User Manual, V 1.7, January 2001

C166S V2

16-Bit Microcontroller

Microcontrollers

Infineon technologies

Never stop thinking.

Edition 2001-01

Published by Infineon Technologies AG,
St.-Martin-Strasse 53,
D-81541 München, Germany

© Infineon Technologies AG 2001.
All Rights Reserved.

Attention please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Infineon Technologies is an approved CECC manufacturer.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office in Germany or our Infineon Technologies Representatives worldwide (see address list).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

C166S V2

16-Bit Microcontroller

Microcontrollers

Never stop thinking.

C166S V2

Revision History:	**2001-01**	V 1.7
Previous Version:	-	

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:

ce.cmd@infineon.com

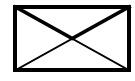


Table of Contents **Page**

1	**Introduction**	9
1.1	Technical Overview	9
1.2	System Description	10
1.2.1	CPU	11
1.2.2	On-Chip Memory Modules	12
1.2.3	Data Management Unit (DMU)	12
1.2.4	Program Memory Unit (PMU)	12
1.2.5	Interrupt and PEC Controller	13
1.2.6	OCDS and JTAG	13
1.2.7	External Bus Controller (EBC)	13
1.2.8	System Control Unit (SCU)	13
1.2.9	Clock Generation Unit (CGU)	14
1.2.10	On-Chip Bootstrap Loader	14
2	**Central Processing Unit**	15
2.1	Register Description Format	17
2.2	CPU Special Function Registers	18
2.3	Instruction Fetch and Program Flow Control	19
2.3.1	Branch Target Addressing Modes	20
2.3.2	Branch Detection and Branch Prediction	22
2.3.3	Sequential and Mispredicted Instruction Flow	24
2.3.3.1	Correctly Predicted Instruction Flow	24
2.3.3.2	Incorrectly Predicted Instruction Flow	26
2.3.4	Atomic and Extend Instructions	27
2.3.5	Code Addressing via Code Segment and Instruction Pointer	28
2.3.6	IFU Control Registers	30
2.3.6.1	The CPU Configuration Register CPUCON1	30
2.3.6.2	The CPU Configuration Register CPUCON2	31
2.4	Use of General Purpose Registers	34
2.4.1	Memory Mapped GPR Banks and the Global Register Bank	36
2.4.2	Local Register Bank	40
2.4.3	Context Switch	40
2.4.3.1	Changing the selected Physical Register Bank	40
2.4.3.2	Context Switching of the Global Register Bank	42
2.5	Data Addressing	45
2.5.1	Short Addressing Modes	46
2.5.2	Long and Indirect Addressing Modes	48
2.5.2.1	Addressing via Data Page Pointer DPP	49
2.5.2.2	DPP Override Mechanism in the C166S V2 CPU	51
2.5.2.3	Long Addressing Mode	52
2.5.2.4	Indirect Addressing Modes	53
2.5.3	DSP Addressing	56
2.5.4	The CoREG Addressing Mode	63

Table of Contents **Page**

2.5.5 The System Stack 64
2.6 Data Processing 68
2.6.1 Data Types 68
2.6.2 Constants 70
2.6.3 16-bit Adder/Subtracter, Barrel Shifter, and 16-bit Logic Unit 70
2.6.4 Bit Manipulation Unit 70
2.6.5 Multiply and Divide Unit 71
2.6.6 The Processor Status Word PSW 74
2.7 Parallel Data Processing 78
2.7.1 Representation of Numbers and Rounding 79
2.7.2 The 16-bit by 16-bit signed/unsigned Multiplier and Scaler 80
2.7.3 Concatenation Unit 80
2.7.4 One-bit Scaler 80
2.7.5 The 40-bit Adder/Subtracter 81
2.7.6 The Data Limiter 81
2.7.7 The Accumulator Shifter 82
2.7.8 The 40-bit Signed Accumulator Register 82
2.7.9 The Repeat Counter MRW 84
2.7.10 The MAC Unit Status Word MSW 85
2.7.11 The MAC Unit Control Word MCW 88
2.8 Dedicated CSFRs 89

3 C166S V2 Memory Organization 91
3.1 Data Organization in Memory 93
3.2 Internal Program Memory 93
3.3 DPRAM, Internal SRAM, and SFR Areas 94
3.3.1 Data Memories 94
3.3.2 Special Function Register Areas 96
3.3.3 IO Area 97
3.3.4 PEC Source and Destination Pointers 97
3.4 External Memory Space 98
3.4.1 Boot and Debug/Monitor Program Memories 98
3.5 Crossing Memory Boundaries 99
3.6 System Stack 99
3.6.1 Data Organization in Global General Purpose Registers 100

4 Instruction Pipeline 103
4.1 Instruction Dependencies in Different Pipeline Stages 104
4.1.1 The General Purpose Registers 104
4.1.2 Indirect Addressing Modes 106
4.1.3 Memory Bandwidth Conflicts 107
4.1.4 CPU-SFRs and the Pipeline 110

5 Interrupt and Exception Handling 117

5.1 Interrupt System and Control . . . 118
5.1.1 General Interrupt System Structure . . . 118
5.1.2 Interrupt Arbitration . . . 120
5.1.3 Interrupt Control . . . 122
5.1.4 Interrupt Vector Table . . . 124
5.1.5 Interrupt Jump Table Cache . . . 125
5.2 Status and Switch Context Control . . . 127
5.2.1 Interrupt Control Functions in the PSW . . . 127
5.2.2 Saving the Status during Interrupt Service . . . 129
5.2.3 Context Switching . . . 130
5.2.4 Fast Bank Switching . . . 131
5.3 Traps . . . 132
5.3.1 Software Traps . . . 132
5.3.2 Hardware Traps . . . 133
5.4 Peripheral Event Controller . . . 138
5.4.1 PEC Control Registers . . . 139
5.4.2 The PEC Source and Destination Pointer . . . 145
5.4.3 PEC Handler Interrupt Actions Summary . . . 147
5.4.4 PEC Channel Assignment and Arbitration . . . 149
5.5 CPU Action Control Unit . . . 151

6 External Bus Controller . . . 153
6.1 Introduction . . . 153
6.2 Timing Principles . . . 154
6.2.1 A Phase . . . 157
6.2.2 B Phase . . . 157
6.2.3 C Phase . . . 157
6.2.4 D Phase . . . 157
6.2.5 E Phase . . . 157
6.2.6 F Phase . . . 158
6.3 Functional Description . . . 158
6.3.1 Configuration Register Overview . . . 158
6.3.2 The EBC MODE Registers EBCMODx . . . 158
6.3.3 The Timing Configuration registers TCONCSx . . . 161
6.3.4 The Function Configuration Registers FCONCSx . . . 163
6.3.5 The Address Window Selection Registers ADDRSELx . . . 164
6.3.5.1 Definition of Address Areas . . . 164
6.3.5.2 Address Window Arbitration . . . 166
6.3.6 Ready Controlled Bus Cycles . . . 167
6.3.6.1 General . . . 167
6.3.6.2 The Synchronous/Asynchronous READY . . . 168
6.3.6.3 Combining the READY function with predefined wait states . . . 168
6.3.7 EBC Idle State . . . 169

6.4 Multi Master Systems . . . 169
6.4.1 External Bus Arbitration . . . 169
6.4.1.1 Initialization of Arbitration . . . 169
6.4.1.2 Arbitration Master Scheme . . . 170
6.4.1.3 Arbitration Slave Scheme . . . 171
6.4.1.4 Locking the Bus . . . 171
6.4.2 Connecting Multimaster Systems . . . 172
6.5 Fastest possible external access . . . 173

7 Instruction Set . . . 175
7.1 Short Instruction Summary . . . 175
7.2 Instruction Set Summary . . . 178
7.3 Instruction Opcodes . . . 192

8 Detailed Instruction Description . . . 205
8.1 Normal Instruction Set . . . 212
8.2 DSP Instruction Set . . . 315
8.3 Instructions for OCDS/ITC injection and System Control . . . 417

9 Summary of CPU/Subsystem Registers . . . 421
9.1 General Purpose Registers (GPRs) . . . 421
9.2 Core Special Function Registers . . . 423
9.2.1 Ordered by Name . . . 423
9.2.2 Ordered by Address . . . 424
9.3 Register Overview Interrupt and Peripheral Event Controller . . . 426
9.3.1 Ordered by Name . . . 426
9.3.2 Ordered by Address . . . 427
9.4 Register Overview External Bus Controller . . . 430
9.4.1 Ordered by Name . . . 430
9.4.2 Ordered by Address . . . 431

10 Keyword Index . . . 433

1 Introduction

C166S V2 is a member of the most recent generation of the popular C166 microcontroller cores. C166S V2 combines high performance with enhanced modular architecture. It was developed to provide easy migration from standard existing C16x to the new C166S V2 core with its impressive DSP performance and advanced interrupt handling. The system architecture inherits successful hardware and software concepts that have been established in the C16x 16-bit microcontroller families. C166 code compatibility enable re-use of existing code. This dramatically reduces the time-to-market for new product development.

The following features position C166S V2 strategically for contemporary and emerging markets for performance-hungry real-time applications:

– High CPU performance. Single clock cycle execution doubles the performance at the same CPU frequency (relative to the performance of the C166).
– Built-in advanced MAC unit dramatically increases DSP performance.
– High Internal Program Memory bandwidth and the instruction fetch pipeline significantly improve program flow regularity and optimize fetches into the execution pipeline.
– Sophisticated Data Memory structure and multiple high-speed data buses provide transparent data access (0 cycles) and broad bandwidth for efficient DSP processing.
– Advanced exceptions handling block with multi-stage arbitration capability yields stellar interrupt performance with extremely small latency.
– Upgraded Peripheral Event Controller supports efficient and flexible DMA features to support a broad range of fast peripherals.
– Highly modular architecture and flexible bus structure provide effective methods of integrating application-specific peripherals to produce customer-oriented derivatives.

This User's Manual describes the new standard C166S V2 core independently from its use for the dedicated product. Differencies to existing standard products are therefore described in the User's Manual (or Target Specification) of the product.

1.1 Technical Overview

– 5-stage execution pipeline
– 2-stage instruction fetch pipeline with FIFO for instruction pre-fetching
– Pipeline with forwarding that controls data dependencies in hardware
– Linear address space for code and data (von Neumann architecture)
– Multiple high bandwidth internal busses for data and instructions
– Enhanced memory map with extended I/O areas
– 16 MBytes total linear address space
– C16x family compatible on-chip special function register area
– Fast multiplication (16-bit x 16-bit) in one CPU clock cycle
– Fast background execution of division (32-bit/16-bit) in 21 CPU clock cycles

– Nearly all instructions executed in one CPU clock cycle
– Enhanced boolean bit manipulation facilities
– Zero cycle jump execution
– Additional instructions to support High Level Language (HLL) and operating systems
– Register-based design with multiple variable register banks
– Two additional fast register banks
– General purpose register architecture
– 16 General-purpose registers (GPRs) for byte operands
– 16 General-purpose registers (GPRs) for integer operands
– Overlapping 8-bit and 16-bit registers
– Opcode fully upward compatible with C166 family
– Variable stack with automatic stack overflow/underflow detection
– High performance branch-, call- and loop processing
– Multiply and accumulate instructions (MAC) executed in one CPU clock cycle
– Extremely short interrupt response time
– "Fast interrupt" and "Fast context switch" features
– Peripheral bus (PDBUS+) with bit protection

1.2 System Description

The basic C166S V2 System consists of the following main units:

- C166S V2 CPU
- On-Chip Data- and Code-Memories
- Data Management Unit (DMU)
- Program Management Unit (PMU)
- Interrupt and Peripheral Event Controller (PEC) Controller
- OCDS and JTAG-Interface
- External Bus Controller (EBC)
- System Control Unit (SCU)
- Clock Generation Unit (CGU)

The powerful C166S V2 core, the peripherals, and the internal memories of the C166S V2 microcontroller are connected to various busses:

- 16-bit high performance system bus
- 16-bit enhanced peripheral bus (PDBUS+)
- 64-bit internal program memory bus
- 16-bit data memory bus

Figure 1-1 shows a typical configuration of a C166S V2-based system.

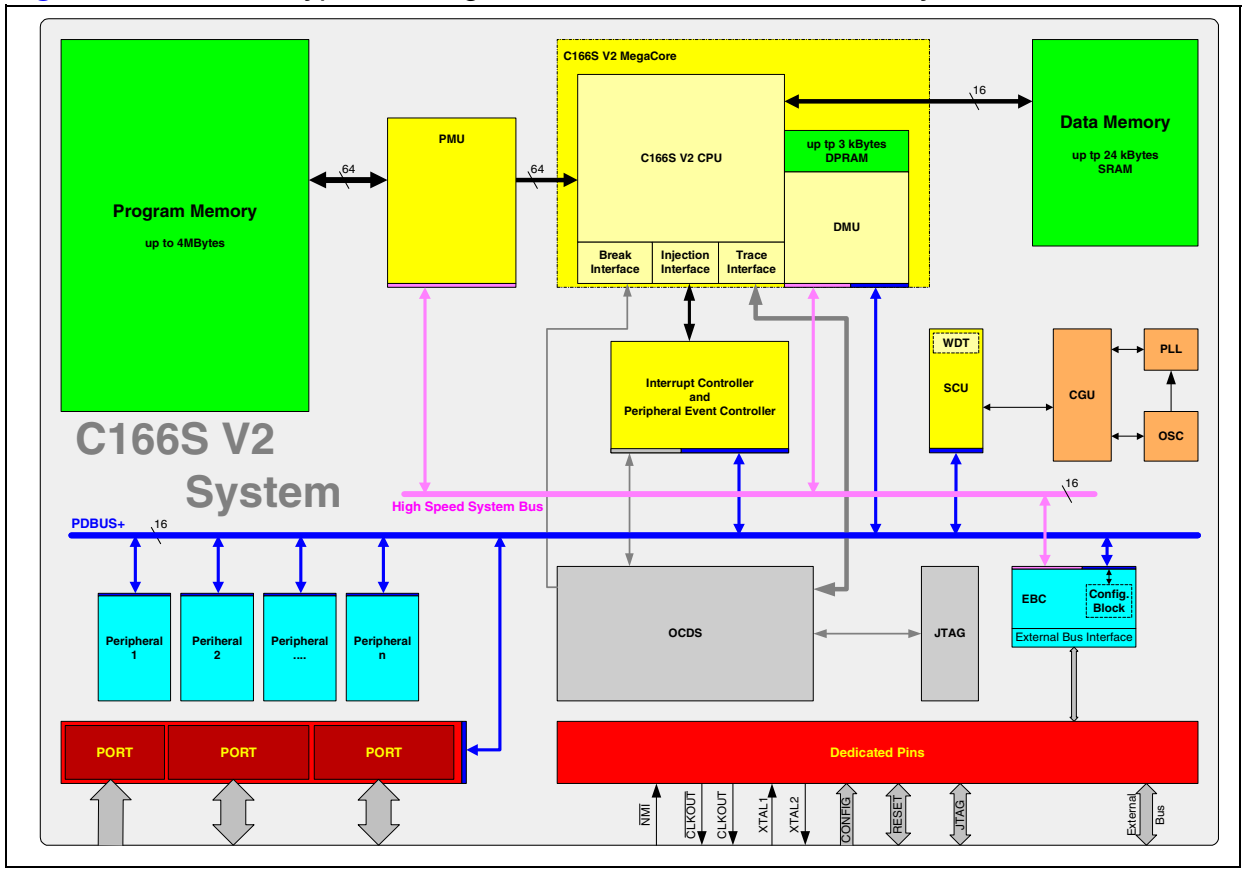


Figure 1-1 C166S V2 System

1.2.1 CPU

- 5-stage execution pipeline
- 2-stage instruction fetch pipeline with FIFO for instruction pre-fetching
- Pipeline with forwarding that controls data dependencies in hardware
- Flexible PMU and DMU with cache capabilities
- Linear address space for code and data (von Neumann architecture)
- Multiple high bandwidth internal busses for data and instructions
- 16 MBytes total linear address space
- Nearly all instructions executed in one CPU clock cycle
- Enhanced boolean bit manipulation facilities
- Zero cycle jump execution
- Additional instructions to support HLL and operating systems
- Register-based design with multiple variable register banks
- Two additional fast register banks
- General purpose register architecture
- 16 General-purpose registers (GPRs) for byte operands
- 16 General-purpose registers (GPRs) for integer operands

– Overlapping 8-bit and 16-bit registers

Multiply Accumulate Unit (MAC)

– Single cycle MAC with zero cycle latency including a 16*16 multiplier plus 40-bit barrel shifter; single clock multiplication is ten times faster than C166 at the same CPU clock
– 40-bit accumulator to handle overflows
– Automatic saturation to 32 bit or rounding included with the MAC instruction
– Fractional numbers supported directly
– One Finite Impulse Response Filter (FIR) tap per cycle with no circular buffer management

1.2.2 On-Chip Memory Modules

– Up to 3 KBytes on-chip dual ported SRAM for DSP data and register banks
– Up to 24 KBytes on-chip internal single ported SRAM module for data storage
– Up to 4 MBytes on-chip memory module for program storage

Note: The on-chip memory configuration may differ from product to product. Product specific on-chip memory configurations are defined in the corresponding product specifications.

1.2.3 Data Management Unit (DMU)

The Data Management Unit (DMU) handles all data transfers external to the core (i.e. external memory or on-chip special function registers on the PDBUS+) and instruction fetches in external memory. The DMU acts as a data mover between the various interfaces. By handling all these interfaces, it incorporates the C166S V2 System Bus. An access prioritization between **E**xternal **B**US **C**ontroller (EBC) accesses from the core and **P**rogram **M**emory **U**nit (PMU) is handled by the DMU. This allows an instruction fetch from external memory in parallel with data access that is not on EBC.

1.2.4 Program Memory Unit (PMU)

The PMU has two basic functions: to provide the CPU with instructions and to provide the CPU (through the DMU) with data located in the Internal Program Memory. The Internal Program Memory is implemented within the PMU.

The instructions requested by the CPU can be located in the Internal Program Memory; in which case, the instructions are requested to the internal memory. Alternatively, they can be located in external memory; in which case, the PMU re-sends this request to the EBC through the DMU, receives the data from the external memory, through the EBC/DMU, and delivers it as the requested instruction to the CPU.

1.2.5 Interrupt and PEC Controller

– 16-Priority-level interrupt system with up to 128 sources on four group levels
– Eight PEC channels with 24-bit source and destination pointers with segment pointer registers
– Enhanced PEC pointers. PEC source pointers and PEC destination pointers can be simultaneously modified
– Independent programmable PEC level and "End of PEC" interrupt

1.2.6 OCDS and JTAG

The OCDS (level 1) provides facilities to the debugger to emulate resources and assist in application program debug. The main features are:

– Real time emulation
– Extended trigger capability including: instruction pointer events, data events on address and/or value, external inputs, counters, chaining of events, timers, etc.
– Software break support
– Break and “break before make” (on IP events only)
– Interrupt servicing during break or monitor mode
– Simple monitor mode or JTAG based debugging through instruction injection

The C166S V2 OCDS is controlled by the debugger[1)] through a set of registers accessible from the JTAG interface. The OCDS also receives informations (such as IP, data, status) from the core for monitoring the activity and generating triggers. Finally, the OCDS interacts with the core through a break interface to suspend program execution, and through an injection interface to allow execution of OCDS generated instructions.

1.2.7 External Bus Controller (EBC)

All external memory accesses are performed by a particular on-chip External Bus Controller (EBC).

1.2.8 System Control Unit (SCU)

The System Control Unit supports all central control tasks and all product specific features. The following typical sub-modules are implemented in this unit:

Reset Control

The reset function is controlled by the reset control unit.

1) Debugger refers to the tool connected to the emulator, and more specifically to the OCDS via the JTAG and which manages the emulation/debugging task.

Power Saving Control

The Power Saving Control block, known from the power management of the C166 derivatives, manages idle mode, power down mode, and sleep mode of the C166S V2.

ID Control

A set of six identification registers is defined for the most important silicon parameters, including the chip manufacturer, the chip type and its properties. These ID registers can be used for automatic test selection.

External Interrupt Control

The C166S V2 System provides asynchronous fast external interrupt inputs.

Central System Control

The central system behavior of the C166S V2 is controlled by this block. The frequency of the PDBUS+ (bus clock) and of all peripherals connected to this bus is programmable according to the maximum physical bus speed and the application requirements. Furthermore, the clock generation status is indicated. Depending on the application state, various security levels (such as protected and unprotected mode) are supported by the security level control state machine.

Watchdog Timer (WDT)

The Watchdog Timer is one of the fail-safe mechanisms that have been implemented to prevent the controller from malfunctioning. However, the Watchdog Timer can detect only long term malfunctions.

1.2.9 Clock Generation Unit (CGU)

The C166S V2 Clock Generation Unit uses either an oscillator or crystal to generate the system clock. A programmable on-chip PLL adds high flexibility to clock generation for the C166S V2.

1.2.10 On-Chip Bootstrap Loader

As in the C166, the on-chip bootstrap loader allows the start code to be moved into internal RAM via the serial interface.

2 Central Processing Unit

C166S V2 CPU represents the third generation of the well known C166 core family. It combines many powerful enhancements with compatibility to the C166 family. The new architecture results in high CPU performance, fast and efficient access to different kinds of memories, and proficient peripheral units integration.

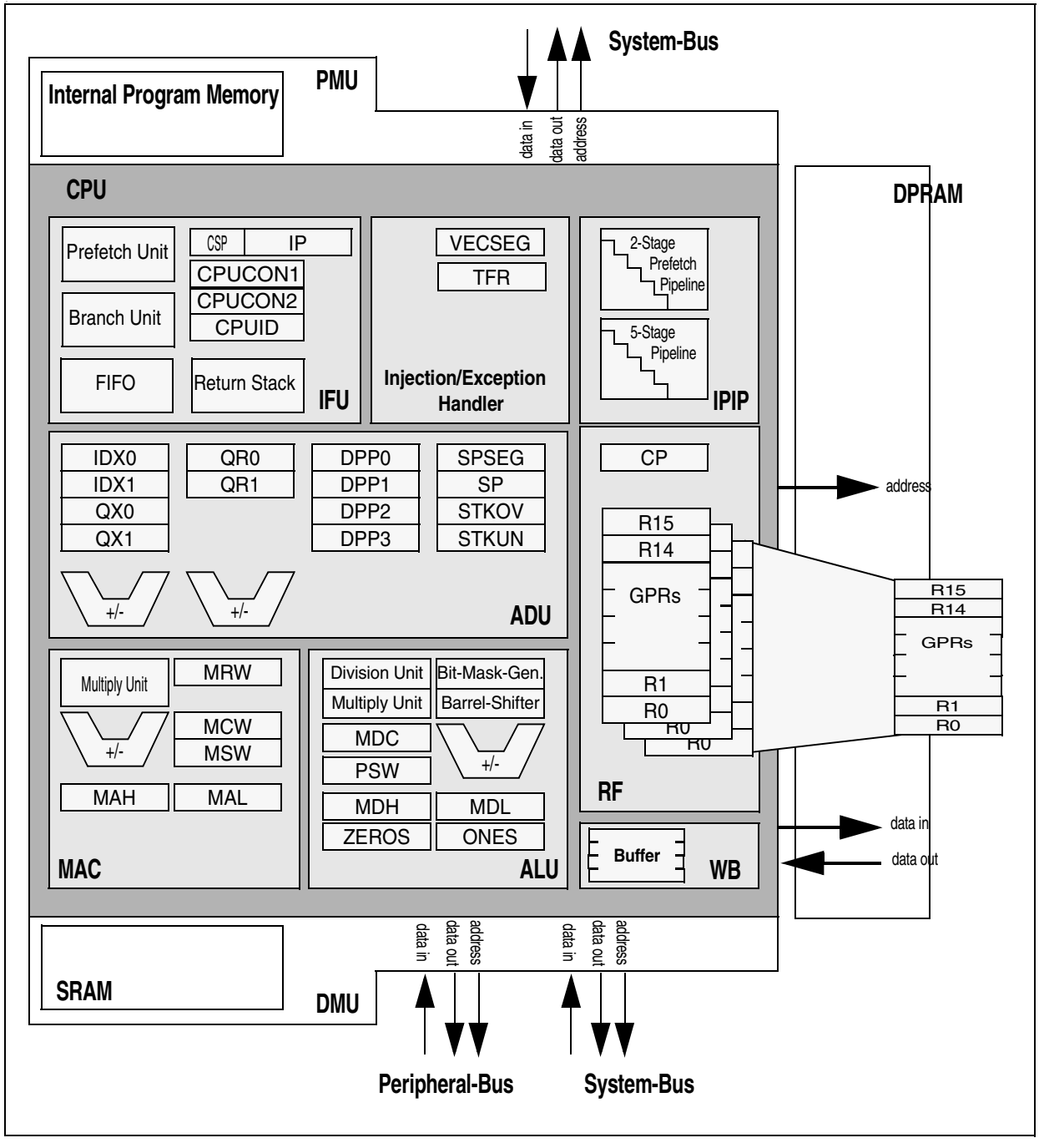


Figure 2-1 CPU Architecture

The new core architecture of the C166S V2 CPU results in higher CPU clock frequencies and reduces the number of clock cycles per executed instruction by half, compared to the C166 core. C166S V2 CPU also integrates a multiplication and accumulation unit which dramatically increases performance of the DSP-intensive tasks.

C166S V2 CPU has eight main units that are listed below. All of these units have been optimized to achieve maximum performance and flexibility.

- High Performance **I**nstruction **F**etch **U**nit (**IFU**)
  - High Bandwidth Fetch Interface
  - Instruction FIFO
  - High Performance Branch-, Call-, and Loop-Processing with instruction flow prediction
- Return Stack
  - Injection/Exception Handler
  - Handling of Interrupt Requests
  - Handling of Hardware Failures
- **I**nstruction **Pip**eline (**IPIP**)
  - Bypassable 2-stage Prefetch Pipeline
  - 5-stage Execution Pipeline
- **A**ddress and **D**ata **U**nit (**ADU**)
  - 16-bit arithmetic unit for address generation
  - DSP address unit with a set of dedicated address- and offset pointers
- **A**rithmetic and **L**ogic **U**nit (**ALU**)
  - 8-bit and 16-bit Arithmetic Unit
  - 16-bit Barrel Shifter
  - Multiplication and Division Unit
  - 8-bit and 16-bit Logic Unit
  - Bit manipulation Unit
- **M**ultiply and **AC**cumulate Unit (**MAC**)
  - 16-bit multiplier with 32-bit result generation[1)]
  - 40-bit Accumulator with 40-bit Barrel Shifter
  - Repeat Control Unit
- **R**egister **F**ile (**RF**)
  - 5-port Register File with three independent register banks
- **W**rite **B**ack **B**uffer (**WB**)
  - 3-entries buffer

1) The same hardware-multiplier is used in the ALU and in the MAC Unit.

2.1 Register Description Format

C166S V2 CPU contains a set of Special Function Register (SFR) and Extended Special Function Registers (ESFR). They are described in the respective chapter of this manual. The example below shows how to interpret the format and notation used to describe SFRs and ESFRs.

A word register looks like this:

REG_NAME
Short Description **SFR(b)/ESFR(b)/XSFR** **Reset Value: aaaa$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	bitfield A					0	0	bit C	bit B	bit A
r	r	r	r	r	r	rwh					r	r	rw	rw	rwh

A byte register looks like this:

REG_NAME
Short Description **SFR(b)/ESFR(b)/XSFR** **Reset Value: aa$_H$**

7	6	5	4	3	2	1	0
0	bitfield A			0	bit C	bit B	bit A
r	rwh			r	rw	rw	rwh

Field	Bits	Type	Description
bitfieldX	[m:n]	type	**Description** value Function off(Default) value Enable Function 1 ... ...
bitX	[n]	type	**Description** 0 Function off(Default) 1 Enable Function

Elements:

REG_NAME	Name of this register
bitX	Name of bit
bitfieldX	Name of bitfield
A16 / A8	Long 16-bit address/Short 8-bit address
SFR(b)/ESFR(b)	Register space (SFR or ESFR (bit addressable) Register)
XSFR	Register located in the internal 4 k IO area

(* *) * *	Register contents after reset	
	'0/1'	: defined value,
	'U'	: unchanged (undefined ('**X**') after power up)
	'?'	: defined by reset configuration
[n]	Bit number	
[m:n]	n	: Bit number first bit of the bitfield
	m	: Bit number of last bit of the bitfield
type	**'r'**	: readable by software
	'w'	: writable by software
	'h'	: writable by hardware
value	**'0/1'**	: defined value,
	'X'	: undefined,
	'0'	: reserved for future purpose, read access delivers 0, must not be set to 1

2.2 CPU Special Function Registers

The core CPU requires a set of CPU Special Function Registers (CSFRs) to maintain the system state information, to control system and bus configuration, and to manage code memory segmentation and data memory paging. The CPU also uses CSFRs to access the General Purpose Registers (GPRs) and the System Stack, to supply the ALU with register-addressable constants, and to support multiply and divide ALU operations.

The access mechanism for these CSFRs in the CPU core is identical to the access mechanism for any other SFR. Since all SFRs can be controlled by any instruction capable of addressing the SFR/CSFR memory space, there is no need for special system control instructions.

However, to ensure proper processor operations, certain restrictions on the user access to some CSFRs must be imposed. For example, the Instruction Pointer (IP) and Code Segment Pointer (CSP) cannot be accessed directly at all. They can only be changed indirectly via branch instructions.

The PSW, SP, and MDC registers can be modified not only explicitly by the programmer, but also implicitly by the CPU during normal instruction processing.

Note: Note that any explicit write request (via software) to an CSFR supersedes a simultaneous modification by hardware of the same register.

Note: All SFRs may be accessed wordwise, or bytewise (some of them even bitwise). Reading bytes from word SFRs is a non-critical operation. Any write operation to a single byte of an CSFR clears the non-addressed complementary byte within the specified CSFR.
Non-implemented (reserved) CSFR bits cannot be modified, and will always supply a read value of 0.

2.3 Instruction Fetch and Program Flow Control

The Instruction Fetch Unit (IFU) pre-fetches and pre-processes instructions to provide a continuous instruction flow. The IFU can fetch simultaneously at least two instructions via a 64-bit wide bus from the Program Management Unit (PMU). The pre-fetched instructions are stored in an instruction FIFO. Pre-processing of branch instructions enables the instruction flow to be predicted. While the CPU is in the process of executing an instruction fetched from the FIFO, the pre-fetcher of the IFU starts to fetch a new instruction at a predicted target address from the PMU. The latency time of this access is hidden by the execution of the instructions which have been buffered in the FIFO before. Even for a non-sequential instruction, execution the IFU can generally provide a continuous instruction flow. The IFU contains two pipeline stages: the Prefetch Stage and the Fetch Stage.

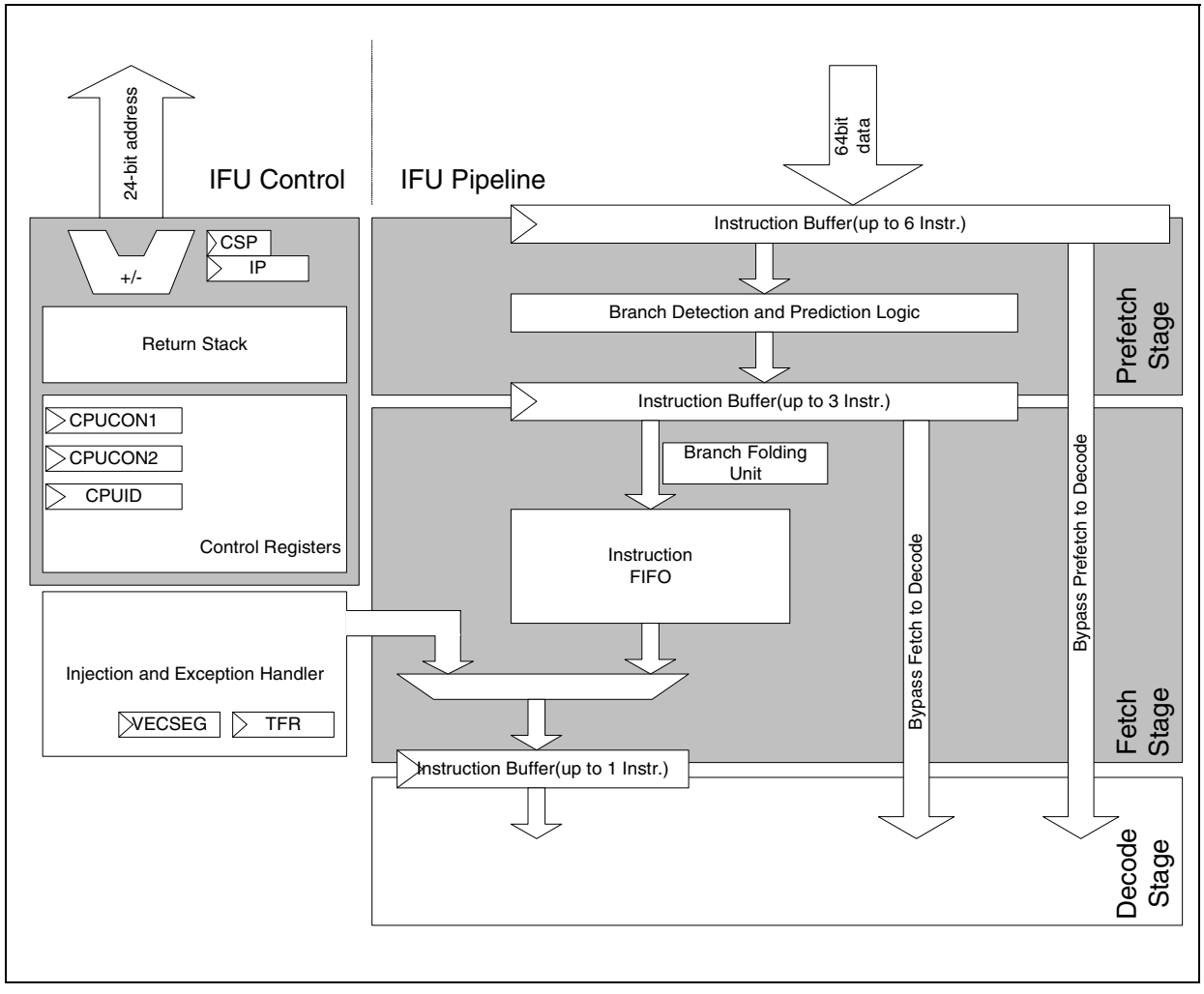


Figure 2-2 IFU Block Diagram

During the pre-fetch stage, the Branch Detection and Prediction Logic analyzes up to three pre-fetched instructions stored in the first Instruction Buffer (up to six instructions). If a branch is detected, then the IFU starts to fetch the next instructions from the PMU according to the prediction rules. After having been analyzed, up to three instructions are stored in the second Instruction Buffer (three instructions) which is the input register of the Fetch Stage.

On the Fetch Stage, the pre-fetched instructions are stored in the instruction FIFO. The Branch Folding Unit (BFU) allows processing of branch instructions in parallel with preceding instructions. To achieve this the BFU pre-processes and re-formats the branch instruction. First, BFU defines (calculates) the absolute target address. This address—after being combined with branch condition and branch attribute bits—is stored in the same FIFO step as the preceding instruction. The target address is also used to pre-fetch the next instructions.

For the Execution Pipeline, both instructions are fetched from the FIFO again and are executed in parallel. If the instruction flow was predicted incorrectly (or FIFO is empty), the two stages of the IFU can be bypassed.

Note: Pipeline behavior in case of a incorrectly predicted instruction flow is described in the following sections.

2.3.1 Branch Target Addressing Modes

The target address and the segment of jump or call instructions can be specified by several addressing modes. The Instruction Pointer register (IP) may be updated using relative, absolute, or indirect modes. The Code Segment Pointer register (CSP) can be updated using an absolute value only. A special mode is provided to address the interrupt and trap jump vector table which resides in the lowest portion of the code segment selected by the VECSEG register contents.

Table 2-1 Branch Target Addressing Modes

Mnemonic	Target Address	Target Segment	Valid Address Range
caddr	(IP) = caddr	-	caddr = 0000_H...$FFFE_H$
rel	(IP) = (IP) + 2*rel (IP) = (IP) + 2*(rel+1)	- -	rel = 00_H...$7F_H$ rel = 80_H...FF_H
[Rw]	(IP) = (Rw)	-	Rw w = 0...15
seg	-	(CSP) = seg	seg = 0...255(3)
#trap7	(IP) = 0000_H + VECSC*trap7	(CSP) = VECSEG	trap7 = 00_H...$7F_H$

caddr: Specifies an absolute 16-bit code address within the current segment. Branches **MAY NOT** be taken to odd code addresses. Therefore, the least significant bit of 'caddr' is not used.

rel: This mnemonic represents an 8-bit signed word offset address relative to the current Instruction Pointer contents, which points to the instruction after the branch instruction. Depending on the offset address range, both forward ('rel'= 00_H to $7F_H$) and backward ('rel'= 80_H to FF_H) branches are possible. The branch instruction itself is repeatedly executed, when 'rel' = '-1' (FF_H) for a word-sized branch instruction, or 'rel' = '-2' (FE_H) for a double-word-sized branch instruction.

[Rw]: In this case, the 16-bit branch target instruction address is determined indirectly by the contents of a word GPR. In contrast to indirect data addresses, indirectly specified code addresses are **NOT** calculated via additional pointer registers (eg. DPP registers). Branches **MAY NOT** be taken to odd code addresses. Therefore, the least significant bit of 'caddr' is not used.

seg: Specifies an absolute code segment number. The C166S V2 CPU supports 256 different code segments, so only the eight lower bits (respectively) of the 'seg' operand value are used to update the CSP register.

#trap7: Specifies a particular interrupt or trap number for branching to the corresponding interrupt or trap service routine via a jump vector table. Trap numbers from 00_H to $7F_H$ can be specified to access any double word code location within the address range xx'0000_H...xx'$15D4_H$ (depending of VECSC) in the selected code segment (see VECSEG, i.e. the interrupt jump vector table), please refer to **Section 5.1.4**.

2.3.2 Branch Detection and Branch Prediction

The Branch Detection Unit pre-processes instructions and classifies detected branches. Depending on the branch class, the Branch Prediction Unit predicts the program flow using the rules in the following table:.

Table 2-2 Branch Target Addressing Modes

Instruction Classes	Instructions	Prediction
Branch instructions with user programmable branch prediction	JMPA- xcc,caddr JMPA+ xcc,caddr CALLA- xcc, caddr CALLA+ xcc,caddr	The User can specify whether the branch should be taken
Branch instructions with branch prediction defined by Assembler	JMPA xcc,caddr CALLA xcc, caddr	Assembler defines whether the branch should be taken based on the jump condition.
Inter-segment branch instructions	JMPS seg, caddr CALLS seg,caddr	The branch is always taken.
Indirect branch instructions	JMPI cc,[Rw] CALLI cc,[Rw]	The branch is taken only if the branch is unconditional.
Relative branches instructions with condition code	JMPR cc,rel	The branch is taken if it is unconditional or if the branch is a backward branch.
Relative branch instructions without condition code	CALLR rel	The branch is always taken.
Branch instructions with bitcondition	JB bitaddr,rel JBC bitaddr,rel JNB bitaddr,rel JNBS bitaddr,rel	The branch is taken if it is a backward branch. Forward branches are always not taken.
Return instructions	RET RETS RETP RETI	The branch is always taken.

Note: For JMPA+/- and CALLA+/- instructions, a static user programmable prediction scheme is used. If bit 8 (’a’) of the instruction long word is cleared, the branch is assumed ‘taken.’ If it is set, the branch is assumed ‘not taken’. The user controls value of bit 8 by entering ’+’ or ’-’ in the instruction mnemonics. This bit can be also set/cleared by the Assembler for JMPA and CALLA instructions depending on the jump condition.

Note: For JMPA instruction, a pre-fetch hint bit is used (the instruction bit 9 = I). This bit is required by the fetch unit to deal efficiently with short backward loops. It must be set if 0 < IP_jmpa - IP_target <= 32, where IP_jmpa is the address of the JMPA instruction and IP_target is the target address of the JMPA. Otherwise, bit 9 must be cleared.

2.3.3 Sequential and Mispredicted Instruction Flow

Because passing through one pipeline stage takes at least one clock cycle, any isolated instruction takes at least five clock cycles to be completed. Pipelining, however, allows parallel (i.e. simultaneous) processing of up to five instructions (with branches up to six instructions). Therefore, most of the instructions appear to be processed during one clock cycle as soon as the pipeline has been filled once after reset.

The pipelining increases the average instruction throughput considered over a certain period of time. In this manual, any execution time specification always refers to the average instruction execution time due to pipelined parallel processing.

2.3.3.1 Correctly Predicted Instruction Flow

Figure 2-3 and **Figure 2-4** show the continuous execution of instructions in principal under the assumption of a fast (0 wait states) Program Memory. In this example, most of the instructions are executed in one CPU cycle while Instruction I_{n+6} takes two CPU cycles for the execution. I_{n+6} is a general example for multicycle instructions (two cycles instruction in this case).

The instructions are fetched from the Instruction FIFO while the IFU pre-fetches the next instructions to fill the FIFO. The Instruction FIFO is being filled with new instructions while the previously stored instructions are being fetched from the FIFO to be executed in the CPU. As long as the instruction flow is correctly predicted by the IFU, both processes are independent.

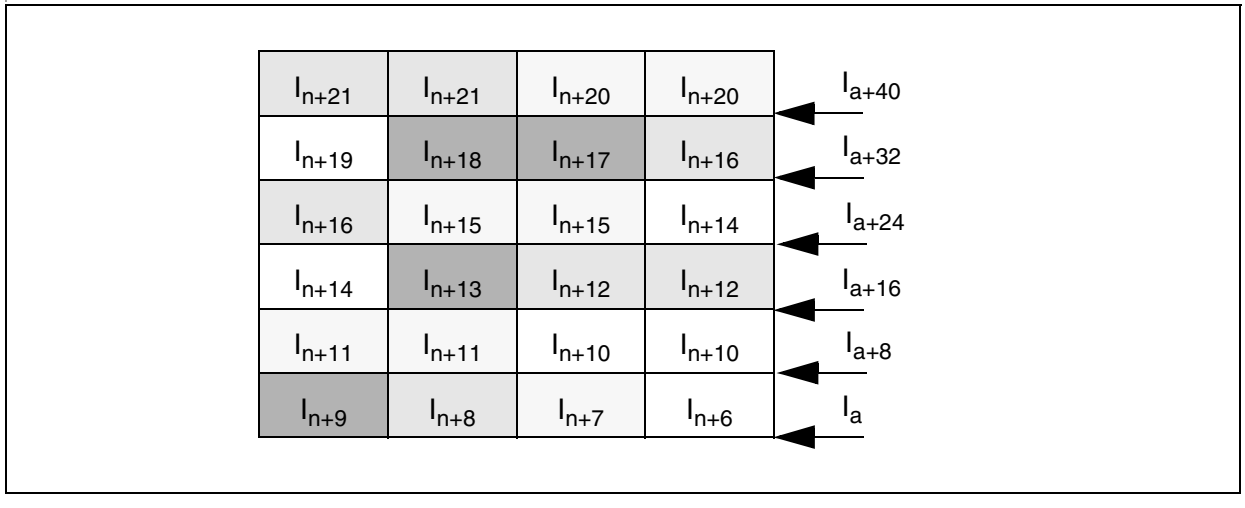


Figure 2-3 Program Memory Contents for Figure 2-4

The diagram shows the sequential instruction flow through the different pipeline stages. While the Prefetcher is prefetching the instruction from the PMU, the processing pipeline is filled with instructions fetched out of the FIFO. In this example with a fast Internal Program Memory, the Prefetcher is able to fetch more instructions than the processing pipeline can execute. In T_{n+4}, the FIFO and prefetch buffer are filled and no further

instructions can be prefetched. The PMU address stays stable (T_{n+4}) until a whole 64-bit double word can be buffered (T_{n+7}) in the 96-bit Prefetch buffer again.

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}	T_{n+6}	T_{n+7}	T_{n+8}
PMU Address	I_{a+16}	I_{a+24}	I_{a+32}	I_{a+40}	I_{a+40}	I_{a+40}	I_{a+40}	I_{a+48}	I_{a+48}
PMU Data 64bit	I_{d+1}	I_{d+2}	I_{d+3}	I_{d+4}	I_{d+5}	I_{d+5}	I_{d+5}	I_{d+5}	I_{d+7}
PREFETCH 96 bit Buffer	I_{n+6} … I_{n+9}	I_{n+9} … I_{n+11}	I_{n+12} I_{n+13}	I_{n+14} I_{n+15}	I_{n+15} … I_{n+19}	I_{n+15} … I_{n+19}	I_{n+16} … I_{n+19}	I_{n+17} … I_{n+19}	I_{n+18} … I_{n+21}
FETCH Instruction Buffer	I_{n+5}	I_{n+6} I_{n+7} I_{n+8}	I_{n+9} I_{n+10} I_{n+11}	I_{n+12} I_{n+13}	I_{n+14}	-	I_{n+15}	I_{n+16}	I_{n+17}
FIFO contents	I_{n+3} … I_{n+5}	I_{n+4} … I_{n+8}	I_{n+5} … I_{n+11}	I_{n+6} … I_{n+13}	I_{n+7} … I_{n+14}	I_{n+7} … I_{n+14}	I_{n+8} … I_{n+15}	I_{n+9} … I_{n+16}	I_{n+10} … I_{n+17}
Fetch from FIFO	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+7}	I_{n+7}	I_{n+8}	I_{n+9}	I_{n+10}	I_{n+11}
DECODE	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}	I_{n+7}	I_{n+8}	I_{n+9}	I_{n+10}
ADDRESS	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}	I_{n+7}	I_{n+8}	I_{n+9}
MEMORY	I_{n+1}	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}	I_{n+7}	I_{n+8}
EXECUTE	I_n	I_{n+1}	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}	I_{n+7}
WRITE BACK		I_n	I_{n+1}	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}

Figure 2-4 Sequential Instruction Execution

2.3.3.2 Incorrectly Predicted Instruction Flow

If the CPU detects that the IFU made an incorrect prediction of the instruction flow, then the pipeline stages and the Instruction FIFO containing the wrong prefetched instructions are canceled. The entire instruction fetch must be restarted at the correct point of the program. **Figure 2-5** and **Figure 2-6** show the behavior in the case of incorrectly predicted instruction flow (0- wait states Internal Program Memory).

During the cycle T_n, the CPU detects an incorrectly prediction case which leads to a canceling of the pipeline. The new address is transferred to the PMU in T_{n+1} which delivers the first data in the next cycle T_{n+2}. But, the target instruction crosses the 64-bit memory boundary and a second fetch in T_{n+3} is required to get the entire 32-bit instruction. In T_{n+4}, the Prefetch Buffer contains two 32-bit instructions while the first instruction I_m is directly forwarded to the Decode stage.

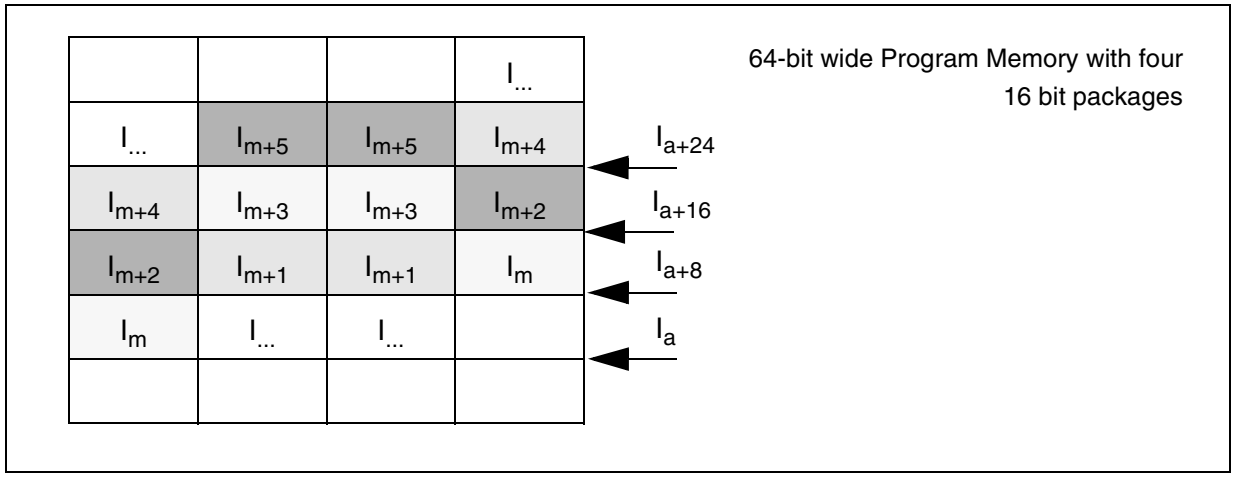


Figure 2-5 Program Memory Contents for Figure 2-6

The prefetcher is now restarted and prefetches further instructions. In T_{n+5}, the instruction I_{m+1} is forwarded from the Fetch Instruction Buffer directly to the Decode stage as well. The Fetch row shows all instructions in the Fetch Instruction Buffer and the instructions fetched from the Instruction FIFO. The instruction I_{m+3} is the first instruction fetched from the FIFO during T_{n+6}. During the same cycle, instruction I_{m+2} was still forwarded from the Fetch Instruction Buffer to the Decode stage.

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}	T_{n+6}	T_{n+7}	T_{n+8}
PMU Address	I...	I_a	I_{a+8}	I_{a+16}	I_{a+24}	I...	I...	I...	I...
PMU Data 64bit	I...		I_d	I_{d+1}	I_{d+2}	I_{d+3}	I...	I...	I...
PREFETCH 96-bit Buffer	I...				I_m I_{m+1}	I_{m+2} I_{m+3}	I_{m+4} I_{m+5}	I...	I...
FETCH Instruction Buffer	I_{next+2}					I_{m+1}	I_{m+2} I_{m+3}	I_{m+4} I_{m+5}	I...
Fetch from FIFO							I_{m+3}	I_{m+4}	I_{m+5}
DECODE	I_{next+1}				I_m	I_{m+1}	I_{m+2}	I_{m+3}	I_{m+4}
ADDRESS	I_{next}					I_m	I_{m+1}	I_{m+2}	I_{m+3}
MEMORY	I_{branch}						I_m	I_{m+1}	I_{m+2}
EXECUTE	I_n	I_{branch}						I_m	I_{m+1}
WRITE BACK		I_n	I_{branch}						I_m

Figure 2-6 Incorrectly Predicted Instruction Flow

2.3.4 Atomic and Extend Instructions

The atomic and extend instructions (ATOMIC, EXTR, EXTP, EXTS, EXTPR, EXTSR) disable the standard and PEC interrupts and class A traps until completion of the immediately following sequence of instructions. The number of instructions in the sequence may vary from 1 to 4. It is coded in the 2-bit constant field #irang2 and takes values from 0 to 3. The EXTended instructions additionally change the addressing mechanism during this sequence (see instruction description).

ATOMIC and EXTended instructions become active immediately, so no additional NOPs are required. All instructions requiring multi cycles or hold states for execution are considered to be one instruction. The ATOMIC and EXTended instructions can be used with any instruction type.

Note: If a class B trap interrupt occurs during an ATOMIC or EXTended sequence, then the sequence is terminated, an interrupt lock is removed, and the standard condition is restored before the trap routine is executed. The remaining instructions of the terminated sequence executed after returning from the trap routine will run under standard conditions.

Note: Certain precautions are required when using nested ATOMIC and EXTended instructions. There is only one counter to control the length of the sequence, i.e.

issuing an ATOMIC or EXTended instruction within a sequence will reload the counter with the value of the new instruction.

2.3.5 Code Addressing via Code Segment and Instruction Pointer

The C166S V2 CPU provides a total addressable memory space of 16 MBytes. This address space is arranged as 256 segments of 64 Kilobytes each. A dedicated 24-bit code address pointer is used to access the memories for instruction fetches. This pointer has two parts: an 8-bit code segment pointer CSP and a 16-bit offset pointer called Instruction Pointer (IP). The concatenation of the CSP and IP results directly in a correct 24-bit physical memory address.

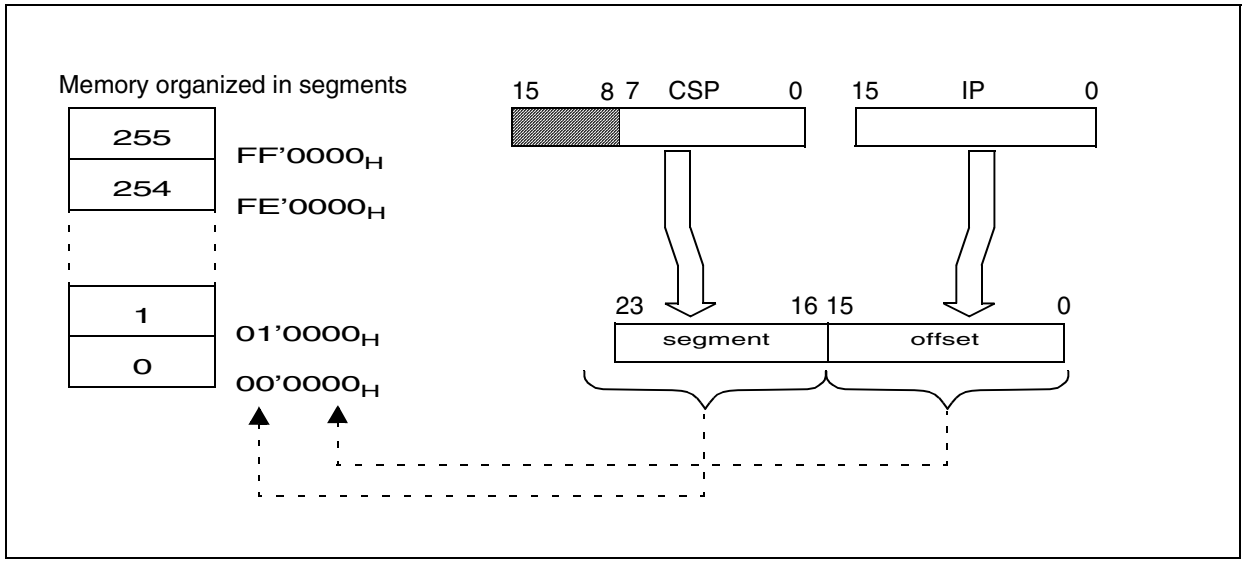


Figure 2-7 Addressing via the Code Segment- and Instruction Pointer

The Instruction Pointer IP

This register determines the 16-bit intra-segment address of the currently fetched instruction within the code segment selected by the CSP register. The IP register is not mapped into the C166S V2 CPU's address space, and thus it is not directly accessible by the programmer. The IP can be modified indirectly via the stack by return instructions. The IP register is implicitly updated by the C166S V2 CPU for branch instructions and after instruction fetch operations.

IP

Instruction Pointer (not addressable) Reset Value: 0000_H

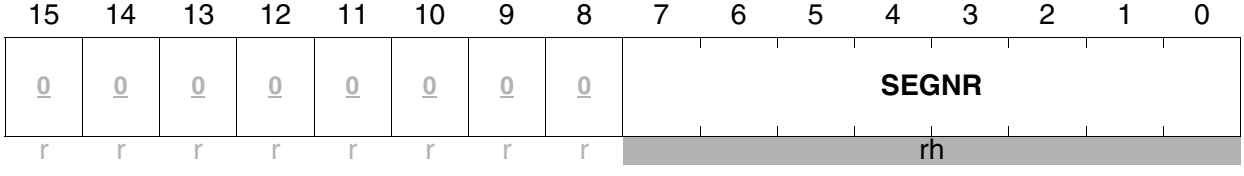
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IP															0
h															-

Field	Bits	Type	Description
IP	[15:1]	h	Specifies the intra segment offset from which the current instruction is to be fetched. IP refers to the current segment <SEGNR>.
0	[0]	-	IP is always word-aligned

The Code Segment Pointer CSP

This non-bit addressable register selects the code segment being used at run-time to access instructions. The lower 8 bits of register CSP select one of up 256 segments of 64 Kilobytes each, while the higher 8 bits are reserved for future use. The reset value is specified by the contents of the VECSEG register (**Section 5.1.4**).

CSP
Code Segment Pointer **SFR** **Reset Value: 0000_H**

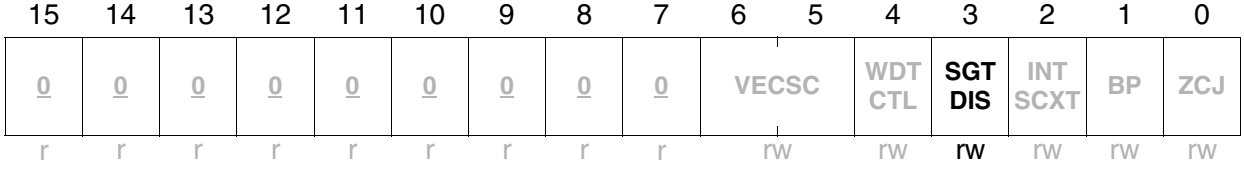


Field	Bits	Type	Description
SEGNR	[7:0]	rh	Specifies the code segment from which the current instruction is to be fetched.

The actual code memory address is generated by direct extension of the 16-bit contents of the IP register by the lower byte of the CSP register as shown in the figure below. The CSP register can be only read and may not be written by data operations.

There are two modes: segmented and non-segmented. The mode is selected with the **SGTDIS** bit in the CPUCON1 register. After reset, the segmented mode is selected.

CPUCON1
CPU Control Register 1 **SFR** **Reset Value: 0000_H**



*Note: For a summary of the CPUCON1 register, please refer to **Section 2.3.6**.*

Field	Bits	Type	Description
SGTDIS	[3]	rw	**Segmentation Disable/Enable Control** 0 Segmentation enabled 1 Segmentation disabled

Segmented Mode

The CSP is modified either directly by the JMPS and CALLS instructions, or indirectly via the stack by the RETS and RETI instructions.
Upon the acceptance of an interrupt or the execution of a software TRAP instruction, the CSP register is automatically loaded with the segment address of the vector location.

Non-Segmented Mode

In non-segmented mode, the CSP is fixed to the CSP value of the instruction that disabled the segmentation. It is no longer possible to modify the CSP either directly by the JMPS or CALLS instructions or indirectly via the stack by the RETS (RETI) instruction.

In case of interrupt processing or a software TRAP instruction, the CSP register is automatically loaded with the segment address of the vector location (VECSEG).

Note: For the correct execution of interrupt tasks, the contents of VECSEG must be the same as the segment selected by the current value of CSP, i.e. the vector table must be located in the segment pointed by the CSP.

Note: For Single Chip Mode, the contents of the CSP register are significant for internal Program Memories accesses.

2.3.6 IFU Control Registers

2.3.6.1 The CPU Configuration Register CPUCON1

This register is used to configure the C166S V2 CPU. Most bits of this register enable dedicated features of the Instruction Fetch Unit (IFU). CPICON1 may not exist in future product derivatives.

CPUCON1
CPU Control Register 1 **SFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	VECSC		WDT CTL	SGT DIS	INT SCXT	BP	ZCJ
r	r	r	r	r	r	r	r	r	rw		rw	rw	rw	rw	rw

Field	Bits	Type	Description
VECSC	[6:5]	rw	**Scaling factor of Vector Table** 00 Space between two vectors is 2 words 01 Space between two vectors is 4 words 10 Space between two vectors is 8 words 11 Space between two vectors is 16 words
WDTCTL	[4]	rw	**Configuration of Watch Dog Timer** 0 DISWDT executable until End of Init[1)] 1 DISWDT/ENWDT always executable
SGTDIS	[3]	rw	**Segmentation Disable/Enable Control** 0 Segmentation enabled 1 Segmentation disabled
INTSCXT	[2]	rw	**Enable Interruptibility of Switch Context** 0 Switch context is not interruptible 1 Switch context is interruptible
BP	[1]	rw	**Enable Branch Prediction Unit** 0 Branch prediction disabled 1 Branch prediction enabled
ZCJ	[0]	rw	**Enable Zero Cycle Jump function** 0 Zero cycle jump function disabled 1 Zero cycle jump function enabled

1) The DISWDT (executed after EINIT) and ENWDT instructions are internally converted in a NOP instruction

Note: Register CPUCON1 is only changeable in supervisor mode. Supervisor mode is finished by executing the EINIT instruction.

2.3.6.2 The CPU Configuration Register CPUCON2

This register is used to configure the C166S V2 CPU. It is an extension of the CPUCON1 register. This register is implemented for test purposes only in the first C166S V2 demonstration devices. This register will not be implemented in production devices.

CPUCON2
CPU Control Register **SFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIFODEPTH				FIFOFED		BYPPF	BYPF	EIOIAEN	STEN	LFIC	OVRUN	RETST	FASTBL[1)]	0	SL
rw				rw		rw	rw	rw	rw	rw	rw	rw	rw	r	rw

1) reserved

Field	Bits	Type	Description
FIFODEPTH	[15:12]	rw	**FIFO Depth configuration** 0000 No FIFO (entries) 0001 One FIFO entry ... . 1000 Eight FIFO entries 1001 reserved ... ... 1111 reserved
FIFOFED	[11:10]	rw	**FIFO Fed configuration** 00 FIFO disabled 01 FIFO filled with up to one instruction per cycle 10 FIFO filled with up to two instructions per cycle 11 FIFO filled with up to three instruction per cycle
BYPPF	[9]	rw	**Prefetch Bypass control** 0 Bypass path from prefetch to decode disabled 1 Bypass path from prefetch to decode available
BYPF	[8]	rw	**Fetch Bypass control** 0 Bypass path from fetch to decode disabled 1 Bypass path from fetch to decode available
EIOIAEN	[7]	rw	**Early IO Injection Acknowledge Enable** 0 Injection acknowledge by destructive read not guaranteed 1 Injection acknowledge by destructive read guaranteed
STEN[1)]	[6]	rw	**Stall Instruction Enable** 0 Stall Instruction disabled 1 Stall Instruction enabled
LFIC	[5]	rw	**Linear Follower Instruction Cache** 0 Linear Follower Instruction Cache disabled 1 Linear Follower Instruction Cache enabled
OVRUN	[4]	rw	**Pipeline control** 0 Overrun of pipeline bubbles not allowed 1 Overrun of pipeline bubbles allowed
RETST	[3]	rw	**Enable return Stack** 0 Return Stack is disabled 1 Return Stack is enabled

Field	Bits	Type	Description
FASTBL[2)]	[2]	rw	**Enables the fast injection of block transfers** 0 Direct injection disabled 1 Direct injection enabled
SL	[0]	rw	**Enables short loop mode** 0 Short loop mode disabled 1 Short loop mode enabled

1) enables dedicated stall debug instructions:
`STALLAM` d_a, h_a, d_m, h_m `Opcode: 44` $d_a h_a d_m h_m$
`STALLEW` d_e, h_e, d_w, h_w `Opcode: 45` $d_e h_e d_w h_w$ `d and h are 6 bit each`
Stalls the corresponding pipeline stage after *d* cycles for *h* cycles.

2) The FASTBL bit is implemented, but reserved. So do not use it. The block feature is implemented in the CPU, but not used by the Interrupt and Injection Unit.

Note: Register CPUCON2 is changeable in supervisor mode only. Supervisor mode is finished by executing the EINIT instruction.

2.4 Use of General Purpose Registers

The C166S V2 CPU uses several banks of sixteen dedicated registers R0, R1, R2... R15, called General Purpose Registers (GPR), which can be accessed in one CPU cycle. The GPRs are the working registers of the arithmetic and logic units and many also serve as address pointers for indirect addressing modes.

There are several banks of GPRs which are memory mapped and two special banks which are not memory-mapped.

The banks of the memory-mapped GPRs are located in the internal DPRAM. One bank uses a block of 16 consecutive words. A Context Pointer (CP) register determines the base address of the current selected bank. Because of the required number of access ports and access time, the GPRs located in the DPRAM cannot be accessed directly. To get the required performance, the GPRs are cached in a 5-port register file for high speed GPR accesses.

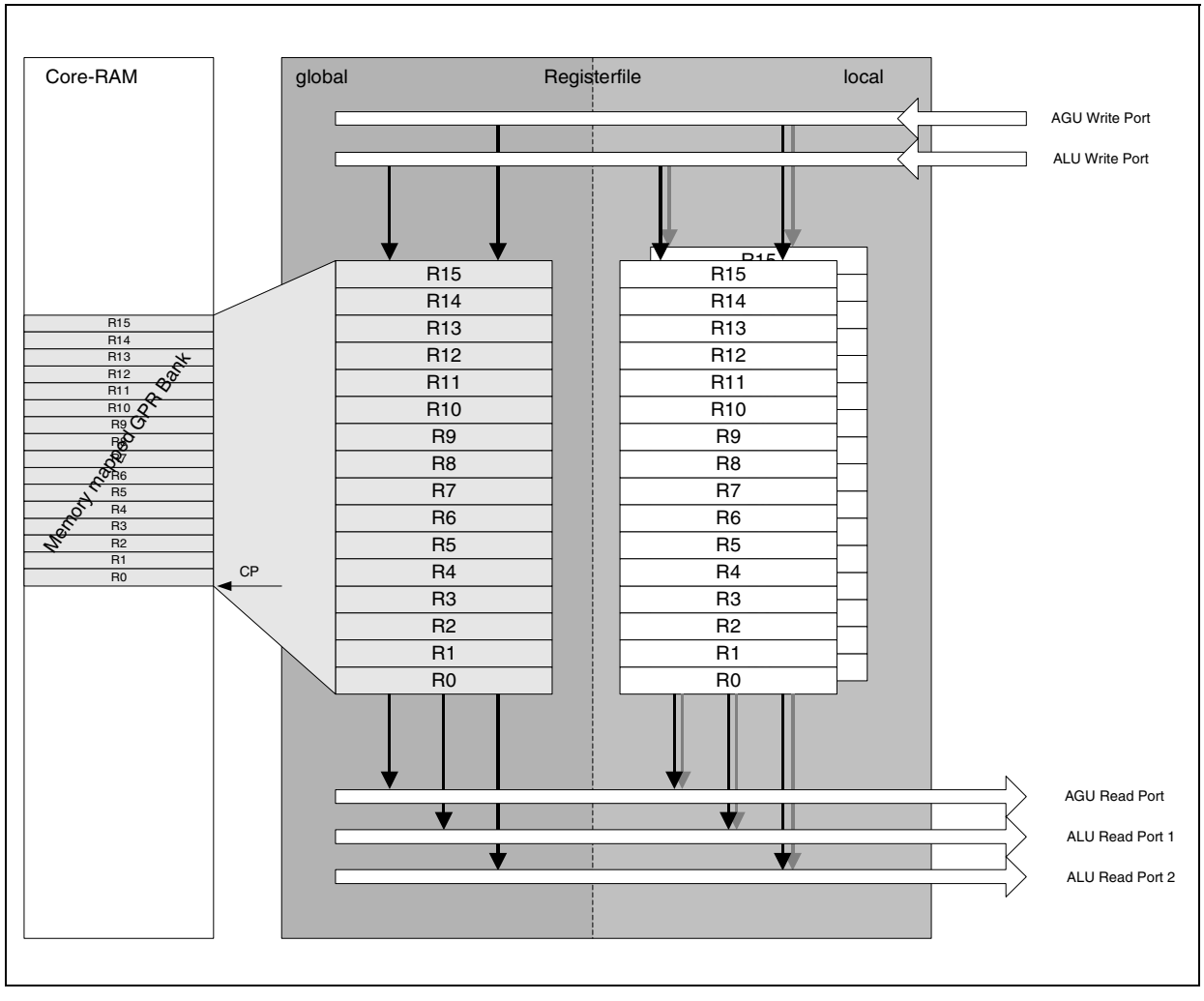


Figure 2-8 Register File

The register file is split into three independent physical register banks. Because of behavior differences, the banks can be distinguished as global and local register banks. There are two local and one global register bank.

The memory-mapped GPR bank selected by the current CP is always cached in the global register bank. Only one memory-mapped GPR bank can be cached at the time. In the case of a context switch, the cache contents must be sequentially saved and restored.

Note: The global register bank is the equivalent of the memory-mapped GPR bank of the C166 family which is selected by the context pointer CP.

To support a very fast context switch for time-critical tasks, two independent not memory mapped GPR banks are available. They are physically and logically located in the two special local register banks. They cannot be accessed via a 24-bit physical memory address.

Only one of the three physical register banks can be activated at the same time. The bank selection is controlled by the **BANK** bitfield of the PSW. The BANK bitfield can be changed explicitly by any instruction which writes to the PSW, or implicitly by a RETI instruction, an interrupt or hardware trap. In case of an interrupt, the selection of the register bank is configured in the Interrupt Controller ITC. Hardware traps always use the global register bank.

2.4.1 Memory Mapped GPR Banks and the Global Register Bank

The C166S V2 CPU uses the global register bank to cache an active memory-mapped GPR bank selected by the Context Pointer (CP). The CP register value determines the address of the first General Purpose Register (GPR) within the DPRAM of up to 16 wordwide and/or bytewide GPRs and selects the memory area which is automatically cached in the global register bank.

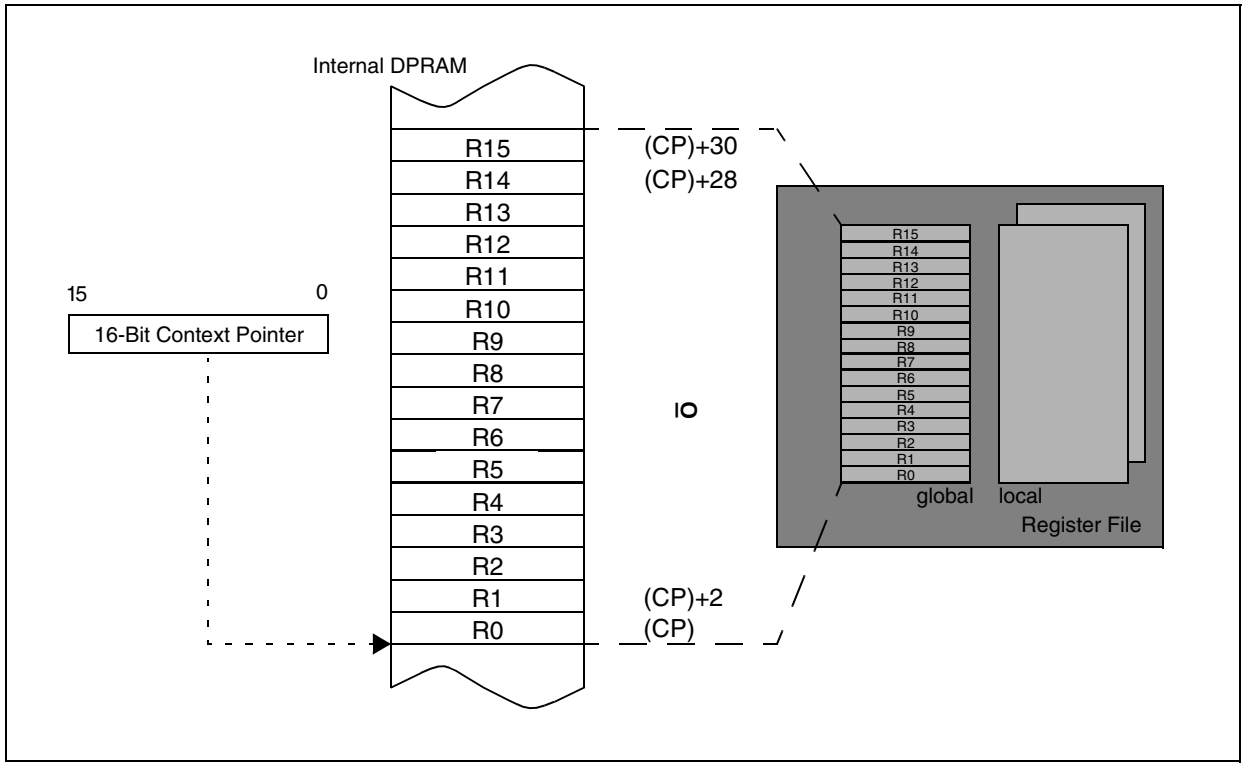


Figure 2-9 Register Bank Selection via Register CP

The General Purpose Registers of a global register bank are **memory-mapped**. The behavior is identical with a cache in which the CP is used as a tag. If the global register bank is activated, the cache will be validated before further instructions are executed. After validation, all further accesses to the GPRs are redirected to the global register bank. If the global register bank is activated, there are three possible ways to access the global register bank:

Short 4-Bit GPR Addresses (mnemonic: Rw or Rb) specify addresses relative to the memory location pointed by the contents of the CP register, i.e. the base of contents of the current global register bank. Both byte and word GPR accesses are possible. The short 4-bit GPR address is logically added to the contents of register CP in the case a byte (Rb) GPR address is specified, or multiplied by two and then added to CP; in case of a word (Rw) GPR address (see figure below).

Note: If GPRs are used as indirect address pointers, they are always accessed wordwise.

For some instructions, only the first four GPRs can be used as indirect address pointers. These GPRs are specified via short 2-bit GPR addresses. The respective physical address calculation is identical with the one for the short 4-bit GPR addresses.

Short 8-Bit Register Addresses (mnemonic: reg or bitoff) within a range from $F0_H$ to FF_H interpret the four least significant bits as short 4-bit GPR addresses, while the four most significant bits are ignored. The respective physical GPR address is calculated similar to the short 4-bit GPR addresses. For single bit GPR accesses, the GPR's word address is calculated in the same way. The accessed bit position within the word is specified by a separate additional 4-bit value.

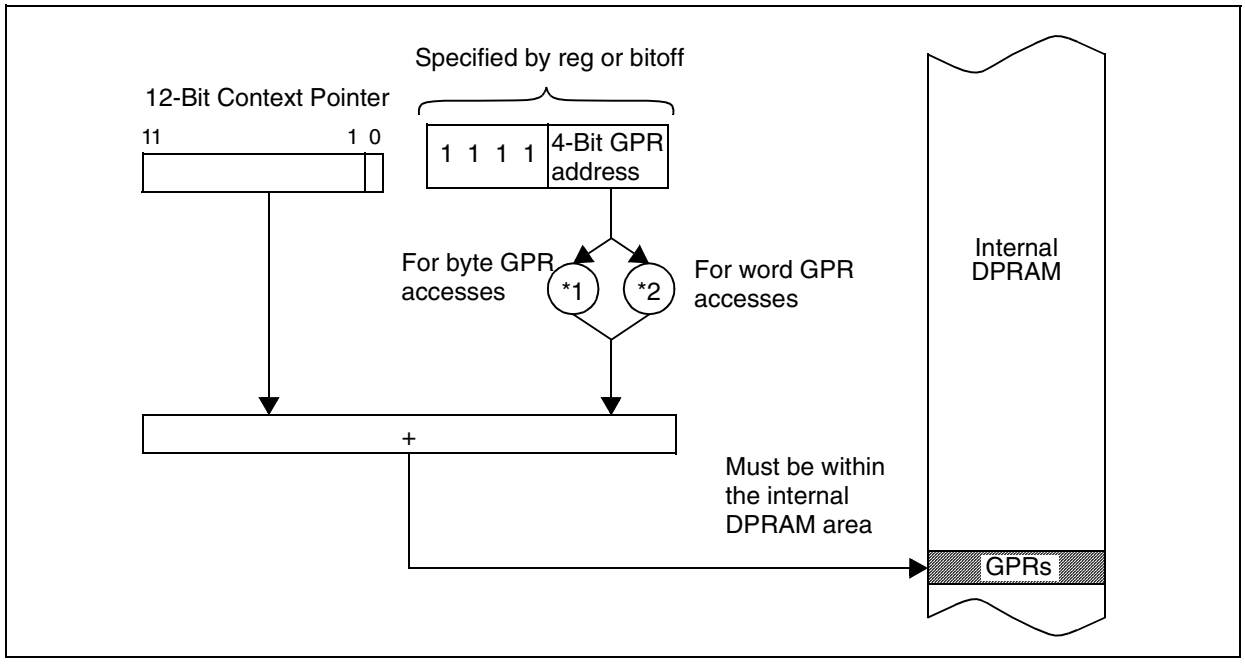


Figure 2-10 Implicit CP Use by logical Short GPR Addressing Modes

24-Bit Memory Addresses can be directly used to access GPRs. In this case, the CPU immediately starts the memory access. At the same time, a hit detection logic checks if the accessed memory location is cached in the global register bank. In case of a cache hit, an additional global register bank read access is initiated. The data that is read from cache will be used and the data that is read from memory will be discarded. This leads to a delay of one CPU cycle (MOV R4,**mem** [CP<=mem<=CP+31]). In case of memory write access, the hit detection logic determines a cache hit in advance. Nevertheless, the address conversion needs one additional CPU cycle. The value is directly written into the global register bank without further delay (MOV **mem**,R4).

Note: The 24-bit GPR addressing mode is not recommended because it requires an extra cycle for the read and write access.

.

Table 2-3 Addressing Modes to Access Word-GPRs

Name	Physical Address 1)	8-Bit Address	4-Bit Address	Description	Reset Value
R0	(CP)+0	$F0_H$	0h	General Purpose Word Register R0	$UUUU_H$
R1	(CP)+2	$F1_H$	1h	General Purpose Word Register R1	$UUUU_H$
R2	(CP)+4	$F2_H$	2h	General Purpose Word Register R2	$UUUU_H$
R3	(CP)+6	$F3_H$	3h	General Purpose Word Register R3	$UUUU_H$
R4	(CP)+8	$F4_H$	4h	General Purpose Word Register R4	$UUUU_H$
R5	(CP)+10	$F5_H$	5h	General Purpose Word Register R5	$UUUU_H$
R6	(CP)+12	$F6_H$	6h	General Purpose Word Register R6	$UUUU_H$
R7	(CP)+14	$F7_H$	7h	General Purpose Word Register R7	$UUUU_H$
R8	(CP)+16	$F8_H$	8h	General Purpose Word Register R8	$UUUU_H$
R9	(CP)+18	$F9_H$	9h	General Purpose Word Register R9	$UUUU_H$
R10	(CP)+20	FA_H	Ah	General Purpose Word Register R10	$UUUU_H$
R11	(CP)+22	FB_H	Bh	General Purpose Word Register R11	$UUUU_H$
R12	(CP)+24	FC_H	Ch	General Purpose Word Register R12	$UUUU_H$
R13	(CP)+26	FD_H	Dh	General Purpose Word Register R13	$UUUU_H$
R14	(CP)+28	FE_H	Eh	General Purpose Word Register R14	$UUUU_H$
R15	(CP)+30	FF_H	Fh	General Purpose Word Register R15	$UUUU_H$

1) Addressing mode only usable if the GPR bank is memory mapped.

Note: The first 8 GPRs (R7...R0) may also be accessed bytewise.

Note: Writing to a GPR byte does not affect the other byte of the respective GPR.

The respective halves of the byte-accessible registers have special names (see Table 2-4). .

Table 2-4 Addressing modes to access Byte-GPRs

Name	Physical Address 1)	8-Bit Address	4-Bit Address	Description	Reset Value
RL0	(CP)+0	$F0_H$	0h	General Purpose Byte Register RL0	UU_H
RH0	(CP)+1	$F1_H$	1h	General Purpose Byte Register RL1	UU_H
RL1	(CP)+2	$F2_H$	2h	General Purpose Byte Register RL2	UU_H
RH1	(CP)+3	$F3_H$	3h	General Purpose Byte Register RL3	UU_H
RL2	(CP)+4	$F4_H$	4h	General Purpose Byte Register RL4	UU_H
RH2	(CP)+5	$F5_H$	5h	General Purpose Byte Register RL5	UU_H
RL3	(CP)+6	$F6_H$	6h	General Purpose Byte Register RL6	UU_H
RH3	(CP)+7	$F7_H$	7h	General Purpose Byte Register RL7	UU_H
RL4	(CP)+8	$F8_H$	8h	General Purpose Byte Register RL8	UU_H
RH4	(CP)+9	$F9_H$	9h	General Purpose Byte Register RL9	UU_H
RL5	(CP)+10	FA_H	Ah	General Purpose Byte Register RL10	UU_H
RH5	(CP)+11	FB_H	Bh	General Purpose Byte Register RL11	UU_H
RL6	(CP)+12	FC_H	Ch	General Purpose Byte Register RL12	UU_H
RH6	(CP)+13	FD_H	Dh	General Purpose Byte Register RL13	UU_H
RL7	(CP)+14	FE_H	Eh	General Purpose Byte Register RL14	UU_H
RH7	(CP)+15	FF_H	Fh	General Purpose Byte Register RL15	UU_H

1) Addressing mode only usable if the GPR bank is memory mapped.

*Note: Even if the local register bank is selected by **BANK,** an old memory-mapped GPR bank can be cached in the global register bank. Memory accesses are still redirected in case of a cache hit.*

2.4.2 Local Register Bank

C166S V2 CPU has two local register banks with sixteen independent GPRs each. Both local register banks are **not memory mapped**. After a switch to a local register bank, the GPRs are directly accessible. There are two different ways to access an activated local register bank.

Short 4-Bit GPR Addresses (mnemonic: Rw or Rb) specify addresses in the local register banks. The local register bank is selected by the **BANK** bitfield of the PSW.

Depending on whether a relative word (Rw) or byte (Rb) GPR address is specified, the short 4-bit GPR address is either multiplied by two or not before it is used to physically access the local register bank. Thus, both byte and word GPR accesses are possible in this way.

Note: If GPRs are used as indirect address pointers, they are always accessed wordwise.

For some instructions, only the first four GPRs can be used as indirect address pointers. These GPRs are specified via short 2-bit GPR addresses. The respective physical address calculation is identical with the one for the short 4-bit GPR addresses.

Short 8-Bit Register Addresses (mnemonic: reg or bitoff) within a range from $F0_H$ to FF_H interpret the four least significant bits as short 4-bit GPR address, while the four most significant bits are ignored. The respective physical GPR address calculation is identical with the one for the short 4-bit GPR addresses. For single bit accesses on a GPR, the GPR's word address is calculated as just described, but the position of the bit within the word is specified by a separate additional 4-bit value.

For a summary of all addressing modes usable to access GPRs, please see **Table 2-3** and **Table 2-4**.

2.4.3 Context Switch

An interrupt service routine or a task scheduler of an operating system usually saves into the stack all the used registers and restores them before returning. The more registers a routine uses, the more time is wasted with saving and restoring. There are two ways to change a context in the C166S V2 core:

- Switching the context by changing the selected register banks.
- Switching the context of the global register bank by changing the context pointer CP.

2.4.3.1 Changing the selected Physical Register Bank

The switch between the three physical register banks is the fastest possible context switch. It is possible to switch between the current memory-mapped GPR bank located in the global register bank and the two not memory-mapped local register banks. The **BANK** bit field of the PSW register determines the selected bank.

PSW
Processor Status Word **SFRb** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				IEN	HLD EN	BANK		USR1	USR0	MUL IP	E	Z	V	C	N
rwh				rw	rw	rwh		rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
BANK	9-8	rwh	**Reserved for register file bank selection** 00 Global register bank 01 Reserved 10 Local register bank 1 11 Local register bank 2

In case of an interrupt service, the bank switch is automatically executed by updating the PSW. The Interrupt Controller (ITC) configuration decides which register bank will be selected. By executing a RETI instruction, the **BANK** bit field of the PSW will automatically be restored and the context will switched to the original register bank.

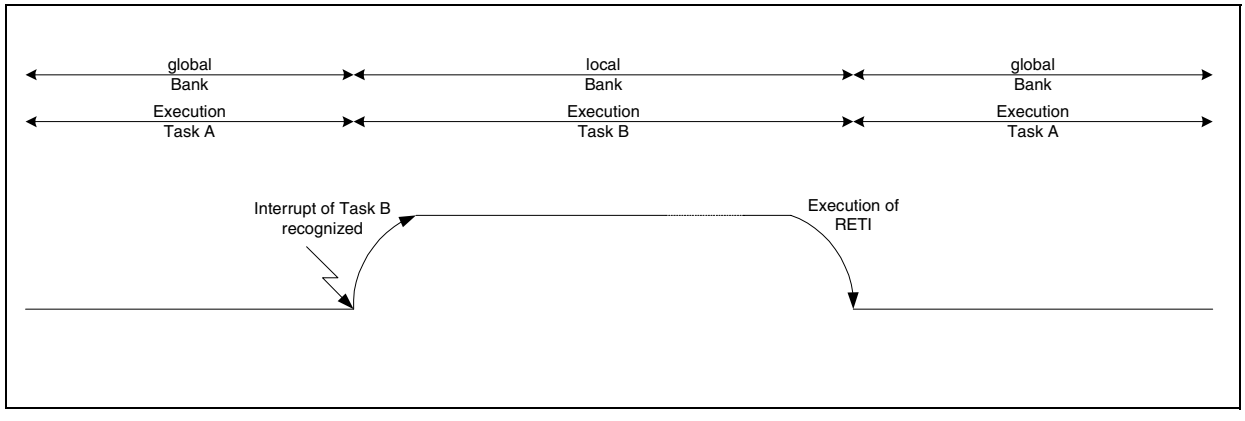


Figure 2-11 Context Switch by Changing the Physical Register Bank

After a switch to a local register bank, the new bank is immediately available. After switching to the global register bank, the cached memory-mapped GPRs must be valid before any further instructions can be executed. If the global register bank is not valid at this time (in case if the context switch process has been interrupted), the cache validation process is repeated automatically. For further explanation, please refer to **Section 2.4.3.2**.

*Note: The switch between the three physical register banks of the register file can also be executed by writing to the **BANK** bitfield of the PSW. Because of pipeline dependencies an explicit change of the PSW must cancel the pipeline.*

2.4.3.2 Context Switching of the Global Register Bank

The contents of the global register bank are switched by changing the base address of the memory mapped GPR bank. The base address is given by the contents of the Context Pointer (CP).

The Context Pointer (CP)

The CP register is non-bit addressable. It can be updated via any instruction capable of modifying SFRs.

CP
Context Pointer **SFR** **Reset Value: $FC00_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CONTEXT POINTER											0
r	r	r	r	rw											r

Field	Bits	Type	Description
1	[15:12]	r	CP always points in the internal DPRAM
CONTEXT POINTER	[11:1]	rw	**Modifiable Portion of register CP** Specifies the (word) base address of the current memory-mapped register bank. When writing a value to register CP with bits CP[11:9] = ’000’, bits CP[11:10] are set to ’11’ by hardware.
0	[0]	r	CP is always word-aligned

Note: It is the user’s responsibility that the physical GPR address specified via CP register plus the short GPR address must always be an internal DPRAM location. If this condition is not met, unexpected results may occur. Do not set CP below the internal DPRAM start address.

Note: Due to the internal instruction pipeline, a write operation to the CP register stalls the instruction flow until the register file context switch is really executed. The instruction immediately following the instruction that updates CP register can use the new value of the changed CP.

The C166S V2 CPU switches the complete memory-mapped GPR bank with a single instruction. After switching, the service routine executes within its own separate context.

The instruction “SCXT CP, #New_Bank” pushes the value of the current context pointer (CP) into the system stack and loads CP with the immediate value “New_Bank”, which selects a new register bank. The service routine may now use its “own registers”. This

memory register bank is preserved when the service routine terminates, i.e. its contents is available on the next call.
Before returning from the service routine (RETI), the previous CP is simply popped from the system stack which returns the registers to the original bank.

Context Pointer Updating

After the CP has been update, a state machine starts to store the old contents of the global register bank and to load the new one. An instruction “SCXT CP, #New_Bank” takes two cycles. The store and load algorithm is executed in nineteen CPU cycles: the execution of the cache validation process takes sixteen cycles plus three cycles to stall an instruction execution to avoid pipeline conflicts upon the completion of the validation process. The context switch process has two phases:

1. Store phase: The contents of the global register bank is stored back into the DPRAM by executing eight injected STORE instructions. After the last STORE instruction the contents of the global register bank are invalidated.

2. Load phase: The global register bank is loaded with the new context by executing eight injected LOAD instructions. After the last LOAD instruction the contents of the global register bank are validated.

The code execution is stopped until the global register bank is valid. A hardware interrupt which also uses a global register bank cannot be executed until the validation process is finished (see **Figure 2-12**).

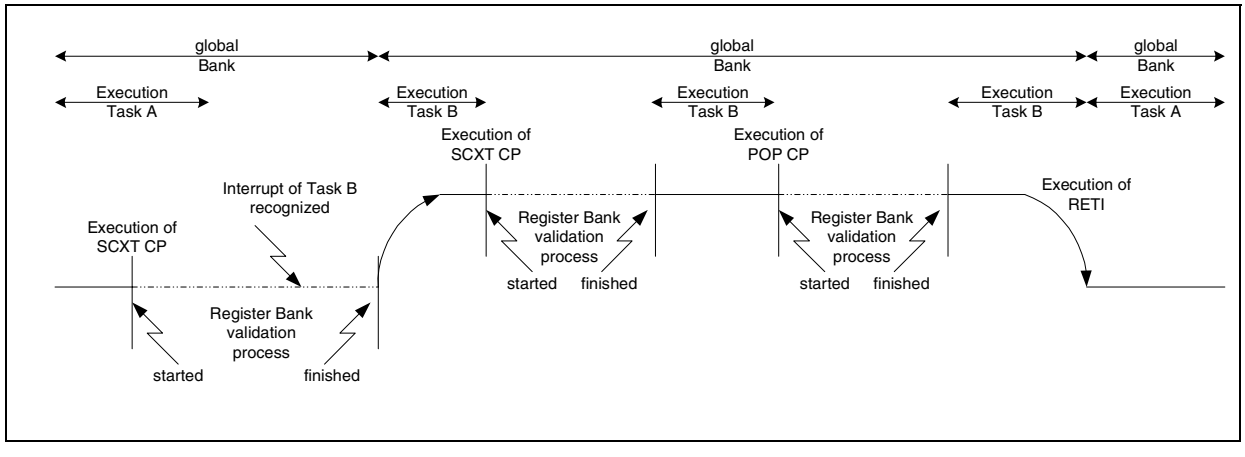
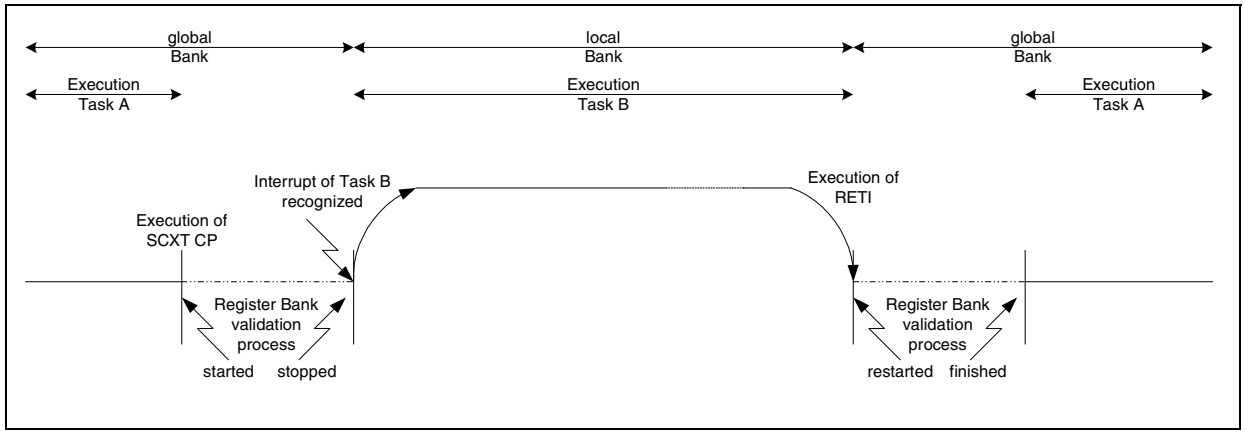


Figure 2-12 Validation process and hardware interrupts using a global register bank

But, the validation process can be interrupted by any hardware interrupt which will work with a local register bank. After switching back to the global register bank, the validation process must be finished. The way the validation process will be restarted depends on the phase in which it has been interrupted.

If the interrupt occurred before the load phase, the entire validation process is restarted from the very beginning. If the store phase has been completed before the interrupt, only the load phase is executed.



Note: Validation Process and Hardware Interrupts using a Local Register Bank

*Note: A cache validation process of Task A can be interrupted by a Task B which uses a local register bank. Task B itself is interrupted again by an interrupt Task C which uses a global register bank again. In this case, the validation process of Task A must be finished before code of Task C can be executed. This means that the validation process of Task A does not affect the interrupt latency of Task B but the latency of Task C. If Task C would immediately interrupt Task A, the register bank validation process of Task A would be finished first. The worst case interrupt latency is identical in both cases (see **Figure 2-12** and **Figure 2-13**).*

.

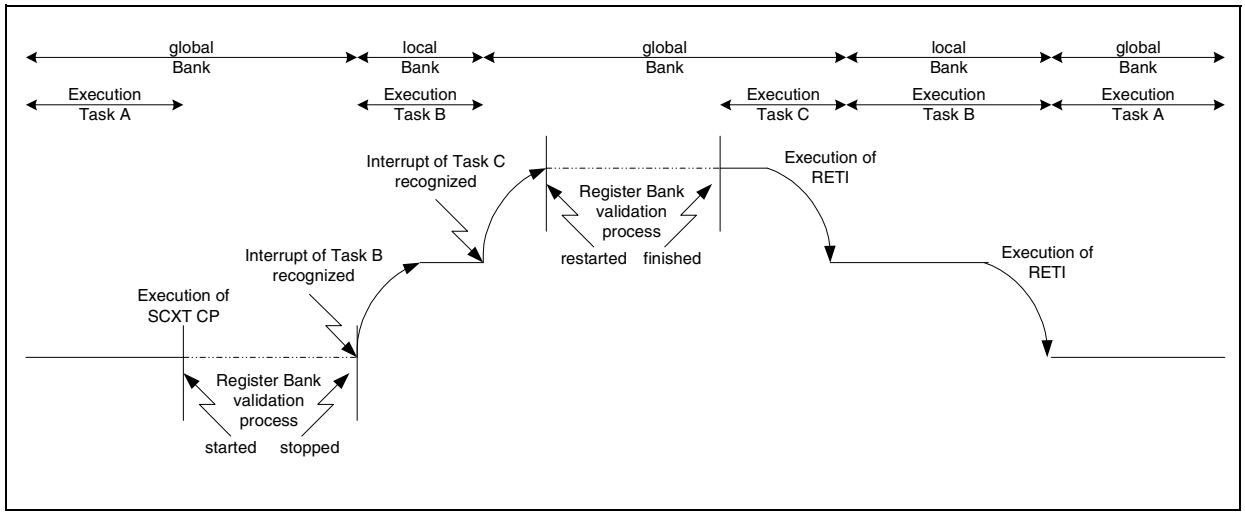


Figure 2-13 Validation Process and Hardware Interrupts using Local and Global Register Bank

2.5 Data Addressing

The Address Data Unit (ADU) of the C166S V2 CPU contains two independent arithmetic units to generate, calculate, and update addresses for data accesses. The ADU performs the following major tasks:

- Standard Address Generation (Standard Address Generation Unit)
- DSP Address Generation (DSP Address Unit)
- Data Paging (Standard Address Unit)
- Stack Handling (Standard Address Unit)

The Standard Address Unit supports linear arithmetic for the indirect addressing modes and also generates the address in case of all other short and long addressing modes. The DSP Address Generation Unit contains an additional set of address pointers and offset registers which are used in conjunction with the CoXXX instructions only.

The C166S V2 CPU provides a lot of powerful addressing modes for word, byte, and bit data accesses (short, long, indirect). The different addressing modes use different formats and have different scopes.

2.5.1 Short Addressing Modes

All of these addressing modes use an implicit base offset address to specify a 24-bit physical address.
Short addressing modes allow access to the GPR, SFR or bit addressable memory space:

Physical Address = Base Address + Δ * Short Address

Note: Δ is 1 for byte GPRs, Δ is 2 for word GPRs..

Table 2-5 Short addressing modes

Mnemonic	Physical Address	Short Address Range	Scope of Access
Rw	(CP) + 2*Rw or local	Rw = 0...15	GPRs(Word)
Rb	(CP) + 1*Rb or local	Rb = 0...15	GPRs(Byte)
reg	$00'FE00_H$ + 2*reg $00'F000_H$ + 2*reg (CP)+2*(reg∧$0F_H$) or local (CP)+1*(reg∧$0F_H$) or local	reg = 00_H...EF_H reg = 00_H...EF_H reg = $F0_H$...FF_H reg = $F0_H$...FF_H	SFRs (Word, Low byte) ESFRs(Word, Low byte) GPRs(Word) GPRs(Bytes)
bitoff	$00'FD00_H$+ 2*bitoff $00'FF00_H$ + 2*(bitoff∧$7F_H$) $00'F100_H$ + 2*(bitoff∧$7F_H$) (CP) + 2*(bitoff∧$0F_H$) or local	bitoff = 00_H...$7F_H$ bitoff = 80_H...EF_H bitoff = 80_H...EF_H bitoff = $F0_H$...FF_H	RAM Bit word offset SFR Bit word offset ESFR Bit word offset GPR Bit word offset
bitaddr	Word offset as with bitoff. Immediate bit position.	bitoff = 00_H...FF_H bitpos= 0...15	Any single bit

Rw, Rb: Specifies direct access to any GPR in the currently active context (global register bank or local register bank). Both 'Rw' and 'Rb' require four bits in the instruction format.The base address of the global register bank is determined by the contents of register CP. 'Rw' specifies a 4-bit word GPR address relative to the base address (CP), while 'Rb' specifies a 4-bit byte GPR address relative to the base address (CP). In case of an active local register bank this 4 bits are used directly to address the GPR.

reg: Specifies direct access to any (E)SFR or GPR in the currently active context (global or local register bank). The 'reg' value requires eight bits in the instruction format. Short 'reg' addresses in the range from 00_H to EF_H always specify (E)SFRs. In that case, the factor 'D' equates 2 and the base address is $00'FE00_H$ for the standard SFR area or $00'F000_H$ for the extended ESFR area. The 'reg' accesses to the ESFR area require a preceding EXT*R instruction to switch the base address. Depending on the opcode, either the total word (for word operations) or the low byte (for byte operations) of an SFR can

be addressed via ’reg’. Note that the high byte of an SFR cannot be accessed via the ’reg’ addressing mode. Short ’reg’ addresses in the range from $F0_H$ to FF_H always specify GPRs. In that case, only the lower four bits of ’reg’ are significant for physical address generation and, therefore, it is identical to the address generation described for the ’Rb’ and ’Rw’ addressing modes.

bitoff: Specifies direct access to any word in the bit addressable memory space. The ’bitoff’ value requires eight bits in the instruction format. Depending on the specified ’bitoff’ range different base addresses are used to generate physical addresses: Short ’bitoff’ addresses in the range from 00_H to $7F_H$ use $00’FD00_H$ as a base address to specify the 128 highest internal RAM word locations in the range from $00’FD00_H$h to $00’FDFE_H$. Short 'bitoff' addresses in the range from 80_H to EF_H use base address $00’FF00_H$ to specify the internal SFR word locations in the range from $00’FF00_H$ to $00’FFDE_H$ or base address $00’F100_H$ to specify the internal ESFR word locations in the range from $00’F100_H$ to $00’F1DE_H$. The ‘bitoff’ accesses to the ESFR area require a preceding EXT*R instruction to switch the base address. For short 'bitoff' addresses from $F0_H$ to FF_H, only the lowest four bits are used to generate the address of the selected word GPR.

bitaddr: Any bit address is specified by a word address within the bit addressable memory space (see 'bitoff'), and by a bit position ('bitpos') within that word. Therefore, 'bitaddr' requires twelve bits in the instruction format.

2.5.2 Long and Indirect Addressing Modes

These addressing modes use one of the four DPP registers to specify a 24-bit address. Any word or byte data within the entire address space can be accessed with these modes.
Any long or indirect 16-bit address contain two parts that have different meanings. Bits 13...0 specify a 14-bit data page offset, while bits 15...14 specify the Data Page Pointer (DPP) (1 of 4) register used to generate the full 24-bit address (see **Figure 2-14**).

The C166S V2 CPU also supports an override mechanism for the DPP addressing scheme (EXTP(R) and EXTS(R) instructions). See following sections for details.

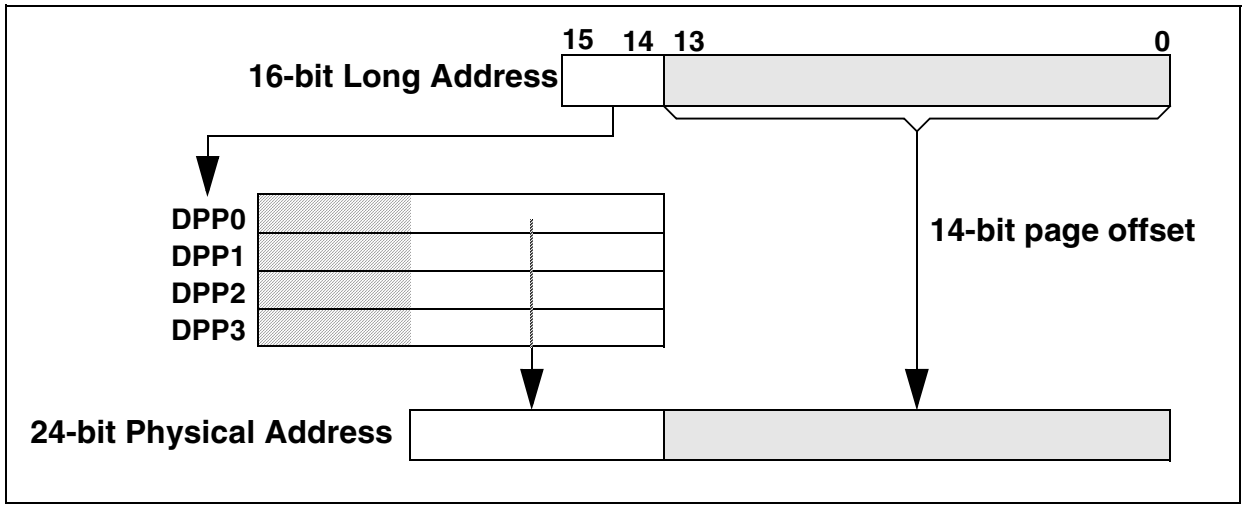


Figure 2-14 Interpretation of a 16-bit Long Address

Note: Word accesses on odd byte addresses are not executed. A hardware trap will be triggered.

2.5.2.1 Addressing via Data Page Pointer DPP

The four non-bit addressable Data Page Pointer registers select up to four different data pages. The lower 10 bits of each DPP register select one of the 1024 possible 16-Kilobyte data pages while the upper 6 bits are reserved for the future use. The DPP registers provide an access to the entire memory space in 16 Kilobytes pages.

The DPP registers are implicitly used whenever data accesses to any memory location are made via indirect or direct long 16-bit addressing modes (except for override accesses via EXTended instructions and PEC data transfers).

Data paging is performed by concatenating the lower 14-bits of an indirect or direct long 16-bit address with the contents of the DDP register selected by the upper two bits of the 16-bit address. The contents of the selected DPP register specifies one of the 1024 possible data pages. This data page base address together with the 14-bit page offset forms the physical 24-bit address.

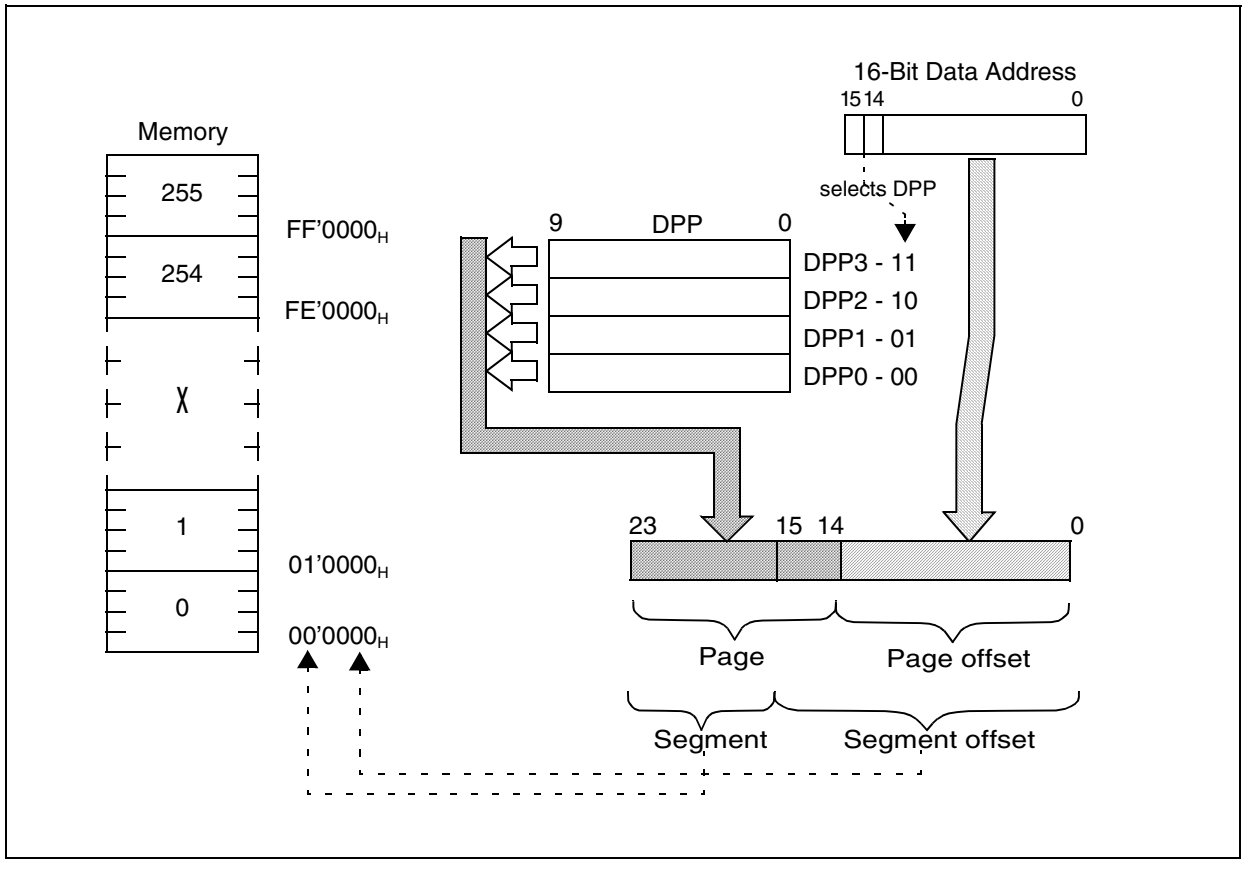


Figure 2-15 Data Page Pointer Addressing

After reset, the DPP registers select data pages 3...0 within segment 0. If the user does not want to use any data paging, no further action is required.

DPP0
Data Page Pointer 0 **SFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	PN									
r	r	r	r	r	r	rw									

DPP1
Data Page Pointer 1 **SFR** **Reset Value: 0001_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	PN									
r	r	r	r	r	r	rw									

DPP2
Data Page Pointer 2 **SFR** **Reset Value: 0002_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	PN									
r	r	r	r	r	r	rw									

DPP3
Data Page Pointer 3 **SFR** **Reset Value: 0003_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	PN									
r	r	r	r	r	r	rw									

Field	Bits	Type	Description
PN	[9:0]	rw	**Data Page Number of DPP** Specifies the data page selected via DPP.

Note: In case of non-segmented memory mode, the entire DPP register is still used for the calculation of the physical 24-bit address.

A DPP register can be updated via any instruction capable of modifying an SFR.

Note: Due to the internal instruction pipeline, a write operation to the DPPx registers could stall the instruction flow until the DPP is actually updated. The instruction that immediately follows the instruction which updates the DPP register can use the new value of the changed DPPx.

2.5.2.2 DPP Override Mechanism in the C166S V2 CPU

The C166S V2 CPU provides an override mechanism for the temporary bypass of the DPP addressing scheme.

The EXTP(R) and EXTS(R) instructions override this addressing mechanism. Instruction EXTP(R) replaces the contents of the respective DPP register, while instruction EXTS(R) concatenates the complete 16-bit long address with the specified segment base address. The overriding page or segment may be specified directly as a constant (#pag, #seg) or via a word GPR (Rw).

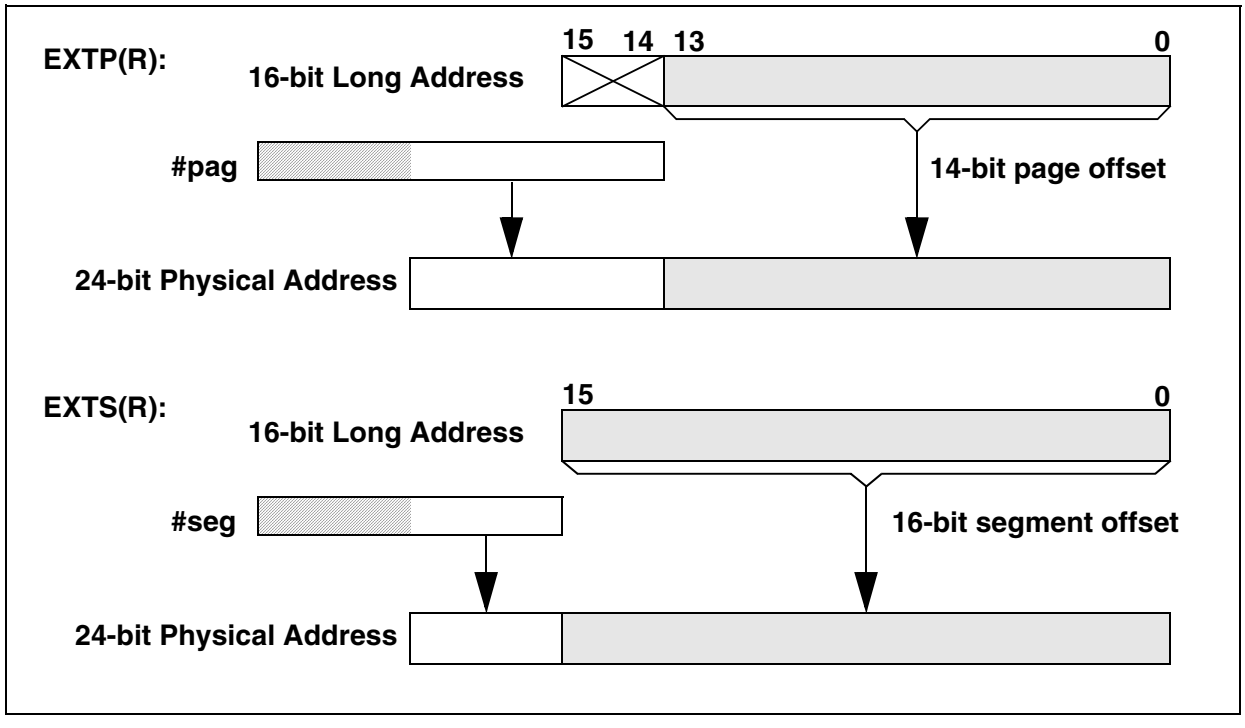


Figure 2-16 Overriding the DPP Mechanism

2.5.2.3 Long Addressing Mode

The long addressing mode uses a 16-bit constant value encoded in the instruction format which specifies the data page offset and the DPP.

The long addressing mode is referred to by the mnemonic ‘mem’. .

Table 2-6 Long Addressing Modes

Mnemonic	Physical Address	Scope of Access
mem	(DPP0) || mem$\wedge 3FFF_H$ (DPP1) || mem$\wedge 3FFF_H$ (DPP2) || mem$\wedge 3FFF_H$ (DPP3) || mem$\wedge 3FFF_H$	Any Word or Byte
mem	pag || mem$\wedge 3FFF_H$	Any Word or Byte
mem	seg || mem	Any Word or Byte

Note: The long addressing may be used with the DPP overriding mechanism (EXTP(R) and EXTS(R)).

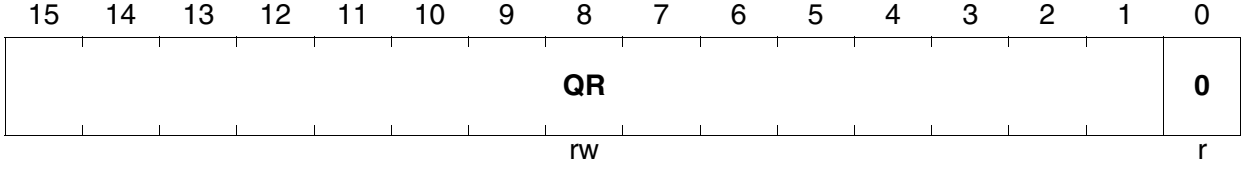
2.5.2.4 Indirect Addressing Modes

These addressing modes can be considered as a combination of short and long addressing modes. This means that long 16-bit address is provided indirectly by the contents of a word GPR which is specified directly by a short 4-bit address (’Rw’=0 to 15). There are indirect addressing modes, which add a constant value to the GPR contents before the long 16-bit address is calculated. Other indirect addressing modes can decrement or increment the indirect address pointers (GPR contents) by 2 or 1 (referring to words or bytes) or by the contents of the offset registers QR0 and QR1.

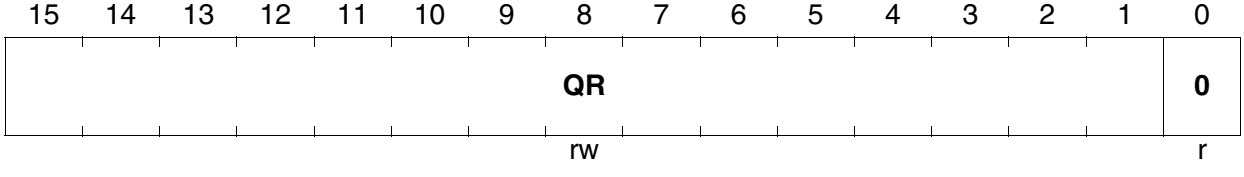
The Offset Register QR0 and QR1

There are two non-bit addressable offset registers QR0 and QR1 which can be used in conjunction with the CoXXX instructions.

QR0
Offset Register **ESFR** **Reset Value: 0000_H**



QR1
Offset Register **ESFR** **Reset Value: 0000_H**



Field	Bits	Type	Description
QR	[15:1]	rw	**Modifiable portion of register QRx** Specifies the 16-bit offset address for indirect addressing modes.
0	[0]	r	Fixed to 0

Note: During initialization of the QR registers, instruction flow stalls are possible. For the proper operation refer to ***Chapter 4.1.4****.*

In each case, one of the four DPP registers is used to specify physical 24-bit addresses. Any word or byte data within the entire memory space can be addressed indirectly.

Note: The indirect addressing may be used with the DPP overriding mechanism (EXTP(R) and EXTS(R)).

Some instructions only use the lowest four word GPRs (R3...R0) as indirect address pointers, which are specified via short 2-bit addresses in that case.

Physical addresses are generated from indirect address pointers using the following algorithm:

1) Calculate the physical address of the word GPR, which is used as indirect address pointer, using the specified short address (’Rw’) and

- the current global register bank

GPR Address = (CP) + 2 * Short Address

- the current local register bank

GPR Address = 2 * Short Address.

2) If required, pre-decremented indirect address pointer (‘-Rw’) by the data-type-dependent value (D=1 for byte operations, D=2 for word operations) before the long 16-bit address is generated:

(GPR Address) = (GPR Address) - D ; [optional step!]

3) Calculate the long 16-bit address by adding a constant value (’Rw+const16’ if selected) to the contents of the indirect address pointer:

Long Address = (GPR Pointer) + Constant ; [+Constant is optional]

4) Calculate the physical 24-bit address using the resulting long address and the corresponding DPP register contents (see long 'mem' addressing modes).

Physical Address = (DPPi) + Page offset

5) - If required, post-in/decrement indirect address pointers (‘Rw±’) by the data-type-dependent value (D=1 for byte operations, D=2 for word operations).
- If required, post-in/decrement indirect address pointers (‘Rw± QRx’) by D=QRx:

(GPR Pointer) = (GPR Pointer) ± D ; [optional step!]

The following indirect addressing modes are provided: .

Table 2-7 Indirect Addressing Modes

Mnemonic	Particularities
[Rw]	Most instructions accept any GPR (R15...R0) as indirect address pointer. Some instructions accept only the lower four GPRs (R3...R0).
[Rw+]	The specified indirect address pointer is automatically post-incremented by 2 or 1 (for word or byte data operations) after the access.
[-Rw]	The specified indirect address pointer is automatically pre-decremented by 2 or 1 (for word or byte data operations) before the access.
[Rw+#data16]	The specified 16-bit constant is added to the indirect address pointer, before the long address is calculated.
[Rw-]	The specified indirect address pointer is automatically post-decremented by 2 (word data operations) after the access.
[Rw+QRx]	The specified indirect address pointer is automatically post-incremented by QRx (word data operations) after the access.
[Rw-QRx]	The specified indirect address pointer is automatically post-decremented by QRX (word data operations) after the access.

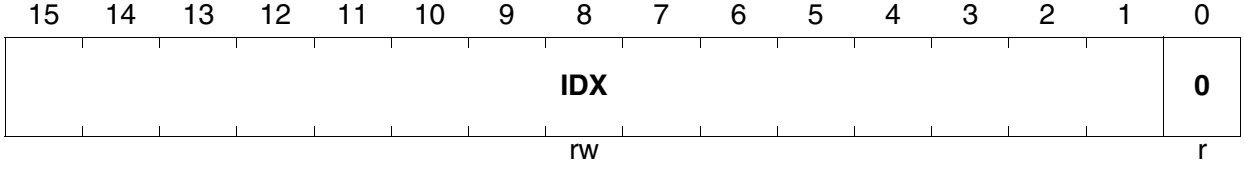
2.5.3 DSP Addressing

In addition to the Standard Address Generation Unit, the DSP Address Generation Unit provides an additional set of pointer and offset registers. An independent arithmetic unit allows the update of these dedicated pointer registers in parallel with the GPR-Pointer modification of the Standard Address Generation Unit. The DSP Address Generation Unit only supports indirect addressing modes that use the special pointer registers IDX0 and IDX1.

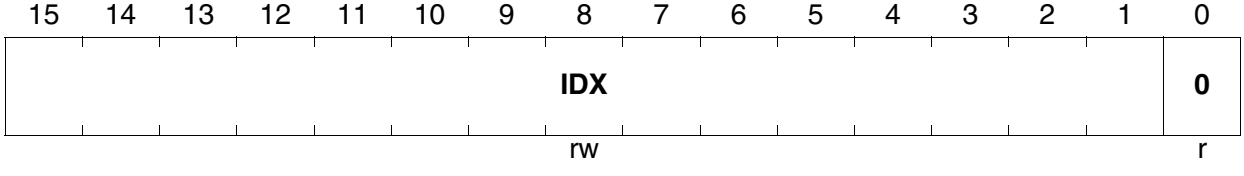
The Pointer Register IDX0 and IDX1

The additional set of pointer registers IDX0 and IDX1 allows the execution of DSP specific CoXXX instruction in one CPU cycle.

IDX0
Address Pointer **SFRb** **Reset Value: 0000_H**



IDX1
Address Pointer **SFRb** **Reset Value: 0000_H**



Field	Bits	Type	Description
IDX	[15:1]	rw	**Modifiable portion of register IDXx** Specifies the 16-bit value of a dedicated address pointer.
0	[0]	r	Fixed to 0

Note: During the initialization of the IDX registers, instruction flow stalls are possible. For the proper operation, refer to the ***Section 4.1.4****.*

The address pointers can be used for arithmetic operations as well as for the special CoMOV instruction. But, the generation of the 24 bit memory address is different.

In case of arithmetic CoXXX operations, the IDX pointers are automatically zero extended to a 24-bit memory address. The IDX address pointers should point to the internal DPRAM area. Even if the IDX address pointers do not point to the internal

DPRAM area, the address is mapped into the DPRAM area. The leading four bits of the IDX pointers are not taken into account as shown in **Figure 2-17**.

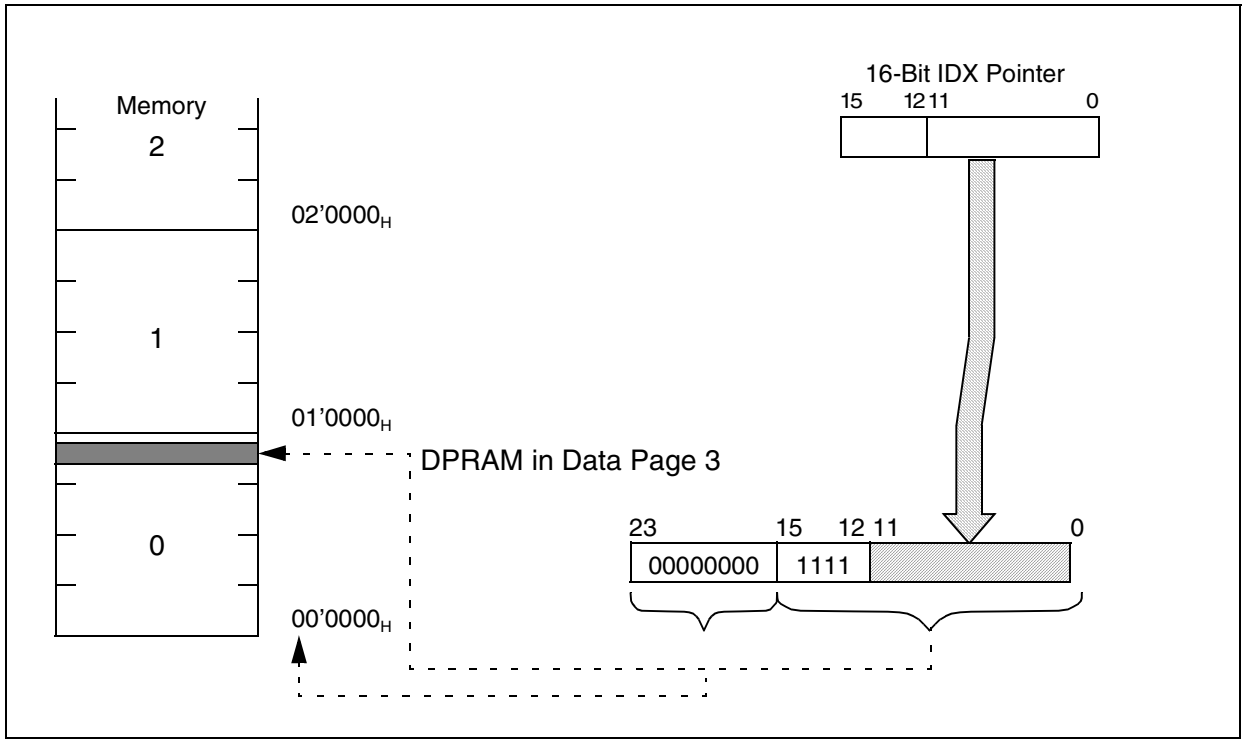


Figure 2-17 Arithmetic MAC Operations and Addressing via the IDX Pointers

For **CoMOV** MAC operation, the IDX pointers are concatenated with the Data Page Pointers, just like normal GPR-Pointers as described in **Section 2.5.2.1**. The IDX pointer can address the entire C166S V2 memory area without any restrictions.

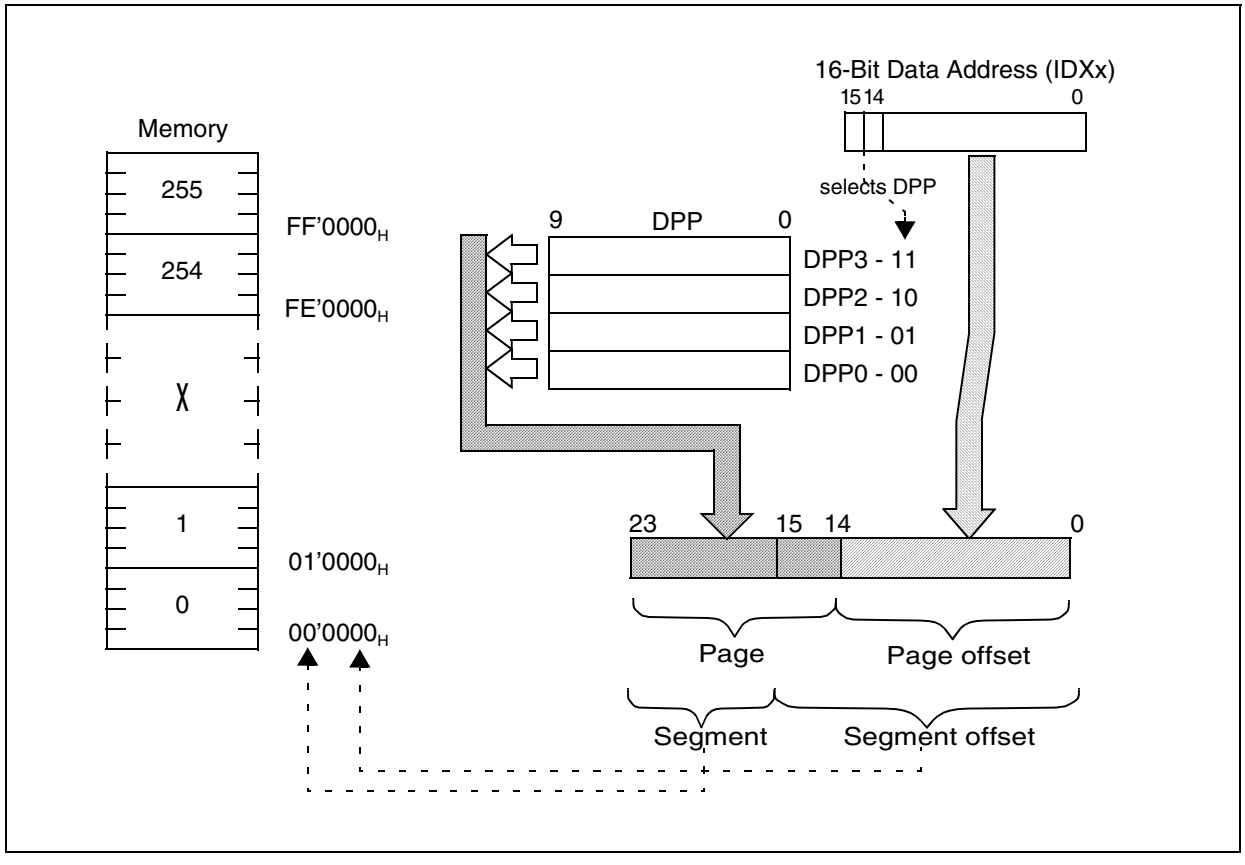


Figure 2-18 CoMOV Operations and Addressing via the IDX Pointers

There are indirect addressing modes which allow parallel data move operations before the long 16-bit address is calculated. Other indirect addressing modes allow decrementing or incrementing the indirect address pointers (IDXx contents) by 2 or by the contents of the offset registers. There are two non-bit addressable offset registers QX0 and QX1 which can be used in conjunction with the CoXXX instructions.

The Offset Register QX0 and QX1

These two non-bit addressable registers are used only for CoXXX operations which access operands using indirect addressing mode. The QX offset registers are used in conjunction with the IDX pointers.

QX0
Offset Register **ESFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QX															0
rw															r

QX1
Offset Register **ESFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QX															0
rw															r

Field	Bits	Type	Description
QX	[15:1]	rw	**Modifiable portion of register QXx** Specifies the 16-bit offset address for indirect addressing modes.
0	[0]	r	Fixed to 0

Note: During the initialization of the QX registers, instruction flow stalls are possible. For the proper operation, refer to the ***Section 4.1.4****.*

Physical addresses are generated from indirect address pointers IDX via the following algorithm:

1) Determine the used IDXx pointer

2) An intermediate long address is calculated for the parallel data move operation of CoXXXM instructions before the long 16-bit address is generated [optional step!]:

- If required, indirect address pointers (‘IDXx±’) are de/incremented by D=2.
- If required, indirect address pointers (‘IDXx± QXx’) are de/incremented by D= QXx.

Intermediate Address = (IDXx Address) ± D ; [optional step!]

3) Calculate long 16-bit address:

Long Address = (IDXx Pointer)

4) Calculate the physical 24-bit address using the resulting long address and the corresponding DPP register contents (see long 'mem' addressing modes and DPPi override mechanism for arithmetic CoXXX instructions).

Physical Address = (DPPi) + Page offset

5) - If required, indirect address pointers ('IDXx**±**') are in/decremented by D=2 for word operations.
- If required, indirect address pointers ('IDXx**±** QXx') are in/decremented by D= QXx for word operations.

(IDX Pointer) = (IDX Pointer) ± D; [optional step!]

The following indirect addressing modes are provided: .

Table 2-8 DSP Addressing Modes

Mnemonic	Particularities
[IDXx]	Most CoXXX instructions accept IDXx (IDX0, IDX1) as an indirect address pointer.
[IDXx+]	The specified indirect address pointer is automatically post-incremented by 2 after the access.
with parallel data move	In case of a CoXXXM instruction, the address stored in the specified indirect address pointer is automatically pre-decremented by 2 for the parallel move operation. The pointer itself is not pre-decremented. Then, the specified indirect address pointer is automatically post-incremented by 2 after the access.
[IDXx-]	The specified indirect address pointer is automatically post-decremented by 2 after the access.

Table 2-8 DSP Addressing Modes (cont'd)

Mnemonic	Particularities
with parallel data move	In case of a CoXXXM instruction, the address stored in the specified indirect address pointer is automatically pre-incremented by 2 for the parallel move operation. The pointer itself is not pre-incremented. Then, the specified indirect address pointer is automatically post-decremented by 2 after the access.
[IDXx+QXx]	The specified indirect address pointer is automatically post-incremented by QXx after the access.
with parallel data move	In case of a CoXXXM instruction, the address stored in the specified indirect address pointer is automatically pre-decremented by QXx for the parallel move operation. The pointer itself is not pre-decremented. Then, the specified indirect address pointer is automatically post-incremented by QXx after the access.
[IDXx-QXx]	The specified indirect address pointer is automatically post-decremented by QXx after the access.
with parallel data move	In case of a CoXXXM instruction, the address stored in the specified indirect address pointer is automatically pre-incremented by QXx for the parallel move operation. The pointer itself is not pre-incremented. Then, the specified indirect address pointer is automatically post-decremented by QXx after the access.

The example in **Figure 2-19** shows the complex operation of CoXXX instructions with a parallel move operation based on the descriptions about addressing modes given in **Section 2.5.2.4 (Indirect Addressing Modes)** and **Section 2.5.3 (DSP Addressing Modes).**

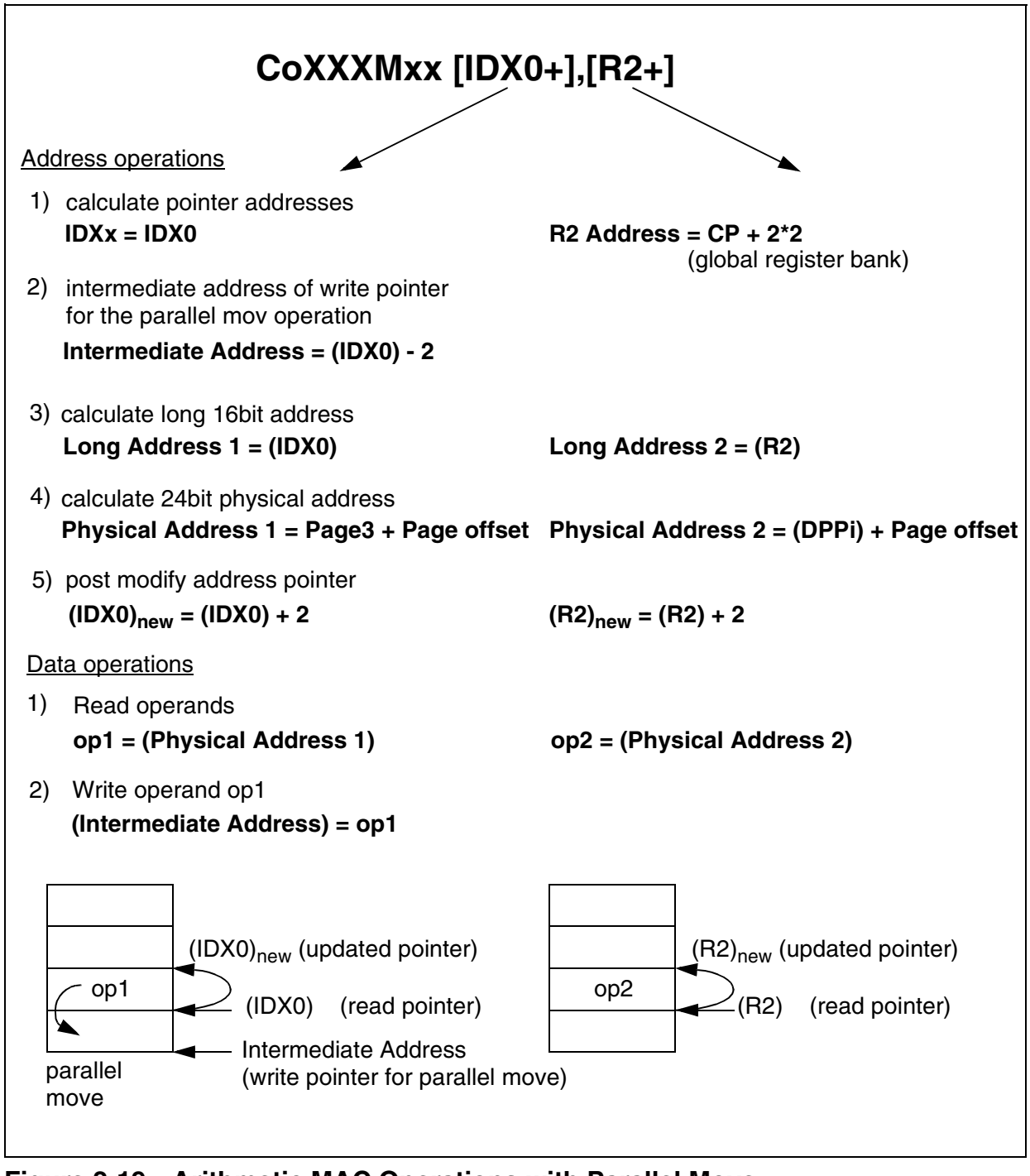


Figure 2-19 Arithmetic MAC Operations with Parallel Move

2.5.4 The CoREG Addressing Mode

The CoSTORE instruction utilizes the special CoREG addressing mode for immediate storage of the MAC-Unit register after a MAC operation. The address of the MAC-Unit register is coded in the CoSTORE instruction format as described in the following table:

Table 2-9 Coding of the CoREG Addressing Mode

Mnemonic	Register	Coding of wwww:w bits [31:27]
MSW	MAC-Unit Status Word	00000
MAH	MAC-Unit Accumulator High Word	00001
MAS	Limited MAC-Unit Accumulator High Word	00010
MAL	MAC-Unit Accumulator Low Word	00100
MCW	MAC-Unit Control Word	00101
MRW	MAC-Unit Repeat Word	00110

2.5.5 The System Stack

The C166S V2 CPU supports a system stack of 64 kBytes. The stack can be located internally in one of the on-chip memories or externally. The 16-bit Stack Pointer (SP) register addresses the stack within a 64 kByte segment. The Stack Pointer Segment Register (SPSG) selects the segment in which the stack is located. A virtual stack (usually bigger then 64 kBytes) can be implemented by software. This mechanism is supported by registers STKOV and STKUN (see descriptions below).

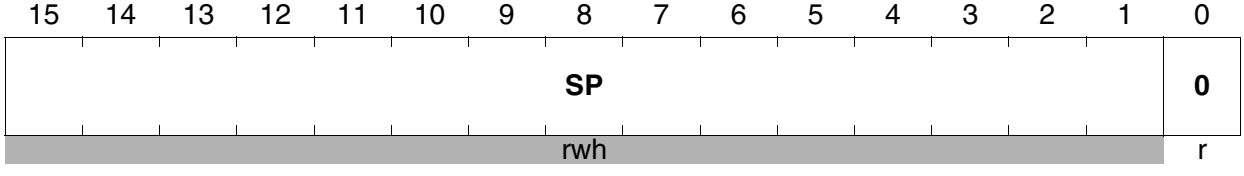
The Stack Pointer Register SP

The non-bit addressable Stack Pointer SP register is used to point to the top of the system stack (TOS). The SP register is pre-decremented whenever data is to be pushed onto the stack, and it is post-incremented whenever data is to be popped from the stack. Therefore, the system stack grows from higher toward lower memory locations.

The SP register can be updated via any instruction capable of modifying an 16-bit SFR.

Note: Due to the internal instruction pipeline, a stack pointer initialization stalls the instruction flow until the operation is finished. A POP and RETURN instruction can immediately follow an instruction updating the SP.

SP
Stack Pointer **SFR** **Reset Value: $FC00_H$**



Field	Bits	Type	Description
SP	[15:1]	rwh	**Modifiable portion of register SP** Specifies the top of the system stack.
0	[0]	r	Fixed to 0

The Stack Pointer Segment Register SPSEG

This non-bit addressable register selects the segment being used at run-time to access system stack. The lower eight bits of register SPSEG select one of up 256 segments of 64-kilobytes each, while the higher 8 bits are reserved for future use.

SPSEG
Stack Pointer Segment **SFRb** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	SPSEGNR							
r	r	r	r	r	r	r	r	rw							

Field	Bits	Type	Description
SPSEGNR	[7:0]	rw	**Stack Pointer Segment Number** Specifies the segment where the stack is located.

System stack addresses are generated by directly extending the 16-bit contents of the SP register by the contents of the SPSG register as shown in **Figure 2-20**.

The system stack cannot cross a 64k byte segment boundary.

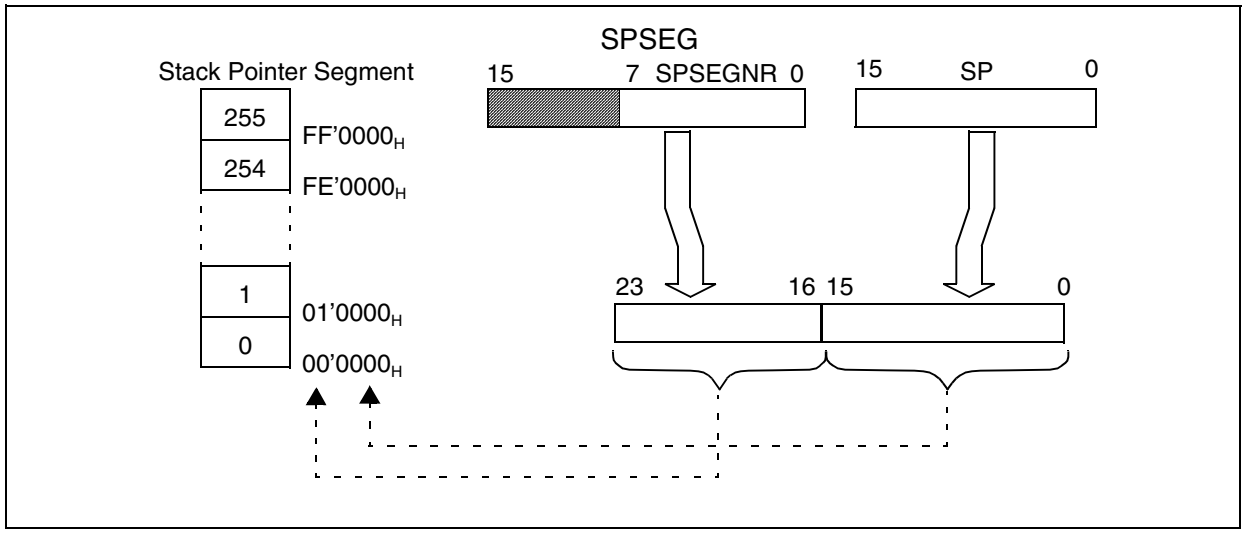


Figure 2-20 Addressing via the Stack Pointer

In case of a non-segmented memory mode, the SPSG register is also used to generate the physical address. If a non-segmented memory model is selected, extreme care should be taken when changing the contents of the SPSG register. Improper SPSG change may result in erroneous system behavior. The SPSG register can be updated via any instruction capable of modifying an SFR.

Note: Due to the internal instruction pipeline, a write operation to the SPSG register stalls the instruction flow until the SPSG register is really updated. The instruction immediately following the instruction updating the SPSG register can use the new value.

The Stack Overflow Pointer STKOV

This non-bit addressable STKOV register is compared with the SP register before each implicit write operation which decrements the contents of the SP register. If the contents of the SP register are equal to the contents of the STKOV register, a stack overflow trap will occur.

STKOV
Stack Overflow Pointer **SFR** **Reset Value: $FA00_H$**

Field	Bits	Type	Description
STKOV	[15:1]	rw	**Modifiable portion of register STKOV** Specifies the segment offset address of the lower limit of the system stack.
0	[0]	r	Fixed to 0

The STKOV register can be updated via any instruction capable of modifying a SFR.

Note: The Stack Pointer Segment Register SPSG is not taken into account for the stack pointer comparison. The system stack cannot cross a 64k segment.

This checking mechanism is triggered before every implicit write access. The contents of the stack pointer is compared with the contents of the overflow register, whenever the SP is to be decremented either by a CALLA, CALLI, CALLR, CALLS, PCALL, TRAP, SCXT or PUSH instruction.

Note: If the Stack Pointer was explicitly changed as a result of move or arithmetic instruction, SP is not compared to the contents of the STKOV. Therefore, if the modified Stack Pointer is below the limit set by STKOV register, the stack violation will not be detected. The stack overflow can be detected only if the contents of SP are equal to (not less than) the contents of the STKOV and only in case of implicit SP modification. This means that SP may be explicitly set to the value below permitted SP range and even be operated there without triggering any traps. However, if SP crosses the limit of the permitted SP range from outside the range as a result of implicit change (PUSH for example), the event (SP) = (STKOV) will

trigger the corresponding trap. Note that event (SP) = (STKOV) resulting from an explicit SP modification does not trigger the trap.

The Stack Overflow Trap is triggered when (SP) = (STKOV) and if SP is to be implicitly decremented. This trap may be used in two different ways:

- **Fatal error indication** treats the stack overflow as a system error and executes associated trap service routine. Under these circumstances, data in the bottom of the stack may have been overwritten by the status information stacked upon servicing the stack overflow trap.
- **Automatic system stack flushing** allows the system stack to be used as a 'Stack Cache' for a bigger external user stack.

The Stack Underflow Pointer STKUN

This non-bit addressable register STKUN is compared with the SP register before each implicit read operation that increments the contents of the SP register. If the contents of the SP register are equal to the contents of the STKUN register, a stack underflow hardware trap will occur.

STKUN
Stack Underflow Pointer **SFR** **Reset Value: $FC00_H$**

Field	Bits	Type	Description
STKUN	[15:1]	rw	**Modifiable portion of register STKUN** Specifies the segment offset address of the upper limit of the system stack.
0	[0]	r	Fixed to 0

The STKUN register can be updated via any instruction capable of modifying a SFR.

Note: The Stack Pointer Segment Register SPSG is not taken into account for the stack pointer comparison. The system stack cannot cross a 64 k segment.

This checking mechanism is triggered before each implicit read access. The contents of the stack pointer are compared to the contents of the underflow register, whenever the SP will be incremented either by a RET, RETS, RETP, RETI or POP instruction.

Note: If the Stack Pointer was explicitly changed as a result of move or arithmetic instruction, SP is not compared to the contents of the STKUN register. Therefore, if the modified Stack Pointer is above the limit set by STKUN register, the stack

violation will not be detected. The stack underflow can be detected only if the contents of SP are equal to (not higher than) the contents of the STKUN and only in case of implicit SP modification. This means that SP may be explicitly set to the value above the permitted SP range and even be operated there without triggering any traps. However, if SP crosses the limit of the permitted SP range from outside the range as a result of an implicit change (POP instruction, for example), the event (SP) = (STKUN) will trigger the corresponding trap. Note that event (SP) = (STKUN) resulting from an explicit SP modification does not trigger the trap.

The Stack Underflow Trap is triggered when (SP) = (STKUN) and if SP is to be implicitly incremented. This trap may be used in two different ways:

Fatal error indication treats the stack underflow as a system error and executes associated trap service routine.

• **Automatic system stack refilling** allows use of the system stack as a 'Stack Cache' for a bigger external user stack.

Scope of Stack Limit Control

The stack limit control implemented by the register pair STKOV and STKUN detects cases in which the Stack Pointer (SP) crosses the defined stack area as a result of implicit change.

Note: If a stack overflow or underflow event occurs in an ATOMIC/EXT sequence, the stack operations that are part of the sequence are completed. The trap is issued after the completion of the entire ATOMIC/EXT sequence.

2.6 Data Processing

All standard arithmetic, shift and logical operations are performed in the 16-bit ALU. In addition to the standard arithmetic and logic unit, the ALU of the C166S V2 CPU includes bit manipulation, multiply and divide unit. Most internal execution blocks have been optimized to perform operations on either 8-bit or 16-bit numbers. After the pipeline has been filled, most instructions are completed in one CPU cycle. The status flags are automatically updated in the PSW register after each ALU operation (see **Section 2.6.6**). These flags allow branching upon specific conditions. Support of both signed and unsigned arithmetic is provided by the user selectable branch test. The status flags are also preserved automatically by the CPU upon entry into an interrupt or trap routine.

2.6.1 Data Types

The C166S V2 CPU supports operations on booleans/bits, bit strings, characters, integers, and signed fraction numbers. Most instructions operate with specific data types, while others are useful for manipulating several data types.

The C166S V2 CPU data formats are able to support all ANSI C data types. Additional to the ANSI C data types, some C-Compilers support new types that allow efficient use of the bit manipulation instructions in embedded control applications.. .

Table 2-10 ANSI C Data Types

ANSI C Data Types	Size (bytes)	Range	CPU Data Format
bit	1 bit	0 or 1	BIT
sfrbit	1 bit	0 or 1	BIT
esfrbit	1 bit	0 or 1	BIT
signed char	1	-128 to +127	BYTE
unsigned char	1	0 to 255U	BYTE
sfr	1	0 to 65535U	WORD
esfr	1	0 to 65535U	WORD
signed short	2	-32768 to 32767	WORD
unsigned short	2	0 to 65535U	WORD
bitword	2	0 to 65535U	WORD or BIT
signed int	2	-32768 to 32767	WORD
unsigned int	2	0 to 65535U	WORD
signed long	4	-2147483648 to +2147483647	Not directly supported
unsigned long	4	0 to 4294967295UL	Not directly supported
float	4	+/-1,176E-38 to +/-3,402E+38	Not directly supported
double	8	+/- 2,225E-308 to +/- 1,797E+308	Not directly supported
long double	8	+/- 2,225E-308 to +/- 1,797E+308	Not directly supported
near pointer	2	16/14 bits depending on memory model	WORD
far pointer	4	14 bits (16 k) in any page	Not directly supported

Table 2-11 CPU Data Formats

CPU Data Format	Size (bytes)	Range
BIT	1 bit	0 or 1
BYTE	1	0 to 255U or -128 to +127
WORD	2	0 to 65535U or -32768 to 32767

2.6.2 Constants

In addition to the powerful addressing modes, the C166S V2 CPU instruction set also supports the use of wordwide or bytewide immediate constants. For optimum utilization of the available code storage, these constants are represented in the instruction formats by either 3, 4, 8, or 16 bits. The short constants are always zero-extended, while the long constants are truncated if necessary, to match the data format required for the particular operation (see table below): .

Table 2-12 Constant Formats

Mnemonic	Word Operation	Byte Operation
#data3	0000_H + data3	00_H + data3
#data4	0000_H + data4	00_H + data4
#data8	0000_H + data8	data8
#data16	data16	data16 $\wedge$ FF_H
#mask	0000_H + mask	mask

Note: Immediate constants are always signified by a leading sign '#'.

2.6.3 16-bit Adder/Subtracter, Barrel Shifter, and 16-bit Logic Unit

All standard arithmetic and logical operations are performed by the 16-bit ALU. In case of byte operations, signals from bits 6 and 7 of the ALU result are used to control the condition flags. Multiple precision arithmetic is supported by a “CARRY-IN” signal to the ALU from previously calculated portions of the desired operation.

A 16-bit barrel shifter provides multiple bit shifts in a single cycle. Rotations and arithmetic shifts are also supported.

2.6.4 Bit Manipulation Unit

C166S V2 CPU offers a large number of instructions for bit processing. The special bit manipulation unit was implemented for this purpose. The bit manipulation instructions enable efficient control and testing of peripherals. Unlike other microcontrollers,

C166S V2 CPU features instructions that provide direct access to two operands in the bit addressable space without requiring them to be moved to temporary locations.

The same logical instructions that are available for words and bytes can also be used for bits. The user can compare and modify a control bit for a peripheral in one instruction. Multiple bit shift instructions have been included to avoid long instruction streams of single bit shift operations. These instruction require a single CPU cycle. Additionally, bit field instructions enable are able to modify the multiple bits in one operand in a single instruction.

All instructions that manipulate single bits or bit groups internally use a read-modify-write sequence that accesses the whole word containing the specified bit(s).

This method has several consequences:

- Bits can be modified only within the internal address areas, i.e. internal RAM and SFRs. External locations cannot be used with bit instructions.

The upper 256 bytes of the SFR area, the ESFR area, and the internal RAM are bit addressable, i.e. those register bits located within the respective sections can be directly manipulated using bit instructions. The other SFRs must be accessed byte/word wise.

Note: All GPRs are bit addressable independent of the allocation of the register bank via the Context Pointer (CP). Even GPRs allocated to not bit addressable RAM locations provide this feature.

- The read-modify-write approach may be critical with hardware-effected bits. In such cases, the hardware may change specific bits while the read-modify-write operation is in progress, where the write back would overwrite the new bit value generated by the hardware. The solution is either the implemented hardware protection (see below) or realized through special programming (see **Section 4.1**).

Protected bits are not changed during the read-modify-write sequence, that is, when hardware sets something like an interrupt request flag between the read and the write of the read-modify-write sequence. The hardware protection logic guarantees that only the intended bit(s) is/are effected by the write-back operation.

Note: If a conflict occurs between a bit manipulation generated by hardware and an intended software access, the software access has priority and determines the final value of the respective bit.

2.6.5 Multiply and Divide Unit

The C166S V2 CPU multiply and divide unit has two separated parts. One is the fast 16x16-bit multiplier that executes a multiplication in one CPU cycle. The other one is a division sub-unit which performs the division algorithm in 21 CPU cycles maximum. According to the data and division types, the division length varies between 18 and 21 cycles. The divide instruction requires four CPU cycles to be executed. For performance reasons, the rest of the division algorithm runs in the background during the following

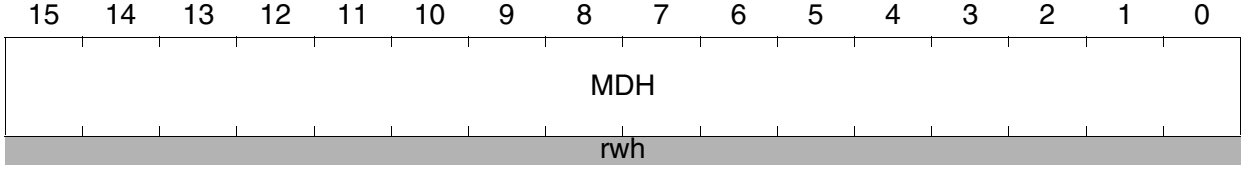
seventeen CPU cycles, while further instructions are executed in parallel. If another instruction tries to use the unit while a division is still running, the execution of this new instruction is stalled until the division is finished.

Interrupt tasks can also be started and executed immediately without any delay. The previous division will be finished in the background. If an instruction of the interrupt task uses the multiply and divide unit before the previous division process is finished, the instruction flow will be stalled as well. To avoid these stalls, the multiply and division unit should not be used during the first fourteen CPU cycles of the interrupt tasks. This requires up to fourteen one-cycle instructions to be executed between the interrupt entry and the first instruction which uses the multiply and divide unit again (worst case).

The Multiply/Divide High Register MDH

The sixteen bit, non-bit addressable MDH register contains the high word of the 32-bit multiply/divide MD register used by the CPU when it performs a multiplication or a division using implicit addressing (DIV, DIVL, DIVLU, DIVU, MUL, MULU). After an implicitly addressed multiplication, this register represents the high order sixteen bits of the 32-bit result. For long divisions, the MDH register must be loaded with the high order sixteen bits of the 32-bit dividend before the division has started. After any division, the MDH register represents the 16-bit remainder.

MDH
Multiply Divide High Word **SFR** **Reset Value: 0000_H**



Field	Bits	Type	Description
MDH	[15:0]	rwh	**High part of MD** The high order sixteen bits of the 32-bit multiply and divide register MD.

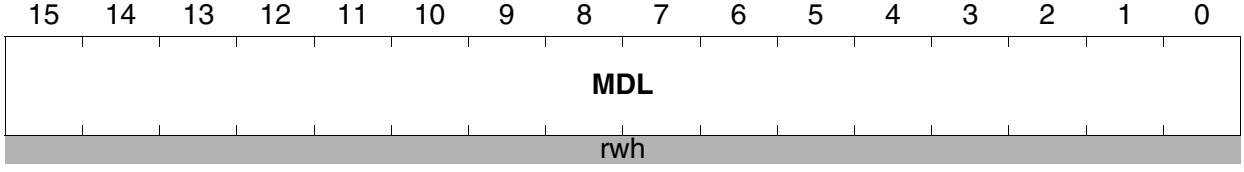
Whenever this register is updated via software, the Multiply/Divide Register In Use (MDRIU) flag in the Multiply/Divide Control register (MDC) is set to 1.

The Multiply/Divide Low Register MDL

The sixteen bit, non-bit addressable MDL register contains the low word of the 32-bit multiply/divide MD register used by the CPU when it performs a multiplication or a division using implicit addressing (DIV, DIVL, DIVLU, DIVU, MUL, MULU). After a

multiplication, this register represents the low order sixteen bits of the 32-bit result. For long divisions, the MDL register must be loaded with the low order sixteen bits of the 32-bit dividend before the division has started. After any division, the MDL register represents the 16-bit quotient.

MDL
Multiply Divide Low Word **SFR** **Reset Value: 0000_H**



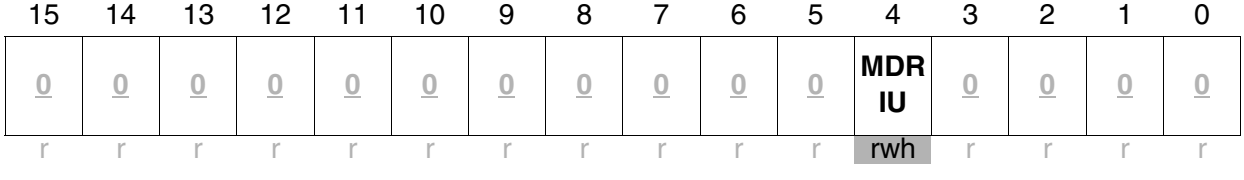
Field	Bits	Type	Description
MDL	[15:0]	rwh	**Low part of MD** The low order 16 bits of the 32-bit multiply and divide register MD.

Whenever this register is updated via software, the Multiply/Divide Register In Use (MDRIU) flag in the Multiply/Divide Control register (MDC) is set to 1. The MDRIU flag is cleared whenever the MDL register is read via software.

The Divide Control Register MDC

This bit addressable 16-bit register is implicitly used by the CPU when it performs a division or multiplication in the ALU.

MDC
Multiply Divide Control **SFRb** **Reset Value: 0000_H**



Field	Bits	Type	Description
MDRIU	[4]	rwh	**Multiply/Divide Register In Use** 0: Cleared when MDL is read via software. 1: Set when MDL or MDH is written via software, or when a multiply or divide instruction is executed.

The MDRIU flag is the only portion of the MDC register used for multiplication and division within the C166S V2 CPU. This bit indicates the usage of the MDL and MDH register. It must be stored prior to a new multiplication or division operation. The remaining portions of the MDC register are never used by the dedicated multiplication and division hardware.

2.6.6 The Processor Status Word PSW

This bit addressable register reflects the current status of the microcontroller. Two groups of bits represent the current ALU status and the current CPU interrupt status. Two separate bits (USR0 and USR1) within register PSW are provided as general purpose flags.

PSW
Processor Status Word **SFRb** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				**IEN**	**HLD EN**	**BANK**		**USR 1**	**USR 0**	**MUL IP**	**E**	**Z**	**V**	**C**	**N**
rwh				rw	rw	rwh		rwh	rwh	r	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
ILVL	[15:12]	rwh	**CPU Priority Level** 0_H Lowest Priority ... ... F_H Highest Priority
IEN	[11]	rw	**Interrupt/PEC Enable Bit (globally)** 0 Interrupt/PEC requests are disabled 1 Interrupt/PEC requests are enabled
HLDEN	[10]	rw	**Hold Enable** 0 external bus arbitration disabled 1 external bus arbitration enabled
BANK	[9:8]	rwh	**Reserved for Register File Bank Selection** 00 Global register bank 01 Reserved 10 Local register bank 1 11 Local register bank 2
USR1	[7]	rwh	**General Purpose Flag** May be used by application
USR0	[6]	rwh	**General Purpose Flag** May be used by application

Field	Bits	Type	Description
MULIP	[5]	r	**Multiplication/Division in progress** Always set to 0
E	[4]	rwh	**End of Table Flag** 0 Source operand is neither 8000_h nor 80_h 1 Source operand is 8000_h or 80_h
Z	[3]	rwh	**Zero Flag** 0 ALU result is not zero 1 ALU result is zero
V	[2]	rwh	**Overflow Flag** 0 No Overflow produced 0 Overflow produced
C	[1]	rwh	**Carry Flag** 0 No carry/borrow bit produced 1 Carry/borrow bit produced
N	[0]	rwh	**Negative Result** 0 ALU result is not negative 1 ALU result is negative

ALU Status (N, C, V, Z, E, MULIP)

The condition flags (N, C, V, Z, E) within the PSW indicate the ALU status resulting from the last performed ALU operation. They are set by the majority of instructions according to the specific rules depending on the ALU operation or data movement.

After execution of an instruction which explicitly updates the PSW register, the condition flags may no longer represent an actual CPU status. An explicit write operation to the PSW register supersedes the condition flag values implicitly generated by the CPU. An explicit read access to the PSW register returns the value of the PSW register after execution of the immediately preceding instruction.

Note: After reset, all of the ALU status bits are cleared.

- **N-Flag:** For the majority of ALU operations, the N-flag is set to 1, if the most significant bit of the result contains a 1; otherwise, it is cleared. In the case of integer operations, the N-flag can be interpreted as the sign bit of the result (negative: N = 1, positive: N = 0). Negative numbers are always represented as the 2s complement of the corresponding positive number. The range of signed numbers extends from '$–8000_H$' to '$+7FFF_H$' for the word data type, or from '$–80_H$' to '$+7F_H$' for the byte data type. For Boolean bit operations with only one operand, the N-flag represents the previous state of the specified bit. For Boolean bit operations with two operands, the N-flag represents the logical XORing of the two specified bits.

- **C-Flag:** After an addition, the C-flag indicates that a “Carry” from the most significant bit of the specified word or byte data type has been generated. After a subtraction or a comparison, the C-flag indicates a “Borrow” which represents the logical negation of a “Carry” for the addition.
  This means that the C-flag is set to 1, if **no** carry from the most significant bit of the specified word or byte data type has been generated during a subtraction. Subtraction is performed by the ALU as a 2s complement addition. The C-flag is cleared when this complement addition causes a “Carry”.

  The C-flag is always cleared for logical, multiply and divide ALU operations, because these operations cannot cause a “Carry” flag to be set.
  For shift and rotate operations, the C-flag represents the value of the bit shifted out last. If a shift count of zero is specified, the C-flag will be cleared. The C-flag is also cleared for a Prioritize operation, because a 1 is never shifted out of the MSB during the normalization of an operand.
  For Boolean bit operations with only one operand, the C-flag is always cleared. For Boolean bit operations with two operands, the C-flag represents the logical ANDing of the two specified bits.

- **V-Flag:** The addition, subtraction and 2's complement operations set the V-flag to '1' if the result exceeds the range of 16 bit signed numbers for word operations ('-8000_H' to '$+7FFF_H$'), or 8 bit signed numbers for byte operations ('-80_H' to '$+7F_H$'). Otherwise, the V-flag is cleared. Note, that the result of an integer addition, integer subtraction, or 2's complement is not valid if the V-flag indicates an arithmetic overflow.
  For multiplication and division the V-flag is set to 1 if the result can not be represented in a word data type, otherwise it is cleared. Note that a division by zero will always cause an overflow. Unlike the division result, the result of multiplication is valid regardless of V-flag value.
  Since the logical ALU operations cannot produce an invalid result, the V-flag is cleared by these operations.

  The V-flag is also used as 'Sticky Bit' for rotate right and shift right operations. Using only the C-flag, a rounding error caused by a shift right operation can be estimated as up to one half of the LSB of the result. In conjunction with the V-flag, the C-flag allows evaluation of the rounding error with a finer resolution (see table below).
  For Boolean bit operations with only one operand, the V-flag is always cleared. For Boolean bit operations with two operands, the V-flag represents the logical ORing of the two specified bits.

Shift Right Rounding Error Evaluation

- **Z-Flag:** The Z-flag is normally set to 1 if the result of an ALU operation equals zero; otherwise, it is cleared.

C-Flag	V-Flag	Rounding Error Quantity	
0	0	No rounding error	
0	1	0 < Rounding error	< $^1/_2$ LSB
1	0	Rounding error	= $^1/_2$ LSB
1	1	Rounding error	> $^1/_2$ LSB

For addition and subtraction with “Carry”, the Z-flag is only set to 1 if the Z-flag already contains a 1 as a result from previous operation and the result of the current ALU operation also equals zero. This mechanism supports the multiple precision calculations.
For Boolean bit operations with only one operand, the Z-flag represents the logical negation of the previous state of the specified bit. For Boolean bit operations with two operands, the Z-flag represents the logical NORing of the two specified bits. For the Prioritize operation, the Z-flag indicates whether the second operand was zero or not.

- **E-Flag:** End of table flag. The E-flag can be altered by the instructions which perform ALU or data movement operations. The E-flag is cleared by those instructions that cannot be reasonably used for table search operations. In all other cases, the E-flag value depends on the value of the source operand to signify whether the end of a search table is reached or not. If the value of the source operand of an instruction equals the lowest negative number which depends on the data format of the corresponding instruction ('8000_H' for the word data type, or '80_H' for the byte data type), the E-flag is set to 1; otherwise, it is cleared.

- **MULIP-Flag:** The MULIP-flag always sticks to 0.

Note: The MULIP flag is a part of the C166 task environment. For compatibility reasons, the bit is still implemented even if not used. A multiply and divide ALU operation of the C166S V2 CPU is no longer interruptible.

- **BANK:** The BANK bitfield of the PSW registers indicates which one of the three physical register banks is activated. The BANK field is updated by hardware upon entry into an interrupt service routine, but it can be also modified by software. The BANK field can be changed explicitly by any instruction which can write to the PSW. Also, it is implicitly updated by the RETI instruction.
- **HLDEN:** Refer to EBC **Chapter 6.4.1**.

CPU Interrupt Status (IEN, ILVL)

The Interrupt Enable bit allows global enable (IEN=1) or disable (IEN=0) of interrupts. The 4-bit Interrupt Level field (ILVL) specifies the priority of the current CPU activity. The interrupt level is updated by hardware upon entry into an interrupt service routine, but it can also be modified via software to prevent other interrupts from being acknowledged. In case an interrupt level '15' has been assigned to the CPU, it has the highest possible

priority, and thus the current CPU operation cannot be interrupted except by hardware traps or external non-maskable interrupts. For details please, refer to **Section 5** “Interrupt and Trap Functions”.

After reset, all interrupts are globally disabled and the lowest priority (ILVL=0) is assigned to the initial CPU activity.

2.7 Parallel Data Processing

The new CoXXX arithmetic instructions are performed in the MAC unit. The MAC unit provides single instruction-cycle, non-pipelined, 32-bit additions; 32-bit subtraction; right and left shifts; 16-bit by 16-bit multiplication; and multiplication with cumultative subtraction/addition. The MAC unit includes the following major components, shown in **Figure 2-21**:

- 16-bit by 16-bit signed/unsigned multiplier with signed result[1)]
- Concatenation Unit
- Scaler (one-bit left shifter) for fractional computing
- 40-bit Adder/Subtracter
- 40-bit Signed Accumulator
- Data Limiter
- Accumulator Shifter
- Repeat Counter

1) The same hardware-multiplier is used in the ALU.

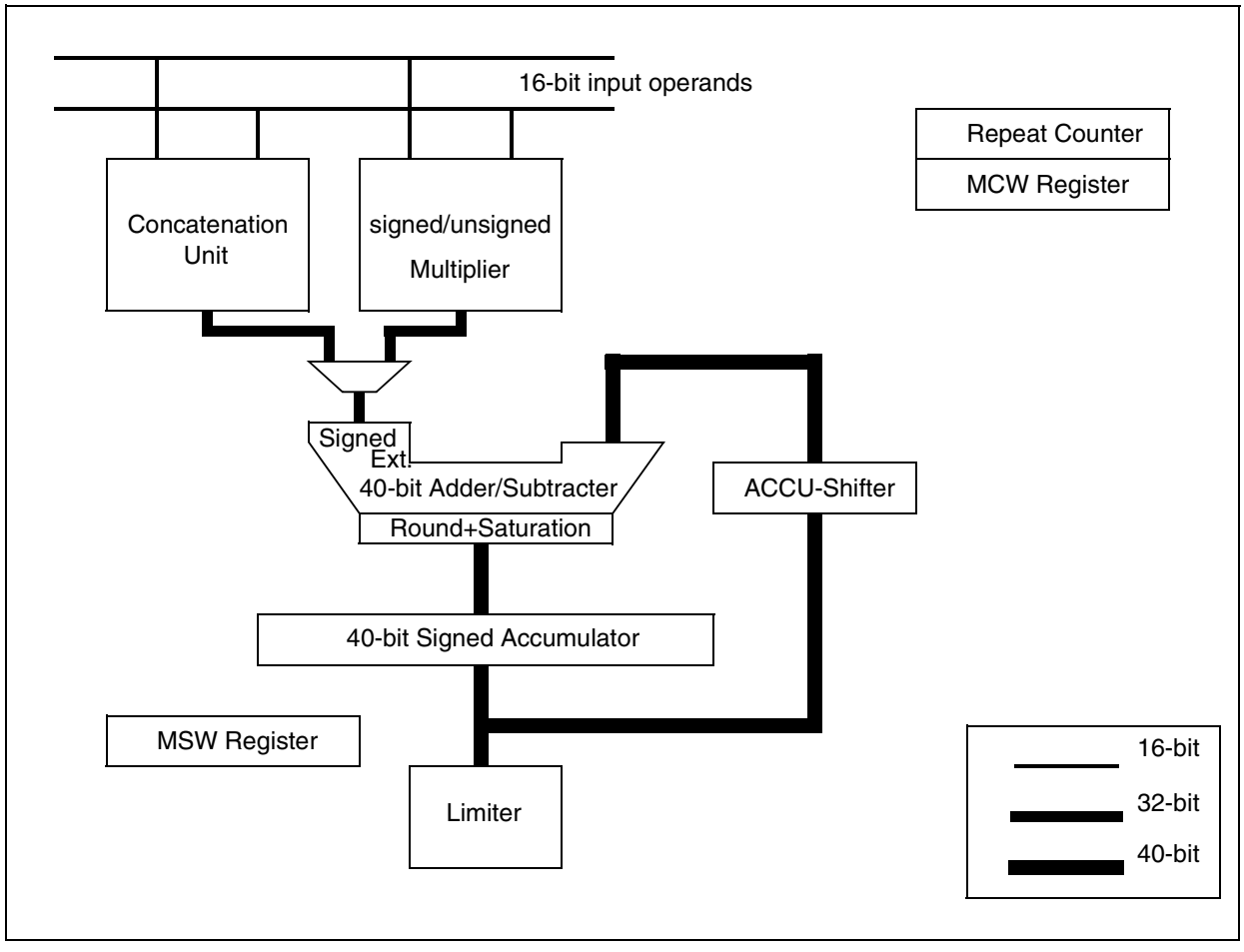


Figure 2-21 Functional MAC Unit Block Diagram

The working register of the MAC Unit is a dedicated 40-bit wide Accumulator register. A set of consistent flags is automatically updated in the MSW register (see **Section 2.7.10**) after each MAC operation. These flags allow branching on specific conditions. Unlike the PSW flags, these flags are not preserved automatically by the CPU upon entry into an interrupt or trap routine. All dedicated MAC registers must be saved on the stack if the MAC unit is shared between different tasks and interrupts.

2.7.1 Representation of Numbers and Rounding

The C166S V2 CPU supports the 2s complement representation of binary numbers. In this format, the sign bit is the MSB of the binary word. This is set to zero for positive numbers and set to one for negative numbers. Unsigned numbers are supported only by multiply/multiply-accumulate instructions which specify whether each operand is signed or unsigned.

In 2s complement fractional format, the N-bit operand is represented using the 1.[N-1] format (1 signed bit, N-1 fractional bits). Such a format can represent numbers between -1 and $+1-2^{-[N-1]}$. This format is supported when MP of MCW is set.

The C166S V2 CPU implements 2s complement rounding'. With this rounding type, one is added to the bit to the right of the rounding point (bit 15 of MAL), before truncation (MAL is cleared).

2.7.2 The 16-bit by 16-bit signed/unsigned Multiplier and Scaler

The multiplier executes 16-bit by 16-bit parallel signed/unsigned fractional and integer multiplication in one CPU-cycle. The multiplier allows the multiplication of unsigned and signed operands. The result is always presented in a signed fractional or integer format.

The result of the multiplication feeds a one-bit Scaler to allow compensation for the extra sign bit gained in multiplying two 16-bit 2s complement numbers.

2.7.3 Concatenation Unit

The Concatenation Unit enables the MAC unit to perform 32-bit arithmetic operations in one CPU cycle. The Concatenation Unit concatenates two 16-bit operands to a 32-bit operand before the 32-bit arithmetic operation is executed in the 40-bit adder/subtracter. The second required operand is always the current Accumulator contents. The Concatenation Unit is also used to pre-load the Accumulator with a 32-bit value.

2.7.4 One-bit Scaler

The One-bit scaler can shift the result of the concatenation unit or the output of the multiplier one bit to the left. The scaler is controlled by the executed instruction for the concatenation or by the MP control bit.

The product is shifted one bit to the left to compensate for the extra sign bit gained in multiplying two 16-bit 2s complement numbers. The enabled automatic shift is performed only if both input operands are signed.

MCW
MAC Control Word **SFRb** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	**MP**	MS	0	0	0	0	0	0	0	0	0
r	r	r	r	r	rw	rw	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
MP	[10]	rw	**One-bit scaler control** 0 Multiplier product shift disabled 1 Multiplier product shift enabled

- **MP-Control Bit:** If the MP mode bit is set and both multiplier operands are signed types, the multiplier output is automatically shifted left by one bit. In the case of a multiply and accumulate operation, the output of the multiplier is shifted before being added to the accumulator.

2.7.5 The 40-bit Adder/Subtracter

The 40-bit adder/Subtracter allows intermediate overflows in a series of multiply/ accumulate operations. The adder/Subtracter has two input ports. The 40-bit port is the feedback of the Accumulator output through the ACCU-Shifter to the Adder/Subtracter. The 32-bit port is the input port for the operand coming from the One-bit Scaler. The 32-bit operands are signed and extended to 40-bits before the addition/subtraction is performed.

The output of the Adder/Subtracter goes to the Accumulator. It is also possible to round the result and to saturate it on a 32-bit value automatically after every accumulation. The round operation is performed by adding $00'00008000_H$ to the result. Automatic saturation is enabled by setting the saturation bit, the MAC Control Word (MCW).

MCW
MAC Control Word **SFRb** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	MP	MS	0	0	0	0	0	0	0	0	0
r	r	r	r	r	rw	rw	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
MS	[9]	rw	**Saturation control** 0 Saturation disabled 1 Saturation enabled

- **MS-Control Bit:** If the MS mode bit is set, the accumulator will be automatically saturated to 32-bits. The MAC Unit supports signed saturation.

When the accumulator is in the overflow saturation mode and an overflow occurs, the accumulator is loaded with either the most positive or the most negative value representable in a 32-bit value, depending on the direction of the overflow as well as the arithmetic used. The value of the accumulator upon saturation is 00'7fff'ffffh (positive) or ff'8000'0000h (negative).

2.7.6 The Data Limiter

Saturation arithmetic is also provided to selectively limit overflow when reading the accumulator by means of a ***CoSTORE <destination>., MAS*** instruction. Limiting is

performed on the MAC-Unit accumulator. If the contents of the Accumulator can be represented in the destination operand size without overflow, then the data limiter is disabled and the operand is not modified. If the contents of the accumulator cannot be represented without overflow in the destination operand size, the limiter will substitute a “limited” data as explained in the next table:

Table 2-13 Limiter Output

ME-flag	MN-flag	Output of Limiter
0	x	unchanged
1	0	$7FFF_H$
1	1	8000_H

Notice that in this particular case, both the accumulator and the status register are not affected. MAS is readable by means of a CoSTORE instruction only.

2.7.7 The Accumulator Shifter

The accumulator shifter is a parallel shifter with a 40-bit input and a 40 bit output. The source accumulator shifting operation are:

- No shift (Unmodified)
- Up to 16-bit Arithmetic Left Shift
- Up to 16-bit Arithmetic Right Shift

Notice that the ME, MSV, and MSL bits from MSW are affected by left shifts; therefore, if the saturation mechanism is enabled (MS), the behavior is similar to the one of the Adder/Subtracter.

Note: Certain precautions are required in case of left shift with saturation enabled. Generally, if MAE contains significant bits, then the 32-bit value in the accumulator is to be saturated. However, it is possible that left shift may move some significant bits out of the Accumulator. The 40-bit result will be misinterpreted and will be either not saturated or saturated incorrectly. There is a chance that the result of left shift may produce a result which can saturate an original positive number to the minimum negative value, or vice versa.

2.7.8 The 40-bit Signed Accumulator Register

The 40-bit Accumulator consists of three smaller registers, MAH, MAL, and MAE. MAH and MAL are 16 bits wide; MAE is 8 bits wide. MAE is the Most Significant Byte of the 40-bit accumulator. This byte performs a guarding function. MAE is accessed as the Least Significant Byte of MSW.

When MAH is written, the value in the accumulator is automatically adjusted to signed extended 40-bit format. That means MAE will be automatically loaded by zeros for the positive number (MAH has 0 in the most significant bit). In the case of the negative

number (MAH has 1 in the most significant bit), the MAE will be loaded with ones, representing the extended 40-bit negative number in 2s compliment notation. One may see that the extended 40-bit value is equal to 32-bit value without extension. In other words, after this extension, MAE does not contain significant bits. Generally, this condition is present when the highest 9 bits of the 40-bit signed result are the same.

During the accumulator operations, an overflow may happen and the result may not fit into 32-bits and the MAE will change. The extension flag “E”, which is the part of the most significant byte of MSW, is set when the signed result in the accumulator has overflowed the 32-bit boundary. This condition is present when the highest 9 bits of the 40-bit signed result are not the same, i.e. MAE contains significant bits.

Most CoXXX operations specify the 40-bit accumulator register as a source and/or a destination operand.

The MAC Unit Accumulator Extension Byte MAE

The MAE register is a part of the 40-bit MAC unit accumulator register. MAE is accessed as the Least Significant Byte of **MSW**. It is implicitly used by the MAC unit for MAC operation. In case a word operand is written into MAH, the MAE register becomes sign-extended. It can be accessed via any instruction capable of accessing an SFR.

MSW
MAC Status Word **SFRb** **Reset Value: 0000_H**

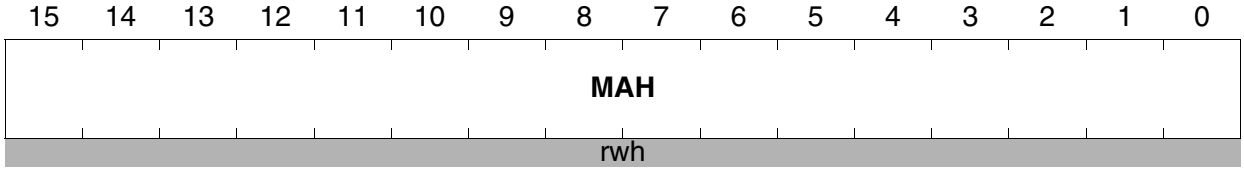
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	MV	MSL	ME	MSV	MC	MZ	MN	MAE							
r	rwh														

Field	Bits	Type	Description
MAE	[7:0]	rwh	The most significant bits of the 40-bit Accumulator

The MAC Unit Accumulator High Word MAH

The MAH register is a part of the 40-bit MAC unit accumulator register. It is implicitly used by the MAC unit for MAC operation. In case the word operand is written into MAH, MAL acquires the zero value and the MAE register becomes sign-extended. It can be accessed via any instruction capable of accessing an SFR.

MAH
Accumulator High Word **SFR** **Reset Value: 0000_H**

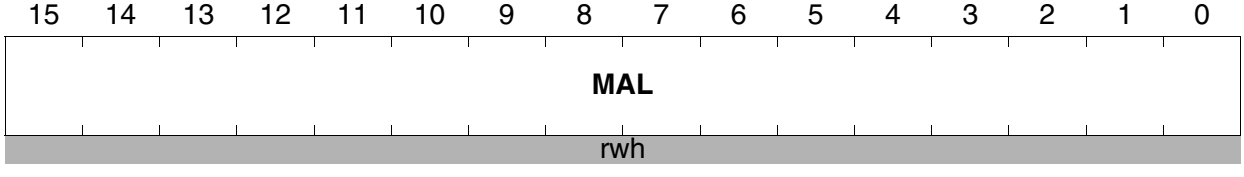


Field	Bits	Type	Description
MAH	[15:0]	rwh	**High part of Accumulator** The middle (bits 31 to 16) word of the 40-bit MAC Accumulator.

The MAC Unit Accumulator Low Word MAL

The MAL register is a part of the 40-bit MAC unit accumulator register. It is implicitly used by the MAC Unit for MAC operation. In case of explicit write access to MAH, MAL receives a zero value. It can be accessed via any instruction capable of accessing an SFR.

MAL
Accumulator Low Word **SFR** **Reset Value: 0000_H**



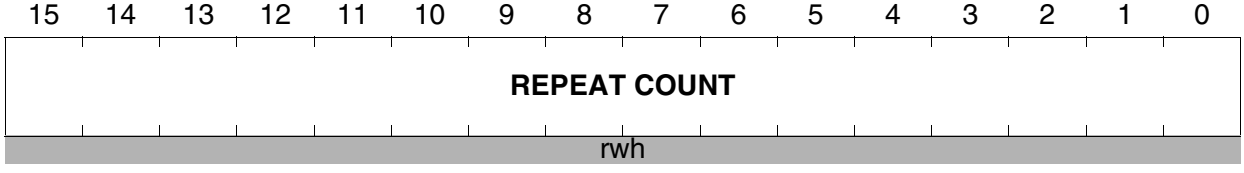
Field	Bits	Type	Description
MAL	[15:0]	rwh	**Low part of Accumulator** The low order 16 bits of the 40-bit MAC Accumulator.

2.7.9 The Repeat Counter MRW

The Repeat Counter MRW controls the number of repetitions a loop must be executed. The register must be pre-loaded before it can be used with -USRx CoXXX operations. MAC operations are able to decrement this counter. When an -USRx CoXXX instruction is executed, the MRW is checked on the zero value **before** the MRW is decremented. If

the MRW equals zero, the USRx bit is set and MRW is not further decremented. The **MRW** can be accessed via any instruction capable of accessing a SFR.

MRW
MAC Repeat Word **SFRb** **Reset Value: 0000_H**



Field	Bits	Type	Description
REPEAT COUNT	[15:0]	rwh	16-bit loop counter

All CoXXX instructions have a 3-bit wide repeat control field 'rrr' in the operand field to control the MRW repeat counter. It is located within CoXXX instructions at bit positions [31:29].

– '000' -> regular CoXXX instruction.
– '001' -> RESERVED
– '010' -> '**-** USR0 CoXXX' instruction, decrements repeat counter.
– '011' -> '**-** USR1 CoXXX' instruction, decrements repeat counter.
– '1xx' -> RESERVED.

The following example shows a loop which is executed 20 times. Every time the CoMACM instruction is executed, the MRW counter is decremented.

```
              mov      MRW, #19
loop01:
- USR1        CoMACM   [IDX0+], [R0+]
              ADD      R2,#2
              JMPA     cc_nusr1, loop01
```

Because correctly predicted JMPA is executed in 0-cycle, it offers the functionality of a repeat instruction.

Note: The USR0 bit should be used carefully because this bit was pre-existing and, therefore, may have been used by programmer or compiler.

2.7.10 The MAC Unit Status Word MSW

The MSW bit addressable register shows the current MAC Unit state. Two groups of bits represent the current MAC Unit status and the eight additional extension bits belonging to the MAC accumulator.

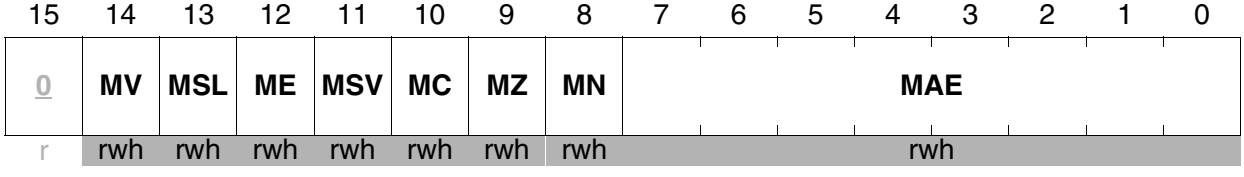
MAC Unit Status (MV, MN, MZ, MC, MSV, ME, MSL)

The condition flags (MV, MN, MZ, MC, MSV, ME, MSL) within the MSW indicate the MAC resulting from the most recently performed MAC operation. These flags are controlled by the majority of the MAC instructions according to specific rules. Those rules depend on the instruction managing the MAC or data movement operation.

After execution of an instruction which explicitly updates the MSW register, the condition flags may no longer represent an actual MAC status. An explicit write operation to the MSW register supersedes the condition flag values implicitly generated by the MAC unit. An explicit read access to the MSW register returns the value of the MSW register after execution of the immediately preceding instruction. The MSW register can be accessed via any instruction capable of accessing an SFR.

Note: After reset, all MAC status bits are cleared.

MSW
MAC Status Word **SFRb** **Reset Value: 0000_H**



Field	Bits	Type	Description
MAE	[7:0]	rwh	The most significant bits of the 40-bit Accumulator
MN	[8]	rwh	**Negative Result** 0 MAC result is positive 1 MAC result is negative
MZ	[9]	rwh	**Zero Flag** 0 MAC result is not zero 1 MAC result is zero
MC	[10]	rwh	**Carry Flag** 0 No carry/borrow produced 1 Carry/borrow produced
MSV	[11]	rwh	**Sticky Overflow Flag** 0 No Overflow occurred 1 Overflow occurred

Field	Bits	Type	Description
ME	[12]	rwh	**MAC Extension Flag** 0 MAE does not contain significant bits 1 MAE contains significant bits
MSL	[13]	rwh	**Sticky Limit Flag** 0 Result was not saturated 1 Result was saturated
MV	[14]	rwh	**Overflow Flag** 0 No Overflow produced 1 Overflow produced

- **Accu Extension MAE:** These 8 bits are part of the 40-bit accumulator register. The MAC Unit implicitly uses these bits during a MAC operation. When writing to the MAH, the MAE is automatically signed extended with the most significant bit of the MAH register.

- **MN-Flag:** For the majority of the MAC operations, the MN-flag is set to 1 if the most significant bit of the result contains a 1; otherwise, it is cleared. In the case of integer operations, the MN-flag can be interpreted as the sign bit of the result (negative: MN=1, positive: MN=0). Negative numbers are always represented as the 2s complement of the corresponding positive number. The range of signed numbers extends from '8000000000_H' to '$7FFFFFFFFF_H$'.

- **MZ-Flag:** The MZ-flag is normally set to 1 if the result of a MAC operation equals zero; otherwise, it is cleared.

- **MC-Flag:** After a MAC addition, the MC-flag indicates that a “Carry” from the most significant bit of the accumulator extension MAE has been generated. After a MAC subtraction or a MAC comparison, the MC-flag indicates a “Borrow” representing the logical negation of a “Carry” for the addition. This means that the MC-flag is set to 1, if **no** “Carry” from the most significant bit of the Accumulator has been generated during a subtraction. Subtraction is performed by the MAC Unit as a 2s complement addition and the MC-flag is cleared when this complement addition caused a “Carry”. For left shift MAC operations, the MC-flag represents the value of the bit shifted out last. Right shift MAC operations always clear the MC-flag. The arithmetic right shift MAC operation can set the MC-flag if the enabled round operation generates a “Carry” from the most significant bit of the Accumulator extension MAE.

- **MSV-Flag:** The addition, subtraction, 2s complement, and round operations always set the MSV-flag to 1 if the MAC result overflows the maximum range of 40-bit signed

numbers. If the MSV-flag indicates an arithmetic overflow, the MAC result of an operation is not valid. The MSV-flag is a 'Sticky Bit'. Once set, other MAC operations cannot affect the status of the MSV-flag. Only a direct write operation can clear the MSV-flag.

- **ME-Flag:** The ME-flag is set if the accumulator extension MAE contains significant bits. The ME-flag is set if the nine highest accumulator bits are not all equal.

- **MSL-Flag:** The MSL-flag is set if an automatic saturation of the accumulator has happened. The automatic saturation is enabled if the MS-bit of the MAC Control Word register MCW is set. The MSL-Flag can be also set by instructions which limit the contents of the accumulator. If the accumulator has been limited, the MSL-Flag is set. The MSL-Flag is a 'Sticky Bit'. Once set, it cannot be affected by the other MAC operations. Only a direct write operation can clear the MSL-flag.

- **MV-Flag:** The addition, subtraction, and accumulation operations set the MV-flag to 1 if the result exceeds the maximum range of signed numbers (80'00000000$_H$ to 7F'FFFFFFFF$_H$); otherwise, the MV-flag is cleared. Note that if the MV-flag indicates an arithmetic overflow, the result of the integer addition, integer subtraction, or accumulation is not valid.

2.7.11 The MAC Unit Control Word MCW

This bit addressable register controls the operation of the MAC Unit. It can be accessed via any instruction capable of addressing an SFR.

MCW
MAC Control Word **SFRb** **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	**MP**	**MS**	0	0	0	0	0	0	0	0	0
r	r	r	r	r	rw	rw	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
MP	[10]	rw	**One-bit scaler control** 0 Multiplier product shift disabled 1 Multiplier product shift enabled
MS	[9]	rw	**Saturation control** 0 Saturation disabled 1 Saturation enabled

- **MS-Control Bit:** If the MS mode bit is set, the accumulator will be automatically saturated to 32 bits. The MAC Unit supports signed saturation.

- MP-Control Bit: If the MP mode bit is set and both multiplier operands are of signed types, the multiplier output is automatically shifted left by one bit. In the case of a multiply and accumulate operation, the output of the multiplier is shifted before being added to the accumulator.

2.8 Dedicated CSFRs

The Constant Zeros Register ZEROS

All bits of this bit addressable register are fixed to 0 by hardware. This register is read-only. Register ZEROS can be used as a register-addressable constant of all zeros for bit manipulation or mask generation. It can be accessed via any instruction which is capable of accessing an SFR.

ZEROS
Constant Zeros Register **SFRb** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
0	[all]	r	Fixed to Zero

The Constant Ones Register ONES

All bits of this bit addressable register are fixed to 1 by hardware. This register is read-only. Register ONES can be used as a register-addressable constant of all ones for bit manipulation or mask generation. It can be accessed via any instruction capable of accessing an SFR.

ONES
Constant Ones Register **SFRb** **Reset Value: FFFF$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
1	[all]	r	Fixed to One

CPU Identification Register CPUID

This 16-bit register contains the module and revision number of the implemented C166S V2 core module.

CPUID
CPU Identification Register **ESFR** **Reset Value: 03??$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODULE NUMBER								**VERSION NUMBER**							
r								r							

Field	Bits	Type	Description
MODULE NUMBER	[15:8]	r	**Module Number** 03_H C166S V2 core module number
VERSION NUMBER	[7:0]	r	**Version Number** Version Number

3 C166S V2 Memory Organization

The memory space of the C166S V2 CPU is configured in a “Von Neumann” architecture. This means that code and data are accessed within the same linear address space. All of the physically separated memory areas, including internal ROM/ Flash/DRAM (if integrated into a specific derivative), internal RAM, internal Special Function Register Areas (SFRs and ESFRs), and external memory are mapped into a single common address space.

The C166S V2 CPU provides a total addressable memory space of 16 MBytes. This address space is arranged as 256 segments of 64 KBytes each. Each segment is again subdivided into four data pages of 16 KBytes each (see **Figure 3-1**).

Most internal memory areas are mirrored into the system segment, segment 0. The upper 4 KBytes of segment 0 ($00'F000_H$...$00'FFFF_H$) hold the Special Function Register Areas (SFR and ESFR) and the DPRAM areas.

Data may be stored in any part of the internal memory areas. Code may be stored in any part of the internal memory areas except the SFR blocks, the DPRAM, and Internal SRAM and internal IO area as these areas may be used for control/data, but not for instructions.

The 64 KByte memory area of segment 191 ($BF'0000_H$...$BF'FFFF_H$) cannot be used to store code and data. It is reserved for “on chip” boot and debug/monitor program memories.

Accesses to internal memory areas on devices without the appropriate internal memories will produce unpredictable results.

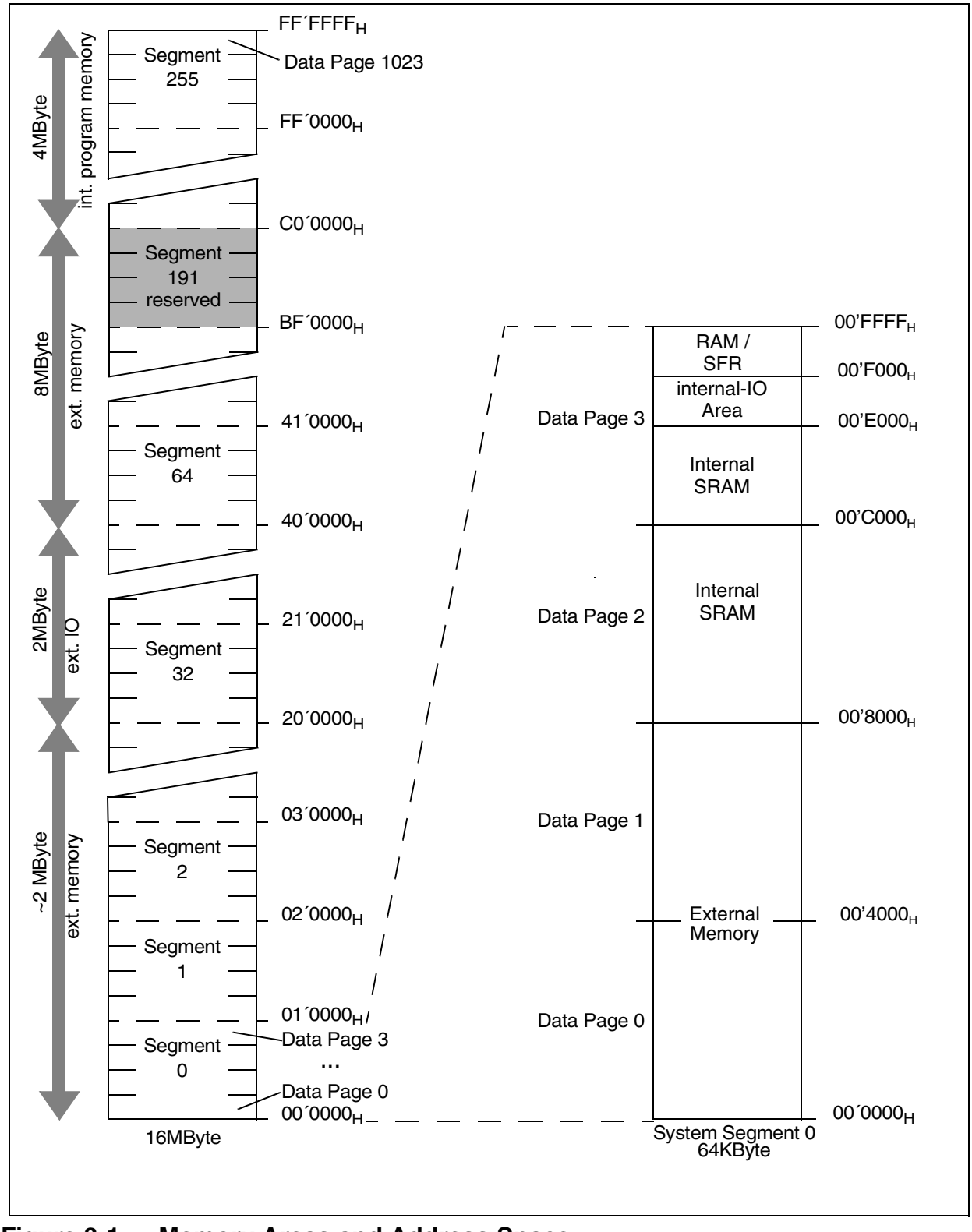


Figure 3-1 Memory Areas and Address Space

3.1 Data Organization in Memory

Bytes are stored at even or odd byte addresses. Words are stored in ascending memory locations with the low byte at an even byte address followed by the high byte at the next odd byte address. Instruction double words are stored in ascending memory locations as two subsequent words, without any restrictions (non aligned). Single bits are always stored in the specified bit position at a word address. The memory and registers store data and instructions in little endian byte order (the least significant bytes are at lower addresses) The byte ordering is illustrated in **Figure 3-2**. Bit position 0 is the least significant bit of the byte at an even byte address, and bit position 15 is the most significant bit of the byte at the next odd byte address. Bit addressing is supported for a part of the Special Function Registers, a part of the internal RAM, and for the General Purpose Registers.

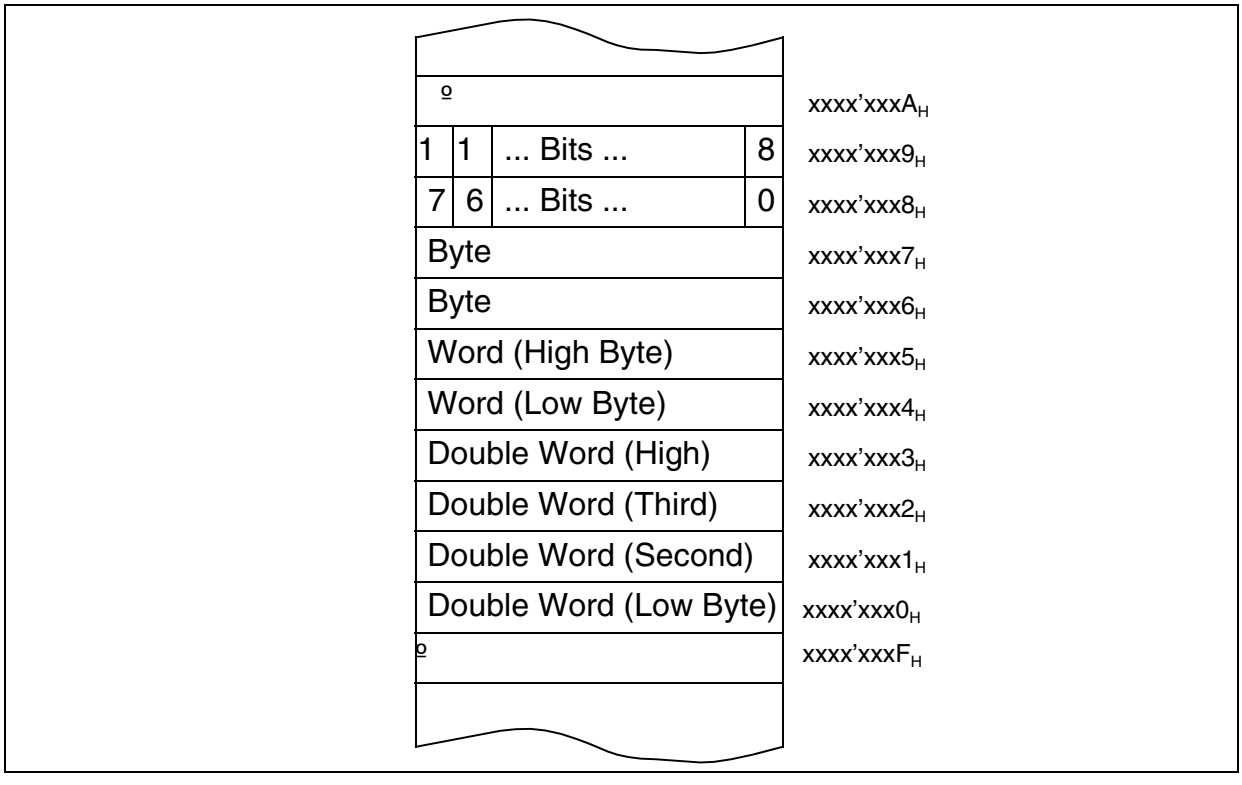


Figure 3-2 Storage of Words, Bytes and Bits in a Byte Organized Memory

Note: Byte units forming a single word must always be stored within the same physical (internal, external, ROM, RAM) and organizational (page, segment) memory area.

3.2 Internal Program Memory

The C166S V2 CPU **reserves** an address area of **4 MBytes** for Internal Program Memory. The internal memory can be ROM, SRAM, Flash or DRAM. Devices with

Internal Program Memory expand the Internal Program Memory area from the beginning of segment 192, i.e. starting at address C0'0000$_H$.

The Internal Program Memory can be used for both code (instructions) and data (constants, tables, etc.) storage.

Code fetches are always made on even word addresses. The highest possible code storage location in the Internal Program Memory is either xx'xxFE$_H$ for single word instructions, or xx'xxFC$_H$, for double word instructions.

Any word and byte data read access may use the indirect or long 16-bit addressing mode. There is no short addressing mode for Internal Program Memory operands. Any word data access is made to an even byte address. Any double word access is made to a modulo 4 address (even word address). The highest possible word data storage location in the Internal Program Memory is xxxx'xxFE$_H$, the highest double word location xxxx'xxFC$_H$.

The Internal Program Memory is not provided for single bit storage, and therefore is not bit addressable.

Note: The 'x' in the locations above depend on the available Internal Program Memory.

3.3 DPRAM, Internal SRAM, and SFR Areas

The C166S V2 CPU differentiates between various internal memory types and internal peripheral areas. These data memories and the IO/SFR areas are located within data page 3 and provide fast accesses using one dedicated Data Page Pointer (see **Figure 3-3**).

Note: Code access is not possible from the DPRAM, the Internal RAM, or the IO/SFR areas.

3.3.1 Data Memories

Two dedicated volatile memories are available for data storage:

- The DPRAM can be used for:
  - General Purpose Register Banks (GPRs)
  - Variable and other data storage, especially for MAC operands
  - System Stack (not recommended if Internal SRAM is integrated)
- The Internal SRAM can be used for:
  - Variable and other data storage
  - System Stack (recommended if Internal SRAM is integrated)

A 3 kByte memory area (00'F200$_H$...000'FE00$_H$) is reserved for the DPRAM. The upper 256 Bytes of the DPRAM (00'FD00$_H$...00'FDFF$_H$) and the GPRs of the current bank are provided for single bit storage, and thus are bit addressable (see shaded blocks in **Figure 3-3**). Any word or byte data in the DPRAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to data page 3. Any word

data access is made on an even byte address. The highest possible word data storage location in the DPRAM is 0000'FDFE$_H$.

A 24 kByte memory area (00'8000$_H$...000'DFFF$_H$) is reserved for the Internal SRAM. Any word and byte data in the Internal SRAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to data page 3 or data page 2. Any word data access is made on an even byte address. The highest possible word data storage location in the Internal SRAM is 0000'DFFE$_H$.

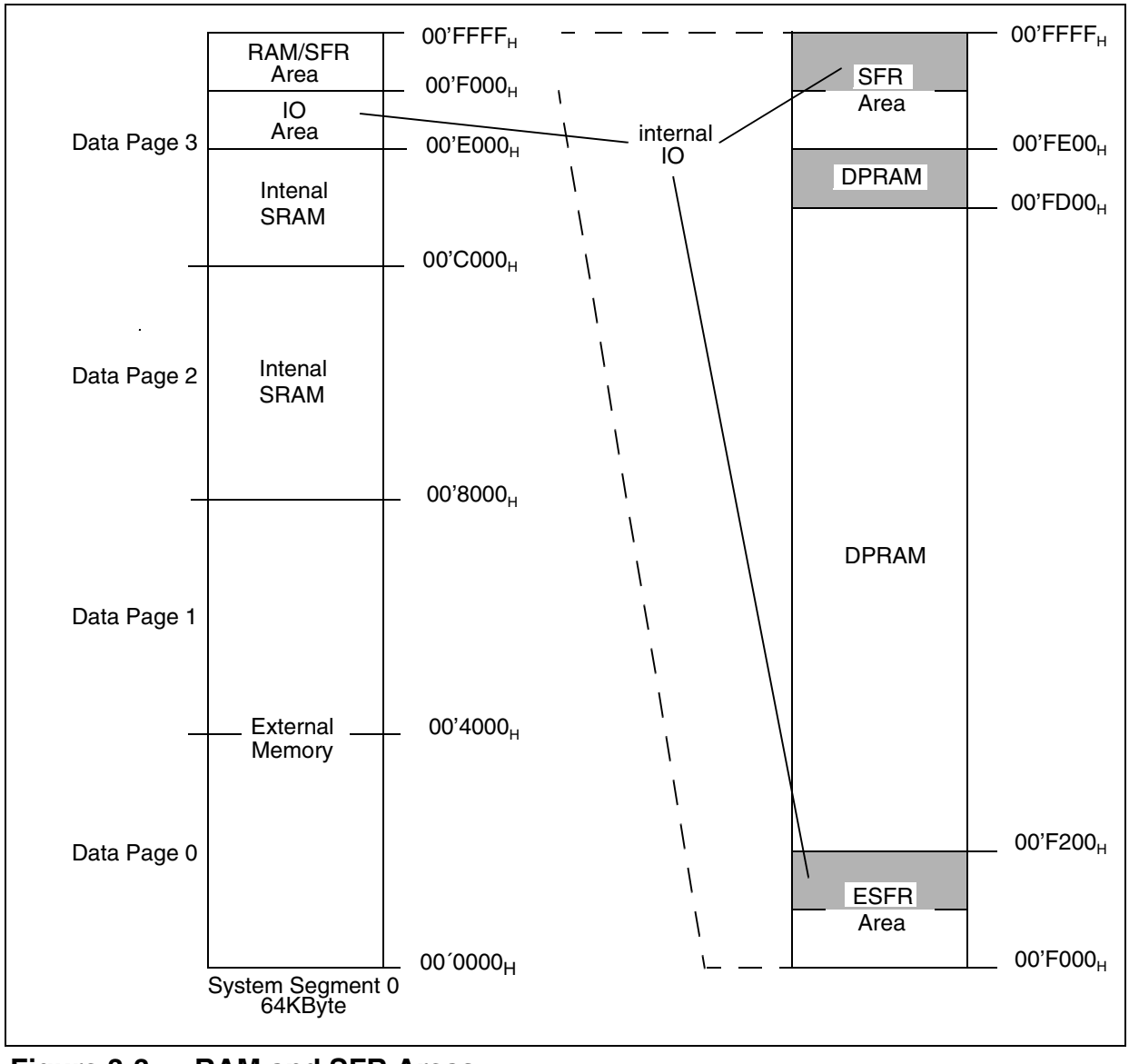


Figure 3-3 RAM and SFR Areas

3.3.2 Special Function Register Areas

The functions of the CPU, the bus interface, the IO ports, and the on-chip peripherals of the C166S V2 device are controlled via a number of so-called Special Function Registers (SFRs). These SFRs are arranged within two areas of 512 Bytes each. The first register block, the SFR area, is located in the 512 Bytes above the DPRAM ($00'FE00_H$...$00'FFFF_H$). The second register block, the Extended SFR (ESFR) area, is located in the 512 Bytes below the DPRAM ($00'F000_H$...$00'F1FF_H$).

Special Function Registers can be addressed via indirect and long 16-bit addressing modes. Using an 8-bit offset together with an implicit base address allows word SFRs and their respective low bytes to be addressed. However, this **does not work** for the respective high bytes!

Note: High byte access of SFRs using the 8-bit offset addressing mode is not possible.

Note: Writing to any byte of an SFR causes the non-addressed complementary byte to be cleared!

Note: GPRs can be accessed using the 8-bit offset addressing mode, but they are not mapped into the SFR and ESFR memory area. an internal peripheral bus access is executed using the respective long address instead of a GPR access.

The upper half of each register block (except the 16 highest words, refer to **Section 2.5.1**) is bit-addressable, so the respective control/status bits can be directly modified or checked using bit addressing.

When accessing registers in the ESFR area using 8-bit addresses or direct bit addressing, the Extend Register (EXTR) instruction is required to switch the short addressing mechanism from the standard SFR area to the Extended SFR area before accessing registers in the ESFR area. This is not required for 16-bit and indirect addresses. GPRs R15...R0 are duplicated, i.e. they are accessible within both register blocks via short 2-, 4- or 8-bit addresses without switching.

Example:

```
EXTR   #4                ;Switch to ESFR area for the next four instructions
MOV    ODP2, #data16     ;ODP2 (ESFR register) uses 8-bit register addressing
BFLDL  DP6, #mask, #data8;DP6 (ESFR register) bit addressing for bit fields
BSET   DP6.7             ;DP6 (ESFR register) bit addressing for single bits
MOV    T8REL, R1         ;T8REL uses 16-bit address, R1 is duplicated¤
                         ;...and also accessible via the ESFR mode
                         ;(EXTR is not required for this access)
;-------;-----------------;The scope of the EXTR #4 instruction ends here!
MOV    T8REL, R1         ;T8REL uses 16-bit address, R1 is duplicated¤
                         ;...and does not require switching
```

To minimize the switching of SFR banks, the ESFR area contains registers that are mainly required for initialization and mode selection. Registers that need to be accessed frequently are allocated to the standard SFR area wherever possible.

Note: The tools are equipped to monitor accesses to the ESFR area and will automatically insert EXTR instructions, switch the SFR bank address, or issue a warning in case of missing or excessive EXTR instructions.

3.3.3 IO Area

Some parts of the C166S V2 CPU memory area are marked as IO. These memory areas have the following special properties:

- Accesses are not buffered and cached
  The write back buffers and caches of the C166S V2 CPU are not used to store IO read and write accesses.
- Special handling of destructive reads
  The pipeline of the C166S V2 CPU allows speculative reads. Memory locations of the IO area are not read until all speculations are solved. Destructive read accesses are delayed.
- Write before read execution
  The pipeline length of the C166S V2 CPU enables a read instruction to read a memory location before a preceding write instruction has executed its write access. Data forwarding guarantees the correct instruction flow execution. In case of an IO read access, the read access will be delayed until all IO writes pending in the pipeline are executed. In case of a write access, peripherals will change their internal states. Write accesses must actually be executed before the next read access is initiated.

*Note: The bit manipulation instructions (BSET, BCLR...) use the **read**-modify-write approach. The IO read access of this instructions will be stalled until all IO write accesses are finished.*

The following memory areas are marked as IO:

- 2 Mbytes of external IO located to $20'0000_H$ to $3F'FFFF_H$
- SFR and ESFR areas located from $00'FE00_H$ to $00'FFFF_H$ and from $00'F000_H$ to $00'F1FF_H$ respectively
- 4 kByte internal IO located from $00'E000_H$ to $00'EFFF_H$

*Note: All external IO areas support real byte accesses. All internal IO areas do not support real byte transfers. For more details on the exception of (E)SFR areas refer to **Section 3.3.2**.*

3.3.4 PEC Source and Destination Pointers

The source and destination pointers for data transfers on the PEC channels are located in the 4-kByte internal IO area. Each channel uses a pair of pointers stored in two

subsequent word registers, with the source pointer (SRCPx) on the lower and the destination pointer (DSTPx) on the higher word address (x = channel number). The PEC registers are part of the PEC itself and are addressed via the internal peripheral bus.

In contrast to the C166 family, the pointers are not located in the internal RAM. The pointers are located in the 4 kByte internal IO.

If a PEC channel is not used, the corresponding pointer locations are not available and cannot be used for word and byte storage.

Writing to any byte of the PEC pointers does cause the non-addressed complementary byte to be cleared!

For more detail about use of the source and destination pointers for PEC data transfer, see the “Interrupt and Exception Execution” section.

3.4 External Memory Space

The C166S V2 CPU is capable of using an address space of up to 16 MBytes. Only portions of this address space are occupied by internal memory areas. All addresses not used for on-chip memory or for registers may reference external memory locations. This external memory is accessed via the external bus interface. This interface may further limit the amount of addressable external memory.

External word and byte data can be accessed only via indirect or long 16-bit addressing modes using one of the four DPP registers. There is no short addressing mode for external operands. Any word data access is made to an even byte address and double word accesses to modulo 4 byte addresses (even word address).

The external memory is not provided for single bit storage and therefore is not bit addressable.

3.4.1 Boot and Debug/Monitor Program Memories

The 64 KByte memory area of segment 191 (BF’0000_H...BF’$FFFF_H$) is reserved for boot and debug/monitor program memories. These “**on chip**” memories are accessed using the EBC and are a part of the EBC‘s external memory space. Accesses are not visible at the port pins of the EBC even if these memories are part of the external memory space. During normal code execution, this segment is not accessible for the C166S V2 CPU. In case of a read access, the EBC will deliver the predefined 0000_H value and write access will not be executed. Only in special boot and emulation modes can the memories of segment 191 be accessed.

Note: Segment 191 (BF’0000_H...BF’$FFFF_H$) is not usable for the system application. External memories and peripherals located in this segment will never be accessed.

3.5 Crossing Memory Boundaries

The address space of the C166S V2 CPU is implicitly divided into logical memory areas and equally sized blocks of different granularity. Crossing the boundaries between these areas or blocks (code or data) requires special attention to ensure that the controller executes the desired operations.

Memory Areas are partitions of the address space that represent different kinds of memory (if provided at all). These memory areas are the internal RAM areas, the internal IO areas, the internal Program Memories (if available), and the external memory.

Accessing subsequent ***data*** locations that belong to different memory areas is not fully supported and may therefore lead to erroneous results. There is no problem if the memory boundaries are word aligned. However, when executing ***code***, the different memory areas (Internal Program Memory areas and external memory) must be switched explicitly via branch instructions. Sequential boundary crossing is not supported and may leads to erroneous results.

Segments are contiguous blocks of 64 KBytes each. They are referenced via the Code Segment Pointer (CSP) for code fetches and via an explicit segment number for data accesses overriding the standard DPP scheme.
During code fetching, segments are not changed automatically, but rather must be switched explicitly. The instructions JMPS, CALLS, and RETS will do this. Larger sequential programs make sure that the highest used code location of a segment contains an unconditional branch instruction to the respective following segment, to prevent the prefetcher from trying to leave the current segment.

Data Pages are contiguous blocks of 16 KBytes each. They are referenced via the data page pointers DPP3...0 and via an explicit data page number for data accesses overriding the standard DPP scheme. Each DPP register can select one of the possible 1024 data pages. The DPP register that is used for the current access is selected via the two upper bits of the 16-bit data address. Subsequent 16-bit data addresses that cross the 16 KByte data page boundaries will use different data page pointers, while the physical locations need not be subsequent within memory.

3.6 System Stack

The system stack may be defined within the internal RAM, but can be also located externally. The size of the system stack is limited to 64 kBytes and must be located in one segment. For all system stack operations, the stack memory is accessed via a 24 bit stack pointer. The Stack Pointer register (SP) represents the low order 16 bits of the 24 bit stack pointer, also referred to as Stack Pointer Offset. The Stack Segment Pointer (SPSEG) represents the high order 8 bits of the stack pointer, also referred to as Stack Segment.

The system stack implementation in the C166S V2 CPU is from high to low memory. The system stack grows downward as it is filled. The SP register is decremented first each

time data is pushed on the system stack, and incremented after each time the data is pulled from the system stack. Only word accesses are supported to the system stack.

The 24 bit stack pointer points to the address of the latest system stack entry, rather than to the next available system stack address.

A stack overflow (STKOV) register and a Stack Underflow (STKUN) register are provided to control the lower and upper limits of the selected stack area. These two stack boundary registers can be used for protection against data destruction.

3.6.1 Data Organization in Global General Purpose Registers

The C166S V2 CPU differentiates between **global** memory mapped General Purpose Register (GPR) banks and **local** not mapped GPR banks. In addition to the memory mapped register banks, the C166S V2 CPU has two local not memory mapped GPR register banks for very fast context switching (see **Section 2.4**).

Note: The local GPR banks are not memory mapped and the GPRs cannot be accessed using a long or indirect memory address.

The C166S V2 CPU supports register bank (context) switching. Multiple global memory mapped register banks can physically exist within the DPRAM at the same time; however, only the global register bank selected by the Context Pointer register (CP) is active at a given time. Selecting a new active global register bank is done by simply updating the CP register.

Mapping of the global General Purpose Registers to DPRAM Addresses is shown here:

DPRAM Address	Byte Registers	Word Register
<CP> + $1E_H$	---	R15
<CP> + $1C_H$	---	R14
<CP> + $1A_H$	---	R13
<CP> + 18_H	---	R12
<CP> + 16_H	---	R11
<CP> + 14_H	---	R10
<CP> + 12_H	---	R9
<CP> + 10_H	---	R8
<CP> + $0E_H$	RH7RL7	R7
<CP> + $0C_H$	RH6RL6	R6
<CP> + $0A_H$	RH5RL5	R5
<CP> + 08_H	RH4RL4	R4
<CP> + 06_H	RH3RL3	R3
<CP> + 04_H	RH2RL2	R2
<CP> + 02_H	RH1RL1	R1
<CP> + 00_H	RH0RL0	R0

A particular Switch Context (SCXT) instruction performs register bank switching and an automatic save of the previous context. The number of implemented register banks (arbitrary sizes) is limited only by the size of the available DPRAM.

The memory mapped GPRs use a block of sixteen consecutive words within DPRAM Segment 0. The Context Pointer (CP) register determines the base address of the currently active register bank. This register bank may consist of up to sixteen word GPRs (R0, R1, .. R15), and/or of up to sixteen byte GPRs (RL0, RH0, º, RL7, RH7). The sixteen byte GPRs are mapped onto the first eight word GPRs (see table above).

In contrast to the system stack, a register bank grows from lower towards higher address locations and occupies a maximum space of 32 bytes. The GPRs are accessed via short 2-, 4- or 8-bit addressing modes using the Context Pointer (CP) register as base address (independent of the current DPP register contents). Additionally, each bit in the currently active register bank can be accessed individually.

4 Instruction Pipeline

The pipeline of the C166S V2 CPU has seven stages. Each stage processes its individual task. The first two stages form the instruction fetch pipeline and the remaining five stages constitute the instruction processing pipeline. The instruction fetch pipeline is used to pre-fetch instructions and to store them into an instruction FIFO. The preprocessing of branch instructions in combination with the instruction FIFO allows filling of the execution pipeline with a continuous flow of instructions. In the case of an incorrectly predicted instruction flow, the instruction fetch pipeline is bypassed to reduce the number of dead cycles. All instructions must pass through each of the five stages of the instruction processing pipeline regardless of the need of some stages to complete an execution of certain instructions. The following illustrates the pipeline stages operation.

1st -> PREFETCH:

This stage pre-fetches instructions from the PMU in the predicted order. The instructions are pre-processed in the branch detection unit to detect branches. The prediction logic decides if the branches are assumed to be taken or not.

2st -> FETCH:

The instruction pointer of the next instruction to be fetched is calculated according to the branch prediction rules. For zero-cycle branch execution, the Branch Folding Unit pre-processes and combines detected branches with the preceding instructions. Pre-fetched instructions are stored in the instruction FIFO. At the same time, instructions are transported out of the instruction FIFO to be executed in the instruction processing pipeline.

3st -> DECODE:

The instructions are decoded and, if required, the register file is accessed to read the GPR used in indirect addressing modes.

4st -> ADDRESS:

All the operand addresses are calculated. The SP register is de/incremented for all instructions which implicitly access the system stack.

5st -> MEMORY:

All the required operands are fetched.

6st -> EXECUTE:

An ALU or MAC-Unit operation is performed on the previously fetched operands. The Condition flags are updated. All explicit write operations to CPU-SFR registers and all auto-in/decrement operations of GPRs used as indirect address pointers are performed.

7st -> WRITE BACK:

All external operands and the remaining operands within the internal DPRAM space are written back. Operands located in the internal SRAM are buffered in the Write Back Buffer.

There are C166S V2 CPU-specific so-called injected instructions. These instructions are generated internally by the machine to provide the time needed to process instructions requiring more than one CPU cycle for processing. They are automatically injected into the decode stage of the pipeline, then they pass through the remaining stages like every standard instruction. Program interrupt, PEC transfer, and OCE operations are also performed by means of injected instructions. Although these internally injected instructions will not be noticed in reality, they are introduced here to ease the explanation of the pipeline operation.

Because up to five different instructions are processed simultaneously, additional hardware has been dedicated in the C166S V2 CPU to deal with dependencies which may exist between instructions in different pipeline stages. This extra hardware supports 'forwarding' of the operand read and write values and resolves most of the possible conflicts—such as multiple usage of buses—in a time optimized way without performance loss. This makes the pipeline unnoticeable for the user in most cases. However, there are some rare cases in which the C166S V2 CPU pipeline requires attention by the programmer. In these cases, the delays caused by the pipeline conflicts can be used for other instructions to optimize performance.

Note: The C166S V2 CPU has a fully interlocked pipeline. Instruction re-ordering is only required for performance reasons.

The following examples describe the pipeline behavior in special cases and give principle rules to improve the performance by re-ordering the execution of instructions.

4.1 Instruction Dependencies in Different Pipeline Stages

Bandwidth limitations and data dependencies between instructions can dramatically decrease the performance of CPUs. The C166S V2 CPU has dedicated hardware to detect and to resolve different kind of dependencies. Some of those dependencies are described in the following section.

4.1.1 The General Purpose Registers

The GPRs are the working registers of the C166S V2 CPU and there are a lot of possible dependencies between instructions using GPRs. A high speed five port register file prevents bandwidth conflicts. The dedicated hardware is implemented to detect and resolve the data dependencies. Special forwarding busses are used to forward GPR values from one pipeline stage to another. This allows the execution of instructions without any delay despite of data dependencies.

```
In     ADD   R0,R1
In+1   ADD   R3,R0
```

```
In+2   ADD    R6,R0
In+3   ADD    R6,R1
In+4   ......
```

.

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n= ADD R0,R1	I_{n+1}= ADD R3,R0	I_{n+2}= ADD R6,R0	I_{n+3}= ADD R6,R1	I_{n+4}	I_{n+5}
ADDRESS	I_{n-1}	I_n= ADD R0,R1	I_{n+1}= ADD R3,R0	I_{n+2}= ADD R6,R0	I_{n+3}= ADD R6,R1	I_{n+4}
MEMORY	I_{n-2}	I_{n-1}	I_n= ADD R0,R1	I_{n+1}= ADD R3,R0	I_{n+2}= ADD R6,R0	I_{n+3}= ADD R6,R1
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n= ADD R0,R1	I_{n+1}= ADD R3,R0	I_{n+2}= ADD R6,R0
WRITE BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n= ADD R0,R1	I_{n+1}= ADD R3,R0

Only in the case in which a GPR is updated in the ALU and then directly used in one of the following instructions as an address pointer will the detection unit force the pipeline to stall. **None of the instructions using indirect addressing modes are capable of using a GPR, which is to be updated by one of the two immediately preceding instructions.** The new value of the GPR is calculated in the execute stage, while the instruction using an indirect addressing mode accesses the GPR already in the Decode Stage. The instruction is stalled in the address stage until the operation in the ALU is executed and the result is forwarded to the address stage.

```
In-1   ........
In     ADD    R0,R1
In+1   MOV    R3,[R0]
In+2   ADD    R6,R0
In+3   ADD    R6,R1
In+4   ........
```

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n= ADD R0,R1	I_{n+1}= MOV R3,[R0]	I_{n+2}	I_{n+2}	I_{n+2}	I_{n+3}
ADDRESS	I_{n-1}	I_n= ADD R0,R1	I_{n+1}= MOV R3,[R0]	I_{n+1}= MOV R3,[R0]	I_{n+1}= MOV R3,[R0]	I_{n+2}
MEMORY	I_{n-2}	I_{n-1}	I_n= ADD R0,R1			I_{n+1}= MOV R3,[R0]
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n= ADD R0,R1		
WRITE BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n= ADD R0,R1	

To avoid stalls, one multicycle or two single cycle instructions may be inserted. These instructions must not update the GPR used for indirect addressing.

```
In-1   ........
In     ADD    R0,R1
In+1   ADD    R6,R0
In+2   ADD    R6,R1
In+3   MOV    R3,[R0]
In+4   ........
```

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n= ADD R0,R1	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,R1	I_{n+3}= MOV R3,[R0]	I_{n+4}	I_{n+5}
ADDRESS	I_{n-1}	I_n= ADD R0,R1	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,R1	I_{n+3}= MOV R3,[R0]	I_{n+4}
MEMORY	I_{n-2}	I_{n-1}	I_n= ADD R0,R1	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,R1	I_{n+3}= MOV R3,[R0]
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n= ADD R0,R1	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,R1
WRITE BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n= ADD R0,R1	I_{n+1}= ADD R6,R0

4.1.2 Indirect Addressing Modes

In the case of read accesses using indirect addressing modes, the Address Generation Unit uses a speculative addressing mechanism. The read data path to one of the different memory areas (DPRAM, Internal SRAM, etc.) is selected according to a history table before the address is decoded. This history table has one entry for each of the

GPRs. The entries store the information of the last accessed memory area using the corresponding GPR. In the case of an incorrect prediction of the memory area, the read access must be restarted.

It is recommended that the GPR used for indirect addressing point to the same memory area. If an updated GPR points to a different memory area, the next read operation will access the wrong memory area. The read access must be repeated, which leads to pipeline stalls.

```
I(n-1).........
I(n)    ADD   R3,[R0]   , points to DPRAM
I(n+1)  MOV   R0,R4
.....
I(i)    MOV   DPPX,... ,change DPPx
.....
I(m)    ADD   R6,[R0]   , points to SRAM
I(m+1)  ADD   R6,R1
I(m+2)  ........
```

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n= MOV R3,[R0]	I_{n+1}= MOV R0,R4	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}
ADDRESS	I_{n-1}	I_n= MOV R3,[R0]	I_{n+1}= MOV R0,R4	I_{n+2}	I_{n+3}	I_{n+4}
MEMORY	I_{n-2}	I_{n-1}	I_n= MOV R3,[R0]	I_{n+1}= MOV R0,R4	I_{n+2}	I_{n+3}
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MOV R3,[R0]	I_{n+1}= MOV R0,R4	I_{n+2}
WRITE BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MOV R3,[R0]	I_{n+1}= MOV R0,R4

	T_m	T_{m+1}	T_{m+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_m= MOV R6,[R0]	I_{m+1}= ADD R6,R1	I_{m+1}= ADD R6,R1	I_{m+2}	I_{m+3}	I_{m+4}
ADDRESS	I_{m-1}	I_m= MOV R6,[R0]	I_m= MOV R6,[R0]	I_{m+1}= ADD R6,R1	I_{m+2}	I_{m+3}
MEMORY	I_{m-2}	I_{m-1}		I_m= MOV R6,[R0]	I_{m+1}= ADD R6,R1	I_{m+2}
EXECUTE	I_{m-3}	I_{m-2}	I_{m-1}		I_m= MOV R6,[R0]	I_{m+1}= ADD R6,R1
WRITE BACK	I_{m-4}	I_{m-3}	I_{m-2}	I_{m-1}		I_m= MOV R6,[R0]

4.1.3 Memory Bandwidth Conflicts

Memory bandwidth conflicts can occur if instructions in the pipeline access the same memory area at the same time. Special access mechanisms are implemented in the C166S V2 CPU to minimize conflicts. The internal DPRAM of the C166S V2 CPU has

two independent read/write ports; this allows parallel read and write operation without delays. Write accesses to the internal SRAM can be buffered in a Write BACK Buffer until read accesses are finished.

- Bandwidth conflicts in the DPRAM Area
  All instructions except the CoXXX instructions can read only one memory operand per cycle. A conflict between the read and one write access cannot occur because the DPRAM has two independent read/write ports.

```
I(n-1)  ........
I(n)    ADD   op1,R1
I(n+1)  ADD   R6,R0
I(n+2)  ADD   R6,op2
I(n+3)  MOV   R3,[R0]
I(n+4)  ........
```

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,op2	I_{n+3}= MOV R3,[R0]	I_{n+4}	I_{n+5}
ADDRESS	I_{n-1}	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,op2	I_{n+3}= MOV R3,[R0]	I_{n+4}
MEMORY	I_{n-2}	I_{n-1}	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,op2	I_{n+3}= MOV R3,[R0]
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,op2
WRITE BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0

Note: Only other pipeline stall conditions can generate a DPRAM bandwidth conflict. The DPRAM is a synchronous pipelined memory. The read access starts with the valid addresses on the address stage. The data are delivered in the Memory stage. If a memory read access is stalled in the Memory stage and the following instruction on the Address stage tries to start a memory read, the new read access must be delayed as well. But, this conflict is hidden by an already existing stall of the pipeline.

- Bandwidth conflicts in the DPRAM Area
  The CoXXX instructions are the only instructions able to read two memory operands per cycle. **A conflict between the two read and one pending write access can occur if all three operands are located in the DPRAM areas.** This is especially important for performance in the case of executing a filter routine. One of the operands should be located in the internal SRAM to guarantee a single cycle execution time of the CoXXX instructions.

```
In-1   ........
In     ADD    op1,R1
In+1   ADD    R6,R0
In+2   CoMAC  [IDX0],[R0]
In+3   MOV    R3,[R0]
In+4   .......
.
```

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0	I_{n+2}= CoMAC	I_{n+3}= MOV R3,[R0]	I_{n+4}	I_{n+4}
ADDRESS	I_{n-1}	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0	I_{n+2}= CoMAC	I_{n+3}= MOV R3,[R0]	I_{n+3}= MOV R3,[R0]
MEMORY	I_{n-2}	I_{n-1}	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0	I_{n+2}= CoMAC	I_{n+2}= CoMAC .....
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0	
WRITE BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0

- Internal SRAM
  The internal SRAM is a single port memory with one read/write port. To reduce the number of bandwidth conflict cases, a Write Back Buffer is implemented. It has three entries for buffer data buffering. **Only if the buffer is filled and a read and write accesses occur at the same time, must the read access be stalled while one of the buffer entries is written back**.

```
I(n-1)  ........
I(n)    ADD    op1,R1
I(n+1)  ADD    R6,R0
I(n+2)  ADD    R6,op2
I(n+3)  MOV    R3,R2
I(n+4)  .........
.
```

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,op2	I_{n+3}= MOV R3,R2	I_{n+4}	I_{n+4}
ADDRESS	I_{n-1}	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,op2	I_{n+3}= MOV R3,R2	I_{n+3}= MOV R3,R2
MEMORY	I_{n-2}	I_{n-1}	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,op2	I_{n+2}= ADD R6,op2
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0	
WRITE BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n= ADD op1,R1	I_{n+1}= ADD R6,R0
Write Back Buffer	full	full	full	full	full	full

4.1.4 CPU-SFRs and the Pipeline

CPU-SFRs control the CPU functionality and behavior. Changes and updates of CSFRs influence the instruction flow in the pipeline. Therefore, special care is required to ensure that instructions in the pipeline always work with the correct CSFRs values. **CSFRs are updated late on the Executed stage of the pipeline. Meanwhile, without conflict detection, the instructions in the Decode, Address, and Memory stages would still work without updated register values**. The C166S V2 CPU detects conflict cases and stalls the pipeline to guarantee a correct execution. For performance reasons, the CPU differentiates between different classes of CPU-SFRs. The flow of instructions through the pipeline can be improved by following the given rules used for instruction re-ordering.

There are three classes of CPU-SFRs:

- The harmless CSFRs (CPUID, ONES, ZEROS, MCW) do not generate pipeline conflict cases. The MCW can be changed without stalling the pipeline. The MCW is

updated in the Execute Stage and is not used for control purposes in the previous stages. CPUID, ONES, and ZEROS are not changeable at all.

```
I n-1   ........
I n     MOV    MCW,#16
I n+1   ADD    R6,R0
I n+2   ADD    R6,R1
I n+3   MOV    R3,[R0]
I n+4   ........
```

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n= MOV MCW,#16	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,R1	I_{n+3}= MOV R3,[R0]	I_{n+4}	I_{n+5}
ADDRESS	I_{n-1}	I_n= MOV MCW,#16	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,R1	I_{n+3}= MOV R3,[R0]	I_{n+4}
MEMORY	I_{n-2}	I_{n-1}	I_n= MOV MCW,#16	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,R1	I_{n+3}= MOV R3,[R0]
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MOV MCW,#16	I_{n+1}= ADD R6,R0	I_{n+2}= ADD R6,R1
WRITE BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MOV MCW,#16	I_{n+1}= ADD R6,R0

- The CSFR result registers MDH, MDL, MSW, MAH, MAL, MRW of the ALU and MAC-Unit are updated late in the Execute stage of the pipeline. If an instruction (except CoSTORE) accesses explicitly these registers in the memory stage, the value cannot be forwarded. The instruction must be stalled for one cycle on the Memory stage.

```
I(n-1)  ........
I(n)    MUL    R0,R1
I(n+1)  MOV    R6,MDL
I(n+2)  ADD    R6,R1
I(n+3)  MOV    R3,[R0]
I(n+4)  ........
```

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n= MUL R0,R1	I_{n+1}= MOV R6,MDL	I_{n+2}= ADD R6,R1	I_{n+3}= MOV R3,[R0]	I_{n+3}= MOV R3,[R0]	I_{n+4}
ADDRESS	I_{n-1}	I_n= MUL R0,R1	I_{n+1}= MOV R6,MDL	I_{n+2}= ADD R6,R1	I_{n+2}= ADD R6,R1	I_{n+3}= MOV R3,[R0]
MEMORY	I_{n-2}	I_{n-1}	I_n= MUL R0,R1	I_{n+1}= MOV R6,MDL	I_{n+1}= MOV R6,MDL	I_{n+2}= ADD R6,R1
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MUL R0,R1		I_{n+1}= MOV R6,MDL
WRITE BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MUL R0,R1	

By reordering instructions, the bubble in the pipeline can be filled with an instruction not using this resource.

```
I(n-1)  ........
I(n)    MUL    R0,R1
I(n+1)  MOV    R3,[R0]
I(n+2)  MOV    R6,MDL
I(n+3)  ADD    R6,R1
I(n+4)  ........
```

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n= MUL R0,R1	I_{n+1}= MOV R3,[R0]	I_{n+2}= MOV R6,MDL	I_{n+3}= ADD R6,R1	I_{n+4}	I_{n+5}
ADDRESS	I_{n-1}	I_n= MUL R0,R1	I_{n+1}= MOV R3,[R0]	I_{n+2}= MOV R6,MDL	I_{n+3}= ADD R6,R1	I_{n+4}
MEMORY	I_{n-2}	I_{n-1}	I_n= MUL R0,R1	I_{n+1}= MOV R3,[R0]	I_{n+2}= MOV R6,MDL	I_{n+3}= ADD R6,R1
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MUL R0,R1	I_{n+1}= MOV R3,[R0]	I_{n+2}= MOV R6,MDL
WRITE BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MUL R0,R1	I_{n+1}= MOV R3,[R0]

- The third class are CSFRs which affect the whole CPU or the pipeline before the Memory stage. The CPU-SFRs CPUCON1, CP, SP, STKUN, STKOV, VECSEG, TFR, and PSW affect the overall CPU functioning while the C-SFRs IDX0, IDX1, QX1, QX0, DPP0, DPP1, DPP2 and DPP3 only affect the Decode, Address, and Memory stage when they are modified **explicitly**.
  If this kind of CSFR has been modified, the pipeline behavior depends on the instruction and addressing modes used to modify the CSFR.

  – In the case of modification of these CSFRs by “POP CSFR” or by instructions using the reg,#data16 addressing mode, a special mechanism is implemented to improve performance during the initialization.
    For further explanation, the instruction which modifies the CSFR can be called “instruction_modify_CSFR”. This special case is detected in the Decode stage when the instruction_modify_CSFR enters the processing pipeline. Further on, instructions described in the following list are held in the decode stage. All other instructions are not held.

    - Instructions using long addressing mode (mem)
    - Instructions using indirect addressing modes ([R_w],]R_w+].....), except JMPI and CALLI
    - ENWDT, DISWDT, EINIT
    - All CoXXX instructions

    If the CPUCON1, CP, SP, STKUN, STKOV, VECSEG, TFR, or the PSW are modified and the instruction_modify_CSFR reaches the execute stage, the pipeline is canceled. The modification affects the entire pipeline and the instruction prefetch. A clean cancel and restart mechanism is required to guarantee a correct instruction flow. In case of modification of IDX0, IDX1, QX1, QX0, DPP0, DPP1, DPP2 or DPP3 only the Decode, Address, and Memory stages are affected and the pipeline must not be canceled. The modification does not affect the instructions in the Address, Memory stage because they are not using this resource. Other kinds of instructions are held in the Decode stage until the CSFR is modified.

    The following example shows a case in which the pipeline is stalled. The instruction MOV R6,R1 after the MOV IDX1,#12 instruction which modifies the CSFR will be held in Decode Stage until the IDX1 register is updated. The next example shows an optimized initialization routine.

```
In-1  ........
In    MOV    IDX1,#12
In+1  MOV    R6,mem
In+2  ADD    R6,R1
In+3  MOV    R3,[R0]
In+4  ........
```

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n= MOV IDX1,#12	I_{n+1}= MOV R6,mem	I_{n+1}= MOV R6,mem	I_{n+1}= MOV R6,mem	I_{n+1}= MOV R6,mem	I_{n+2}= ADD R6,R1
ADDRESS	I_{n-1}	I_n= MOV IDX1,#12				I_{n+1}= MOV R6,mem
MEMORY	I_{n-2}	I_{n-1}	I_n= MOV IDX1,#12			
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MOV IDX1,#12		
WRITE BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MOV IDX1,#12	

```
In-1  ........
In    MOV    IDX1,#12
In+1  MOV    MAH,#23
In+2  MOV    MAL,#25
In+3  MOV    R3,#08
In+4  ........
```

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n= MOV IDX1,#12	I_{n+1}= MOV MAH,#23	I_{n+2}= MOV MAL,#25	I_{n+3}= MOV R3,#08	I_{n+4}	I_{n+5}
ADDRESS	I_{n-1}	I_n= MOV IDX1,#12	I_{n+1}= MOV MAH,#23	I_{n+2}= MOV MAL,#25	I_{n+3}= MOV R3,#08	I_{n+4}
MEMORY	I_{n-2}	I_{n-1}	I_n= MOV IDX1,#12	I_{n+1}= MOV MAH,#23	I_{n+2}= MOV MAL,#25	I_{n+3}= MOV R3,#08
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MOV IDX1,#12	I_{n+1}= MOV MAH,#23	I_{n+2}= MOV MAL,#25
WRITE BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MOV IDX1,#12	I_{n+1}= MOV MAH,#23

– For all the other instructions that modify this kind of CSFR, a simple stall and cancel mechanism guarantees the correct instruction flow.
A possible explicit write-operation to this kind of CSFRs is detected on the Memory stage of the pipeline. The following instructions on the Address and Decode Stage are stalled. If the instruction reaches the execute stage, the entire pipeline and the Instruction FIFO of the IFU are canceled. The instruction flow is completely re-started.

```
I(n-1)  ........
I(n)    MOV    PSW,R4
I(n+1)  MOV    R6,R1
I(n+2)  ADD    R6,R1
I(n+3)  MOV    R3,[R0]
I(n+4)  ........
```

	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}	T_{n+6}
DECODE	I_{n+1}= MOV R6,R1	I_{n+2}= ADD R6,R1	I_{n+2}= ADD R6,R1			I_{n+1}= MOV R6,R1
ADDRESS	I_n= MOV PSW,R4	I_{n+1}= MOV R6,R1	I_{n+1}= MOV R6,R1			
MEMORY	I_{n-1}	I_n= MOV PSW,R4				
EXECUTE	I_{n-2}	I_{n-1}	I_n= MOV PSW,R4			
WRITE BACK	I_{n-3}	I_{n-2}	I_{n-1}	I_n= MOV PSW,R4		

5 Interrupt and Exception Handling

The Interrupt and Exception Handler is responsible for managing all system and core exceptions. Four kinds of exceptions are executed in a similar manner:

- Interrupts generated by the Interrupt Controller ITC
- DMA transfers issued by the Peripheral Event Controller PEC.
- Software Traps caused by the TRAP instruction
- Hardware Traps issued by faults or specific system states

Normal Interrupt Processing

The CPU temporarily suspends current program execution and branches to an interrupt service routine to service a device requesting an interrupt. The current program status (IP and PSW; in segmentation mode, also CSP) is saved in the internal system stack. A prioritization scheme with sixteen priority levels specifies the order for handling multiple interrupt requests.

Software and Hardware Traps

Trap functions are activated in response to special conditions that occur during the execution of instructions. A trap can also be caused externally by the Non-Maskable Interrupt pin, NMI. Several hardware trap functions are provided to handle erroneous conditions and exceptions that arise during program execution. Hardware traps always have the highest priority and cause immediate system response. The software trap function is invoked by the TRAP instruction that generates a software interrupt for a specified interrupt vector. For all types of traps, the current program status is saved in the system stack.

Interrupt Processing via the Peripheral Event Controller (PEC)

A faster alternative to normal interrupt processing uses the C166S V2 CPU's integrated Peripheral Event Controller (PEC) to service an interrupt requesting device. Triggered by an interrupt request, the PEC performs a single word or byte data transfer between any two memory locations. During a PEC transfer, the normal program execution of the CPU is halted. No internal program status information needs to be saved. The same prioritization scheme is used for PEC service as for normal interrupt processing.

5.1 Interrupt System and Control

5.1.1 General Interrupt System Structure

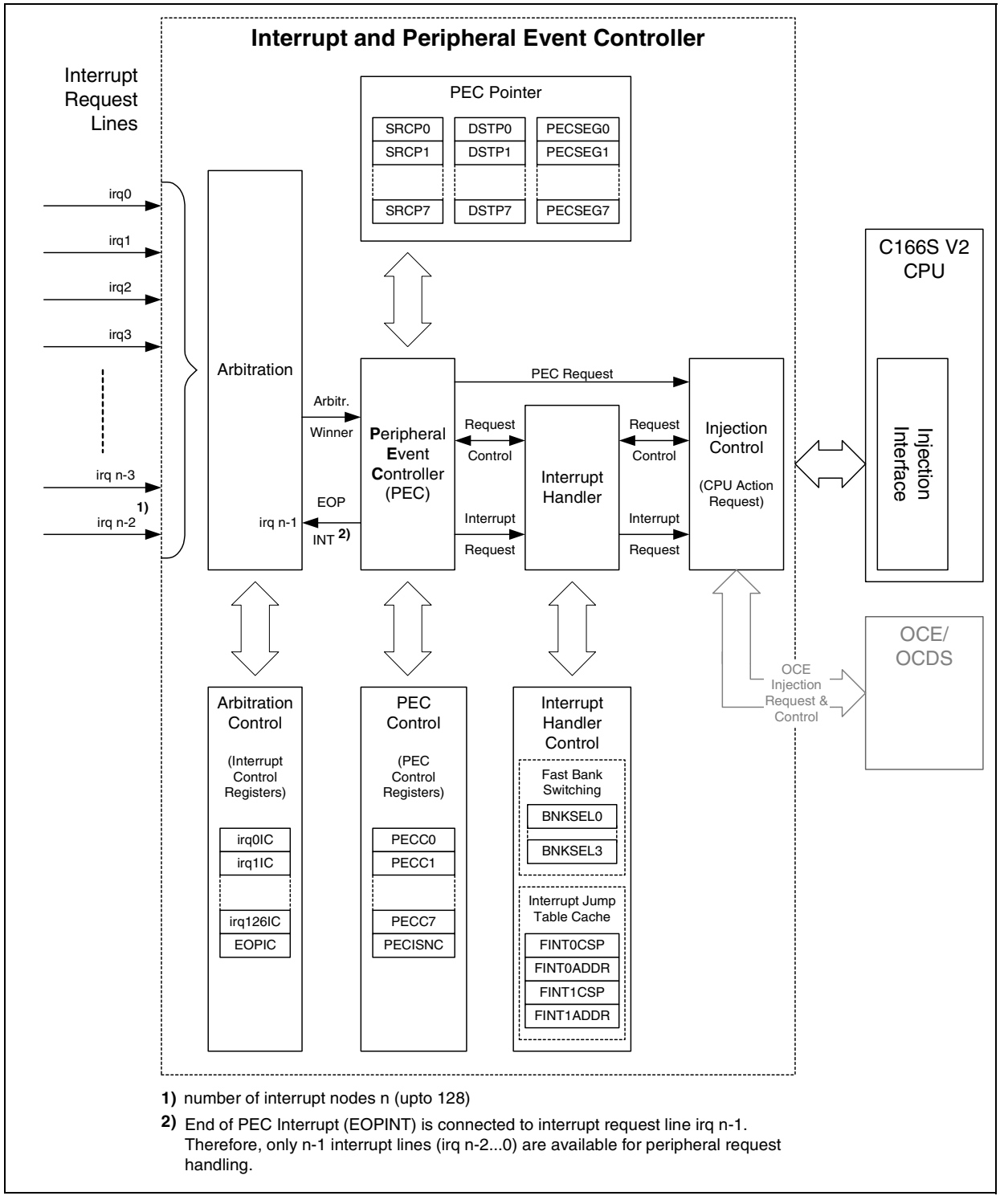
The C166S V2 CPU can provide up to 128 separate interrupt nodes that may be assigned to sixteen interrupt priority levels with four sub-priorities inside each level (group priority) for up to 64 interrupt nodes or with eight sub-priorities inside each level (group priority) in the case of more than 64 interrupt nodes. To support modular and consistent software design techniques, most sources of an interrupt or PEC request are supplied with separate interrupt control registers and interrupt vectors. The control register contains an interrupt request flag, an interrupt enable bit, and an interrupt priority of the associated source. Each source request is activated by one specific event, determined by the selected operating mode of the requesting device. In some cases, multi-source interrupt nodes are incorporated for efficient use of system resources. These nodes can be activated by various source requests.

The C166S V2 CPU provides a vectored interrupt system. This system reserves specific vector locations in the memory space for the reset, trap, and interrupt service functions. Whenever a request occurs, the CPU branches to the location associated with the respective interrupt source. The reserved vector locations build a jump table in the address space of the C166S V2 CPU.

All pending interrupt requests are arbitrated. The arbitration winner is sent to the CPU together with its priority level and action request. The CPU triggers the corresponding action based on the required functionality (normal interrupt, PEC, jump table cache, etc.) of the arbitration winner.

An action request will be accepted by the CPU if the requesting source has a higher priority than the current CPU priority level and interrupts are globally enabled. If the requesting source has a lower (or equal) interrupt level priority than the current CPU task, it remains pending.

The basic functionality of the interrupt and peripheral event controller can be seen in **Figure 5-1**:



1) number of interrupt nodes n (upto 128)

2) End of PEC Interrupt (EOPINT) is connected to interrupt request line irq n-1. Therefore, only n-1 interrupt lines (irq n-2...0) are available for peripheral request handling.

Figure 5-1 Block Diagram of the Interrupt and PEC Controller

5.1.2 Interrupt Arbitration

The C166S V2 interrupt arbitration system can handle interrupt requests from up to 128 sources. Interrupt requests may be triggered either by the C166S V2 peripherals or by external inputs. The “End of PEC” interrupt for supporting enhanced PEC functionality is connected internally to one interrupt request line.

The arbitration process starts with an enabled interrupt request and stays active as long as an interrupt request is pending. If nothing is pending, the arbitration logic switches to the idle state to save power.

Each interrupt request line is controlled by its interrupt control register xxIC (here and below ‘xx’ stands for the mnemonic of the respective interrupt source). An interrupt request event sets the interrupt request flag in the corresponding interrupt control register (bit xxIC.xxIR). The interrupt request can also be triggered by the software if the program sets the respective interrupt request bit. This feature is specifically used by operating systems.

If the request bit has been set and the corresponding interrupt request is enabled by the interrupt enable bit of the same control register (bit xxIC.xxIE), an arbitration cycle starts with the next clock cycle. However, if an arbitration cycle is currently in progress, the new interrupt request will be delayed until the next arbitration cycle. If an interrupt request (or PEC request) is accepted by the core, the respective interrupt request flag is cleared automatically.

All interrupt requests pending at the beginning of a new arbitration cycle are considered simultaneously. Within the arbitration cycle, the arbitration is independent of the actual request time.

C166S V2 uses a three-stage interrupt prioritization scheme for interrupt arbitration as shown in **Figure 5-2**.

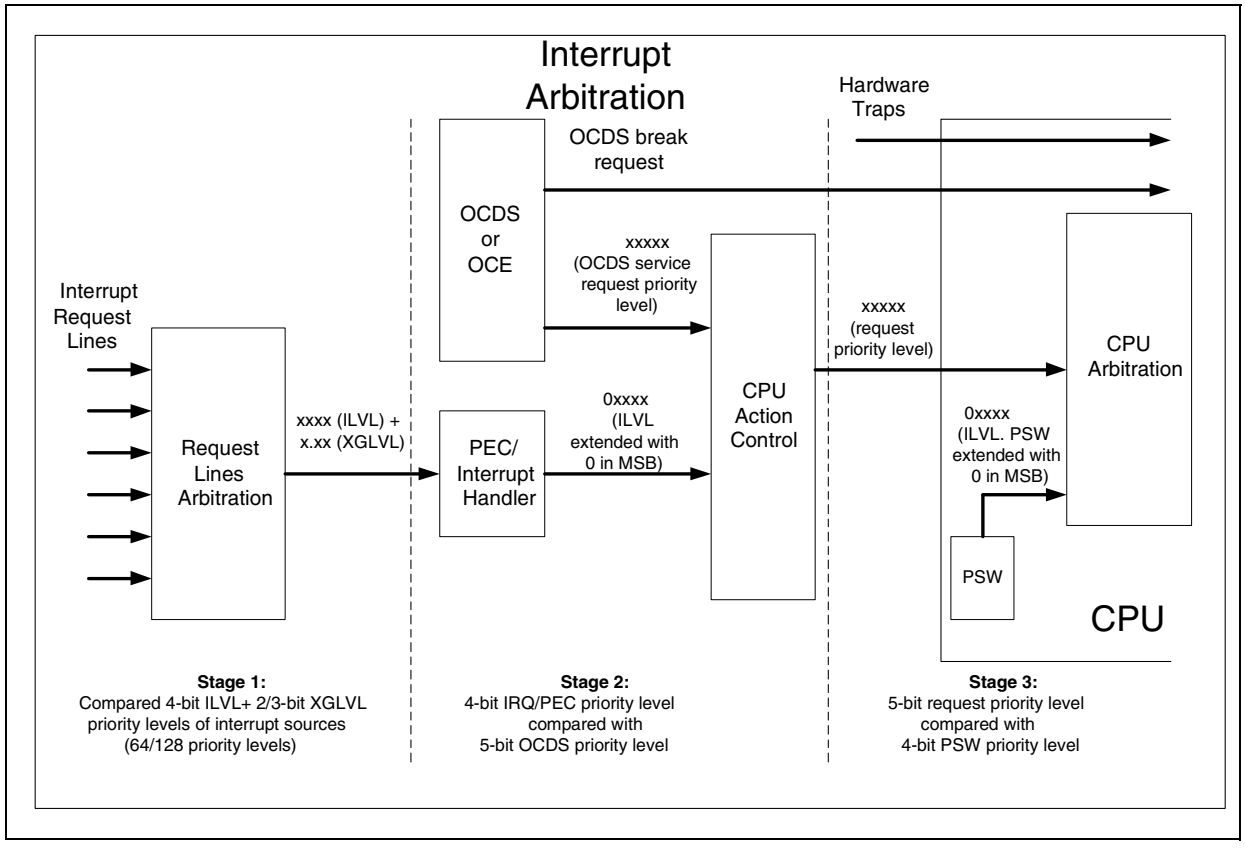


Figure 5-2 Interrupt Arbitration

The first arbitration stage compares the priority levels of interrupt request lines. The priority level of each requestor consists of interrupt priority level and group priority level. An interrupt priority level is programmed for each interrupt request line by the 4-bit bit field ILVL of the respective xxIC register. The group priority level is programmed for each interrupt request line by the 2-bit bit field GLVL—and, in the case of more than 64 interrupt nodes, by the extension bit GPX of the register xxIC. GPX and GLVL combined form the 3-bit (extended) group priority level XGLVL, controlling up to eight interrupt sub-priorities within one of the sixteen interrupt levels.

Note: All interrupt request sources that are enabled and programmed to the same interrupt priority level (ILVL) must have different group priority levels. Otherwise, an incorrect interrupt vector may be generated.

The second arbitration stage compares the priority of the first stage winner with the priority of OCDS service requests. C166S V2 OCDS service requests bypass the first stage of arbitration and go directly to the CPU Action Control Unit. The CPU Action Control Unit disregards the group priority level of interrupt/PEC requests and deals only with interrupt priority levels (ILVL). For comparison with an OCDS service request priority programmed with a 5-bit value, the 4-bit ILVL of the interrupt/PEC request is extended to a 5-bit value with MSB=0. This means that any OCDS request with MSB=1 will always

win the second stage arbitration. However, if there is a OCDS request with MSB=0 conflicting with the same priority interrupt/PEC request, the latter is sent to the CPU.

On the third arbitration stage, the priority level of the second stage winner is compared with the priority of the current CPU task. An action request will be accepted by the CPU if the requesting source has a priority level higher than the current CPU priority level (bits ILVL of the PSW register) and for interrupt and PEC requests if they are globally enabled by the global interrupt enable flag IEN in PSW. The CPU denies all interrupt/PEC requests in case of a cleared IEN flag and an injection level between 0 to 15. To compare with the 5-bit priority level of the second stage winner, the 4-bit ILVL.PSW is extended to a 5-bit value with MSB=0. This means that any request with MSB=1 will always interrupt the current CPU task. If the requester has a priority level lower than or equal to the current CPU task, the request remains pending.

Note: Priority level 0000_B is the default level of the CPU. Therefore, a request on interrupt priority level 0000_B will be arbitrated, but the CPU will never accept an action request on this level. However, every enabled interrupt request (including all denied interrupt requests as well as priority level 0000_B requests) triggers a CPU wake-up from idle state independent of the setting of the global interrupt enable bit PSW.IEN.

Both the OCDS break requests and the hardware traps bypass the arbitration scheme and go directly to the core.

5.1.3 Interrupt Control

All interrupt control registers are organized identically. The lower eight bits of an interrupt control register contain the complete interrupt control and status information of the associated source required during one round of prioritization (arbitration cycle). The upper eight bits of the respective register are reserved. All interrupt control registers are bit addressable and all bits can be read or written via software. Therefore, each interrupt source can be programmed or modified with just one instruction. In the case of reading the interrupt control registers with instructions that operate with word data types, the upper 7 bits (15...9) will return zeroes. It is recommended to always write zeroes to these bit positions. The layout of the interrupt control registers shown below is applicable to all xxIC registers.

xxIC
Interrupt Control Register **SFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	GPX	xxIR	xxIE	ILVL				GLVL	
r	r	r	r	r	r	r	rw	rwh	rw	rw				rw	

Field	Bits	Type	Description
GPX	[8]	rw	**Group Priority Extension** Defines the value of high-order group level bit
xxIR[1)]	[7]	rwh	**Interrupt Request Flag** 0 No request pending 1 This source has raised an interrupt request
xxIE	[6]	rw	**Interrupt Enable Control Bit** (individually enables/disables a specific source) 0 Interrupt request is disabled 1 Interrupt request is enabled
ILVL	[5:2]	rw	**Interrupt Priority Level** F_H Highest priority level ... ... 0_H Lowest priority level
GLVL	[1:0]	rw	**Group Priority Level** 3_H Highest priority level ... ... 0_H Lowest priority level
XGLVL	[8],[1:0]		**Extended Group Priority Level** 7_H Highest priority level ... ... 0_H Lowest priority level

1) Bit xxIR supports bit-protection

The arbitration scheme allows nesting of up to fifteen interrupt service routines of different priority levels (Level 0 cannot be used; see note above).

Note: To reduce power, the arbitration is stopped when no interrupt request is active.

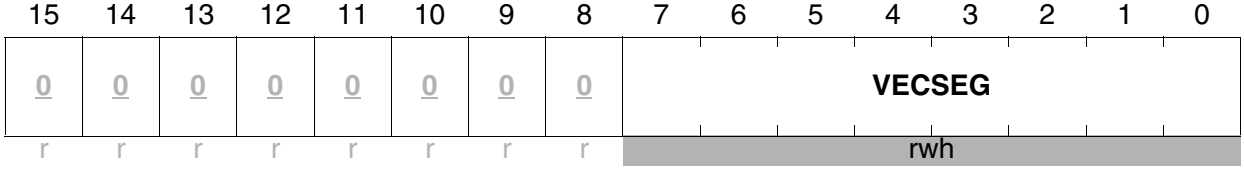
5.1.4 Interrupt Vector Table

The C166S V2 provides a vectored interrupt system. This system reserves the specific vector locations in the memory space for the reset, trap, and interrupt service functions. Whenever a request occurs, the CPU branches to the location associated with the respective interrupt source. This vector position directly identifies the source causing the request.

Note: Class B hardware traps all share the same interrupt vector. The status flags in the Trap Flag Register (TFR) are used to determine which exception caused the trap. For details, see ***Section 5.3****.*

The reserved vector locations are assembled into a vector table located in the address space of the C166S V2. The vector table contains the appropriate jump instructions that transfer control to the interrupt or trap service routines. These routines may be located anywhere within the address space. The location and organization of the vector table is programmable. The vector table can be located in all segments with exception of the reserved segment 191. The Vector Segment register VECSEG specifies the segment of the Vector Table.

VECSEG
Vector Segment Pointer **bSFR** **Reset Value: $xxxx_H$**



Field	Bits	Type	Description
VECSEG	[7:0]	rwh	Segment number of the Vector Table

The reset value of VECSEG can be configured during system reset or can be set depending on the particular product. The C166S V2 supports the following reset values:

– Start from Internal Program Memory (**C0'0000$_H$**)
– Start from Boot memory (**BF'0000$_H$**)
– Start from external memory (**00'0000$_H$**)
– Start from a segment specified from the system (**xx'0000$_H$**)[1)]

The **VECSC** bit field of the CPUCON1 register controls the number of word locations separating two vectors. The space between two vectors can be programmed to 2, 4, 8, or 16 words.

1) The current startup routine does not support this reset configuration.

Each vector location has an offset address to the segment base address of the vector table. The address can be easily calculated. The segment part is given by the VECSEG register and the offset is the trap number shifted by the space programmed in the VECSC bit field.

CPUCON1
CPU Control Register **SFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	VECSC		WDT CTL	SGT DIS	INT SCXT	BP	ZCJ
r	r	r	r	r	r	r	r	r	rw		rw	rw	rw	rw	rw

Field	Bits	Type	Description
VECSC	[6:5]	rw	**Scaling factor of Vector Table** 00 Space between two vectors is 2 words 01 Space between two vectors is 4 words 10 Space between two vectors is 8 words 11 Space between two vectors is 16 words

*Note: For a summary of the CPUCON1 register, please refer to **Chapter 2.3.6**.*

5.1.5 Interrupt Jump Table Cache

The mechanism that uses the vector table location as the entry point for the interrupt service routines can be overwritten by the Interrupt Controller (ITC). For a very fast interrupt response time, the C166S V2 offers a new feature of the interrupt system—Interrupt Jump Table Cache (also called “fast interrupt”). The ITC can transfer a 24-bit vector to the CPU that is used directly as a start address for the service routine. This feature skips the path through the vector table which normally saves the execution of at least one branch. Due to the random nature of interrupt requests, execution of these branches requires several CPU cycles, especially if memories with a high latency are used, such as DRAMs. Therefore, avoiding the vector table may significantly improve interrupt response time. However, the number of 24-bit vectors in the ITC is limited.

Fast interrupt is available for two interrupt sources with interrupt priority levels greater than or equal to 12. The Interrupt Jump Table Cache skips the instruction fetches from the interrupt vector table and executes a direct jump to the interrupt service routines entry point. This feature is controlled by a set of two interrupt jump table cache registers (FINTxCSP, FINTxADDR) for each of the two jump table entries.

Every interrupt jump table cache entry contains an enable bit, an associated arbitration priority level (ILVL and GLVL), and the 24-bit address of the interrupt service routine. Note that only the two lower bits of the interrupt priority level are selectable in the

respective control registers. The two upper bits of the interrupt priority level are set to '11_B', which limits the allowed interrupt priority level to be greater than or equal to 12.

FINT0CSP
Fast Interrupt Control Register 0 **XSFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	0	0	GPX	ILVL		GLVL		SEG							
rw	r	r	rw	rw		rw		rw							

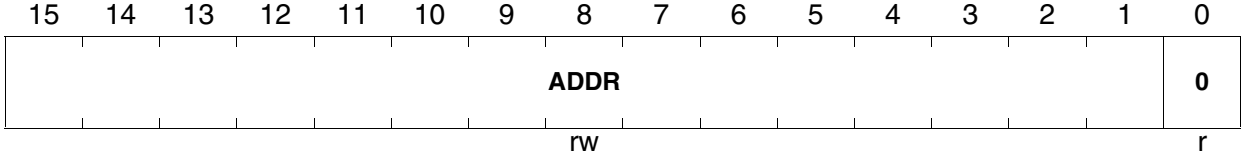
FINT1CSP
Fast Interrupt Control Register 1 **XSFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	0	0	GPX	ILVL		GLVL		SEG							
rw	r	r	rw	rw		rw		rw							

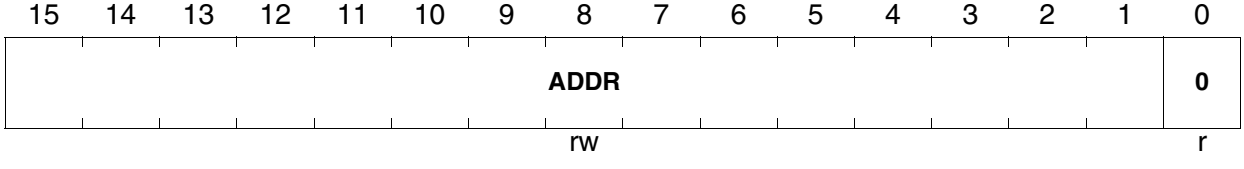
Field	Bits	Type	Description
EN	[15]	rw	**Fast Interrupt Enable** 0 The interrupt jump table cache is disabled. No fast interrupt is used. 1 The interrupt jump table cache is enabled. A fast interrupt (direct jump to the interrupt service routine) is used instead of the normal fetch from the interrupt vector table.
GPX	[12]	rw	**Group Priority Extension** This bit enables group extension for fast interrupts. (hardwired to 0 for fewer than 64 interrupt nodes)
ILVL	[11:10]	rw	**Interrupt Priority Level** This bit field selects the lower two bits of the interrupt priority level associated with this interrupt jump table cache entry. *Note: The two upper bits of the interrupt priority level are set to '11_B', which ends in an interrupt priority level greater than or equal to 12.*

Field	Bits	Type	Description
GLVL	[9:8]	rw	**Group Priority Level** This bit field selects the group priority level of the associated interrupt jump table cache entry.
SEG	[7:0]	rw	**Segment Number of Interrupt Service Routine** This bit field specifies address bits 23:16 of the interrupt service routine´s entry point.

FINT0ADDR
Fast Interrupt Address Register 0 **XSFR** **Reset Value: 0000_H**



FINT1ADDR
Fast Interrupt Address Register 1 **XSFR** **Reset Value: 0000_H**



Field	Bits	Type	Description
ADDR	[15:1]	rw	**Address of Interrupt Service Routine** This bit field specifies address bits 15:1 of the interrupt service routine's entry point.
0	[0]	r	**Interrupt Service Routine Address Bit 0** LSB of the interrupt service routine's entry point address is 0 because of word alignment.

5.2 Status and Switch Context Control

5.2.1 Interrupt Control Functions in the PSW

The Processor Status Word (PSW) is functionally divided into two parts: the lower byte of the PSW represents the arithmetic status of the CPU, the upper byte of the PSW controls the interrupt system of the C166S V2 CPU.

Note: For a summary of the PSW register, please refer to ***Section 2.6.6***

PSW
Processor Status Word **bSFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				IEN	HLD EN	BANK		USR1	USR0	MUL IP	E	Z	V	C	N
rwh				rw	rw	rwh		rwh	rwh	r	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
ILVL	[15:12]	rwh	**CPU Priority Level** 0_H Lowest Priority ... ... F_H Highest Priority
IEN	[11]	rw	**Interrupt/PEC Enable Bit (globally)** 0 Interrupt/PEC requests are disabled 1 Interrupt/PEC requests are enabled
BANK	[9:8]	rwh	**Reserved for register file bank selection** 00 Global register bank 01 Reserved 10 Local register bank 1 11 Local register bank 2

CPU Priority ILVL defines the current level for the CPU operation, thus, this bit field reflects the priority level of the currently executed routine. When the CPU enters an interrupt service routine this bit field is set to the priority level of the request that is being serviced. The previous PSW is saved in the system stack before entering interrupt service routine. To be serviced, any interrupt request must have a higher priority level than the current CPU priority level. Any request of the same or a lower level will not be acknowledged.
The current CPU priority level may be adjusted via software to select interrupt request sources that can be serviced.

PEC transfers do not really interrupt the CPU, but rather “steal” some CPU cycle, so PEC services do not influence the ILVL field in the PSW.

Hardware traps set the CPU level to the maximum priority (15). Therefore, no interrupt or PEC requests will be acknowledged while an exception trap service routine is being executed.

The TRAP instruction does not change the CPU level, so software trap service routines may be interrupted by higher requests.

Register Bank BANK defines the currently used register bank for the CPU operation. When the CPU enters an interrupt service routine, this bit field is updated to select the register bank associated with the serviced request.

Note: The TRAP instruction does not change the register bank.

Note: Hardware traps always use the global register bank.

Interrupt Enable bit IEN globally enables or disables interrupts and PEC operations. When IEN is cleared, no new interrupt requests are accepted by the CPU after IEN was set to 0. However, requests that have already entered the pipeline will be completed. If IEN is set to 1, then all interrupt sources are globally enabled.

Note: To generate requests, interrupt sources must be also enabled by the interrupt enable bits in their associated control register.

Note: Traps are non-maskable and, therefore, are not controlled by the IEN bit.

5.2.2 Saving the Status during Interrupt Service

Before an operating system or ITC can actually service a task switch request or interrupt, the CPU must save the current task status. The C166S V2 CPU saves the CPU status (PSW) along with the return address in the system stack. The return address defines the point at which the execution of the interrupted task is to be resumed after returning from the service routine. This return address is specified by the Instruction Pointer (IP) and, in the case of a segmented memory model, also by the Code Segment Pointer (CSP). Bit SGTDIS in the CPUCON1 register defines which memory model is used and, therefore, controls how the return address is stored.

In the case of non-segmented mode, the system stack stores PSW first and then IP. In segmented mode, PSW is followed by CSP and the IP. This order optimizes the use of the system stack if segmentation is disabled.

The CPU priority field (ILVL in PSW) is updated with the priority of the interrupt request that is to be serviced, so the CPU now executes on the new level.

The BANK field in the PSW register is changed to select the register bank associated with the interrupt request. The associations between interrupt requests and register banks are programmed in the Interrupt Controller (ITC).

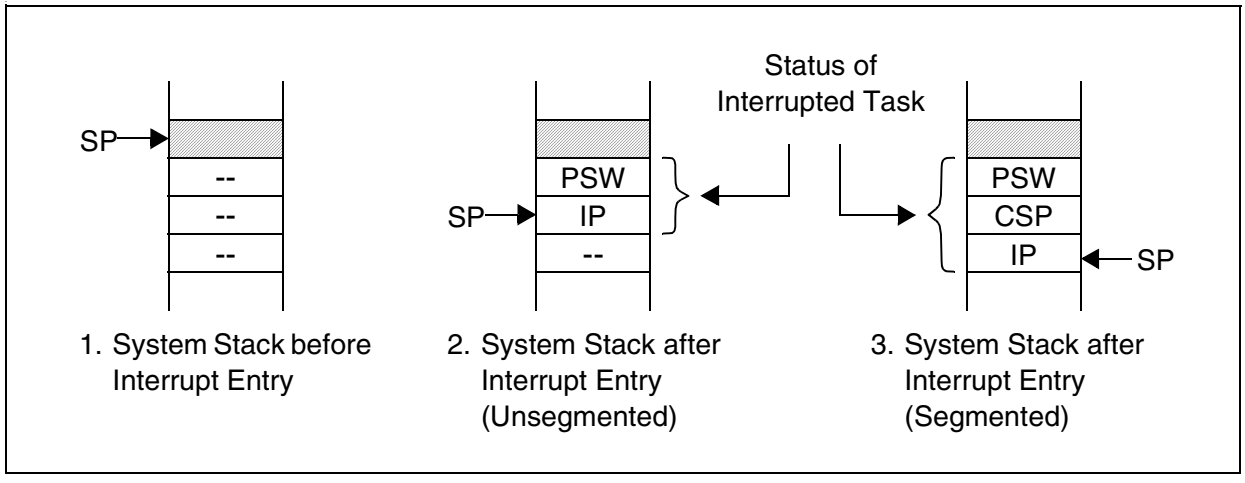


Figure 5-3 Task Status Saved on the System Stack

After accepting an interrupt request, the C166S V2 CPU sends an acknowledge to the ITC that the requested interrupt is being serviced. The vector associated with the requesting source is loaded into the IP and CSP and the first instruction of the service routine is fetched. All other CPU resources, such as data page pointers and the context pointer, are not affected.

When the CPU returns from the interrupt service routine (RETI is executed), the status information is popped from the system stack in reverse order. The status information contents depend on the SGTDIS bit value (see **Figure 5-3**).

5.2.3 Context Switching

An interrupt service routine usually saves all the registers it uses in the stack, and restores them before returning. The more registers a routine uses, the more time is wasted by saving and restoring. The C166S V2 CPU allows the complete bank of CPU registers (GPRs) to be switched, so the service routine executes within its own separate context. There are two ways to switch a context in the C166S V2 core (for details, see **Section 2.4.3**):

1. **Switching Context by Changing the Selected Register Banks**
   Selection of the register bank used in the interrupt task is programmed in the Interrupt Controller. During the execution of the interrupt entry procedure, the change of the register bank is automatically executed. After switching to one of the two local register banks, the service routine may now use its “own registers” directly. This local register bank is preserved when the service routine is terminated; thus, its contents are available on the next call.
   When switching to the global register bank, the service routine must also switch the context of the global register bank (see the next section) to get a private set of GPRs.

2. **Switching Context of the Global Register Bank by Changing Context Pointer**
   The C166S V2 CPU allows the complete global register bank of CPU registers (GPRs) to be changed with a single instruction; so, the service routine executes within its own separate context. The instruction “SCXT CP, #New_Bank” pushes the contents of the context pointer (CP) into the system stack and loads CP with the immediate value “New_Bank”. The new CP value sets a new global register bank. The service routine may now use its “own registers”. This global register bank is preserved when the service routine is terminated; thus, its contents are available for the next call. Before returning (RETI), the previous CP is simply popped from the system stack; thus, returning the registers to the original global register bank.

Note: Resources used by the interrupting program must eventually be saved and restored, such as the DPPs and the registers of the MUL/DIV unit.
There are certain timing restrictions during context switching that are associated with pipeline behavior. For details, see ***Section 2.4.3.2****.*

5.2.4 Fast Bank Switching

The interrupt handler of the C166S V2 CPU supports an additional enhanced feature (compared to other members of the C166 family) for normal interrupts called Fast Bank Switching. To speed up interrupt handling, the core can use fast General Purpose Register (GPR) bank switching for interrupts with an interrupt level greater or equal 12. For every arbitration priority level with ILVL = ‘15_D’-‘12_D’ and XGLVL = ‘7_D’-‘0_D’, the register bank can be selected via two bits. These bits are located in the two register bank selection registers BNKSELx (x = 0,..,3). The BNKSEL2 and BNKSEL3 registers are only implemented in configurations using the GPX extension bit.

BNKSELx (x = 0... 3)
Register Bank Selection Register x **XSFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPRSEL7		GPRSEL6		GPRSEL5		GPRSEL4		GPRSEL3		GPRSEL2		GPRSEL1		GPRSEL0	
rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
GPRSELx (x = 0... 7)	[x+1:x]	rw	**Register Bank Selection** 00 Global register bank 01 Reserved 10 Local register bank 1 11 Local register bank 2

Note: The GPRSELx value of the current triggered interrupt is automatically transferred into the Program Status Word (PSW).

Table 5-1 identifies the arbitration priority level assignment to the respective bit fields within the four register bank selection registers:

Table 5-1 Register Bank Assignment

Interrupt Priority Level (ILVL)	Group Priority Level (XGLVL)	Assigned GPRSELx Register	Interrupt Priority Level (ILVL)	Group Priority Level (XGLVL)	Assigned GPRSELx Register
15	7	BNKSEL3.GPRSEL7	13	7	BNKSEL2.GPRSEL7
15	6	BNKSEL3.GPRSEL6	13	6	BNKSEL2.GPRSEL6
15	5	BNKSEL3.GPRSEL5	13	5	BNKSEL2.GPRSEL5
15	4	BNKSEL3.GPRSEL4	13	4	BNKSEL2.GPRSEL4
15	3	BNKSEL1.GPRSEL7	13	3	BNKSEL0.GPRSEL7
15	2	BNKSEL1.GPRSEL6	13	2	BNKSEL0.GPRSEL6
15	1	BNKSEL1.GPRSEL5	13	1	BNKSEL0.GPRSEL5
15	0	BNKSEL1.GPRSEL4	13	0	BNKSEL0.GPRSEL4
14	7	BNKSEL3.GPRSEL3	12	7	BNKSEL2.GPRSEL3
14	6	BNKSEL3.GPRSEL2	12	6	BNKSEL2.GPRSEL2
14	5	BNKSEL3.GPRSEL1	12	5	BNKSEL2.GPRSEL1
14	4	BNKSEL3.GPRSEL0	12	4	BNKSEL2.GPRSEL0
14	3	BNKSEL1.GPRSEL3	12	3	BNKSEL0.GPRSEL3
14	2	BNKSEL1.GPRSEL2	12	2	BNKSEL0.GPRSEL2
14	1	BNKSEL1.GPRSEL1	12	1	BNKSEL0.GPRSEL1
14	0	BNKSEL1.GPRSEL0	12	0	BNKSEL0.GPRSEL0

5.3 Traps

A software trap is initiated by the TRAP instruction. The TRAP instruction can call an interrupt service routine by its associated vector number. The trap number specified in the operand field of the trap instruction determines which vector location of the vector table will be used.

5.3.1 Software Traps

The TRAP instruction is used to cause a software call to an interrupt service routine. The trap number specified in the operand field of the trap instruction determines which vector location of the vector table will be used.

The TRAP instruction has an effect similar to an interrupt request at the same vector. PSW, CSP (in segmentation mode), and IP are pushed into the system stack and then a jump is taken to the specified vector location. When a software trap is executed, the CSP for the trap service routine is loaded with the value of the VECSEG register. No Interrupt Request flags are affected by the TRAP instruction. The interrupt service routine called by a TRAP instruction must be terminated with a RETI (return from interrupt) instruction to ensure correct operation.

Note: The CPU priority level and the selected register bank in PSW register are not modified by the TRAP instruction; so, the service routine is executed with the same priority level as the interrupt task. Therefore, the service routine entered by the TRAP instruction can be interrupted by other traps or by higher priority interrupts, unless triggered by a real hardware event. The service routine also works with an unchanged register bank. If the hardware triggers the same service routine, register bank can be selected by the ITC and may be different.

5.3.2 Hardware Traps

Hardware Traps are issued by faults or specific system states that occur during runtime (not identified at compile time). The C166S V2 CPU distinguishes eight different hardware trap functions. When a hardware trap condition has been detected, the CPU branches to the trap vector location for the respective trap condition. The instruction causing the trap event is completed before the trap handling routine is entered.

Hardware traps are not-maskable and always have a priority higher than any other CPU task. If several hardware trap conditions are detected within the same instruction cycle, the highest priority trap is serviced. In case of a hardware trap, the injection unit injects an ITRAP instruction into the pipeline.

The ITRAP instruction performs the following actions:

- Pushes PSW, CSP (in segmented mode) and IP into the System Stack
- Sets CPU level in the PSW register to the highest possible priority level, which disables all interrupts and DMA transfers
- Selects the global register bank for the trap service routine
- Branches to the trap vector location specified by the trap number of the trap condition

The eight hardware functions of the C166S V2 CPU are divided in two classes: Class A and Class B.

Class A traps are:

- External Non-Maskable Interrupts NMI
- Stack Overflow
- Stack Underflow
- Software Break

These traps share the same trap priority, but have an individual vector address.

Class B traps are:

– Undefined Opcode
– Parity Fault
– Protection Fault
– Illegal Word Operand Access

The Class B traps share the same interrupt node and interrupt vector. The bit addressable Trap Flag Register (TFR) allows a trap service routine to identify the trap that caused the exception.

The Trap Flag Register TFR

Each trap function is indicated by a separate request flag. When a hardware trap occurs, the corresponding request flag in register TFR is set to 1.

TFR
Trap Flag Register **bSFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NMI	**STK OF**	**STK UF**	**SOFT BRK**	0	0	0	0	**UND OPC**	0	0	**PAR FLT**	**PRT FLT**	**ILL OPA**	0	0
rwh	rwh	rwh	rwh	r	r	r	r	rwh	r	r	rwh	rwh	rwh	r	r

Field	Bits	Type	Description
NMI[1)]	[15]	rwh	**Non maskable interrupt flag** 0 No non-maskable interrupt detected 1 Non-maskable interrupt detected
STKOF[1)]	[14]	rwh	**Stack overflow flag** 0 No stack overflow event detected 1 Stack overflow event detected
STKUF[1)]	[13]	rwh	**Stack underflow flag** 0 No stack underflow event detected 1 Stack underflow event detected
SOFTBRK[1)]	[12]	rwh	**Software Break** 0 No software break event detected 1 Software break event detected
UNDOPC[1)]	[7]	rwh	**Undefined Opcode** 0 No undefined opcode event detected 1 Undefined opcode event detected
PARFLT[1)]	[4]	rwh	**Parity Fault**[2)] 0 No parity fault event detected 1 Parity fault event detected

Field	Bits	Type	Description
PRTFLT[1]	[3]	rwh	**Protection Fault** 0 No protection fault event detected 1 Protection fault event detected
ILLOPA[1]	[2]	rwh	**Illegal word operand access** 0 No illegal word operand access event detected 1 Illegal word operand access event detected

1) This Bit supports bit-protection

2) Parity fault on instruction fetch interface, usable for memories with parity check.

Note: The trap service routine must clear the respective trap flag; otherwise, a new trap will be requested after exiting the service routine. Setting a trap request flag by software causes the same effects as if it had been set by hardware.

The reset functions (hardware, software, watchdog) may be also regarded as a type of trap. Reset functions have the highest priority (trap priority III). Class A traps have the second highest priority (trap priority II). At the third rank are Class B traps (trap priority I); thus, a Class A trap can interrupt a Class B trap.

Table 5-2 Hardware Trap Summary

Exception Condition	Trap Flag	Trap Vector	Trap Number	Trap Priority
Reset Functions:				
Hardware Reset		RESET	00_H	III
Software Reset		RESET	00_H	III
Watchdog Timer Overflow		RESET	00_H	III
Class A Hardware Traps:				
Non-Maskable Interrupt	NMI	NMITRAP	02_H	II.3
Stack Overflow	STKOF	STOTRAP	04_H	II.2
Stack Underflow	STKUF	STUTRAP	06_H	II.1
Software Break	SOFTBRK	SBRKTRAP	08_H	II.0
Class B Hardware Traps:				
Undefined Opcode	UNDOPC	BTRAP	$0A_H$	I
Parity Fault	PARFLT	BTRAP	$0A_H$	I
Protection Fault	PRTFLT	BTRAP	$0A_H$	I
Illegal Word Operand Access	ILLOPA	BTRAP	$0A_H$	I

Class A Trap

Class A traps are generated by the high priority system NMI or by special CPU events such as the software break, a stack overflow, or an underflow event. Class A traps are

not used to indicate hardware failures. After a Class A event, a dedicated service routine is called to react to the events. Each Class A trap has its own vector location in the vector table. After finishing the service routine, the instruction flow must be further correctly executed. This explains why Class A traps cannot interrupt atomic/extend sequences and I/O accesses in progress. For example, an interrupted extend sequence cannot be restarted.

All Class A traps are generated in the pipeline during the execution of instructions, with an exception of NMI, which is an asynchronous external event. It is not possible for two different instructions in the pipeline to generate traps in the same CPU cycle. Class A trap events can be generated only during the memory stage of execution. The execution of instructions which caused a Class A trap event is always completed. In the case of a Class A trap, the pipeline is directly canceled and the IP of the instruction following the last executed one is pushed into the stack. In the case of an atomic/extend sequence or I/O read access in progress, the execution continues till the sequence completion. Upon completion of the sequence, the IP of the instruction following the last one executed is pushed into the stack. Therefore, in the case of a Class A trap, the stack always contains the IP of the first not-executed instruction in the instruction flow.

Note: The Branch Folding Unit allows an execution of branch instructions in parallel with the preceding instruction. The pre-processed branch instruction is combined with the preceding instruction. The branch is executed together with the instruction which caused the Class A trap. The IP of the first following not-executed instruction in the instruction flow is then pushed on the stack.

If more than one Class A trap occurs at a same time, they are prioritized internally. The NMI trap has the highest priority and the software break has the lowest.

Note: In the case of two different Class A traps occurring simultaneously, both trap flags are set. The IP of the instruction following the last one executed is pushed into the stack. The trap with the higher priority is executed. After return from the service routine, the IP is popped from the stack and immediately pushed again because of the other pending Class A trap (unless the trap related to the second trap flag in TFR has been cleared by the first trap service routine).

Class B Trap

Class B traps are generated by unrecoverable hardware failures. In the case of a hardware failure, the CPU must immediately start a failure service routine. Class B traps can interrupt an atomic/extend sequence and an I/O read access. After finishing the Class B service routine, a restoration of the interrupted instruction flow is not possible.

All Class B traps have the same priority (trap priority I). When several Class B traps become active at the same time, the corresponding flags in the TFR register are set and the trap service routine is entered. Because all Class B traps have the same vector, the priority of service of simultaneously occurring Class B traps is determined by the software in the trap service routine.

The Parity Fault is an asynchronous external event while all other Class B traps are generated in the pipeline during the execution of instructions. It is not possible for two different instructions in the pipeline to generate Class A and Class B traps in the same CPU cycle. Class B trap events can be generated only during memory stage execution.

Instructions which caused a Class B trap event are always executed. In the case of a class B trap, the pipeline is directly canceled and the IP of the instruction following the one which caused the trap is pushed on the stack. Therefore, the stack always contains the IP of the first following not-executed instruction in the instruction flow.

Note: The Branch Folding Unit allows the execution of branch instructions in parallel with the preceding instruction. The pre-processed branch instruction is combined with the preceding instruction. The branch is executed together with the instruction causing the Class B trap. The IP of the first following not-executed instruction in the instruction flow is pushed into the stack.

During execution of a Class A trap service routine, any Class B trap will not be serviced until the Class A trap service routine is exited with a RETI instruction. In this case, the Class B trap condition is stored in the TFR register, but the IP value of the instruction which caused this trap will be lost.

Note: If a Class A trap occurs simultaneously with a Class B trap, both trap flags are set. The IP of the instruction following the one which caused the trap is pushed into the stack, and the Class A trap is executed. If this occurs during execution of an atomic/extend sequence or I/O read access in progress, then the presence of the Class B trap breaks the protection of atomic/extend operations and the class A trap will be executed immediately without waiting for the sequence completion. After return from the service routine, the IP is popped from the system stack and immediately pushed again because of the other pending Class B trap. In this situation, the restoration of the interrupted instruction flow is not possible.

- **External NMI Trap:** Whenever a high to low transition on the dedicated external NMI pin (Non-Maskable Interrupt) is detected, the NMI flag in register TFR is set and the CPU will enter the NMI trap routine.
- **Stack Overflow Trap:** Whenever the stack pointer is implicitly decremented and the stack pointer is equal to the value in the stack overflow register STKOV, the STKOF flag in register TFR is set and the CPU will enter the stack overflow trap routine.
- **Stack Underflow Trap:** Whenever the stack pointer is implicitly incremented and the stack pointer is equal to the value in the stack underflow register STKUN, the STKUF flag is set in register TFR, and the CPU will enter the stack underflow trap routine.
- **Software Break Trap:** When the instruction currently being executed by the CPU is a SBRK instruction, the SOFTBRK flag is set in register TFR and the CPU enters the software break debug routine. The flag generation of the software break instruction can be disabled by an On-chip Emulation Module. In this case, the instruction only breaks the instruction flow and signals this event to the debugger. The flag is not set and the trap will not be executed.

- **Undefined Opcode Trap:** When the instruction currently being decoded by the CPU does not contain a valid C166S V2 CPU opcode, the UNDOPC flag is set in register TFR and the CPU enters the undefined opcode trap routine. The instruction that causes the undefined opcode trap is executed as a NOP.
- **Parity Fault Trap:** When a parity error is detected in the system, the PARFLT flag is set in register TFR and the CPU enters the parity fault trap routine. For the C166S V2 CPU, the parity fault is an asynchronous system event. There is no link between the fault and the instruction flow itself.
- **Protection Fault Trap:** Whenever one of the special protected instructions is executed where the opcode of that instruction is not repeated twice in the second word of the instruction and the byte following the opcode is not the complement of the opcode, the PRTFLT flag in register TFR is set and the CPU enters the protection fault trap routine. The protected instructions include DISWDT, EINIT, IDLE, PWRDN, SRST, ENWDT and SRVWDT. The instruction that causes the protection fault trap is executed like a NOP.
- **Illegal Word Operand Access Trap:** Whenever a word operand read or write access is attempted to an odd byte address, the ILLOPA flag in register TFR is set and the CPU enters the illegal word operand access trap routine.

5.4 Peripheral Event Controller

The Peripheral Event Controller (PEC) makes a decision about the CPU action required to manage an interrupt request. It may be either normal interrupt service or fast data transfer between two memory locations. The C166S V2 PEC controls eight fast data transfer channels.

If normal interrupt is requested, the CPU temporarily suspends the current program execution and branches to an interrupt service routine. The current program status and context must be preserved.

If a PEC channel is selected for servicing an interrupt request, a single word or byte data transfer between any two memory locations is to be performed. During a PEC transfer, the normal program execution of the CPU is halted. No internal program status information needs to be saved. The PEC transfer is the fastest possible interrupt response. In many cases, a PEC transfer is sufficient to service the peripheral request (serial channels, for example).

The PEC channels can perform the following actions:

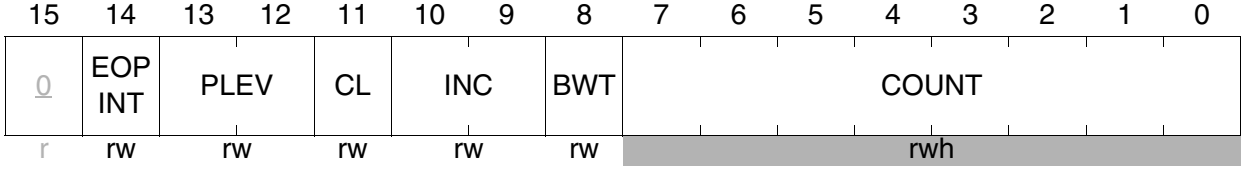
- Byte or word transfer
- Continuous data transfer
- PEC channel-specific interrupt request upon data transfer completion or common for all channels “End of PEC” interrupt for enhanced handling
- Automatic increment of source or/and destination pointers with support of memory to memory transfer

Note: PEC transfer is executed if its priority level is higher than current CPU priority level.

5.4.1 PEC Control Registers

Each PEC channel is controlled by the respective **PEC** channel **C**ontrol register (PECCx) and a set of source and destination pointers (SRCPx, DSTPx and PECSEGx), where ‘x’ stands for the PEC channel number. The PECCx registers control the arbitration priority level assignment to the PEC channels and the action to be performed.

PECCx
PEC Channel Control Register (x=7-0) SFR Reset Value: 0000_H



Field	Bits	Type	Description
EOPINT	[14]	rw	**End of PEC Interrupt Selection** 0 End of PEC interrupt with the same level as the PEC transfer is trigger 1 End of PEC interrupt is serviced by a separate interrupt node with programmable interrupt level (EOPIC) and interrupt sharing control register (PECISNC)
PLEV	[13:12]	rw	**PEC Level Selection** This bit field controls the PEC channel assignment to an arbitration priority level. (see section below)
CL	[11]		**Channel Link Control** 0 PEC channels work independently 1 Pairs of PEC channels are linked together
	[10:9]	rw	**Increment Control** (Modification of source and destination pointer after PEC transfer) 00 No modification 01 Increment of destination pointer DSTPx by 1 (BWT = 1) or by 2 (BWT = 0) 10 Increment of source pointer SRCPx by 1 (BWT = 1) or by 2 (BWT = 0) 11 Increment of destination pointer DSTPx and source pointer SRCPx by 1 (BWT = 1) or by 2 (BWT = 0)

Field	Bits	Type	Description
BWT	[8]	rw	**Byte/Word Transfer Selection** 0 Transfer a word 1 Transfer a byte
COUNT	[7:0]	rwh	**PEC Transfer Count** Counts PEC transfers and influences the channel´s action (see section below)

The **Byte/Word Transfer bit (**BWT**)** of the PECCx register determines if a byte or a word is to be moved during a PEC service cycle and defines an increment step size for the pointer(s) to be modified.

The **PEC Transfer Count field** (COUNT) of the PECCx directly controls the action of the respective PEC channel. The contents of the bit field COUNT may specify a certain number of PEC transfers, unlimited transfers, or no PEC service at all.

– If the PEC transfer counter COUNT value is set to $\mathbf{00_H}$, the normal interrupt requests are processed instead of PEC data transfers and the corresponding PEC channel remains idle.
– **Continuous data transfers** are selected by setting the bit field COUNT to $\mathbf{FF_H}$ value. In this case, COUNT is not decremented by the transfers and the respective PEC channel can serve unlimited number of PEC requests until it is modified by the program.
– If the bit field COUNT is set to service a specified number of requests by the respective PEC channel, it is decremented with each PEC transfer and the request flag is cleared to indicate that the request has been serviced. When COUNT reaches 00_H, it immediately activates the interrupt service routine that has the same priority level (EOPINT = 0) or triggers the “End of PEC” interrupt with a different priority level (EOPINT = 1). When COUNT is **decremented from $\mathbf{01_H}$ to $\mathbf{00_H}$** after a data transfer, the request flag will be cleared if EOPINT is set to 1. If EOPINT is 0, the request flag will not be cleared and another interrupt request will be generated on the same priority level. The respective PEC channel remains idle and the associated interrupt service routine is activated instead of PEC transfer because COUNT contains the $\mathbf{00_H}$ value. (see **Section 5.4.3**).

The EOPIC register is the interrupt control register of the End Of PEC interrupt.

The Register **PECISNC** contains flags of the “End of PEC” interrupt node. This node is used when enhanced “End of PEC” interrupt feature was invoked and control bit EOPINT is set to 1 in the corresponding **PECCx.**

Figure 5-4 shows the usage of the “End of PEC” interrupt subnode:

EOPIC
Interrupt Control Register[1)] bESFR Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	GPX	EOP IR	EOP IE	ILVL				GLVL	
r	r	r	r	r	r	r	rw	rwh	rw	rw				rw	

1) The EOPIC register is assigned to one of the interrupt nodes. The assignment is product specific.

Field	Bits	Type	Description
GPX	[8]	rw	**Group Priority Extension** Defines the value of high-order group level bit
EOPIR[1)]	[7]	rwh	**Interrupt Request Flag** 0 No request pending 1 The source has raised an interrupt request
EOPIE	[6]	rw	**Interrupt Enable Control Bit** 0 Interrupt request is disabled 1 Interrupt request is enabled
ILVL	[5:2]	rw	**Interrupt Priority Level** F_H Highest priority level ... ... 0_H Lowest priority level
GLVL	[1:0]	rw	**Group Priority Level** 3_H Highest priority level ... ... 0_H Lowest priority level
XGLVL	[8],[1:0]		**Extended Group Priority Level** 7_H Highest priority level ... ... 0_H Lowest priority level

1) Bit EOPIR supports bit-protection

PECISNC
PEC Interrupt Sub Node Control bSFR Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C7IR	C7IE	C6IR	C6IE	C5IR	C5IE	C4IR	C4IE	C3IR	C3IE	C2IR	C2IE	C1IR	C1IE	C0IR	C0IE
rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw

Field	Bits	Type	Description
CxIR	15, 13, 11, 9, 7, 5, 3, 1	rwh	**Interrupt Sub Node Request Flag of PEC Channel x** [1) 2)] 0 No special end of PEC interrupt request is pending for PEC channel x 1 PEC channel x has raised an end of PEC interrupt request
CxIE	14, 12, 10, 8, 6, 4, 2, 0	rw	**Interrupt Sub Node Enable Control Bit of PEC Channel x** [1) 3)] (individually enables/disables a specific source) 0 End of PEC interrupt request of PEC channel x is disabled 1 End of PEC interrupt request of PEC channel x is enabled

1) x = 7...0

2) NOTE:
The “End of PEC” sub-node interrupt request flags are not cleared by hardware when entering the interrupt service routine (interrupt has been accepted by the CPU), unlike the interrupt request flags of the interrupt nodes (request flags xxIC.xxIR). The interrupt service routine must check the request flags and clear them before executing the RETI instruction.

3) It is recommended to clear an interrupt request flag (CxIR) before setting the respective enable flag (CxIE). Otherwise, former requests still pending will immediately trigger an interrupt request after setting the enable bit.

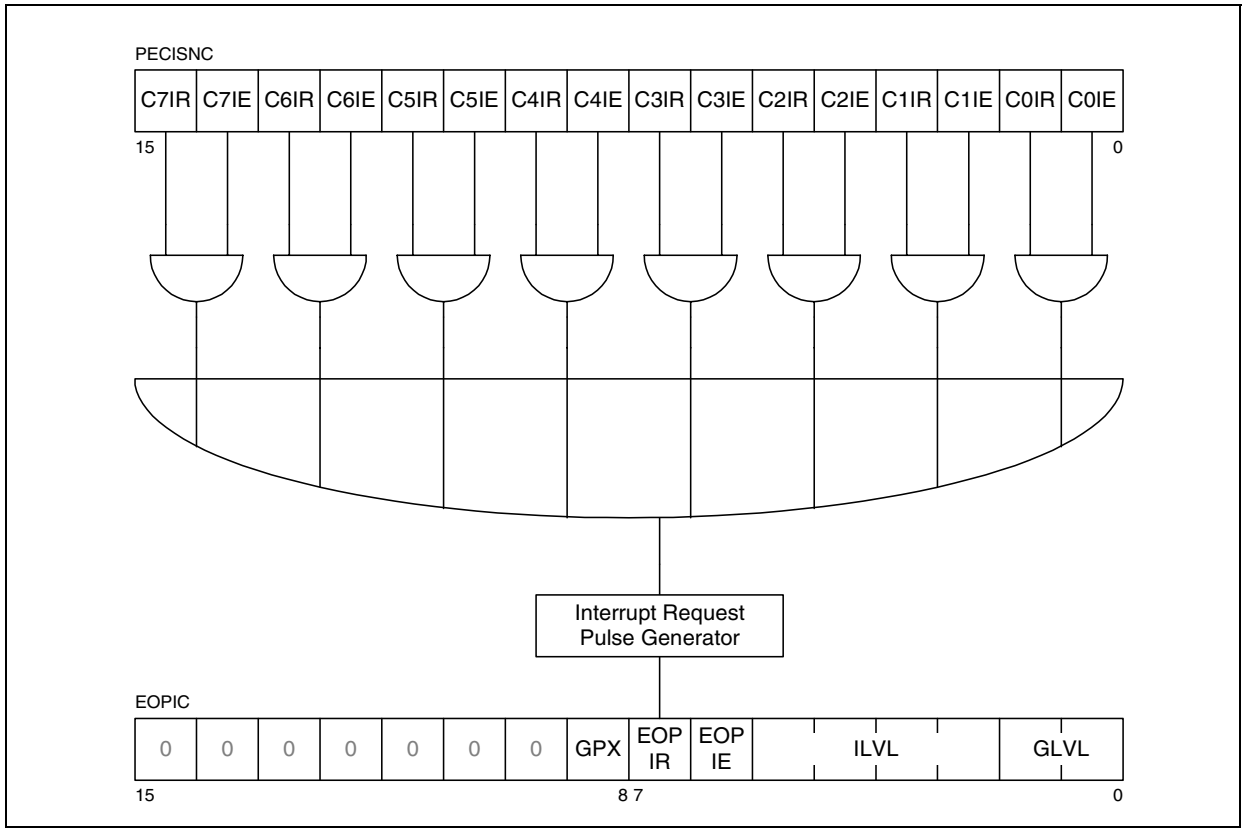


Figure 5-4 End of PEC Interrupt Sub Node

Table 5-3 summarizes the values the bit field COUNT and the corresponding PEC channel actions.

Table 5-3 PEC Channel Actions

Previous COUNT Field Value	Modified COUNT Field Value	Action of PEC Channel and Comments
FF_H	FF_H	**Move a Byte/Word** Continuous transfer mode; COUNT is not modified
FE_H...02_H	FD_H...01_H	Move a Byte/Word and decrement COUNT
01_H	00_H	**Move a Byte/Word** Depending on bit EOPINT, one of two different actions are taken: **EOPINT = 0** (compatible mode) The service request flag (xxIR) of the respective interrupt remains set (it is cleared for all other COUNT values). Therefore, an additional interrupt request is triggered on the next arbitration cycle with a COUNT field value of '00_H' (see next raw) **EOPINT = 1** The service request flag (xxIR) of the respective interrupt is cleared. Additionally, the interrupt request flag of the EOP sub node (PECISNC.CxIR) is set. Furthermore, the interrupt request flag of the end of PEC interrupt node (EOPIC.EOPIR) is automatically set if the sub node request is enabled (PECISNC.CxIE = 1). (see also **Section 5.4.3**)
00_H	00_H	**No PEC action** A normal interrupt is requested instead of a PEC data transfer (see also **Section 5.4.3**).

The **Increment Control Field (INC)** of the PECCx register defines when ether one or both of the PEC pointers must be incremented after the PEC transfer. If the pointers are not to be modified (INC='00'), the respective channel will always move data from the same source to the same destination.

Channel Link Mode (CL bit)

Channel linking allows to perform PEC data transfers via a pair of two PEC channels, that are switched rotationaly, to provide the possibility of data chaining. The linked transfer is in principal the same as described for standard PEC but if the transfer of a linked channel has finished by decrementing the transfer count to zero the PEC controller automatically switches to the partner channel of the pair. While the data

transfers are then controlled by the partner channel the finished channel can be reconfigured. The termination of the transfers of a linked channel is indicated by the triggering of an interrupt. If the channel link bit CL of the active channel or the EOPINT flag is set a End of PEC interrupt is called. Otherwise, the standard interrupt connected to the even channel is requested.

The switch to the PEC channel partner is only possible if channel linking is enabled by setting the PECCx.CL bit of the current channel x. If for a channel the link bit is set but its count value is zero no switch is performed but the normal interrupt of the PEC channel calling node is requested when a new interrupt request occurs for the corresponding node. So the complete linked transfer is terminated if either in the active channel the count value is 0 or the CL flag is 0. Possible channel pairs are only the combinations of channels 0/1, 2/3, 4/5 and 6/7. The PEC channel assignment of the odd numbered channels is ignored if at least one of the channel linking bits (CL) of the channel pair is set. This means an interrupt request connected to the odd channel triggers only the standard interrupt, but no PEC transfer. So, the channel pair is assigned to the interrupt and group level of the even numbered channel partner. After the first initialization for linked transfer the transfer is started with the even numbered channel. The channels toggle as long as CL bit of the currently active channel is set on the transition of the PEC transfer count value from 1 to 0. The even channel is automatically selected if both CL flags are 0 or both transfer counts are 0. In all other cases the last active channel stays selected. A reset of the CL bits during a programmed channel link mode may cause a corruption of the sequence.

A chained PEC sequence should be programmed so that as long the sequence is not finished, the CL bit is set, together with a new transfer count value. For the transfer before the last transfer, the called END of PEC interrupt routine should not reconfigure the count value and should not reset the CL bit. The last transfer channel should not have the CL bit set. So, at the end of the complete transfer, either a standard or an END of PEC trap can be selected by the EOPINT bit of the last channel.

5.4.2 The PEC Source and Destination Pointer

The PEC channels source and destination pointers specify the locations between which the data is to be moved. All pointers are 24-bits wide. The 24-bit source address is stored in the register SRCPx (lower 16 bits of address) and in the high byte of register PECSEGx (highest 8 address bits).

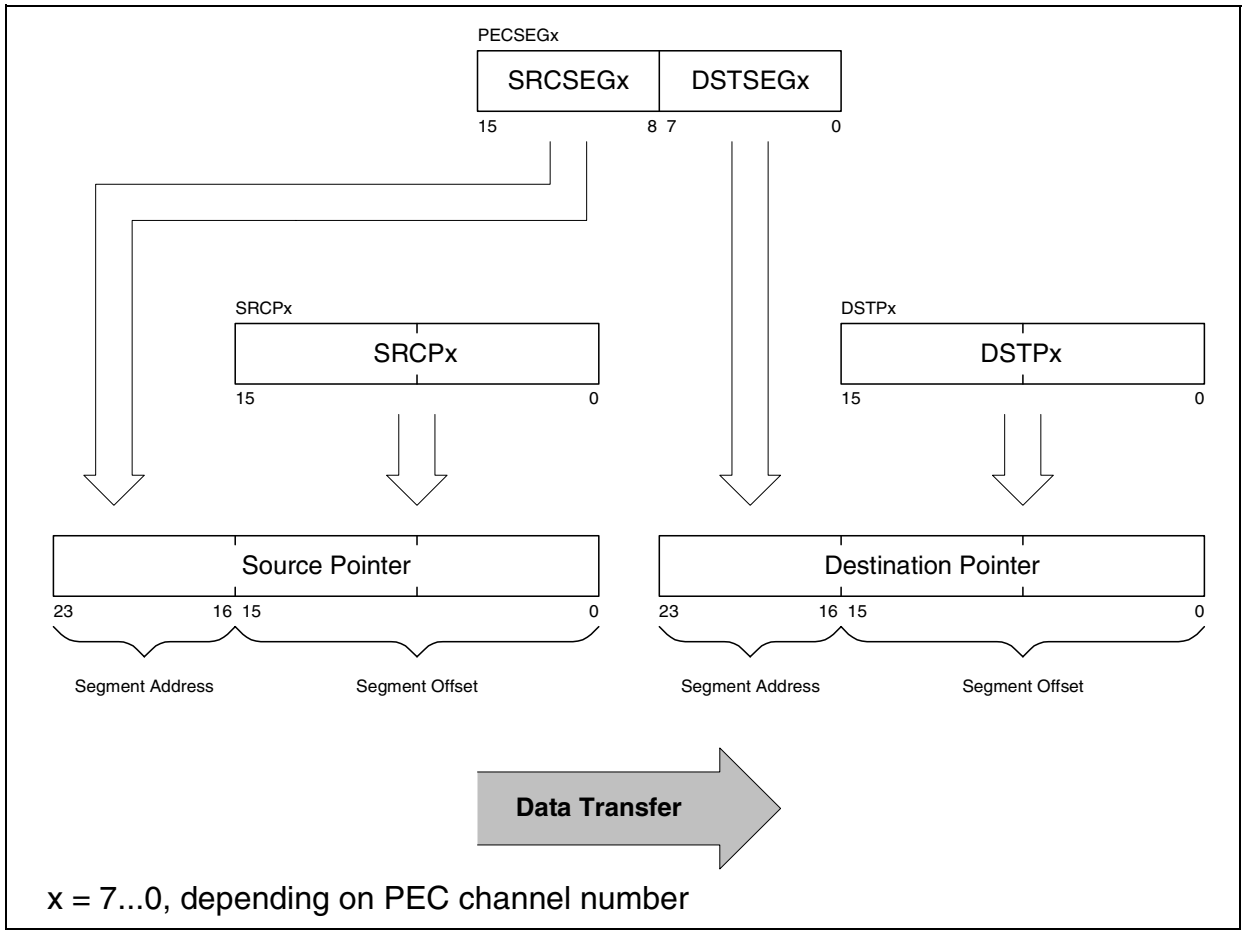


Figure 5-5 PEC Pointer Address Handling

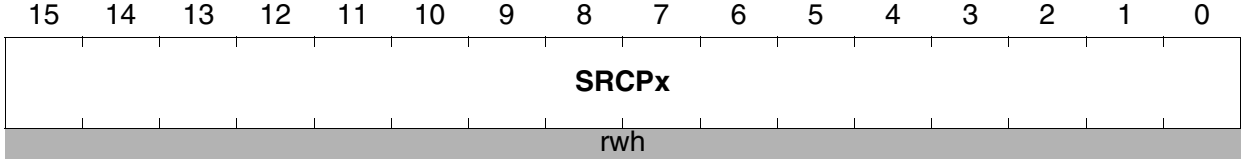
The 24-bit destination address is stored in the register DSTPx (lower 16 bits of address) and in the low byte of register PECSEGx (highest 8 address bits). Only the lower 16 bits of the PEC address pointers (segment offset) can be modified (incremented) by the PEC transfer mechanism. The highest 8 bits, which represent the segment number, are not modified by hardware. Therefore, the PEC pointers may be incremented within the address space of one segment and may not cross the segment border. If the offset address pointer gets the '$FFFF_H$' value in the case of byte transfers (BWT = 1) or '$FFFE_H$' in the case of word transfers (BWT = 0), the next increment will be disregarded. The address register will keep one of these maximum values and no overflow will happen. The described behavior protects the memory from unintentional overwriting. No explicit error event is generated by the system in case of address pointer(s) saturation; therefore, it is the user's responsibility to prevent this condition.

Note: PEC data transfers do not use the data page pointers DPP3...DPP0.

Note: If a word data transfer is selected for a specific PEC channel (BWT = 0), the respective source and destination pointers must both contain a valid word address

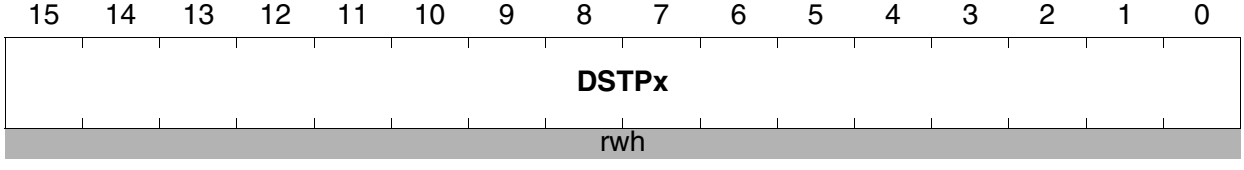
that points to an even byte boundary. Otherwise, the Illegal Word Access trap will be invoked when this channel is used.

SRCPx
PEC Source Pointer (x=7-0) **XSFR** **Reset Value: 0000_H**



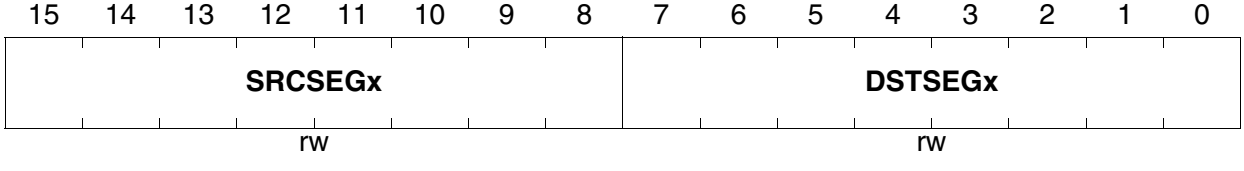
Field	Bits	Type	Description
SRCPx	[15:0]	rwh	**Source Pointer Address of Channel x** Source Address bits 15-0

DSTPx
PEC Destination Pointer (x=7-0) **xSFR** **Reset Value: 0000_H**



Field	Bits	Type	Description
DSTPx	[15:0]	rwh	**Destination Pointer Address of Channel x** Destination Address bits 15-0

PECSEGx
PEC Segment Pointer (x=7-0) **xSFR** **Reset Value: 0000_H**



5.4.3 PEC Handler Interrupt Actions Summary

As described above, two different kinds of interrupts can be triggered by the PEC handler depending on the status of the bitfield COUNT.

Field	Bits	Type	Description
SRCSEGx	[15:8]	rw	**Source Pointer Segment Address of Channel x** Source Address bits 23-16
DSTSEGx	[7:0]	rw	**Destination Pointer Segment Address of Channel x** Destination Address bits 23-16

- PEC channel is enabled[1)] and the bit field COUNT has a value higher than '01_H'.
  a) Control bit EOPINT = 0 or 1
  ACTIONS:
  – PEC request is proceeded
  – No other interrupt activity

- PEC channel is enabled and the bit field COUNT gets a decrement from '01_H' to '00_H' triggered by a service request.
  a) Control bit EOPINT = 0 (compatible with C166)
  ACTIONS:
  – PEC request is proceeded
  – Interrupt request flag (xxIR) of the requesting interrupt node (arbitration winner) is not cleared, participates on the next arbitration cycle, and triggers a normal interrupt on the same level as the PEC request is served.

  b) Control bit EOPINT = 1 (enhanced end of PEC interrupt feature)
  ACTIONS:
  – PEC request is proceeded
  – Interrupt request flag (xxIR) of requesting interrupt node (arbitration winner) is cleared and will not trigger more actions.
  – Interrupt request flag of the end of PEC interrupt subnode will be set (PECISNC.CxIR = 1)
  – If the respective interrupt enable flag of the end of PEC interrupt subnode was set before by software (PECISNC.CxIE = 1), an end of PEC interrupt is requested (EOPIC.EOPIR = 1). This end of PEC interrupt participates on the next arbitration cycle with its priority (selected via EOPIC.ILVL and EOPIC.GLVL), if this interrupt source was enabled before by software (EOPIC.EOPIR = 1). With this behavior, an end of PEC interrupt can be triggered on a level lower than the respective PEC requests have been serviced.

- PEC channel is disabled if the bit field COUNT is cleared (either by hardware or by software).

1) Every PEC channel is automatically enabled when its COUNT value is greater than 00_H.

a) Control bit EOPINT = 0 or 1
ACTIONS:
– A normal interrupt service routine is requested on the PEC channel priority level.

5.4.4 PEC Channel Assignment and Arbitration

The C166S V2 PEC channels can be assigned to a certain arbitration priority level. All requests with interrupt priority levels 8 to 15 and group levels 0 to 3 can be associated with the PEC functionality (eight PEC channels in total). The group extension is not supported for PEC requests, because the 8 PEC channels are assigned to two interrupt levels for compatibility to the C16x family.

The following mechanism shows how to program the bit field PECCx.PLEV to set up a link to a certain interrupt priority level and a group priority level:

```
PEC Channel x
is linked to:
Interrupt priority level (in IC register): (1, ~PLEV.1, ~PLEV.0, x.0)
Extended Group priority level:             (0,     x.1,     x.0)
```

For an easier understanding of this formula, **Table 5-4** lists all possible combinations.

Table 5-4 PEC Channel Assignment

Arbitration Priority Level		PEC Channel x		Arbitration Priority Level		PEC Channel x	
Interrupt Priority Level xxIC.ILVL	Group Priority Level xxIC.XGLVL	PLEV	Ch	Interrupt Priority Level xxIC.ILVL	Group Priority Level xxIC.XGLVL	PLEV	Ch
15	3	**00**	7	11	3	**10**	7
15	2		6	11	2		6
15	1		5	11	1		5
15	0		4	11	0		4
14	3		3	10	3		3
14	2		2	10	2		2
14	1		1	10	1		1
14	0		0	10	0		0
13	3	**01**	7	9	3	**11**	7
13	2		6	9	2		6
13	1		5	9	1		5
13	0		4	9	0		4
12	3		3	8	3		3
12	2		2	8	2		2
12	1		1	8	1		1
12	0		0	8	0		0

All interrupt requests not assigned to a PEC channel go directly to the interrupt handler.

5.5 CPU Action Control Unit

The CPU Action Control Unit multiplexes interrupt/PEC requests with OCDS requests and forwards them to the CPU demanding the corresponding action. It also routes request acknowledges and denies from the core to the corresponding requester. The OCDS requests have programmable priority levels. If another interrupt request that has won an arbitration conflicts with an OCDS request, the one with the higher priority will trigger the CPU action first. However, if both requests (Interrupt/PEC and OCDS) have the same priority level, the interrupt/PEC request wins.

The OCDS break request is sent directly from the OCDS module to the CPU (where it is prioritized) and ignores the CPU Action Control Unit (or any other module of the interrupt and Peripheral Event Controller).

6 External Bus Controller

6.1 Introduction

Although the C166S V2 products provide a powerful set of on-chip peripherals and on-chip program and data memories, these internal units only cover a small fraction of the C166S V2´s address space of up to 16 MByte. The external bus interface allows access to external[1)] peripherals and additional volatile and non-volatile memories. The external bus interface provides a number of configurations, so it can be tailored to fit perfectly into a given application system.

Accesses to external memories or peripherals are executed by the integrated External Bus Controller (EBC). The function of the EBC is controlled via a set of configuration registers. The basic behavior can be programmed via the mode selection registers EBCMODx.

The EBC supports up to eight external chip select channels. Each of these chip select signals is programmable via a set of registers. The FCONCSx registers specify the external bus cycles in terms of address (mux/demux), data (16-bit/8-bit), chip select enable and READY control. The timing of the bus access is controlled by the timing configuration registers TCONCSx, which specify the length of the different access phases. All these parameters are used for accesses within a specific address area which is defined via the corresponding address select register ADDRSELx.

The seven register sets FCONCS1/TCONCS1/ADDRSEL1 to FCONCS7/TCONCS7/ ADDRSEL7 define seven independent ‘address windows’, while all external accesses outside these windows are controlled via the registers FCONCS0 and TCONCS0. Two additional chip select channels with fixed address ranges are defined for the startup and the monitor memory.

The external bus timing is related to the reference clock output CLKOUT. All bus signals are generated in relation to the rising edge of this clock. This behavior eases the timing specification drastically and allows high EBC operating frequencies above 100 MHz. The external bus protocol is compatible with the C16x ones. However, the external bus timing is improved in terms of wait state granularity.

Note: For supporting these improvements, an extended configuration scheme compared to the C16x is defined. The C16x registers SYSCON and BUSCONx are no longer used. In principle the configuration of the external bus controller is done during the application initialization. Therefore, only some initialization code has to be adapted for using the C166S V2 EBC module instead of the C16x external bus controller.

1) C166S V2:’External’ means off-chip However, modules like customer ASIC, startup memory and additional peripherals and memories can be connected on-chip to the external bus module as well. These modules are from the controller sub-system point of view also external, but on-chip.

6.2 Timing Principles

The EBC supports four different access types. Reads and Writes in multiplexed and demultiplexed mode. Multiplexed mode means that the data bus is used in a ‘time-multiplex’ for address (the 16 LSBs) and for data. In demultiplexed mode the data bus is used for data only and an additional 16 bit address bus is available.

Naming Conventions

- **ALE** Address Latch Enable (high active)
  indicates that the applied address is valid
- **$\overline{\text{WR}}$/ $\overline{\text{WRL}}$** Write Strobe (low active)/
  Write Low Byte Strobe (low active)
  configured either to a general write request or a write request for the low byte (see **Table 6-1**)
- **$\overline{\text{BHE}}$/ $\overline{\text{WRH}}$** Byte High Enable (low active)/
  Write High Byte Strobe (low active)
  configured either to an enable for the high byte or a write request for the high byte (see **Table 6-1**)
- **$\overline{\text{RD}}$** Read Strobe (low active)
- **READY** Ready to indicated end of actions (programmable polarity)
- **ADDR** Address Bus split to a part [23:16] and [15:0]
- **DATA** Data Bus [15:0] or shared Data/Address [15:0] Bus
- **$\overline{\text{HOLD}}$** Hold input for foreign bus requests (low active)
- **$\overline{\text{HLDA}}$** Hold Acknowledge (low active)
  master output to grant bus / slave input
- **$\overline{\text{BREQ}}$** Bus Request (low active)

Table 6-1 Write Configurations (see Chapter 6.3.2)

written byte		general write configuration			separated byte low/high writes		
low	high	$\overline{\text{WR}}$	$\overline{\text{BHE}}$	ADDR[0]	$\overline{\text{WRL}}$	$\overline{\text{WRH}}$	ADDR[0]
-	-	inactive	don’t care	0/1	inactive	inactive	0/1
write	-	active	inactive	0	active	inactive	0/1
-	write	active	active	1	inactive	active	0/1
write	write	active	active	0	active	active	0/1

The timings of the external bus can be split up into six phases:

Figure 6-1 Demultiplexed Bus Read

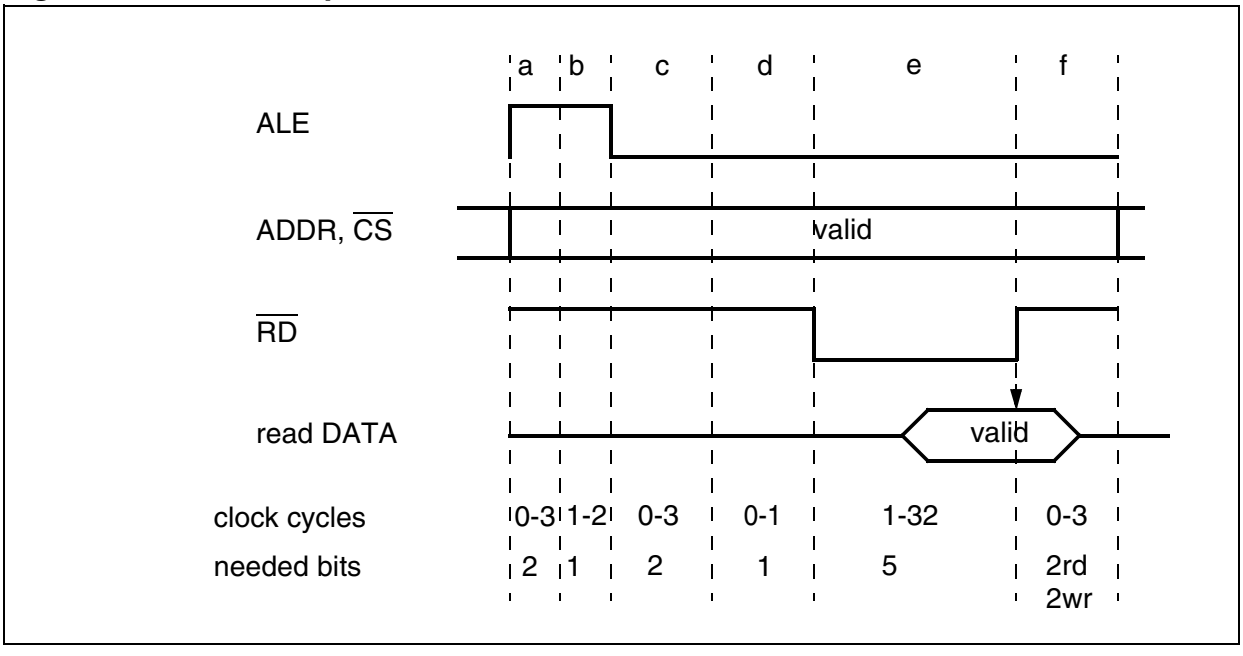
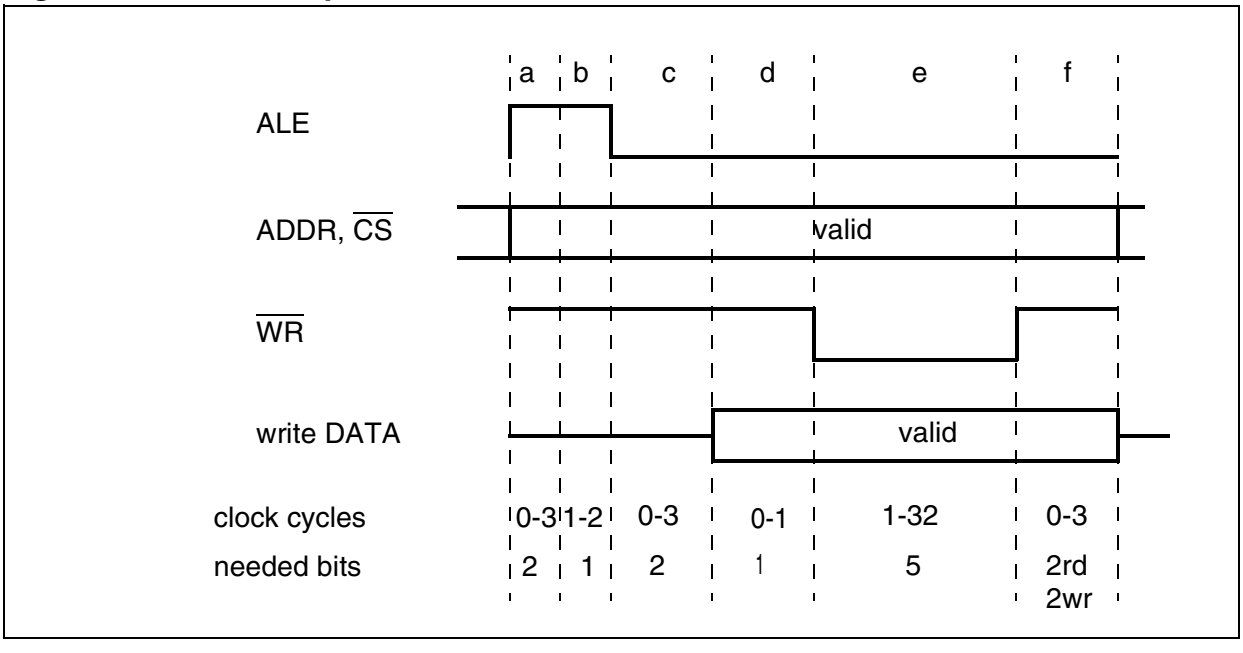


Figure 6-2 Demultiplexed Bus Write



- a phase: addresses valid, ALE high, no command. CS switch tristate wait states
- b phase: addresses valid, ALE high, no command. ALE length
- c phase: addresses valid, ALE low, no command. R/W delay
- d phase: write data valid, ALE low, no command. Data valid for write cycles
- e phase: command (read or write) active. Access time
- f phase: command inactive, address hold. Read data tristate time, write data hold time

Figure 6-3 Multiplexed Bus Read

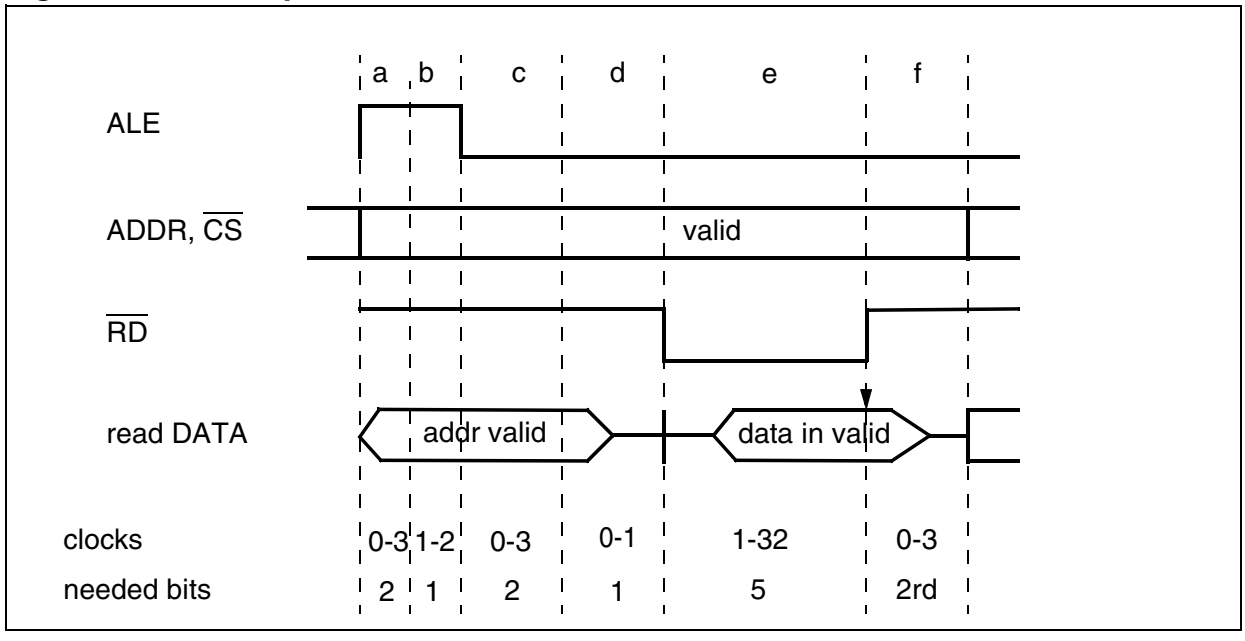
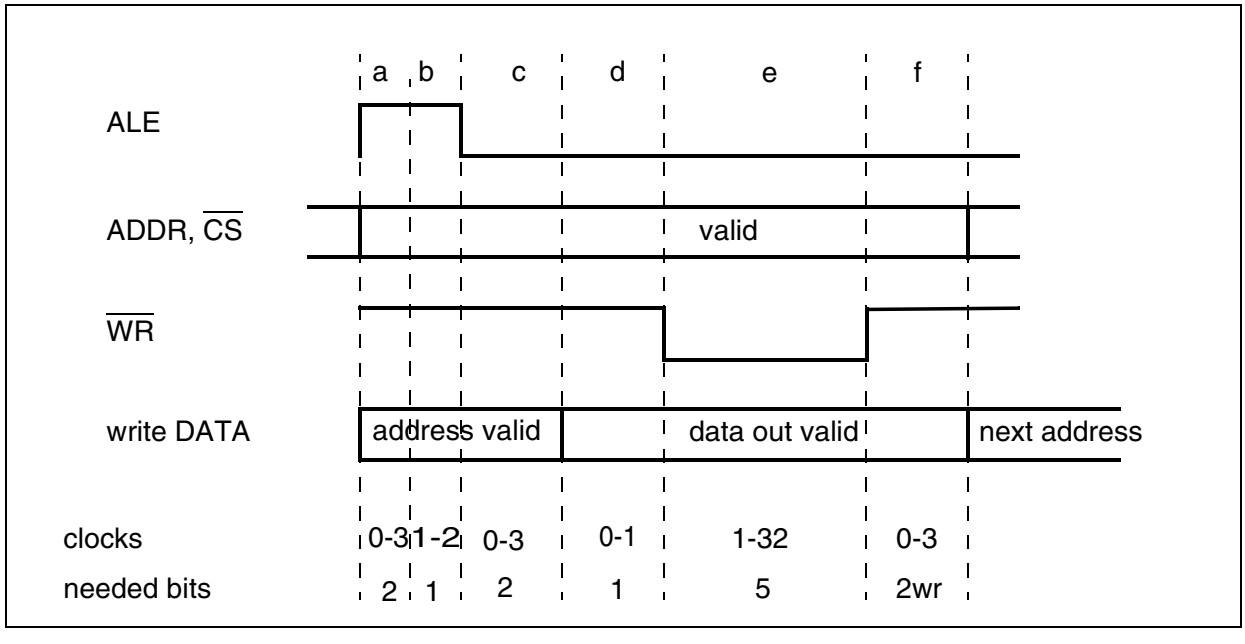


Figure 6-4 Multiplexed Bus Write



- a phase: addresses valid, ALE high, no command. CS switch tristate wait states
- b phase: addresses valid, ALE high, no command. ALE length
- c phase: addresses valid, ALE low, no command. Address hold, R/W delay
- d phase: address tristate for read cycles, data valid for write cycles, ALE low, no command
- e phase: command (read or write) active. Access time
- f phase: command inactive, address hold. Read data tristate time, write data hold time.

6.2.1 A Phase

The **A phase** can take 0-3 clocks. It is used to tristate the databus drivers activated in the previous cycle (tristate wait states after CS switch).
Phase A cycles are not inserted in every access cycle but only when changing the CS. If an access using one chipselect CSx was finished and the next access with a different chipselect CSy is started then Phase A cycles are performed according to the PHA bits set for the first chipselect CSx. This feature is used to optimize wait states with devices having a long turn off delay at their databus drivers like EPROM and FLASH.
The A Phase cycles are inserted while the addresses and ALE of the next cycle are already applied.
If there are idle cycles in between two accesses these clock cycles are taken into account and the A Phase is shortened accordingly. For example if there are three tristate cycles programmed and two idle cycles occurred then the A Phase takes only one clock cycle.

6.2.2 B Phase

The **B phase** can take 1-2 clocks. It is used for selecting devices and registers before giving a command and to define the length of the active ALE. In multiplexed bus mode the address is applied on the data bus for latching.

6.2.3 C Phase

The **C phase** is similar to the A and B phases but ALE is already low. It can take 0-3 clock cycles.
In multiplexed bus mode the address is held for being latched safely. Phase C cycles can be used to delay the command signals (RW delay).

6.2.4 D Phase

The **D phase** can take 0-1 clocks. It is used to tristate the address on the multiplexed bus when a read cycle is performed. For all write cycles it is used to have the data valid on the bus before the command is applied.

6.2.5 E Phase

The **E phase** is the command respectively access phase and takes 1-32 clocks. Read data is fetched, write data is put onto the bus; the command signals are active. Read data is registered with the terminating clock cycle of this phase.
The READY function is lengthening this phase, too (see **Table 6.3.6**). READY controlled access cycles have a random cycle time.

6.2.6 F Phase

The **F phase** can take 0-3 clocks. Addresses and write data are held while the command is inactive. The number of wait states being inserted at the F phase is programmable independently for read and write accesses. The F phase is used for data reads to program tristate wait states on the bidirectional data bus in order to avoid bus conflicts.

6.3 Functional Description

6.3.1 Configuration Register Overview

The EBC registers are functionally split up into three groups:

- EBC mode registers that have influence on global functions.
- Chip select related registers to configure the functionality, timing and size of the chipselect windows.
- Startup and Monitor Memory registers to control the access to these dedicated memories.

CS0 is the default chip select that selects all address space not addressed by another chip select or occupied by internal address space. Therefore CS0 has no ADDRSEL register.

All EBC registers are write protected by the EINIT protection mechanism. This means that after execution of the EINIT instruction by the C166S V2 CPU these registers are not writeable anymore.

For a list of all EBC control registers refer to **Chapter 9.4**. All EBC registers are located in a 128 byte segment.

6.3.2 The EBC MODE Registers EBCMODx

EBC Mode Register 0
EBCMOD0 **XSFR** **Reset value: $00F0_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RDY POL	RDY DIS	ALE DIS	BYT DIS	WR CFG	EBC DIS	SLA VE	ARB EN	CSPEN				SAPEN			
rw	rw	rw	rw	rw	rw	rw	rw	rw				rw			

The EBC Mode Register 0 controls the alternate function of the pins.

Field	Bits	Typ	Description
RDYPOL	15	rw	**READY pin Polarity** 0 READY is active low 1 READY is active high
RDYDIS	14	rw	**READY pin Disable** 0 READY enabled 1 READY disabled[1)]
ALEDIS	13	rw	**ALE pin Disable** 0 ALE enabled 1 ALE disabled[1)]
BYTDIS	12	rw	**BHE pin Disable** 0 BHE enabled 1 BHE disabled[1)]
WRCFG[2)]	11	rw	**Configuration for pins $\overline{\text{WR/WRL}}$, $\overline{\text{BHE/WRH}}$** 0 $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ 1 $\overline{\text{WRL}}$ and $\overline{\text{WRH}}$
EBCDIS	10	rw	**EBC pins Disable** 0 EBC is using the pins for external bus 1 EBC pins disabled[1)]
SLAVE	9	rw	**SLAVE mode enable** 0 Bus arbiter acts in master mode 1 Bus arbiter acts in slave mode
ARBEN	8	rw	**BUS Arbitration Pins enable** 0 $\overline{\text{HOLD}}$, $\overline{\text{HLDA}}$ and $\overline{\text{BREQ}}$ disabled[1)] 1 pins act as $\overline{\text{HOLD}}$, $\overline{\text{HLDA}}$ and $\overline{\text{BREQ}}$

Field	Bits	Typ	Description
CSPEN	[7:4]	rw	**CS Pins Enable**[1)] 0000 no chipselect pins enabled 0001 enables pin CS0 ... 1000 enables pins CS7, ..., CS0 else reserved
SAPEN	[3:0]	rw	**Segment Addresses Pins Enable**[1)] 0000 no segment address pin enabled 0001 enables address pin A[16] ... 1000 enables address pins A[23:16] else reserved

1) disabled pins are tristate and/or usable as General Purpose IO (GPIO)

2) A change of the bit content is not valid before the next external bus access cycle.

The **EBC Mode register 1** controls the general behaviour of the EBC.

EBC Mode Register 1
EBCMOD1 **XSFR** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	DHP DIS	0	APDIS				
r	r	r	r	r	r	r	r	r	rw	r	rw				

	Bits	Typ	Description
0	[15:7] [5]	r	**Reserved** The software always reads a '0'. Although these bits are read only, the software should always write a '0' in case of a write access.
DHPDIS	[6]	rw	**Data High Pins Disable** 0 AD Bus Pins[15:8] enabled 1 AD Bus Pins[15:8] disabled, can be used as GPIO
APDIS[1)]	[4:0]	rw	**Address Pins Disable** 00000 Address Bus Pins [15:0] enabled 11111 Address Bus Pins [15:0] disabled, can be used as GPIO others reserved (do not use)

1) For a demultiplexed external bus access with the address pins disabled no address will be available.

6.3.3 The Timing Configuration registers TCONCSx

The timing control registers are used to program the described cycle timing for the different access phases. The timing control registers may be reprogrammed during code fetches from the affected address window. The new settings are first valid for the next access.

Timing Configuration Register for Chip Select Channel 0

TCONCS0 **XSFR** **Reset value: 6243_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WRPHF		RDPHF		PHE					PHD	PHC		PHB	PHA	
r	rw		rw		rw					rw	rw		rw	rw	

Timing Configuration Register for Chip Select Channel x

TCONCSx **XSFR** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WRPHF		RDPHF		PHE					PHD	PHC		PHB	PHA	
r	rw		rw		rw					rw	rw		rw	rw	

x = 1 ... 7

For controlling accesses to the monitor memory and start up memory there are two timing control registers TCONCSMM and TCONCSSM. The functional control selection and address windows are fixed and not changeable for the built-in memories.

Timing Configuration Register for Chip Select Monitor Memory

TCONCSMM **XSFR** **Reset value: 6243_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WRPHF		RDPHF		PHE					PHD	PHC		PHB	PHA	
r	rw		rw		rw					rw	rw		rw	rw	

Timing Configuration Register for Chip Select Startup Memory

TCONCSSM **XSFR** **Reset value: 6243_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WRPHF		RDPHF		PHE					PHD	PHC		PHB	PHA	
r	rw		rw		rw					rw	rw		rw	rw	

Field	Bits	Typ	Description
0	15	r	**Reserved** The software always reads a ’0’. Although this bit is read only, the software should always write a ’0’ in case of a write access.
WRPHF	[14:13]	rw	**Write Phase F** 00 0 clock cycles ... 11 3 clock cycles
RDPHF	[12:11]	rw	**Read Phase F** 00 0 clock cycles ... 11 3 clock cycles
PHE	[10:6]	rw	**Phase E** 00000 1 clock cycle ... 11111 32 clock cycles
PHD	5	rw	**Phase D** 0 0 clock cycles 1 1 clock cycle
PHC	[4:3]	rw	**Phase C** 00 0 clock cycles ... 11 3 clock cycles
PHB	2	rw	**Phase B** 0 1 clock cycle 1 2 clock cycles
PHA	[1:0]	rw	**Phase A** 00 0 clock cycles ... 11 3 clock cycles

6.3.4 The Function Configuration Registers FCONCSx

The Function Control registers are used to control the bus and ready functionality for a selected address window. It can be distinguished between 8 and 16 bit bus and multiplexed and demulitplexed accesses. Furthermore the READY functionality can be programmed and defined whether the address window is enabled or not.

Function Configuration Register for Chip Select Channel 0

FCONCS0 **XSFR** **Reset value: 0021_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	BTYP		0	RDY MOD	RDY EN	EN CS
r	r	r	r	r	r	r	r	r	r	rw		r	rw	rw	rw

Function Configuration Register for Chip Select Channel x

FCONCSx **XSFR** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	BTYP		0	RDY MOD	RDY EN	EN CS
r	r	r	r	r	r	r	r	r	r	rw		r	rw	rw	rw

x = 1 ... 7

Field	Bits	Typ	Description
0	[15:6]	r	**Reserved** The software always reads a ’0’. Although these bits are read only, the software should always write a ’0’ in case of a write access.
BTYP	[5:4]	rw	**Bus Type Selection** 00 8 bit Demultiplexed 01 8 bit Multiplexed 10 16 bit Demultiplexed 11 16 bit Multiplexed
0	3	r	**Reserved** The software always reads a ’0’. Although this bit is read only, the software should always write a ’0’ in case of a write access.

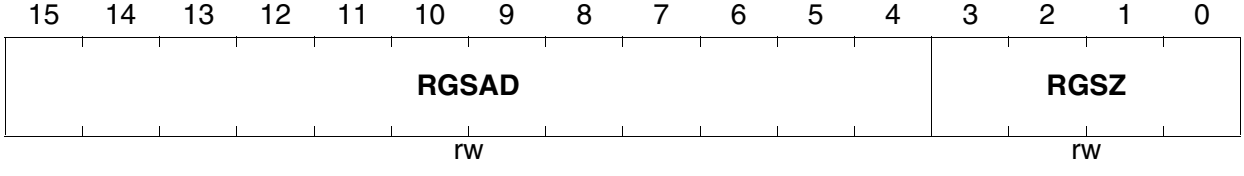
Field	Bits	Typ	Description
RDYMOD	2	rw	**Ready Mode** 0 asynchronous READY 1 synchronous READY
RDYEN	1	rw	**Ready enable** 0 access time is controlled by bitfield PHEx 1 access time is controlled by bitfield PHEx and READY signal
ENCS[1)]	0	rw	**Enable Chip Select** 0 disable 1 enable

1) Disabling a Chip Select not only effects the chip select output signal; it also deactivates the respective address window of the disabled chip select. A disabled address window is also ignored by an address window arbitration (see **Chapter 6.3.5.2**).

6.3.5 The Address Window Selection Registers ADDRSELx

Address range and size Select for Chip Select Channel x

ADDRSELx **XSFR** **Reset value: 0000_H**



x = 1 ... 7

Field	Bits	Typ	Description
RGSAD	[15:4]	rw	Address Range Start Address Selection
RGSZ	[3:0]	rw	Address Range Size Selection (see **Table 6-2**)

Note: There is no register ADDRSEL0, as register set FCONCS0 / TCONCS0 controls all external accesses outside the seven address windows built by the seven address selects ADDRSEL1 to ADDRSEL7.

6.3.5.1 Definition of Address Areas

The seven register sets FCONCS1/TCONCS1/ADDRSEL1 to FCONCS7/TCONCS7/ ADDRSEL7 define seven separate address areas within the address space of the C166S V2. Within each of these address areas external accesses can be driven in one

of the four different bus modes independently. Each ADDRSELx register cuts out an address window, where the corresponding parameters of the registers FCONCSx and TCONCSx are used to control external accesses. The range start address of such a window defines the most significant address bits of the selected window which are consequently not needed to address the memory/module in this window (**Table 6-2**). The size of the window chosen by ADDRSELx.RGSZ defines the relevant bits of ADDRSELx.RGSAD (marked with 'R') which are used to select with the most significant bits of the request address the corresponding window. The other bits of the request address are used to address the memory locations inside this window. The lower bits of ADDRSELx.RGSAD (marked 'x') are disregarded.

Two additional chip select channels, which are used for accessing the startup and the monitor memory, are located in a predefined address range. The size of these two address areas is fixed to 32 kByte.

The address area from 00'8000$_H$ to 00'FFFF$_H$ (32 kbyte) is reserved for C166S V2 CPU internal I/O, the area from BF'0000$_H$ to BF'FFFF$_H$ (64 kbyte) for startup and monitor memory and the area from C0'0000$_H$ to FF'FFFF$_H$ (4 Mbyte) is used by the internal program memory. Therefore, these address areas cannot be used by external resources connected to the external bus.

Table 6-2 Address range and size for ADDRSELx

ADDRSELx		Address Window	
Range Size RGSZ	**Relevant (R) bits of RGSAD**	**Selected address range**	**Range start address A[23:0] selected with R-bits of RGSAD**
3..0	15 ... 4	size	A23 ... A0
0000	RRRR RRRR RRRR	4 KBytes	RRRR RRRR RRRR 0000 0000 0000
0001	RRRR RRRR RRRx	8 KBytes	RRRR RRRR RRR0 0000 0000 0000
0010	RRRR RRRR RRxx	16 KBytes	RRRR RRRR RR00 0000 0000 0000
0011	RRRR RRRR Rxxx	32 KBytes	RRRR RRRR R000 0000 0000 0000
0100	RRRR RRRR xxxx	64 KBytes	RRRR RRRR 0000 0000 0000 0000
0101	RRRR RRRx xxxx	128 KBytes	RRRR RRR0 0000 0000 0000 0000
0110	RRRR RRxx xxxx	256 KBytes	RRRR RR00 0000 0000 0000 0000
0111	RRRR Rxxx xxxx	512 KBytes	RRRR R000 0000 0000 0000 0000
1000	RRRR xxxx xxxx	1 MBytes	RRRR 0000 0000 0000 0000 0000
1001	RRRx xxxx xxxx	2 MBytes	RRR0 0000 0000 0000 0000 0000
1010	RRxx xxxx xxxx	4 MBytes	RR00 0000 0000 0000 0000 0000
1011	Rxxx xxxx xxxx	8 MBytes	R000 0000 0000 0000 0000 0000
11xx	xxxx xxxx xxxx	reserved[1)]	---- ---- ---- ---- ---- ----

1) The complete address space of 12 MByte can be selected by the default chip select CS0.

*Note: The range start address can only be on boundaries specified by the selected range size according to **Table 6-2**.*

6.3.5.2 Address Window Arbitration

For each external access the EBC compares the current address with all address select registers (programmable ADDRSELx and hardwired address select registers for startup and monitor memory) of enabled windows. This comparison is done in four levels.

Priority 1: The hardwired address select registers for startup and monitor memories are evaluated first. A match with one of these two address ranges directs the access to the respective memory using the corresponding chip select with its timing control register. The window of monitor and start up is not accessible by other chip selects.

Priority 2: Registers ADDRSELx [x = 2, 4, 6] are evaluated first. A window match with one of these registers directs the access to the respective external area using the corresponding set of control registers FCONCSx/TCONCSx and ignoring registers ADDRSELy. An overlapping of windows of this group will lead to an undefined behaviour.

Priority 3: A match with registers ADDRSELy [y = 1, 3, 5, 7] directs the access to the respective external area using the corresponding set of control registers FCONCSy/TCONCSy. An overlapping of windows of this group will lead to an undefined behaviour. Overlaps with priority 2 ADDRSELx are only allowed for the (x,y) pairs (2,1), (4,3) and (6,5).

Priority 4: If there is no match with any address select register (neither the hardwired ones nor the programmable ADDRSEL) the access to the external bus uses the general set of control registers FCONCS0/TCONCS0 if enabled.

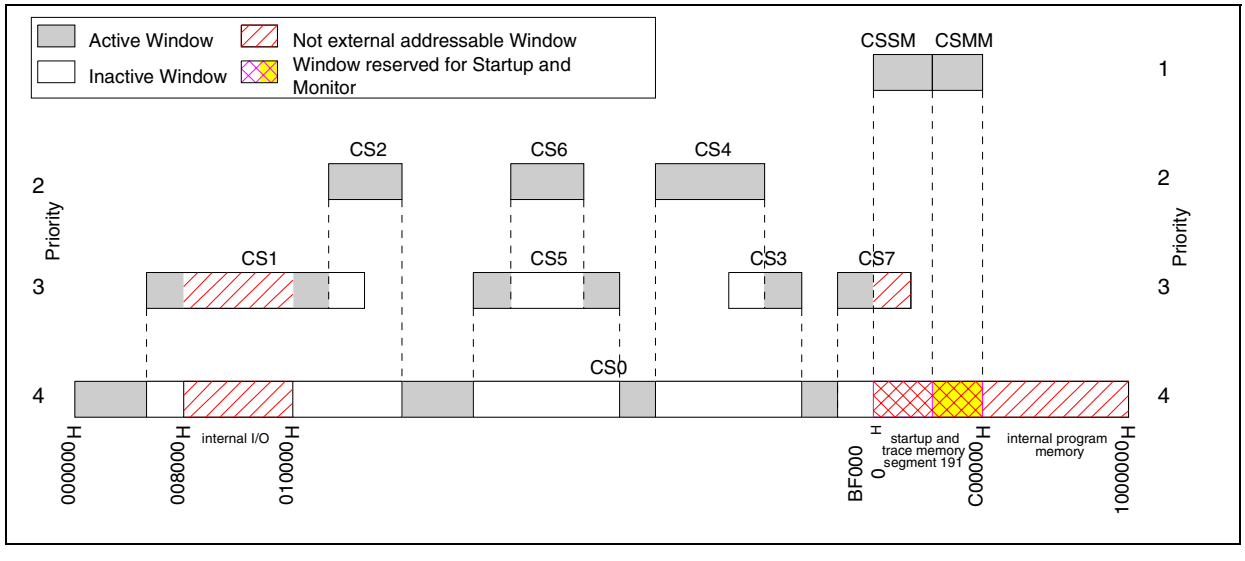


Figure 6-5 Address Window Arbitration

Note: Only the indicated overlaps are allowed. All other overlaps lead to erroneous bus cycles. E.g. ADDSEL4 may not overlap ADDRSEL2 or ADDRSEL1. The

hardwired address ranges for the startup memory and the monitor memory are defined non-overlapping.

6.3.6 Ready Controlled Bus Cycles

6.3.6.1 General

For situations, where the response (access) time of a peripheral is not constant, or where the programmable wait states are not enough, the C166S V2 EBC provides external bus cycles that are terminated via a READY input signal. In this case during phase E the C166S V2 EBC first counts a programmable number of clock cycles (1...32) and starts in the last wait cycle to monitor the internal READY line (see **Figure 6-6**) to determine the actual end of the current bus cycle. The external device drives READY active in order to indicate that data has been latched (write cycle) or is available (read cycle).

The READY pin is generally enabled by setting the bit RDYDIS in EBCMOD0 to ’0’ in order to switch the corresponding port pin. Also the polarity of the READY is defined inside the EBCMOD0 register on the RDYPOL bit.

For a specific access the READY function is enabled via the RDYEN bit in the FCONCSx registers. With FCONCSx.RDYMOD the READY is handled either in synchronous or in asynchronous mode (see also **Figure 6-6**).

When the READY function is enabled for a specific address window, each bus cycle within this window must be terminated with an active READY signal. Otherwise the controller hangs until the next reset. This is also the case for an enabled RDYEN but a disabled READY port pin.

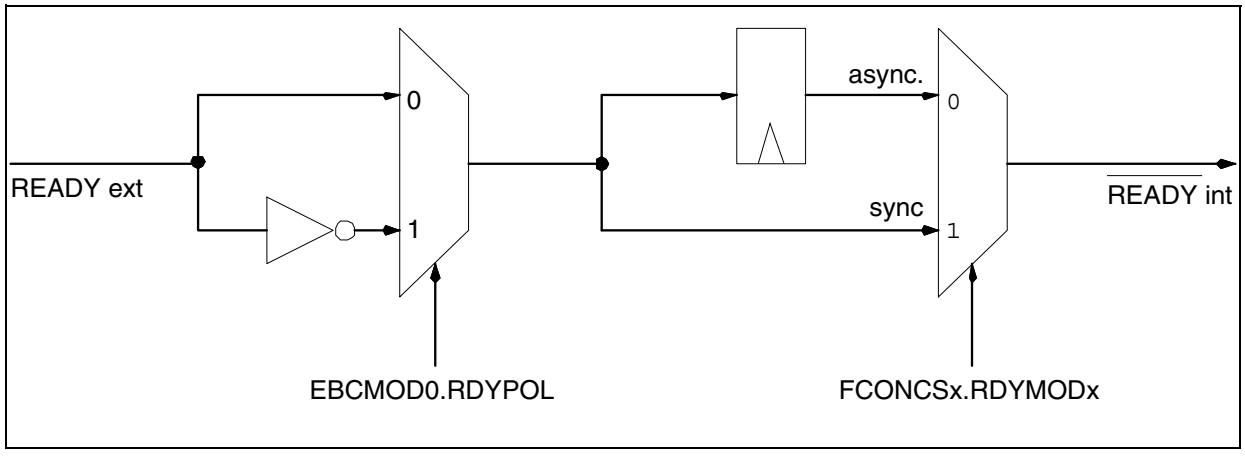


Figure 6-6 External to internal READY conversion

6.3.6.2 The Synchronous/Asynchronous READY

The **synchronous** READY provides the fastest bus cycles, but requires setup and hold times to be met. The CLKOUT signal should be enabled and may be used by the peripheral logic to control the READY timing in this case.

The **asynchronous** READY is less restrictive, but requires one additional wait state caused by the internal synchronization. As the asynchronous READY is sampled earlier programmed wait states may be necessary to provide proper bus cycles

A READY signal (especially asynchronous READY) that has been activated by an external device may be deactivated in response to the trailing (rising) edge of the respective command ($\overline{RD}$ or $\overline{WR}$).

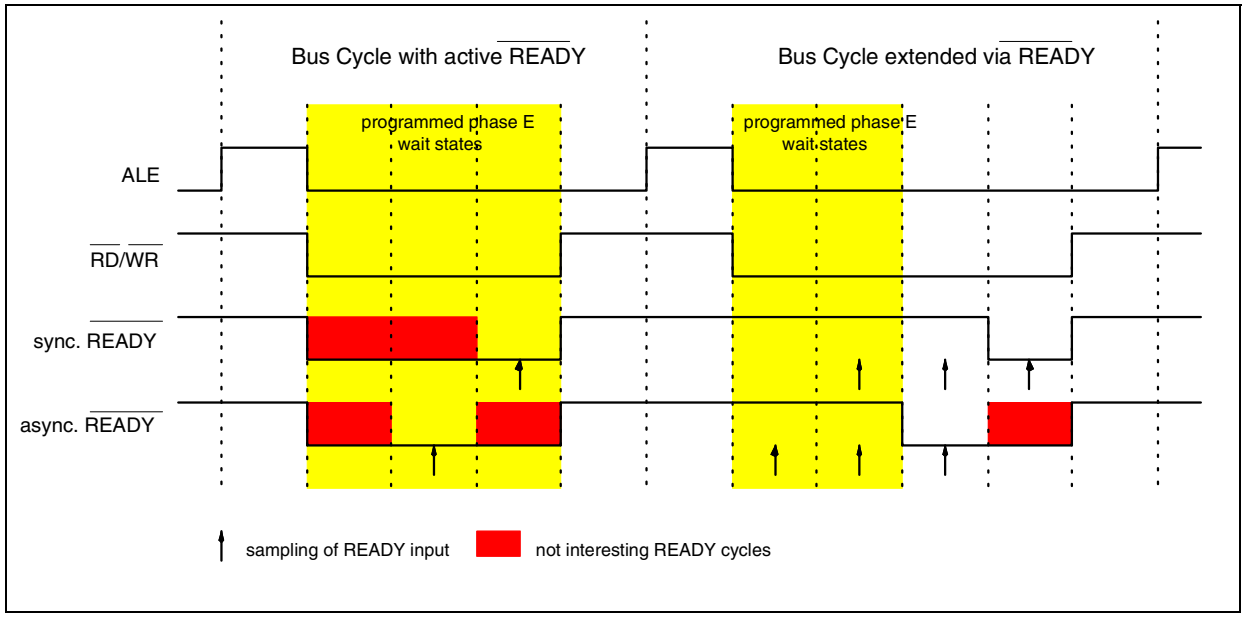


Figure 6-7 Ready controlled bus cycles

6.3.6.3 Combining the READY function with predefined wait states

Typically an external wait state or READY control logic takes a while to generate the READY signal when a cycle was started. After a predefined number of clock cycles the C166S V2 will start checking its READY line to determine the end of the bus cycle.

When using the READY function with so-called ‘normally-ready’ peripherals, it may lead to erroneous bus cycles, if the READY line is sampled too early. These peripherals pull their READY output active, while they are idle. When they are accessed, they drive READY inactive until the bus cycle is complete, then drive it active again. If, however, the peripheral drives READY inactive a little late, after the first sample point of the C166S V2, the controller samples an active READY and terminates the current bus cycle too early. By inserting predefined wait states the first READY sample point can be shifted to a time, where the peripheral has safely controlled the READY line.

6.3.7 EBC Idle State

When the external bus interface is enabled, but no external access is currently executed, the EBC is idle. As long as only internal resources (from a CPU point of view) like RAM, peripherals or registers, etc. are used, the external bus interface remains unchanged (see **Table 6-3**). The external control signals ($\overline{RD}$ and $\overline{WR}$ or $\overline{WRL}$/$\overline{WRH}$ if enabled) remain inactive (high).

Table 6-3 Status of the External Bus Interface during EBC Idle State

Pins	Internal accesses only
AD15 to AD0	Tristate (floating)
A15 to A0	Undefined address (if used for the bus interface)
A23 to A16	Undefined segment address (on selected pins)
$\overline{CS7}$ to $\overline{CS0}$	Inactive (high)
$\overline{BHE}$	Level corresponding to last external access
ALE	Inactive (low)
$\overline{RD}$	Inactive (high)
$\overline{WR}$/$\overline{WRL}$	Inactive (high)
$\overline{WRH}$	Inactive (high)

6.4 Multi Master Systems

6.4.1 External Bus Arbitration

The C166S V2 supports multi master systems on the external bus by its external bus arbitration. This bus arbitration allows an external master to request the C166S V2's bus. The C166S V2 will release the external bus and will float the data and address bus lines and force the control signals via pull ups/downs to their inactive state.

6.4.1.1 Initialization of Arbitration

During reset all arbitration pins are tristate, except pin $\overline{BREQ}$ which is pulled inactive. After reset the C166S V2 EBC always starts in 'init mode' where the external bus is available but no arbitration is enabled. All arbitration pins are ignored in this state. Other to the external bus connected C166S V2 EBCs assume to have the bus also, so potential bus conflicts are not resolved. For a multimaster system the arbitration should be initialized first before starting any bus access. The EBC can either be chosen as arbitration master or as arbitration slave by programming the EBCMOD0 bit SLAVE. The selected mode and the arbitration gets active by the first setting of the HLDEN bit inside the CPUs PSW register. Afterwards a change of the slave/master mode is not possible

without resetting the device. Of course for arbitration the dedicated pins have to be activated by setting EBCMOD0.ARBEN.

6.4.1.2 Arbitration Master Scheme

If the C166S V2 EBC is configured as arbitration master, it is default owner of the external bus, controls the arbitration protocol and drives the bus also during idle phases with no bus requests. To perform the arbitration handshake a HOLD input allows the request of the external bus from the arbitration master. When the arbitration master hands over the bus to the requester this is signaled by driving the hold acknowledge pin HLDA low, which remains at this level until the arbitration slave frees the bus by releasing its request on the HOLD input. If the arbitration master is not the owner of the bus it treats the external bus interface as follows:

- Address and data bus(es) float to tristate
- Command lines are pulled high by internal pull-up devices (RD, WR/WRL, BHE/WRH)
- Address latch control line ALE is pulled low by an internal pull-down device
- CSx outputs are pulled high by internal pull-up devices.

In this state the arbitration slave can take over the bus.

If the arbitration master requires the bus again, it can request the bus via the bus request signal BREQ. As soon as the arbitration master regains the bus it releases the BREQ signal and drives HLDA to high.

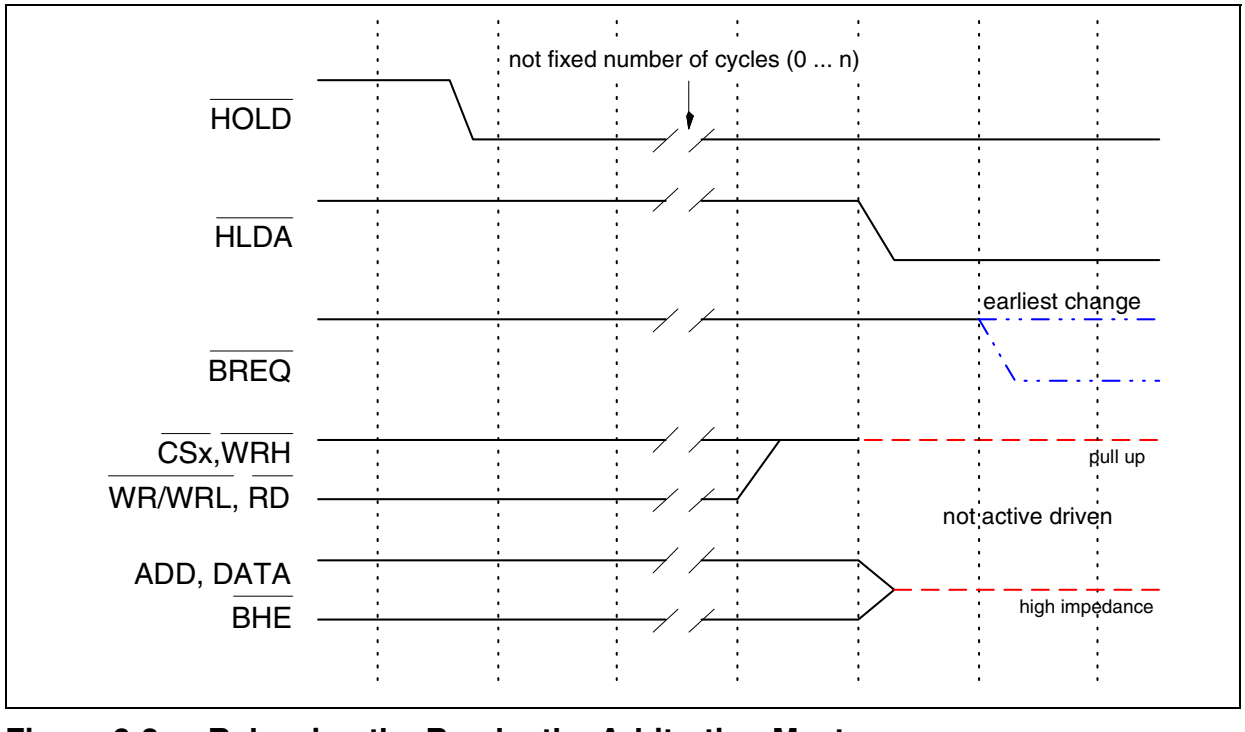
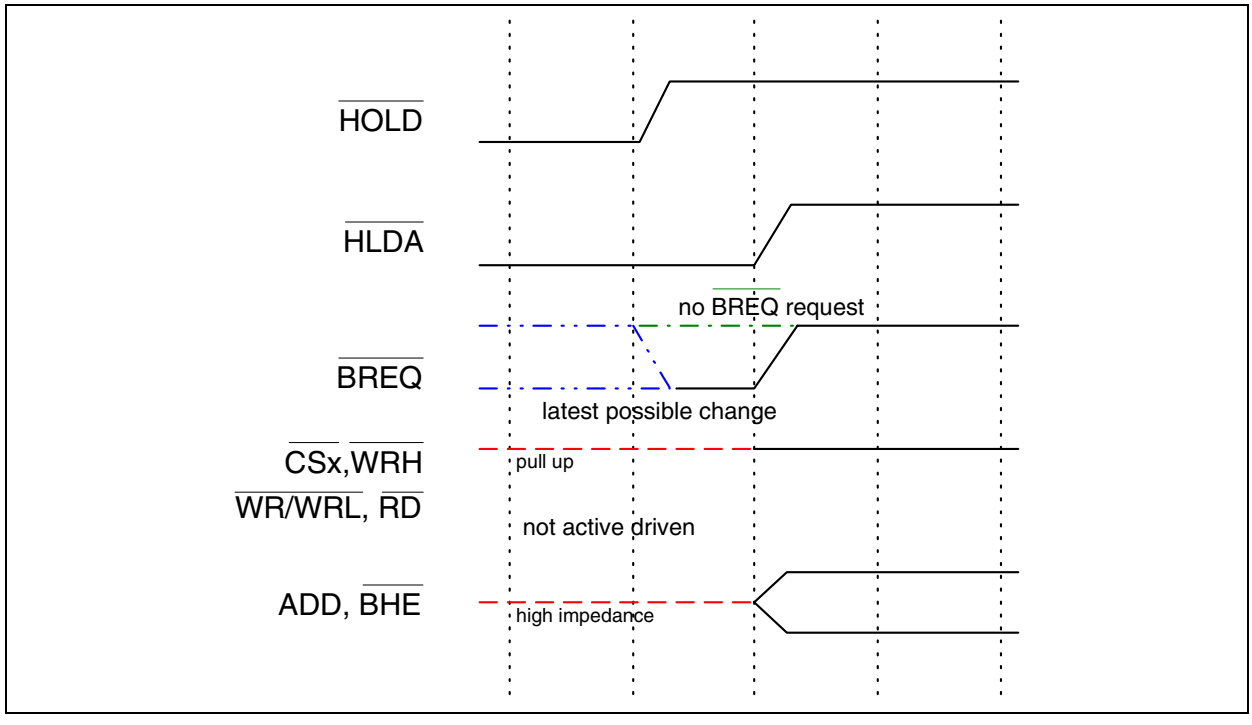


Figure 6-8 Releasing the Bus by the Arbitration Master

Note: The figure above shows the first possibility for BREQ to get active. The C166S V2 will complete the currently running bus cycle before granting the external bus as indicated by the broken lines.

Figure 6-9 Regaining the Bus by the Arbitration Master



Note: The falling BREQ edge shows the last chance for BREQ to trigger the indicated regain-sequence. Even if BREQ is activated earlier the regain-sequence is initiated by HOLD going high. Please note that HOLD may also be deactivated without the C166S V2 requesting the bus.

6.4.1.3 Arbitration Slave Scheme

If the C166S V2 EBC is configured as arbitration slave it is by default not owner of the external bus and has to request the bus first. As long as it has not finished all its queued requests and the arbitration master is not requesting the bus the arbitration slave stays owner of the bus. For the description of the signal handling of the handshake see **Chapter 6.4.1.2**. For the arbitration slave the hold acknowledge pin HLDA is configured as input.

6.4.1.4 Locking the Bus

If an application in a multimaster system requires a sequence of undisturbed bus access it has the possibility (independently of being arbitration slave or master) to lock[1)] the bus

[1)] It is not allowed to lock the bus by resetting the EBCMOD0.ARBEN bit, as this can lead to bus conflicts.

by setting the PSW bit HLDEN to ‘0’. In this case the looked C166S V2 EBC will not answer to HOLD requests from other external bus master until HLDEN is set to ‘1’ again. Of course a looked bus master not owning the bus can request the external bus. If a master and a slave are requesting the external bus at the same time for several accesses, they toggle the ownership after each access cycle if the bus is not locked.

6.4.2 Connecting Multimaster Systems

Two C166S V2s where one is configured as arbitration master and the other as arbitration slave can be connected directly together as shown in **Figure 6-10**. As both EBCs assume after reset to own the external bus, the ‘slave’ CPU has to be released from reset and initialized first, before starting the ‘master’ CPU. The other way is to start both systems at the same time but then both EBC must be configured and the PSW.HLDEN bits set before the first external bus request.

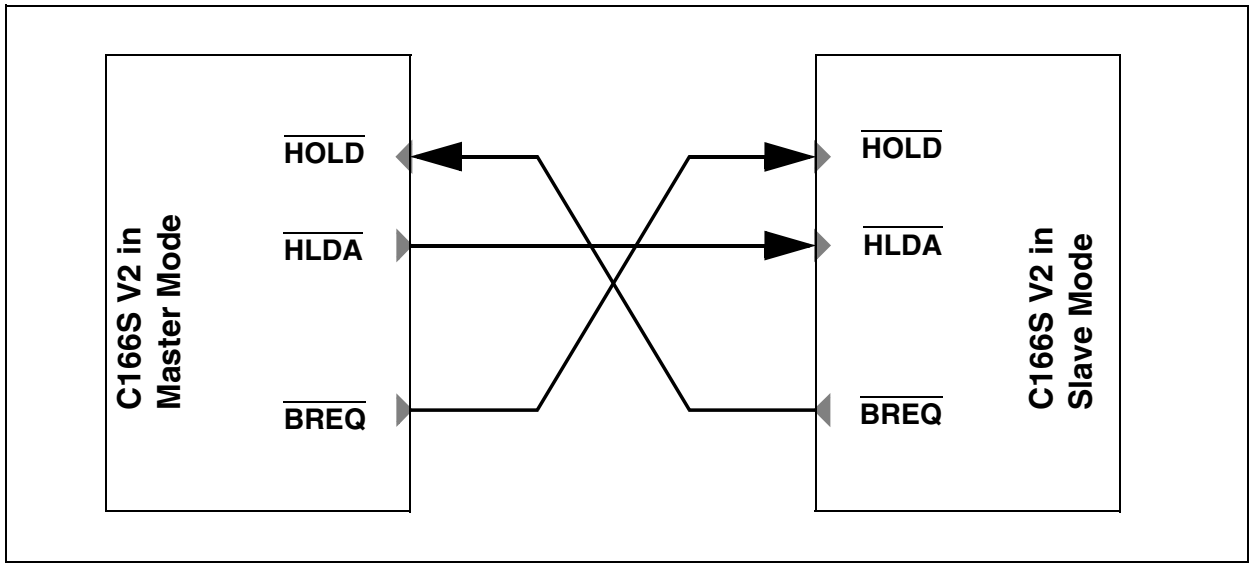


Figure 6-10 Connecting two C166S V2s using Master/Slave Arbitration

When more than two C166S V2s or other compatible bus masters are connected together additional interconnection/arbitration logic is required. In this case the slave/master selection has to be done according to the introduced logic.

6.5 Fastest possible external access

The following four figures show the principal possible fastest access type for the EBC.

Figure 6-11 Fastest Read Cycle Demultiplexed Bus

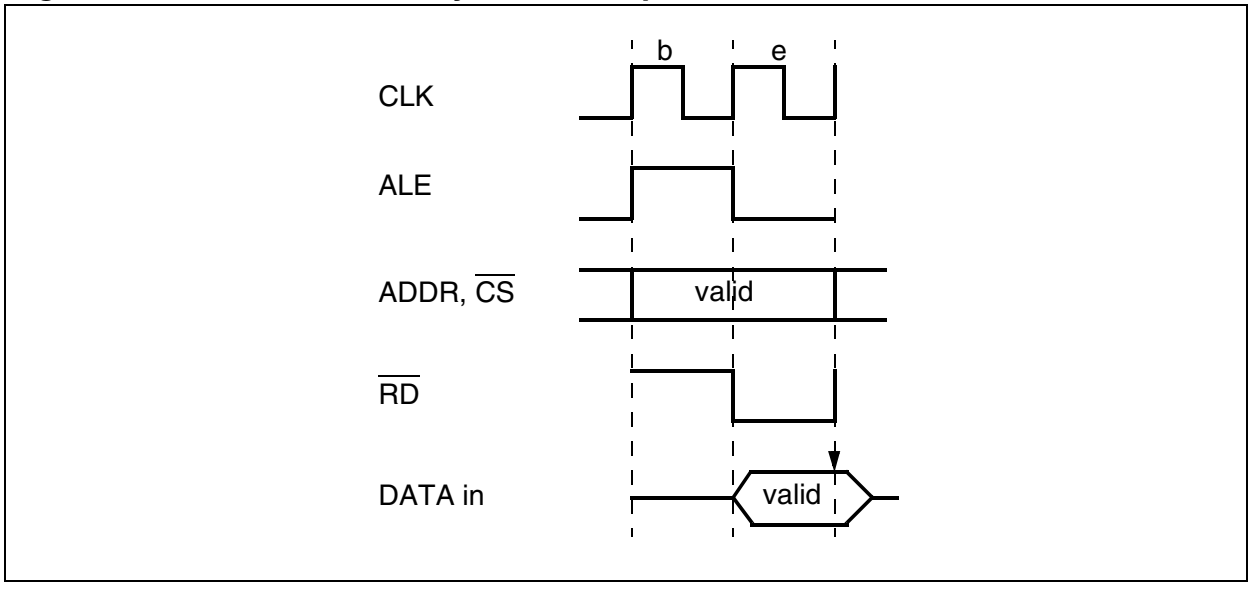


Figure 6-12 Fastest Write Cycle Demultiplexed Bus

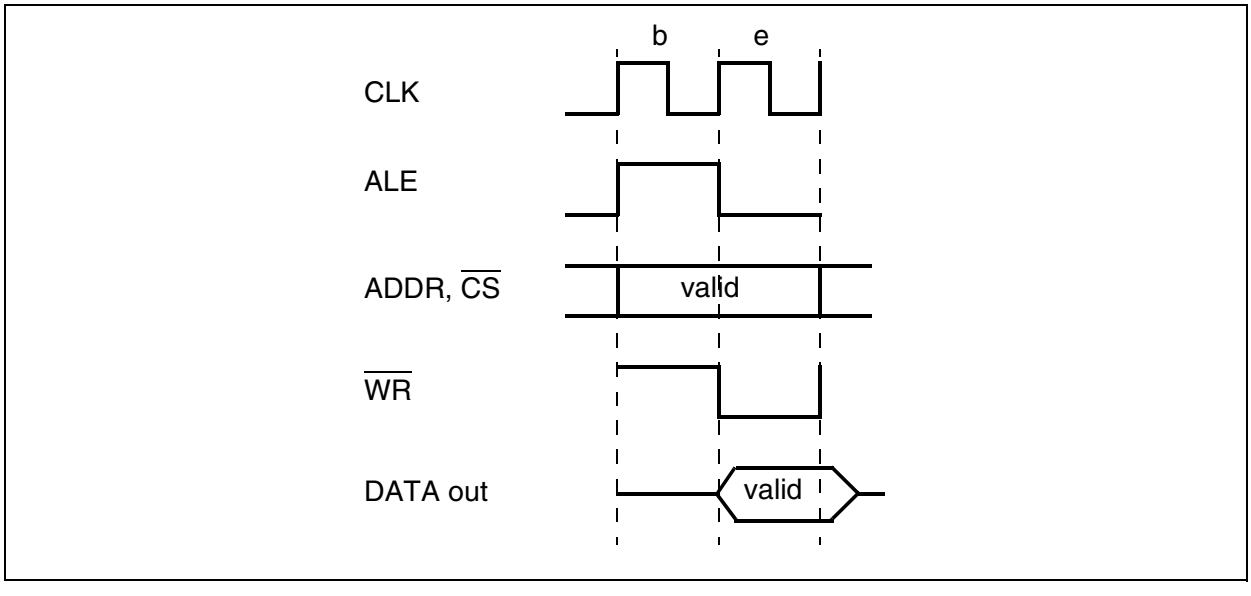


Figure 6-13 Fastest Read Cycle Multiplexed Bus

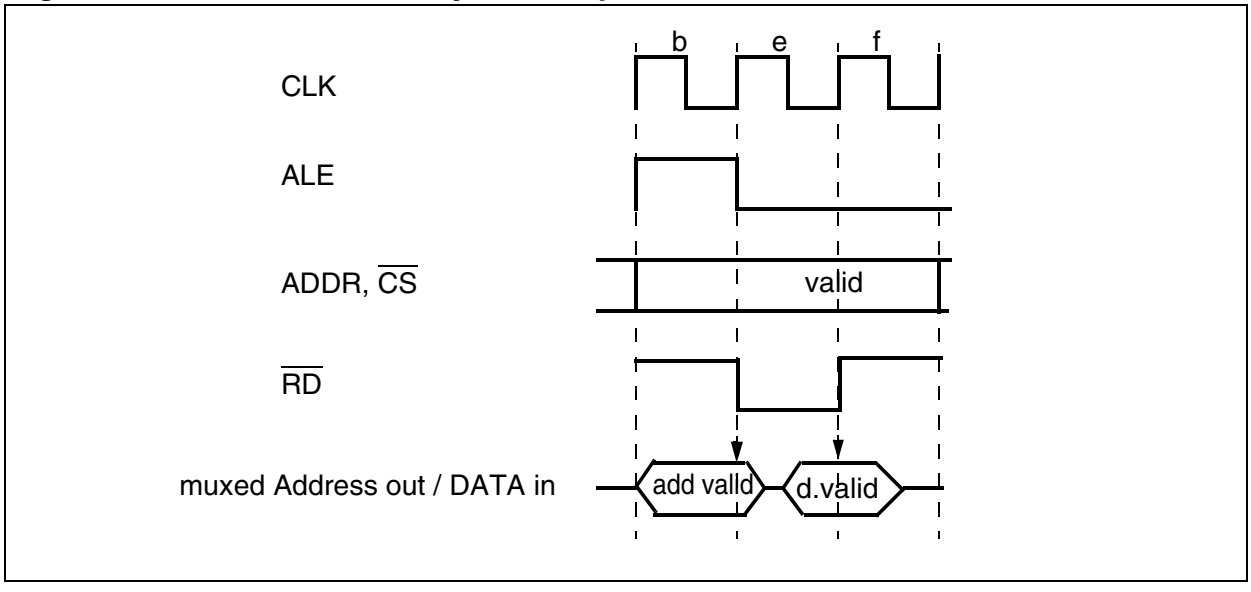
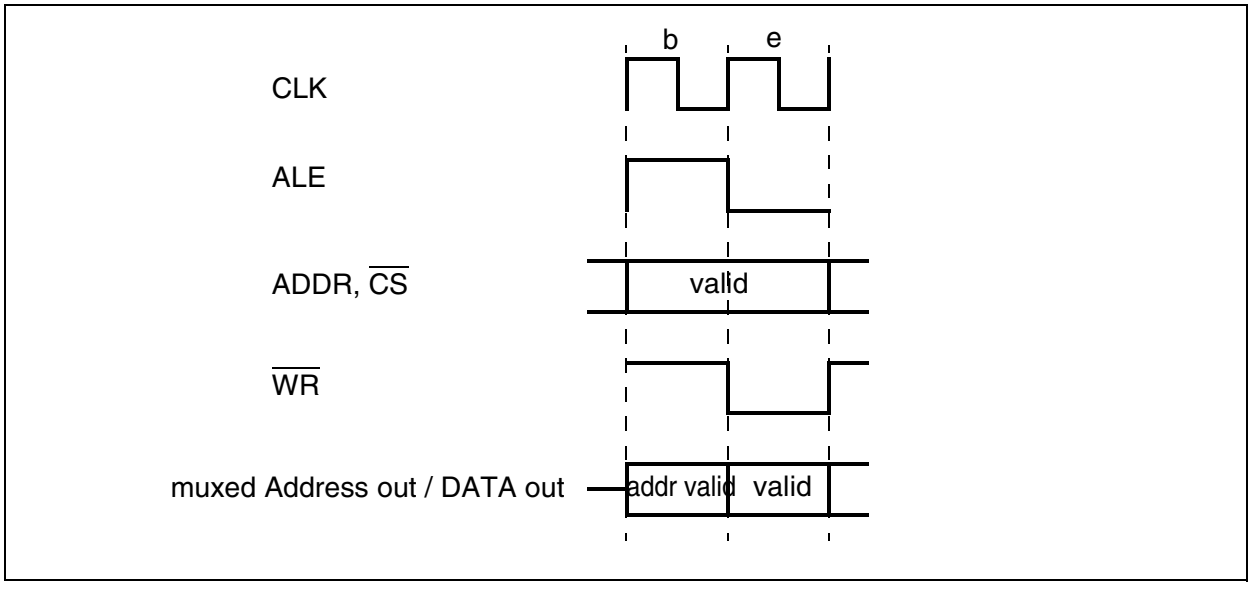


Figure 6-14 Fastest Write Cycle Multiplexed Bus



7 Instruction Set

7.1 Short Instruction Summary

The following compressed cross-reference tables quickly identify specific instructions and provide basic information about them Two ordering schemes are included:

The first table (two pages) is a compressed cross-reference table that quickly associates specific hexadecimal opcodes with the corresponding mnemonics.

The second table lists instructions by their mnemonic and identifies the addressing modes that may be used with the specific instructions and indicates the instruction length for the selected addressing mode. This reference helps to optimize instruction sequences in terms of code size and/or execution time.

Description Levels

In the following sections the instructions are compiled according to different criteria in order to provide different levels of precision:

- **Cross Reference Tables** summarize all instructions in condensed tables
- **The Instruction Set Summary** groups the individual instructions into functional groups
- **The Opcode Table** references the instructions by their hexadecimal opcode

	0x	1x	2x	3x	4x	5x	6x	7x
x0	ADD	ADDC	SUB	SUBC	CMP	XOR	AND	OR
x1	ADDB	ADDCB	SUBB	SUBCB	CMPB	XORB	ANDB	ORB
x2	ADD	ADDC	SUB	SUBC	CMP	XOR	AND	OR
x3	ADDB	ADDCB	SUBB	SUBCB	CMPB	XORB	ANDB	ORB
x4	ADD	ADDC	SUB	SUBC	-	XOR	AND	OR
x5	ADDB	ADDCB	SUBB	SUBCB	-	XORB	ANDB	ORB
x6	ADD	ADDC	SUB	SUBC	CMP	XOR	AND	OR
x7	ADDB	ADDCB	SUBB	SUBCB	CMPB	XORB	ANDB	ORB
x8	ADD	ADDC	SUB	SUBC	CMP	XOR	AND	OR
x9	ADDB	ADDCB	SUBB	SUBCB	CMPB	XORB	ANDB	ORB
xA	BFLDL	BFLDH	BCMP	BMOVN	BMOV	BOR	BAND	BXOR
xB	MUL	MULU	PRIOR	-	DIV	DIVU	DIVL	DIVLU
xC	ROL	ROL	ROR	ROR	SHL	SHL	SHR	SHR
xD	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR
xE	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR
xF	BSET	BSET	BSET	BSET	BSET	BSET	BSET	BSET

	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	CMPI1	CMPI2	CMPD1	CMPD2	MOVBZ	MOVBS	MOV	MOV
x1	NEG	CPL	NEGB	CPLB	-	AT/ EXTR	MOVB	MOVB
x2	CMPI1	CMPI2	CMPD1	CMPD2	MOVBZ	MOVBS	PCALL	MOV
x3	CoXXX	CoXXX	CoXXX	Co STORE	Co STORE	CMOV	-	MOVB
x4	MOV	MOV	MOVB	MOVB	MOV	MOV	MOVB	MOVB
x5	ENWDT	-	DIS WDT	EINIT	MOVBZ	MOVBS	-	-
x6	CMPI1	CMPI2	CMPD1	CMPD2	SCXT	SCXT	MOV	MOV
x7	IDLE	PWRDN	SRV WDT	SRST	-	EXTP/S/ R	MOVB	MOVB
x8	MOV	MOV	MOV	MOV	MOV	MOV	MOV	-
x9	MOVB	MOVB	MOVB	MOVB	MOVB	MOVB	MOVB	-
xA	JB	JNB	JBC	JNBS	CALLA	CALLS	JMPA	JMPS
xB	-	TRAP	CALLI	CALLR	RET	RETS	RETP	RETI
xC	SBRK	JMPI	ASHR	ASHR	NOP	EXTP/S/ R	PUSH	POP
xD	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR
xE	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR
xF	BSET	BSET	BSET	BSET	BSET	BSET	BSET	BSET

7.2 Instruction Set Summary

This section summarizes the instructions and lists them by functional class. This enables quick identification of the right instruction(s) for a specific function.

The following notes apply to this summary:

Data Addressing Modes

Rw:	–	Word GPR (R0, R1, … , R15)
Rb:	–	Byte GPR (RL0, RH0, …, RL7, RH7)
IDX:	–	Address Pointer IDX (IDX0, IDX1)
QX:	–	Address Offset Register QX (QX0, QX1)
QR:	–	Address Offset Register QR (QR0, QR1)
reg:	–	SFR or GPR (in case of a byte operation on an SFR, only the low byte can be accessed via ‘reg’)
mem:	–	Direct word or byte memory location
[…]:	–	Indirect word or byte memory location (Any word GPR can be used as indirect address pointer, except for the arithmetic, logical and compare instructions, where only R0 to R3 are allowed)
bitaddr:	–	Direct bit in the bit-addressable memory area
bitoff:	–	Direct word in the bit-addressable memory area
#data:	–	Immediate constant (The number of significant bits which can be specified by the user is represented by the respective appendix ’x’)
#mask8:	–	Immediate 8-bit mask used for bit-field modifications

Table 7-1 shows the various combinations of pointer post-modification for the addressing modes of the CoXXX instructions. The symbols "[Rw_n*]" and "[IDX_i*]" will be used to refer to these addressing modes.

Table 7-1 Pointer Post-Modification Combinations for IDXi and Rwn

Symbol	Mnemonic	Address Pointer Operation
"[IDX_i⊗]" stands for	[IDX_i]	(IDX_i) ← (IDX_i) (no-operation)
	[IDX_i+]	(IDX_i) ← (IDX_i) +2 (i=0,1)
	[IDX_i -]	(IDX_i) ← (IDX_i) -2 (i=0,1)
	[IDX_i + QX_j]	(IDX_i) ← (IDX_i) + (QX_j) (i, j =0,1)
	[IDX_i - QX_j]	(IDX_i) ← (IDX_i) - (QX_j) (i, j =0,1)
"[Rw_n⊗]" stands for	[Rwn]	(Rwn) ← (Rwn) (no-operation)
	[Rwn+]	(Rwn) ← (Rwn) +2 (n=0-15)
	[Rwn-]	(Rwn) ← (Rwn) -2 (n=0-15)
	[Rwn+QR_j]	(Rwn) ← (Rwn) + (QR_j) (n=0-15;j =0,1)
	[Rwn - QR_j]	(Rwn) ← (Rwn) - (QR_j) (n=0-15; j =0,1)

Multiply and Divide Operations

The MDL and MDH registers are implicit source and/or destination operands of the multiply and divide instructions.

Branch Target Addressing Modes

caddr: – Direct 16-bit jump target address (Updates the Instruction Pointer)

seg: – Direct 2-bit segment address
(Updates the Code Segment Pointer)

rel: – Signed 8-bit jump target word offset address relative to the Instruction Pointer of the following instruction

#trap7: – Immediate 7-bit trap or interrupt number.

Extension Operations

The EXT* instructions override the standard DPP addressing scheme:

#pag10: – Immediate 10-bit page address.

#seg8: – Immediate 8-bit segment address.

Branch Condition Codes

cc: Symbolically specifiable condition codes

cc_UC	–Unconditional
cc_Z	–Zero
cc_NZ	–Not Zero
cc_V	–Overflow
cc_NV	–No Overflow
cc_N	–Negative
cc_NN	–Not Negative
cc_C	–Carry
cc_NC	–No Carry
cc_EQ	–Equal
cc_NE	–Not Equal
cc_ULT	–Unsigned Less Than
cc_ULE	–Unsigned Less Than or Equal
cc_UGE	–Unsigned Greater Than or Equal
cc_UGT	–Unsigned Greater Than
cc_SLE	–Signed Less Than or Equal
cc_SGE	–Signed Greater Than or Equal
cc_SGT	–Signed Greater Than
cc_NET	–Not Equal and Not End-of-Table
cc_nusr0	–USR-bit 0 is cleared[1)]
cc_nusr1	–USR-bit 1 is cleared[1)]
cc_usr0	–USR-bit 0 is set[1)]
cc_usr1	–USR-bit 1 is set[1)]

1) Only usable with the JMPA and CALLA instructions

Mnemonic	Addressing Modes		Bytes
ADD[B] ADDC[B] AND[B] OR[B] SUB[B] SUBC[B] XOR[B]	Rwn Rwn Rwn Rwn reg reg mem	Rwm [1] [Rwi] [1] [Rwi+] [1] #data3 [1] #data16 mem reg	2 2 2 2 4 4 4
ASHR ROL / ROR SHL / SHR	Rwn Rwn	Rwm #data4	2 2
BAND BCMP BMOV BMOVN BOR / BXOR	bitaddrZ.z	bitaddrQ.q	4
BCLR BSET	bitaddrQ.q		2
BFLDH BFLDL	bitoffQ	#mask8 #data8	4
MOV[B]	Rwn Rwn Rwn Rwn [Rwm] [-Rwm] [Rwn] [Rwn+] [Rwn] reg Rwn [Rwm+#d16] [Rwn] mem reg mem	Rwm [1] #data4 [1] [Rwm] [1] [Rwm+] [1] Rwn [1] Rwn [1] [Rwm] [Rwm] [Rwm+] #data16 [2] [Rwm+#d16] [1] Rwn [1] mem [Rwn] mem reg	2 2 2 2 2 2 2 2 2 4 4 4 4 4 4 4
MOVBS MOVBZ	Rwn reg mem	Rbm mem reg	2 4 4
EXTS EXTSR	Rwm #seg	#irang2 #irang2	2 4
NOP RET RETI RETS SBRK	-		2

Mnemonic	Addressing Modes		Bytes
CPL[B] NEG[B]	Rwn [1]		2
DIV DIVL DIVLU DIVU	Rwn		2
MUL MULU	Rwn	Rwm	2
CMPD1/2 CMPI1/2	Rwn Rwn Rwn	#data4 #data16 mem	2 4 4
CMP[B]	Rwn Rwn Rwn Rwn reg reg	Rwm [1] [Rwi] [1] [Rwi+] [1] #data3 [1] #data16 [2] mem	2 2 2 2 4 4
CALLA JMPA	cc	caddr	4
CALLI JMPI	cc	[Rwn]	2
CALLS JMPS	seg	caddr	4
CALLR	rel		2
JMPR	cc	rel	2
JB JBC JNB JNBS	bitaddrQ.q	rel	4
PCALL	reg	caddr	4
POP PUSH RETP	reg		2
SCXT	reg reg	#data16 mem	4 4
PRIOR	Rwn	Rwm	2
TRAP	#trap7		2
ATOMIC EXTR	#irang2		2
EXTP EXTPR	Rwm #pag	#irang2 #irang2	2 4
SRST/IDLE PWRDN SRVWDT DISWDT ENWDT EINIT	-		4

[1] Byte oriented instructions (suffix ‘B’) use Rb instead of Rw (not with [Rwn]!).
[2] Byte oriented instructions (suffix ‘B’) use #data8 instead of #data16.

Instruction Set Summary

Mnemonic		Description	Bytes
Arithmetic Operations			
ADD	Rw, Rw	Add direct word GPR to direct GPR	2
ADD	Rw, [Rw]	Add indirect word memory to direct GPR	2
ADD	Rw, [Rw +]	Add indirect word memory to direct GPR and post-increment source pointer by 2	2
ADD	Rw, #data3	Add immediate word data to direct GPR	2
ADD	reg, #data16	Add immediate word data to direct register	4
ADD	reg, mem	Add direct word memory to direct register	4
ADD	mem, reg	Add direct word register to direct memory	4
ADDB	Rb, Rb	Add direct byte GPR to direct GPR	2
ADDB	Rb, [Rw]	Add indirect byte memory to direct GPR	2
ADDB	Rb, [Rw +]	Add indirect byte memory to direct GPR and post-increment source pointer by 1	2
ADDB	Rb, #data3	Add immediate byte data to direct GPR	2
ADDB	reg, #data8	Add immediate byte data to direct register	4
ADDB	reg, mem	Add direct byte memory to direct register	4
ADDB	mem, reg	Add direct byte register to direct memory	4
ADDC	Rw, Rw	Add direct word GPR to direct GPR with Carry	2
ADDC	Rw, [Rw]	Add indirect word memory to direct GPR with Carry	2
ADDC	Rw, [Rw +]	Add indirect word memory to direct GPR with Carry and post-increment source pointer by 2	2
ADDC	Rw, #data3	Add immediate word data to direct GPR with Carry	2
ADDC	reg, #data16	Add immediate word data to direct register with Carry	4
ADDC	reg, mem	Add direct word memory to direct register with Carry	4
ADDC	mem, reg	Add direct word register to direct memory with Carry	4
ADDCB	Rb, Rb	Add direct byte GPR to direct GPR with Carry	2
ADDCB	Rb, [Rw]	Add indirect byte memory to direct GPR with Carry	2
ADDCB	Rb, [Rw +]	Add indirect byte memory to direct GPR with Carry and post-increment source pointer by 1	2
ADDCB	Rb, #data3	Add immediate byte data to direct GPR with Carry	2
ADDCB	reg, #data8	Add immediate byte data to direct register with Carry	4
ADDCB	reg, mem	Add direct byte memory to direct register with Carry	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Arithmetic Operations (cont'd)			
ADDCB	mem, reg	Add direct byte register to direct memory with Carry	4
SUB	Rw, Rw	Subtract direct word GPR from direct GPR	2
SUB	Rw, [Rw]	Subtract indirect word memory from direct GPR	2
SUB	Rw, [Rw +]	Subtract indirect word memory from direct GPR and post-increment source pointer by 2	2
SUB	Rw, #data3	Subtract immediate word data from direct GPR	2
SUB	reg, #data16	Subtract immediate word data from direct register	4
SUB	reg, mem	Subtract direct word memory from direct register	4
SUB	mem, reg	Subtract direct word register from direct memory	4
SUBB	Rb, Rb	Subtract direct byte GPR from direct GPR	2
SUBB	Rb, [Rw]	Subtract indirect byte memory from direct GPR	2
SUBB	Rb, [Rw +]	Subtract indirect byte memory from direct GPR and post-increment source pointer by 1	2
SUBB	Rb, #data3	Subtract immediate byte data from direct GPR	2
SUBB	reg, #data8	Subtract immediate byte data from direct register	4
SUBB	reg, mem	Subtract direct byte memory from direct register	4
SUBB	mem, reg	Subtract direct byte register from direct memory	4
SUBC	Rw, Rw	Subtract direct word GPR from direct GPR with Carry	2
SUBC	Rw, [Rw]	Subtract indirect word memory from direct GPR with Carry	2
SUBC	Rw, [Rw +]	Subtract indirect word memory from direct GPR with Carry and post-increment source pointer by 2	2
SUBC	Rw, #data3	Subtract immediate word data from direct GPR with Carry	2
SUBC	reg, #data16	Subtract immediate word data from direct register with Carry	4
SUBC	reg, mem	Subtract direct word memory from direct register with Carry	4
SUBC	mem, reg	Subtract direct word register from direct memory with Carry	4
SUBCB	Rb, Rb	Subtract direct byte GPR from direct GPR with Carry	2
SUBCB	Rb, [Rw]	Subtract indirect byte memory from direct GPR with Carry	2
SUBCB	Rb, [Rw +]	Subtract indirect byte memory from direct GPR with Carry and post-increment source pointer by 1	2
SUBCB	Rb, #data3	Subtract immediate byte data from direct GPR with Carry	2
SUBCB	reg, #data8	Subtract immediate byte data from direct register with Carry	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Arithmetic Operations (cont'd)			
SUBCB	reg, mem	Subtract direct byte memory from direct register with Carry	4
SUBCB	mem, reg	Subtract direct byte register from direct memory with Carry	4
MUL	Rw, Rw	Signed multiply direct GPR by direct GPR (16-16-bit)	2
MULU	Rw, Rw	Unsigned multiply direct GPR by direct GPR (16-16-bit)	2
DIV	Rw	Signed divide register MDL by direct GPR (16-/16-bit)	2
DIVL	Rw	Signed long divide register MD by direct GPR (32-/16-bit)	2
DIVLU	Rw	Unsigned long divide register MD by direct GPR (32-/16-bit)	2
DIVU	Rw	Unsigned divide register MDL by direct GPR (16-/16-bit)	2
CPL	Rw	Complement direct word GPR	2
CPLB	Rb	Complement direct byte GPR	2
NEG	Rw	Negate direct word GPR	2
NEGB	Rb	Negate direct byte GPR	2
Logical Instructions			
AND	Rw, Rw	Bitwise AND direct word GPR with direct GPR	2
AND	Rw, [Rw]	Bitwise AND indirect word memory with direct GPR	2
AND	Rw, [Rw +]	Bitwise AND indirect word memory with direct GPR and post-increment source pointer by 2	2
AND	Rw, #data3	Bitwise AND immediate word data with direct GPR	2
AND	reg, #data16	Bitwise AND immediate word data with direct register	4
AND	reg, mem	Bitwise AND direct word memory with direct register	4
AND	mem, reg	Bitwise AND direct word register with direct memory	4
ANDB	Rb, Rb	Bitwise AND direct byte GPR with direct GPR	2
ANDB	Rb, [Rw]	Bitwise AND indirect byte memory with direct GPR	2
ANDB	Rb, [Rw +]	Bitwise AND indirect byte memory with direct GPR and post-increment source pointer by 1	2
ANDB	Rb, #data3	Bitwise AND immediate byte data with direct GPR	2
ANDB	reg, #data8	Bitwise AND immediate byte data with direct register	4
ANDB	reg, mem	Bitwise AND direct byte memory with direct register	4
ANDB	mem, reg	Bitwise AND direct byte register with direct memory	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Logical Instructions (cont'd)			
OR	Rw, Rw	Bitwise OR direct word GPR with direct GPR	2
OR	Rw, [Rw]	Bitwise OR indirect word memory with direct GPR	2
OR	Rw, [Rw +]	Bitwise OR indirect word memory with direct GPR and post-increment source pointer by 2	2
OR	Rw, #data3	Bitwise OR immediate word data with direct GPR	2
OR	reg, #data16	Bitwise OR immediate word data with direct register	4
OR	reg, mem	Bitwise OR direct word memory with direct register	4
OR	mem, reg	Bitwise OR direct word register with direct memory	4
ORB	Rb, Rb	Bitwise OR direct byte GPR with direct GPR	2
ORB	Rb, [Rw]	Bitwise OR indirect byte memory with direct GPR	2
ORB	Rb, [Rw +]	Bitwise OR indirect byte memory with direct GPR and post-increment source pointer by 1	2
ORB	Rb, #data3	Bitwise OR immediate byte data with direct GPR	2
ORB	reg, #data8	Bitwise OR immediate byte data with direct register	4
ORB	reg, mem	Bitwise OR direct byte memory with direct register	4
ORB	mem, reg	Bitwise OR direct byte register with direct memory	4
XOR	Rw, Rw	Bitwise XOR direct word GPR with direct GPR	2
XOR	Rw, [Rw]	Bitwise XOR indirect word memory with direct GPR	2
XOR	Rw, [Rw +]	Bitwise XOR indirect word memory with direct GPR and post-increment source pointer by 2	2
XOR	Rw, #data3	Bitwise XOR immediate word data with direct GPR	2
XOR	reg, #data16	Bitwise XOR immediate word data with direct register	4
XOR	reg, mem	Bitwise XOR direct word memory with direct register	4
XOR	mem, reg	Bitwise XOR direct word register with direct memory	4
XORB	Rb, Rb	Bitwise XOR direct byte GPR with direct GPR	2
XORB	Rb, [Rw]	Bitwise XOR indirect byte memory with direct GPR	2
XORB	Rb, [Rw +]	Bitwise XOR indirect byte memory with direct GPR and post-increment source pointer by 1	2
XORB	Rb, #data3	Bitwise XOR immediate byte data with direct GPR	2
XORB	reg, #data8	Bitwise XOR immediate byte data with direct register	4
XORB	reg, mem	Bitwise XOR direct byte memory with direct register	4
XORB	mem, reg	Bitwise XOR direct byte register with direct memory	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Boolean Bit Manipulation Operations			
BCLR	bitaddr	Clear direct bit	2
BSET	bitaddr	Set direct bit	2
BMOV	bitaddr, bitaddr	Move direct bit to direct bit	4
BMOVN	bitaddr, bitaddr	Move negated direct bit to direct bit	4
BAND	bitaddr, bitaddr	AND direct bit with direct bit	4
BOR	bitaddr, bitaddr	OR direct bit with direct bit	4
BXOR	bitaddr, bitaddr	XOR direct bit with direct bit	4
BCMP	bitaddr, bitaddr	Compare direct bit to direct bit	4
BFLDH	bitoff, #mask8, #data8	Bitwise modify masked high byte of bit-addressable direct word memory with immediate data	4
BFLDL	bitoff, #mask8, #data8	Bitwise modify masked low byte of bit-addressable direct word memory with immediate data	4
CMP	Rw, Rw	Compare direct word GPR to direct GPR	2
CMP	Rw, [Rw]	Compare indirect word memory to direct GPR	2
CMP	Rw, [Rw +]	Compare indirect word memory to direct GPR and post-increment source pointer by 2	2
CMP	Rw, #data3	Compare immediate word data to direct GPR	2
CMP	reg, #data16	Compare immediate word data to direct register	4
CMP	reg, mem	Compare direct word memory to direct register	4
CMPB	Rb, Rb	Compare direct byte GPR to direct GPR	2
CMPB	Rb, [Rw]	Compare indirect byte memory to direct GPR	2
CMPB	Rb, [Rw +]	Compare indirect byte memory to direct GPR and post-increment source pointer by 1	2
CMPB	Rb, #data3	Compare immediate byte data to direct GPR	2
CMPB	reg, #data8	Compare immediate byte data to direct register	4
CMPB	reg, mem	Compare direct byte memory to direct register	4
Compare and Loop Control Instructions			
CMPD1	Rw, #data4	Compare immediate word data to direct GPR and decrement GPR by 1	2
CMPD1	Rw, #data16	Compare immediate word data to direct GPR and decrement GPR by 1	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Compare and Loop Control Instructions (cont'd)			
CMPD1	Rw, mem	Compare direct word memory to direct GPR and decrement GPR by 1	4
CMPD2	Rw, #data4	Compare immediate word data to direct GPR and decrement GPR by 2	2
CMPD2	Rw, #data16	Compare immediate word data to direct GPR and decrement GPR by 2	4
CMPD2	Rw, mem	Compare direct word memory to direct GPR and decrement GPR by 2	4
CMPI1	Rw, #data4	Compare immediate word data to direct GPR and increment GPR by 1	2
CMPI1	Rw, #data16	Compare immediate word data to direct GPR and increment GPR by 1	4
CMPI1	Rw, mem	Compare direct word memory to direct GPR and increment GPR by 1	4
CMPI2	Rw, #data4	Compare immediate word data to direct GPR and increment GPR by 2	2
CMPI2	Rw, #data16	Compare immediate word data to direct GPR and increment GPR by 2	4
CMPI2	Rw, mem	Compare direct word memory to direct GPR and increment GPR by 2	4
Prioritize Instruction			
PRIOR	Rw, Rw	Determine number of shift cycles to normalize direct word GPR and store result in direct word GPR	2
Shift and Rotate Instructions			
SHL	Rw, Rw	Shift left direct word GPR; number of shift cycles specified by direct GPR	2
SHL	Rw, #data4	Shift left direct word GPR; number of shift cycles specified by immediate data	2
SHR	Rw, Rw	Shift right direct word GPR; number of shift cycles specified by direct GPR	2

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Shift and Rotate Instructions (cont'd)			
SHR	Rw, #data4	Shift right direct word GPR; number of shift cycles specified by immediate data	2
ROL	Rw, Rw	Rotate left direct word GPR; number of shift cycles specified by direct GPR	2
ROL	Rw, #data4	Rotate left direct word GPR; number of shift cycles specified by immediate data	2
ROR	Rw, Rw	Rotate right direct word GPR; number of shift cycles specified by direct GPR	2
ROR	Rw, #data4	Rotate right direct word GPR; number of shift cycles specified by immediate data	2
ASHR	Rw, Rw	Arithmetic (sign bit) shift right direct word GPR; number of shift cycles specified by direct GPR	2
ASHR	Rw, #data4	Arithmetic (sign bit) shift right direct word GPR; number of shift cycles specified by immediate data	2
Data Movement			
MOV	Rw, Rw	Move direct word GPR to direct GPR	2
MOV	Rw, #data4	Move immediate word data to direct GPR	2
MOV	reg, #data16	Move immediate word data to direct register	4
MOV	Rw, [Rw]	Move indirect word memory to direct GPR	2
MOV	Rw, [Rw +]	Move indirect word memory to direct GPR and post-increment source pointer by 2	2
MOV	[Rw], Rw	Move direct word GPR to indirect memory	2
MOV	[-Rw], Rw	Pre-decrement destination pointer by 2 and move direct word GPR to indirect memory	2
MOV	[Rw], [Rw]	Move indirect word memory to indirect memory	2
MOV	[Rw +], [Rw]	Move indirect word memory to indirect memory and post-increment destination pointer by 2	2
MOV	[Rw], [Rw +]	Move indirect word memory to indirect memory and post-increment source pointer by 2	2
MOV	Rw, [Rw + #data16]	Move indirect word memory by base plus constant to direct GPR	4
MOV	[Rw + #data16], Rw	Move direct word GPR to indirect memory by base plus constant	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Data Movement (cont'd)			
MOV	[Rw], mem	Move direct word memory to indirect memory	4
MOV	mem, [Rw]	Move indirect word memory to direct memory	4
MOV	reg, mem	Move direct word memory to direct register	4
MOV	mem, reg	Move direct word register to direct memory	4
MOVB	Rb, Rb	Move direct byte GPR to direct GPR	2
MOVB	Rb, #data4	Move immediate byte data to direct GPR	2
MOVB	reg, #data8	Move immediate byte data to direct register	4
MOVB	Rb, [Rw]	Move indirect byte memory to direct GPR	2
MOVB	Rb, [Rw +]	Move indirect byte memory to direct GPR and post-increment source pointer by 1	2
MOVB	[Rw], Rb	Move direct byte GPR to indirect memory	2
MOVB	[-Rw], Rb	Pre-decrement destination pointer by 1 and move direct byte GPR to indirect memory	2
MOVB	[Rw], [Rw]	Move indirect byte memory to indirect memory	2
MOVB	[Rw +], [Rw]	Move indirect byte memory to indirect memory and post-increment destination pointer by 1	2
MOVB	[Rw], [Rw +]	Move indirect byte memory to indirect memory and post-increment source pointer by 1	2
MOVB	Rb, [Rw + #data16]	Move indirect byte memory by base plus constant to direct GPR	4
MOVB	[Rw + #data16], Rb	Move direct byte GPR to indirect memory by base plus constant	4
MOVB	[Rw], mem	Move direct byte memory to indirect memory	4
MOVB	mem, [Rw]	Move indirect byte memory to direct memory	4
MOVB	reg, mem	Move direct byte memory to direct register	4
MOVB	mem, reg	Move direct byte register to direct memory	4
MOVBS	Rw, Rb	Move direct byte GPR with sign extension to direct word GPR	2
MOVBS	reg, mem	Move direct byte memory with sign extension to direct word register	4
MOVBS	mem, reg	Move direct byte register with sign extension to direct word memory	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Data Movement (cont'd)			
MOVBZ	Rw, Rb	Move direct byte GPR with zero extension to direct word GPR	2
MOVBZ	reg, mem	Move direct byte memory with zero extension to direct word register	4
MOVBZ	mem, reg	Move direct byte register with zero extension to direct word memory	4
Jump and Call Operations			
JMPA	cc, caddr	Jump absolute if condition is met	4
JMPI	cc, [Rw]	Jump indirect if condition is met	2
JMPR	cc, rel	Jump relative if condition is met	2
JMPS	seg, caddr	Jump absolute to a code segment	4
JB	bitaddr, rel	Jump relative if direct bit is set	4
JBC	bitaddr, rel	Jump relative and clear bit if direct bit is set	4
JNB	bitaddr, rel	Jump relative if direct bit is not set	4
JNBS	bitaddr, rel	Jump relative and set bit if direct bit is not set	4
CALLA	cc, caddr	Call absolute subroutine if condition is met	4
CALLI	cc, [Rw]	Call indirect subroutine if condition is met	2
CALLR	rel	Call relative subroutine	2
CALLS	seg, caddr	Call absolute subroutine in any code segment	4
PCALL	reg, caddr	Push direct word register onto system stack and call absolute subroutine	4
TRAP	#trap7	Call interrupt service routine via immediate trap number	2
System Stack Operations			
POP	reg	Pop direct word register from system stack	2
PUSH	reg	Push direct word register onto system stack	2
SCXT	reg, #data16	Push direct word register onto system stack und update register with immediate data	4
SCXT	reg, mem	Push direct word register onto system stack und update register with direct memory	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Return Operations			
RET		Return from intra-segment subroutine	2
RETS		Return from inter-segment subroutine	2
RETP	reg	Return from intra-segment subroutine and pop direct word register from system stack	2
RETI		Return from interrupt service subroutine	2
System Control			
SRST		Software Reset	4
SBRK		Software Break	2
IDLE		Enter Idle Mode	4
PWRDN		Enter Power Down Mode (supposes $\overline{\text{NMI}}$-pin being low)	4
SRVWDT		Service Watchdog Timer	4
DISWDT		Disable Watchdog Timer	4
ENWDT		Enable Watchdog Timer	4
EINIT		Signify End-of-Initialization on RSTOUT-pin	4
ATOMIC	#irang2	Begin ATOMIC sequence *)	2
EXTR	#irang2	Begin EXTended Register sequence *)	2
EXTP	Rw, #irang2	Begin EXTended Page sequence *)	2
EXTP	#pag10, #irang2	Begin EXTended Page sequence *)	4
EXTPR	Rw, #irang2	Begin EXTended Page and Register sequence *)	2
EXTPR	#pag10, #irang2	Begin EXTended Page and Register sequence *)	4
EXTS	Rw, #irang2	Begin EXTended Segment sequence *)	2
EXTS	#seg8, #irang2	Begin EXTended Segment sequence *)	4
EXTSR	Rw, #irang2	Begin EXTended Segment and Register sequence *)	2
EXTSR	#seg8, #irang2	Begin EXTended Segment and Register sequence *)	4
Miscellaneous			
NOP		Null operation	2

7.3 Instruction Opcodes

This section lists the C166S V2 CPU instructions by hexadecimal opcodes to help identify specific instructions when reading executable code, ie. during the debugging phase.

Notes for Opcode Lists

- These instructions are encoded by means of additional bits in the operand field of the instruction

$x0_H – x7_H$:	Rw, #data3	or	Rb, #data3
$x8_H – xB_H$:	Rw, [Rw]	or	Rb, [Rw]
$xC_H – xF_H$:	Rw, [Rw +]	or	Rb, [Rw +]

For these instructions, only the lowest four GPRs (R0 to R3) can be used as indirect address pointers.

- These instructions are encoded by means of additional bits in the operand field of the instruction

$00xx.xxxx_B$:	EXTS	or	ATOMIC
$01xx.xxxx_B$:	EXTP		
$10xx.xxxx_B$:	EXTSR	or	EXTR
$11xx.xxxx_B$:	EXTPR		

Notes on the JMPR Instructions

The condition code to be tested for the JMPR instructions is specified by the opcode. Two mnemonic representation alternatives exist for some of the condition codes.

Notes on the JMPA and CALLA Instructions

For JMPA+/- and CALLA+/- instructions, a static user programmable prediction scheme is used. If bit 8 (’a’) of the instruction long word is cleared, then the branch is assumed ‘taken’. If it is set, then the branch is assumed ‘not taken’. The user controls bit 8 value by entering ’+’ or ’-’ in the instruction mnemonics. This bit can be also set/cleared by the Assembler for JMPA and CALLA instructions depending on the jump condition.

For JMPA instruction, a pre-fetch hint bit is used (the instruction bit 9 ’I’). This bit is required by the fetch unit to deal efficiently with short backward loops. It must be set if 0 < IP_jmpa - IP_target <= 32, where IP_jmpa is the address of the JMPA instruction and IP_target is the target address of the JMPA. Otherwise, bit 9 must be cleared.

Notes on the BCLR and BSET Instructions

The position of the bit to be set or cleared is specified by the opcode. The operand ‘bitoff.n’ (n = 0 to 15) refers to a particular bit within a bit-addressable word.

Notes on CoXXX instructions

All CoXXX instructions have a 3-bit wide extended control field ’rrr’ in the operand field to control the MRW repeat counter. It is located within the CoXXX instructions at bit positions [31:29].

– ‘000’ -> regular CoXXX instruction.
– ‘001’ -> RESERVED
– ‘010’ -> ‘- USR0 CoXXX’ instruction.
– ‘011’ -> ‘- USR1 CoXXX’ instruction.
– ’1xx’ -> RESERVED.

Notes on CoXXX instructions using indirect addressing modes

These CoXXX instructions have extended control fields in the operand field to specify the special indirect addressing mode.

Bitfield ’X’ is 4-bits wide and is located within CoXXX instructions at bit positions [15:12]. Bit [15] specifies one of the two IDX address pointers; the bitfield [14:12] specifies the operation concerning the IDX pointer.

Bit[15]:

– ‘0’ -> IDX0
– ‘1’ -> IDX1

Bitfield[14:12]

– ‘000’ -> RESERVED
– ‘001’ -> no-operation
– ‘010’ -> IDX +2
– ‘011’ -> IDX -2
– ’100’ -> IDX + QX0
– ’101’ -> IDX - QX0
– ’110’ -> IDX + QX1
– ’111’ -> IDX - QX1

Bitfield ’qqq’ is 3-bits wide and is located within CoXXX instructions at bit positions [26:24]. It specifies the operation concerning the Rw pointer.

Bitfield[26:24]

– ‘000’ -> RESERVED
– ‘001’ -> no-operation
– ‘010’ -> Rw +2
– ‘011’ -> Rw -2
– ’100’ -> Rw + QR0
– ’101’ -> Rw - QR0
– ’110’ -> Rw + QR1
– ’111’ -> Rw - QR1

Notes on the Undefined Opcodes

A hardware trap occurs when one of the undefined opcodes signified by '----' is decoded by the CPU.

In the following table used symbols for instruction cycle times:

reg	1 cycle, if short register addressing uses GPR
	2 cycles, else
bit	1 cycle if at least one bit address is a GPR
	2 cycles, else
co	1 to 2 cycle (see table for MAC instructions)
0-1	0 cycles, if branch is executed zerocycle
	1 cycle, else
2-3	2 cycles, if CPUCON1.SGTDIS = 1
	3 cycles, else
5-6	5 cycles, if CPUCON1.SGTDIS = 1
	6 cycles, else
4+15	4 visible cycles to calculate PSW for division,
	plus 15 invisible cycle where the result is not available
1-31	1 to 31 cycles for 'multicycle' NOP (opcode CC 000d:dddd)

Hex-code	Bytes/ Cycles	Mnemonic	Operands		Hex-code	Bytes/ Cycles	Mnemonic	Operands
00	2/1	ADD	Rw, Rw		20	2/1	SUB	Rw, Rw
01	2/1	ADDB	Rb, Rb		21	2/1	SUBB	Rb, Rb
02	4/reg	ADD	reg, mem		22	4/reg	SUB	reg, mem
03	4/reg	ADDB	reg, mem		23	4/reg	SUBB	reg, mem
04	4/reg	ADD	mem, reg		24	4/reg	SUB	mem, reg
05	4/reg	ADDB	mem, reg		25	4/reg	SUBB	mem, reg
06	4/1	ADD	reg, #data16		26	4/1	SUB	reg, #data16
07	4/1	ADDB	reg, #data8		27	4/1	SUBB	reg, #data8
08	2/1	ADD	Rw, [Rw +] or Rw, [Rw] or Rw, #data3		28	2/1	SUB	Rw, [Rw +] or Rw, [Rw] or Rw, #data3
09	2/1	ADDB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3		29	2/1	SUBB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3
0A	4/1	BFLDL	bitoff, #mask8, #data8		2A	4/bit	BCMP	bitaddr, bitaddr
0B	2/1	MUL	Rw, Rw		2B	2/1	PRIOR	Rw, Rw
0C	2/1	ROL	Rw, Rw		2C	2/1	ROR	Rw, Rw
0D	2/0-1	JMPR	cc_UC, rel		2D	2/0-1	JMPR	cc_EQ, rel or cc_Z, rel
0E	2/1	BCLR	bitoff.0		2E	2/1	BCLR	bitoff.2
0F	2/1	BSET	bitoff.0		2F	2/1	BSET	bitoff.2
10	2/1	ADDC	Rw, Rw		30	2/1	SUBC	Rw, Rw
11	2/1	ADDCB	Rb, Rb		31	2/1	SUBCB	Rb, Rb
12	4/reg	ADDC	reg, mem		32	4/reg	SUBC	reg, mem
13	4/reg	ADDCB	reg, mem		33	4/reg	SUBCB	reg, mem
14	4/reg	ADDC	mem, reg		34	4/reg	SUBC	mem, reg
15	4/reg	ADDCB	mem, reg		35	4/reg	SUBCB	mem, reg
16	4/1	ADDC	reg, #data16		36	4/1	SUBC	reg, #data16
17	4/1	ADDCB	reg, #data8		37	4/1	SUBCB	reg, #data8
18	2/1	ADDC	Rw, [Rw +] or Rw, [Rw] or Rw, #data3		38	2/1	SUBC	Rw, [Rw +] or Rw, [Rw] or Rw, #data3
19	2/1	ADDCB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3		39	2/1	SUBCB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3
1A	4/1	BFLDH	bitoff, #mask8, #data8		3A	4/bit	BMOVN	bitaddr, bitaddr
1B	2/1	MULU	Rw, Rw		3B	-/-	-	-
1C	2/1	ROL	Rw, #data4		3C	2/1	ROR	Rw, #data4
1D	2/0-1	JMPR	cc_NET, rel		3D	2/0-1	JMPR	cc_NE, rel or cc_NZ, rel
1E	2/1	BCLR	bitoff.1		3E	2/1	BCLR	bitoff.3
1F	2/1	BSET	bitoff.1		3F	2/1	BSET	bitoff.3

Hex-code	Bytes/ Cycles	Mnemonic	Operands		Hex-code	Bytes/ Cycles	Mnemonic	Operands
40	2/1	CMP	Rw, Rw		60	2/1	AND	Rw, Rw
41	2/1	CMPB	Rb, Rb		61	2/1	ANDB	Rb, Rb
42	4/reg	CMP	reg, mem		62	4/reg	AND	reg, mem
43	4/reg	CMPB	reg, mem		63	4/reg	ANDB	reg, mem
44	-/-	-	-		64	4/reg	AND	mem, reg
45	-/-	-	-		65	4/reg	ANDB	mem, reg
46	4/1	CMP	reg, #data16		66	4/1	AND	reg, #data16
47	4/1	CMPB	reg, #data8		67	4/1	ANDB	reg, #data8
48	2/1	CMP	Rw, [Rw +] or Rw, [Rw] or Rw, #data3		68	2/1	AND	Rw, [Rw +] or Rw, [Rw] or Rw, #data3
49	2/1	CMPB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3		69	2/1	ANDB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3
4A	4/bit	BMOV	bitaddr, bitaddr		6A	4/bit	BAND	bitaddr, bitaddr
4B	2/4+15	DIV	Rw		6B	2/4+15	DIVL	Rw
4C	2/1	SHL	Rw, Rw		6C	2/1	SHR	Rw, Rw
4D	2/0-1	JMPR	cc_V, rel		6D	2/0-1	JMPR	cc_N, rel
4E	2/1	BCLR	bitoff.4		6E	2/1	BCLR	bitoff.6
4F	2/1	BSET	bitoff.4		6F	2/1	BSET	bitoff.6
50	2/1	XOR	Rw, Rw		70	2/1	OR	Rw, Rw
51	2/1	XORB	Rb, Rb		71	2/1	ORB	Rb, Rb
52	4/reg	XOR	reg, mem		72	4/reg	OR	reg, mem
53	4/reg	XORB	reg, mem		73	4/reg	ORB	reg, mem
54	4/reg	XOR	mem, reg		74	4/reg	OR	mem, reg
55	4/reg	XORB	mem, reg		75	4/reg	ORB	mem, reg
56	4/1	XOR	reg, #data16		76	4/1	OR	reg, #data16
57	4/1	XORB	reg, #data8		77	4/1	ORB	reg, #data8
58	2/1	XOR	Rw, [Rw +] or Rw, [Rw] or Rw, #data3		78	2/1	OR	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 [1]
59	2/1	XORB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3		79	2/1	ORB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3
5A	4/bit	BOR	bitaddr, bitaddr		7A	4/bit	BXOR	bitaddr, bitaddr
5B	2/4+15	DIVU	Rw		7B	2/4+15	DIVLU	Rw
5C	2/1	SHL	Rw, #data4		7C	2/1	SHR	Rw, #data4
5D	2/0-1	JMPR	cc_NV, rel		7D	2/0-1	JMPR	cc_NN, rel
5E	2/1	BCLR	bitoff.5		7E	2/1	BCLR	bitoff.7
5F	2/1	BSET	bitoff.5		7F	2/1	BSET	bitoff.7

Hex-code	Bytes/ Cycles	Mnemonic	Operands		Hex-code	Bytes/ Cycles	Mnemonic	Operands
80	2/1	CMPI1	Rw, #data4		A0	2/1	CMPD1	Rw, #data4
81	2/1	NEG	Rw		A1	2/1	NEGB	Rb
82	4/1	CMPI1	Rw, mem		A2	4/1	CMPD1	Rw, mem
83	4/co	CoXXX	xx		A3	4/co	CoXXX	xx
84	4/2	MOV	[Rw], mem		A4	4/2	MOVB	[Rw], mem
85	4/1	ENWDT			A5	4/1	DISWDT	
86	4/1	CMPI1	Rw, #data16		A6	4/1	CMPD1	Rw, #data16
87	4/5	IDLE			A7	4/1	SRVWDT	
88	2/1	MOV	[-Rw], Rw		A8	2/1	MOV	Rw, [Rw]
89	2/1	MOVB	[-Rw], Rb		A9	2/1	MOVB	Rb, [Rw]
8A	4/1	JB	bitaddr, rel		AA	4/1	JBC	bitaddr, rel
8B	-/-	-	-		AB	2/2	CALLI	cc, [Rw]
8C	2/1	SBRK			AC	2/1	ASHR	Rw, Rw
8D	2/0-1	JMPR	cc_C, rel or cc_ULT, rel		AD	2/0-1	JMPR	cc_SGT, rel
8E	2/1	BCLR	bitoff.8		AE	2/1	BCLR	bitoff.10
8F	2/1	BSET	bitoff.8		AF	2/1	BSET	bitoff.10
90	2/1	CMPI2	Rw, #data4		B0	2/1	CMPD2	Rw, #data4
91	2/1	CPL	Rw		B1	2/1	CPLB	Rb
92	4/1	CMPI2	Rw, mem		B2	4/1	CMPD2	Rw, mem
93	4/co	CoXXX	xxx		B3	4/1	CoSTORE	[Rw*], CoREG
94	4/2	MOV	mem, [Rw]		B4	4/2	MOVB	mem, [Rw]
95	-/-	-	-		B5	4/1	EINIT	
96	4/1	CMPI2	Rw, #data16		B6	4/1	CMPD2	Rw, #data16
97	4/5	PWRDN			B7	4/5	SRST	
98	2/1	MOV	Rw, [Rw+]		B8	2/1	MOV	[Rw], Rw
99	2/1	MOVB	Rb, [Rw+]		B9	2/1	MOVB	[Rw], Rb
9A	4/1	JNB	bitaddr, rel		BA	4/1	JNBS	bitaddr, rel
9B	2/2-3	TRAP	#trap7		BB	2/1	CALLR	rel
9C	2/1	JMPI	cc, [Rw]		BC	2/1	ASHR	Rw, #data4
9D	2/0-1	JMPR	cc_NC, rel or cc_UGE, rel		BD	2/0-1	JMPR	cc_SLE, rel
9E	2/1	BCLR	bitoff.9		BE	2/1	BCLR	bitoff.11
9F	2/1	BSET	bitoff.9		BF	2/1	BSET	bitoff.11

Hex-code	Bytes/ Cycles	Mnemonic	Operands		Hex-code	Bytes/ Cycles	Mnemonic	Operands
C0	2/1	MOVBZ	Rw, Rb		E0	2/1	MOV	Rw, #data4
C1	-/1	-	-		E1	2/1	MOVB	Rb, #data4
C2	4/1	MOVBZ	reg, mem		E2	4/2	PCALL	reg, caddr
C3	4/1	CoSTORE	Rw, CoREG		E3	-/-	-	-
C4	4/1	MOV	[Rw+#data16], Rw		E4	4/1	MOVB	[Rw+#data16], Rb
C5	4/1	MOVBZ	mem, reg		E5	-/-	-	-
C6	4/2	SCXT	reg, #data16		E6	4/1	MOV	reg, #data16
C7	-/-	-	-		E7	4/1	MOVB	reg, #data8
C8	2/2	MOV	[Rw], [Rw]		E8	2/2	MOV	[Rw], [Rw+]
C9	2/2	MOVB	[Rw], [Rw]		E9	2/2	MOVB	[Rw], [Rw+]
CA	4/1	CALLA	cc, addr		EA	4/0-1	JMPA	cc, caddr
CB	2/1	RET			EB	2/2	RETP	reg
CC	2/1-31	NOP			EC	2/1	PUSH	reg
CD	2/0-1	JMPR	cc_SLT, rel		ED	2/0-1	JMPR	cc_UGT, rel
CE	2/1	BCLR	bitoff.12		EE	2/1	BCLR	bitoff.14
CF	2/1	BSET	bitoff.12		EF	2/1	BSET	bitoff.14
D0	2/1	MOVBS	Rw, Rb		F0	2/1	MOV	Rw, Rw
D1	2/1	ATOMIC or EXTR	#irang2		F1	2/1	MOVB	Rb, Rb
D2	4/1	MOVBS	reg, mem		F2	4/1	MOV	reg, mem
D3	4/2	CoMOV	[IDX*], [Rw*]		F3	4/1	MOVB	reg, mem
D4	4/1	MOV	Rw, [Rw + #data16]		F4	4/1	MOVB	Rb, [Rw + #data16]
D5	4/1	MOVBS	mem, reg		F5	-/-	-	-
D6	4/2	SCXT	reg, mem		F6	4/1	MOV	mem, reg
D7	4/1	EXTP(R), EXTS(R)	#pag10,#irang2 #seg8, #irang2		F7	4/1	MOVB	mem, reg
D8	2/2	MOV	[Rw+], [Rw]		F8	-/-	-	-
D9	2/2	MOVB	[Rw+], [Rw]		F9	-/-	-	-
DA	4/2	CALLS	seg, caddr		FA	4/0-1	JMPS	seg, caddr
DB	2/2	RETS			FB	2/5-6	RETI	
DC	2/1	EXTP(R), EXTS(R)	Rw, #irang2		FC	2/1	POP	reg
DD	2/0-1	JMPR	cc_SGE, rel		FD	2/0-1	JMPR	cc_ULE, rel
DE	2/1	BCLR	bitoff.13		FE	2/1	BCLR	bitoff.15
DF	2/1	BSET	bitoff.13		FF	2/1	BSET	bitoff.15

Hex-code	Extended Hex-code	Cycles	Mnemonic	Operands
83	00	1	CoMULu	RWn, [RWm*]
83	01	2	CoMULu	RWn, [RWm*], rnd
83	02	1	CoADD	RWn, [RWm*]
83	08	1	CoMULu-	RWn, [RWm*]
83	0A	1	CoSUB	RWn, [RWm*]
83	10	1	CoMACu	RWn, [RWm*]
83	11	2	CoMACu	RWn, [RWm*], rnd
83	12	1	CoSUBR	RWn, [RWm*]
83	20	1	CoMACu-	RWn, [RWm*]
83	22	1	CoLOAD	RWn, [RWm*]
83	2A	1	CoLOAD-	RWn, [RWm*]
83	30	1	CoMACRu	RWn, [RWm*]
83	31	2	CoMACRu	RWn, [RWm*], rnd
83	3A	1	CoMAX	RWn, [RWm*]
83	40	1	CoMULsu	RWn, [RWm*]
83	41	2	CoMULsu	RWn, [RWm*], rnd
83	42	1	CoADD2	RWn, [RWm*]
83	48	1	CoMULsu-	RWn, [RWm*]
83	4A	1	CoSUB2	RWn, [RWm*]
83	50	1	CoMACsu	RWn, [RWm*]
83	51	2	CoMACsu	RWn, [RWm*], rnd
83	52	1	CoSUB2R	RWn, [RWm*]
83	60	1	CoMACsu-	RWn, [RWm*]
83	62	1	CoLOAD2	RWn, [RWm*]
83	6A	1	CoLOAD2-	RWn, [RWm*]
83	70	1	CoMACRsu	RWn, [RWm*]
83	71	2	CoMACRsu	RWn, [RWm*], rnd
83	7A	1	CoMIN	RWn, [RWm*]
83	80	1	CoMULus	RWn, [RWm*]
83	81	2	CoMULus	RWn, [RWm*], rnd
83	88	1	CoMULus-	RWn, [RWm*]
83	8A	1	CoSHL	[RWm*]
83	90	1	CoMACus	RWn, [RWm*]
83	91	2	CoMACus	RWn, [RWm*], rnd
83	9A	1	CoSHR	[RWm*]
83	A0	1	CoMACus-	RWn, [RWm*]
83	AA	1	CoASHR	[RWm*]
83	B0	1	CoMACRus	RWn, [RWm*]
83	B1	2	CoMACRus	RWn, [RWm*], rnd
83	BA	1	CoASHR	[RWm*] , rnd
83	C0	1	CoMUL	RWn, [RWm*]
83	C1	2	CoMUL	RWn, [RWm*], rnd
83	C2	1	CoCMP	RWn, [RWm*]
83	C8	1	CoMUL-	RWn, [RWm*]

Hex-code	Extended Hex-code	Cycles	Mnemonic	Operands
83	CA	1	CoABS	RWn, [RWm*]
83	D0	1	CoMAC	RWn, [RWm*]
83	D1	2	CoMAC	RWn, [RWm*], rnd
83	E0	1	CoMAC-	RWn, [RWm*]
83	F0	1	CoMACR	RWn, [RWm*]
83	F1	2	CoMACR	RWn, [RWm*], rnd
93	00	1	CoMULu	[IDXi*], [RWm*]
93	01	2	CoMULu	[IDXi*], [RWm*], rnd
93	02	1	CoADD	[IDXi*], [RWm*]
93	08	1	CoMULu-	[IDXi*], [RWm*]
93	0A	1	CoSUB	[IDXi*], [RWm*]
93	10	1	CoMACu	[IDXi*], [RWm*]
93	11	2	CoMACu	[IDXi*], [RWm*], rnd
93	12	1	CoSUBR	[IDXi*], [RWm*]
93	18	1	CoMACMu	[IDXi*], [RWm*]
93	19	2	CoMACMu	[IDXi*], [RWm*], rnd
93	20	1	CoMACu-	[IDXi*], [RWm*]
93	22	1	CoLOAD	[IDXi*], [RWm*]
93	28	1	CoMACMu-	[IDXi*], [RWm*]
93	2A	1	CoLOAD-	[IDXi*], [RWm*]
93	30	1	CoMACRu	[IDXi*], [RWm*]
93	31	2	CoMACRu	[IDXi*], [RWm*], rnd
93	38	1	CoMACMRu	[IDXi*], [RWm*]
93	39	2	CoMACMRu	[IDXi*], [RWm*], rnd
93	3A	1	CoMAX	[IDXi*], [RWm*]
93	40	1	CoMULsu	[IDXi*], [RWm*]
93	41	2	CoMULsu	[IDXi*], [RWm*], rnd
93	42	1	CoADD2	[IDXi*], [RWm*]
93	48	1	CoMULsu-	[IDXi*], [RWm*]
93	4A	1	CoSUB2	[IDXi*], [RWm*]
93	50	1	CoMACsu	[IDXi*], [RWm*]
93	51	2	CoMACsu	[IDXi*], [RWm*], rnd
93	52	1	CoSUB2R	[IDXi*], [RWm*]
93	58	1	CoMACMsu	[IDXi*], [RWm*]
93	59	2	CoMACMsu	[IDXi*], [RWm*], rnd
93	5A	1	CoNOP	[IDXi*]
93	5A	1	CoNOP	[IDXi*], [RWm*]
93	5A	1	CoNOP	[RWm*]
93	60	1	CoMACsu-	[IDXi*], [RWm*]
93	62	1	CoLOAD2	[IDXi*], [RWm*]
93	68	1	CoMACMsu-	[IDXi*], [RWm*]
93	6A	1	CoLOAD2-	[IDXi*], [RWm*]
93	70	1	CoMACRsu	[IDXi*], [RWm*]
93	71	2	CoMACRsu	[IDXi*], [RWm*], rnd
93	78	1	CoMACMRsu	[IDXi*], [RWm*]

Hex-code	Extended Hex-code	Cycles	Mnemonic	Operands
93	79	2	CoMACMRsu	[IDXi*], [RWm*], rnd
93	7A	1	CoMIN	[IDXi*], [RWm*]
93	80	1	CoMULus	[IDXi*], [RWm*]
93	81	2	CoMULus	[IDXi*], [RWm*], rnd
93	88	1	CoMULus-	[IDXi*], [RWm*]
93	90	1	CoMACus	[IDXi*], [RWm*]
93	91	2	CoMACus	[IDXi*], [RWm*], rnd
93	98	1	CoMACMus	[IDXi*], [RWm*]
93	99	2	CoMACMus	[IDXi*], [RWm*], rnd
93	A0	1	CoMACus-	[IDXi*], [RWm*]
93	A8	1	CoMACMus-	[IDXi*], [RWm*]
93	B0	1	CoMACRus	[IDXi*], [RWm*]
93	B1	2	CoMACRus	[IDXi*], [RWm*], rnd
93	B8	1	CoMACMRus	[IDXi*], [RWm*]
93	B9	2	CoMACMRus	[IDXi*], [RWm*], rnd
93	C0	1	CoMUL	[IDXi*], [RWm*]
93	C1	2	CoMUL	[IDXi*], [RWm*] , rnd
93	C2	1	CoCMP	[IDXi*], [RWm*]
93	C8	1	CoMUL-	[IDXi*], [RWm*]
93	CA	1	CoABS	[IDXi*], [RWm*]
93	D0	1	CoMAC	[IDXi*], [RWm*]
93	D1	2	CoMAC	[IDXi*], [RWm*], rnd
93	D8	1	CoMACM	[IDXi*], [RWm*]
93	D9	2	CoMACM	[IDXi*], [RWm*], rnd
93	E0	1	CoMAC-	[IDXi*], [RWm*]
93	E8	1	CoMACM-	[IDXi*], [RWm*]
93	F0	1	CoMACR	[IDXi*], [RWm*]
93	F1	2	CoMACR	[IDXi*], [RWm*], rnd
93	F8	1	CoMACMR	[IDXi*], [RWm*]
93	F9	2	CoMACMR	[IDXi*], [RWm*] , rnd
A3	00	1	CoMULu	RWn, RWm
A3	01	2	CoMULu	RWn, RWm, rnd
A3	02	1	CoADD	RWn, RWm
A3	08	1	CoMULu-	RWn, RWm
A3	0A	1	CoSUB	RWn, RWm
A3	10	1	CoMACu	RWn, RWm
A3	11	2	CoMACu	RWn, RWm, rnd
A3	12	1	CoSUBR	RWn, RWm
A3	1A	1	CoABS	
A3	20	1	CoMACu-	RWn, RWm
A3	22	1	CoLOAD	RWn, RWm
A3	2A	1	CoLOAD-	RWn, RWm
A3	30	1	CoMACRu	RWn, RWm
A3	31	2	CoMACRu	RWn, RWm , rnd
A3	32	1	CoNEG	

Hex-code	Extended Hex-code	Cycles	Mnemonic	Operands
A3	3A	1	CoMAX	RWn, RWm
A3	40	1	CoMULsu	RWn, RWm
A3	41	2	CoMULsu	RWn, RWm , rnd
A3	42	1	CoADD2	RWn, RWm
A3	48	1	CoMULsu-	RWn, RWm
A3	4A	1	CoSUB2	RWn, RWm
A3	50	1	CoMACsu	RWn, RWm
A3	51	2	CoMACsu	RWn, RWm , rnd
A3	52	1	CoSUB2R	RWn, RWm
A3	60	1	CoMACsu-	RWn, RWm
A3	62	1	CoLOAD2	RWn, RWm
A3	6A	1	CoLOAD2-	RWn, RWm
A3	70	1	CoMACRsu	RWn, RWm
A3	71	2	CoMACRsu	RWn, RWm , rnd
A3	72	1	CoNEG	rnd
A3	7A	1	CoMIN	RWn, RWm
A3	80	1	CoMULus	RWn, RWm
A3	81	2	CoMULus	RWn, RWm, rnd
A3	82	1	CoSHL	#data5
A3	88	1	CoMULus-	RWn, RWm
A3	8A	1	CoSHL	RWn
A3	90	1	CoMACus	RWn, RWm
A3	91	2	CoMACus	RWn, RWm, rnd
A3	92	1	CoSHR	#data5
A3	9A	1	CoSHR	RWn
A3	A0	1	CoMACus-	RWn, RWm
A3	A2	1	CoASHR	#data5
A3	AA	1	CoASHR	RWn
A3	B0	1	CoMACRus	RWn, RWm
A3	B1	2	CoMACRus	RWn, RWm, rnd
A3	B2	1	CoASHR	#data5, rnd
A3	B2	1	CoRND	
A3	BA	1	CoASHR	RWn, rnd
A3	C0	1	CoMUL	RWn, RWm
A3	C1	2	CoMUL	RWn, RWm, rnd
A3	C2	1	CoCMP	RWn, RWm
A3	C8	1	CoMUL-	RWn, RWm
A3	CA	1	CoABS	RWn, RWm
A3	D0	1	CoMAC	RWn, RWm
A3	D1	2	CoMAC	RWn, RWm, rnd
A3	E0	1	CoMAC-	RWn, RWm
A3	F0	1	CoMACR	RWn, RWm

Hex-code	Extended Hex-code	Cycles	Mnemonic	Operands
A3	F1	2	CoMACR	RWn, RWm, rnd
B3		1	CoSTORE	[RWn*], CoReg
C3		1	CoSTORE	RWn, CoReg
D3	00	2	CoMOV	[IDXi*], [RWm*]

8 Detailed Instruction Description

This section describes each instruction in detail. The instructions are listed alphabetically, and the description contains the following elements.

- **Instruction Name:** Specifies the mnemonic opcode of the instruction in oversized bold lettering for easy reference. The mnemonics have been chosen with regard to the particular operation performed by the instruction.
- **Syntax:** Specifies the mnemonic opcode and the required formal operands of the instruction as used in the following subsection 'Operation'. There are instructions with either none, one, two or three operands, which must be separated from each other by commas:

MNEMONIC {op1 {,op2 {,op3 } } }

The syntax for the actual operands of an instruction depends on the selected addressing mode. All of the available addressing modes are summarized at the end of each single instruction description. In contrast to the syntax for the instructions described in the following material, the assembler provides much more flexibility in writing C166S V2 CPU programs (e.g. by generic instructions and by automatically selecting appropriate addressing modes whenever possible). Thus, it eases the use of the instruction set.

- **Operation:** This part presents a logical description of the operation performed by an instruction as a symbolic formula or a high level language construct.

The following symbols are used to represent data movement, arithmetic, or logical operators.

Diadic operations:	(opX)		operator (opY)
¨	(opY)	is	**MOVED** into (opX)
+	(opX)	is	**ADDED** to (opY)
-	(opY)	is	**SUBTRACTED** from (opX)
*	(opX)	is	**MULTIPLIED** by (opY)
/	(opX)	is	**DIVIDED** by (opY)
Ÿ	(opX)	is	logically **AND**ed with (opY)
⁄	(opX)	is	logically **OR**ed with (opY)
Ý	(opX)	is	logically **EXCLUSIVELY OR**ed with (opY)
¤	(opX)	is	**COMPARED** against (opY)
mod	(opX)	is	divided **MODULO** (opY)
||	(opX)	is	**CONCATENATED** (opY)
Monadic operations:			**operator (opX)**

ÿ (opX) is logically **COMPLEMENTED**

Parentheses indicate a method of addressing the used operand as follows:

opX	Specifies the immediate constant value of opX
(opX)	Specifies the contents of opX
(opX[n])	Specifies the contents of bit n of opX
((opX))	Specifies the contents of the contents of opX (ie. opX is used as pointer to the actual operand)

The following operands notation will also be used in the operational description:

CP	Context Pointer
CSP	Code Segment Pointer
IP	Instruction Pointer
MD	Multiply/Divide register (32 bits wide, consists of MDH and MDL)
MDL, MDH	Multiply/Divide Low and High registers (each 16 bit wide)
ACC	Accumulator (40 bits wide, consists of MAE, MAH and MDL)
MAH, MAL	Accumulator Low and High registers (each 16 bits wide)
MAE	Accumulator extension register (one byte wide)
PSW	Program Status Word
SP	System Stack Pointer
CPUCON1	CPU Configuration register
C	Carry condition flag in the PSW register
V	Overflow condition flag in the PSW register
SGTDIS	Segmentation Disable bit in the SYSCON register
count	Temporary variable for an intermediate storage of the number of shift or rotate cycles which remain to complete the shift or rotate operation
tmp	Temporary variable for an intermediate result
0, 1, 2,...	Constant values due to the data format of the specified operation

Data Types: This part specifies the particular data type according to the instruction. Basically, the following data types are possible:

BIT, BYTE, WORD, DOUBLEWORD, ACC = 40-bit signed value

Only CoXXX instructions and instructions which extend byte data to word data can change the data type. Note that the data types mentioned in this subsection do not cover accesses to indirect address pointers or to the system stack. These accesses are always performed with word data. Moreover, no data type is specified for System Control Instructions and for those branch instructions which do not access any explicitly addressed data.

- **Description:** This part provides a brief description of the action that is executed by the respective instruction.
- **Condition Code:** The Condition code indicates that the respective instruction is executed if the specified condition exists, and is skipped if it does not. The table below summarizes the sixteen possible condition codes that can be used within Call and Branch instructions. The table shows the abbreviations, the test that is executed for a specific condition, and a 4/5-bit number associated with condition code.

Condition Code Mnemonic cc	Test	Description	Condition Code Number c	Condition Code Number d
cc_UC	1 = 1	Unconditional	0_H	0_H
cc_Z	Z = 1	Zero	2_H	4_H
cc_NZ	Z = 0	Not zero	3_H	6_H
cc_V	V = 1	Overflow	4_H	8_H
cc_NV	V = 0	No overflow	5_H	A_H
cc_N	N = 1	Negative	6_H	C_H
cc_NN	N = 0	Not negative	7_H	E_H
cc_C	C = 1	Carry	8_H	10_H
cc_NC	C = 0	No carry	9_H	12_H
cc_EQ	Z = 1	Equal	2_H	4_H
cc_NE	Z = 0	Not equal	3_H	6_H
cc_ULT	C = 1	Unsigned less than	8_H	10_H
cc_ULE	(Z∨C) = 1	Unsigned less than or equal	F_H	$1E_H$
cc_UGE	C = 0	Unsigned greater than or equal	9_H	12_H
cc_UGT	(Z∨C) = 0	Unsigned greater than	E_H	$1C_H$

Condition Code Mnemonic cc	Test	Description	Condition Code Number c	Condition Code Number d
cc_SLT	(N⊕V) = 1	Signed less than	C_H	18_H
cc_SLE	(Z∨(N⊕V)) = 1	Signed less than or equal	B_H	16_H
cc_SGE	(N⊕V) = 0	Signed greater than or equal	D_H	$1A_H$
cc_SGT	(Z∨(N⊕V)) = 0	Signed greater than	A_H	14_H
cc_NET	(Z∨E) = 0	Not equal AND not end of table	1_H	02_H
cc_nusr0[1)]	usr0 = 0	usr0 is cleared		1_H
cc_nusr1[1)]	usr1 = 0	usr1 is cleared		3_H
cc_usr0[1)]	usr0 = 1	usr0 is set		5_H
cc_usr1[1)]	usr1 = 1	usr1 is set		7_H

1) Only usable with the JMA and CALLA instructions.

- **Condition Flags:** This part reflects the state of the N, C, V, Z, and E flags in the PSW register which is the state after execution of the corresponding instruction, except if the PSW register itself was specified as the destination operand of that instruction (see Note).

The resulting state of the flags is represented by symbols as follows:

'*' The flag is set due to the following standard rules for the corresponding flag:

N = 1 : MSB of the result is set

N = 0 : MSB of the result is not set

C = 1 : Carry occurred during operation

C = 0 : No Carry occurred during operation

V = 1 : Arithmetic Overflow occurred during operation

V = 0 : No Arithmetic Overflow occurred during operation

Z = 1 : Result equals zero

Z = 0 : Result does not equal zero

E = 1 : Source operand represents the lowest negative number (either 8000h for word data or 80h for byte data)

E = 0 : Source operand does not represent the lowest negative number for the specified data type

'S' The flag is set due to rules which deviate from the described standard. For more details see instruction pages (below) or the ALU status flags description.

'-' The flag is not affected by the operation.

'0' The flag is cleared by the operation.

'NOR' The flag contains the logical NORing of the two specified bit operands.

'AND' The flag contains the logical ANDing of the two specified bit operands.

'OR' The flag contains the logical ORing of the two specified bit operands.

'XOR' The flag contains the logical XORing of the two specified bit operands.

'B' The flag contains the original value of the specified bit operand.

'$\overline{B}$' The flag contains the complemented value of the specified bit operand.

Note: If the PSW register was specified as the destination operand of an instruction, the condition flags can not be interpreted as just described, because the PSW register is modified depending on the data format of the instruction as follows:
For word operations, the PSW register is overwritten with the word result. For byte operations, the non-addressed byte is cleared and the addressed byte is overwritten. For bit or bit-field operations on the PSW register, only the specified bits are modified. Supposed that the condition flags were not selected as destination bits, they stay unchanged. This means that they keep the state after execution of the previous instruction.
In any case, if the PSW was the destination operand of an instruction, the PSW flags do NOT represent the condition flags of this instruction as usual.

- **Addressing Modes:** This part specifies which combinations of different addressing modes are available for the required operands. The selected addressing mode combination is usually specified by the opcode of the corresponding instruction. However, there are some arithmetic and logical instructions for which the addressing mode combination is not specified by the (identical) opcodes but by particular bits within the operand field.

The addressing mode entries are made up of three elements:

Mnemonic Shows accepted operands for the respective instruction.

Format This part specifies the format of the instructions as it is represented in the assembler listing. **Figure 8-1** shows the relation between the instruction format representation of the assembler and the corresponding internal organization of such an instruction format (N = nibble = 4 bits).

The following symbols are used to describe the instruction formats:

00_H through FF_H: Instruction Opcodes

0, 1 : Constant Values

:.... : Each of the 4 characters immediately following a colon represents a single bit

:..ii : 2-bit short GPR address (Rwi)

SS : Code segment number

:..## : 2-bit immediate constant (#irang2)

:.### : 3-bit immediate constant (#data3)

...#:# : 5-bit immediate constant (#data5)

c : 4-bit condition code specification (cc)

d : 5-bit condition code specification (xcc)

n : 4-bit short GPR address (Rwn or Rbn)

m : 4-bit short GPR address (Rwm or Rbm)

q : 4-bit position of the source bit within the word specified by QQ

qqq : 3-bit addressing mode specifier for CoXXX instructions

z : 4-bit position of the destination bit within the word specified by ZZ

: 4-bit immediate constant (#data4)

t:ttt0 : 7-bit trap number (#trap7)

QQ : 8-bit word address of the source bit (bitoff)

rr : 8-bit relative target address word offset (rel)

rrr : 3-bit repeat control for CoXXX instructions

RR : 8-bit word address reg

wwww:w : 5-bit word address CoREG

X : 4-bit addressing mode specifier for CoXXX instructions

ZZ : 8-bit word address of the destination bit (bitoff)

: 8-bit immediate constant (#data8)

xx : 8-bit immediate constant (represented by #data16, byte xx is not significant)

@ @ : 8-bit immediate constant (#mask8)

MM MM : 16-bit address (mem or caddr; low byte, high byte)

: 16-bit immediate constant (#data16; low byte, high byte)

a : 1-bit branch assumption bit

l : 1-bit short backward loop bit

Number of Bytes All C166S V2 CPU instructions are either 2 or 4 bytes. According to the instruction size, all instructions can be classified as either single word or double word instructions.

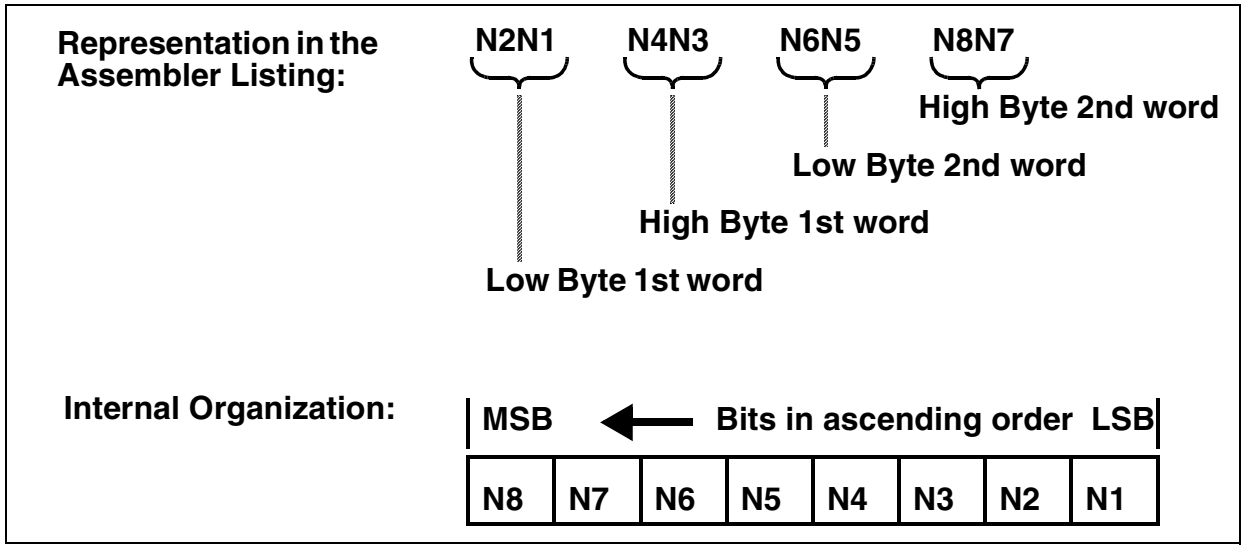


Figure 8-1 Instruction Format Representation

The following pages contain a detailed description of each normal arithmetic, logic, branch or system instruction in alphabetical order followed by a list of the dedicated DSP instructions:

8.1 Normal Instruction Set

ADD Integer Addition ADD

Group Arithmetic Instructions

Syntax **ADD op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) + (op2)

Description

Performs a 2s complement binary addition of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	*	*	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a carry is generated from the most significant bit of the word data type. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ADD	Rw_n , #data3	08 n:0###	2
ADD	Rw_n , Rw_m	00 nm	2
ADD	Rw_n , [Rw_i+]	08 n:11ii	2
ADD	Rw_n , [Rw_i]	08 n:10ii	2
ADD	mem , reg	04 RR MM MM	4
ADD	reg , #data16	06 RR ## ##	4
ADD	reg , mem	02 RR MM MM	4

ADDB Integer Addition ADDB

Group Arithmetic Instructions

Syntax **ADDB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) + (op2)

Description

Performs a 2s complement binary addition of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	*	*	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the byte data type. Cleared otherwise.
- C Set if a carry is generated from the most significant bit of the byte data type. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ADDB	Rb_n , #data3	09 n:0###	2
ADDB	Rb_n , Rb_m	01 nm	2
ADDB	Rb_n , [Rw_i+]	09 n:11ii	2
ADDB	Rb_n , [Rw_i]	09 n:10ii	2
ADDB	mem , reg	05 RR MM MM	4
ADDB	reg , #data8	07 RR ## xx	4
ADDB	reg , mem	03 RR MM MM	4

ADDC Integer Addition with Carry ADDC

Group Arithmetic Instructions

Syntax **ADDC op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) + (op2) + (C)

Description

Performs a 2s complement binary addition of the source operand specified by op2, the destination operand specified by op1 and the previously generated carry bit. The sum is then stored in op1. This instruction can be used to perform multiple precision arithmetic.

CPU Flags

E	Z	V	C	N
*	S	*	*	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero and previous Z flag was set. Cleared otherwise.
- V Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a carry is generated from the most significant bit of the word data type. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ADDC	Rw_n , #data3	18 n:0###	2
ADDC	Rw_n , Rw_m	10 nm	2
ADDC	Rw_n , [Rw_i+]	18 n:11ii	2
ADDC	Rw_n , [Rw_i]	18 n:10ii	2
ADDC	mem , reg	14 RR MM MM	4
ADDC	reg , #data16	16 RR ## ##	4
ADDC	reg , mem	12 RR MM MM	4

ADDCB Integer Addition with Carry ADDCB

Group Arithmetic Instructions

Syntax **ADDCB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) + (op2) + (C)

Description

Performs a 2s complement binary addition of the source operand specified by op2, the destination operand specified by op1 and the previously generated carry bit. The sum is then stored in op1. This instruction can be used to perform multiple precision arithmetic.

CPU Flags

E	Z	V	C	N
*	S	*	*	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero and previous Z flag was set. Cleared otherwise.
- V Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the byte data type. Cleared otherwise.
- C Set if a carry is generated from the most significant bit of the byte data type. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ADDCB	Rb_n , #data3	19 n:0###	2
ADDCB	Rb_n , Rb_m	11 nm	2
ADDCB	Rb_n , [Rw_i+]	19 n:11ii	2
ADDCB	Rb_n , [Rw_i]	19 n:10ii	2
ADDCB	mem , reg	15 RR MM MM	4
ADDCB	reg , #data8	17 RR ## xx	4
ADDCB	reg , mem	13 RR MM MM	4

AND Logical AND AND

Group Logical Instructions

Syntax **AND op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) ∧ (op2)

Description

Performs a bitwise logical AND of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
AND	Rw_n , #data3	68 n:0###	2
AND	Rw_n , Rw_m	60 nm	2
AND	Rw_n , [Rw_i+]	68 n:11ii	2
AND	Rw_n , [Rw_i]	68 n:10ii	2
AND	mem , reg	64 RR MM MM	4
AND	reg , #data16	66 RR ## ##	4
AND	reg , mem	62 RR MM MM	4

ANDB Logical AND ANDB

Group Logical Instructions

Syntax **ANDB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) ∧ (op2)

Description

Performs a bitwise logical AND of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ANDB	Rb_n , #data3	69 n:0###	2
ANDB	Rb_n , Rb_m	61 nm	2
ANDB	Rb_n , [Rw_i+]	69 n:11ii	2
ANDB	Rb_n , [Rw_i]	69 n:10ii	2
ANDB	mem , reg	65 RR MM MM	4
ANDB	reg , #data8	67 RR ## xx	4
ANDB	reg , mem	63 RR MM MM	4

ASHR Arithmetic Shift Right ASHR

Group Shift and Rotate Instructions

Syntax **ASHR op1, op2**

Source Operand(s) op1 → WORD
op2 → shift counter

Destination Operand(s) op1 → WORD

Operation

```
(count) ← (op2)
(V) ← 0
(C) ← 0
DO WHILE ((count) ≠ 0)
        (V) ← (C) ∨ (V)
        (C) ← (op1[0])
        (op1[n]) ← (op1[n+1]) [n=0...14]
        (count) ← (count) - 1
END WHILE
```

Description

Arithmetically shifts the destination word operand op1 right by the number of times as specified by the source operand op2. To preserve the sign of the original operand op1, the most significant bits of the result are filled with zeros if the original most significant bit was a 0 or with ones if the original most significant bit was a 1. The Overflow flag is used as a Rounding flag. The least significant bit is shifted into the Carry. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant 4 bits are used.

CPU Flags

E	Z	V	C	N
0	*	*	*	*

- E Always cleared.
- Z Set if result equals zero. Cleared otherwise.
- V Set if in any cycle of the shift operation a 1 is shifted out of the carry flag. Cleared in case of a shift count equal 0.
- C The carry flag is set according to the last least significant bit shifted out of op1. Cleared for a shift count of zero.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ASHR	Rw_n , #data4	BC #n	2
ASHR	Rw_n , Rw_m	AC nm	2

ATOMIC Begin ATOMIC Sequence ATOMIC

Group System Control Instructions

Syntax **ATOMIC op1**

Source Operand(s) op1 → 2-bit instruction counter

Destination Operand(s) none

Operation

(count) ← (op1) [1 ≤ op1 ≤ 4]
Disable interrupts and Class A traps
DO WHILE ((count) ≠ 0 AND Class_B_Trap_Condition ≠ TRUE)
Next Instruction
(count) ← (count) - 1
END WHILE
(count) ← 0
Enable interrupts and traps

Description

Causes standard and PEC interrupts and class A hardware traps to be disabled for a specified number of instructions. The ATOMIC instruction becomes immediately active. No NOPs are required for normal ATOMIC execution. Depending on the value of op1, the period of validity of the ATOMIC sequence extends over the sequence of the next one to four instructions being executed after the ATOMIC instruction. All instructions requiring multiple cycles or hold states to be executed are regarded as one instruction in this sense. Any instruction type can be used with the ATOMIC instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
ATOMIC	#irang2	D1 :00##-0	2

BAND Bit Logical AND BAND

Group Boolean Bit Manipulation Instructions

Syntax **BAND op1, op2**

Source Operand(s) op1, op2 → BIT

Destination Operand(s) op1 → BIT

Operation

(op1) ← (op1) ∧ (op2)

Description

Performs a single bit logical AND of the source bit specified by op2 and the destination bit specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
0	NOR	OR	AND	XOR

E Always cleared.
Z Contains the logical NOR of the two specified bits.
V Contains the logical OR of the two specified bits.
C Contains the logical AND of the two specified bits.
N Contains the logical XOR of the two specified bits.

Encoding

Mnemonic		Format	Bytes
BAND	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	6A QQ ZZ qz	4

BCLR Bit Clear BCLR

Group Boolean Bit Manipulation Instructions

Syntax **BCLR op1**

Source Operand(s) none

Destination Operand(s) op1 → BIT

Operation

(op1) ← 0

Description

Clears the bit specified by op1. This instruction is primarily used for peripheral and system control.

CPU Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

E Always cleared.

Z Contains the logical negation of the previous state of the specified bit.

V Always cleared.

C Always cleared.

N Contains the previous state of the specified bit.

Encoding

Mnemonic		Format	Bytes
BCLR	$bitaddr_{Q.q}$	qE QQ	2

BCMP Bit to Bit Compare BCMP

Group Boolean Bit Manipulation Instructions

Syntax **BCMP op1, op2**

Source Operand(s) op1, op2 → BIT

Destination Operand(s) none

Operation

(op1) ⇔ (op2)

Description

Performs a single bit comparison of the source bit specified by op1 and the source bit specified by op2. No result is written by this instruction. Only the flags are updated.

CPU Flags

E	Z	V	C	N
0	NOR	OR	AND	XOR

E Always cleared.
Z Contains the logical NOR of the two specified bits.
V Contains the logical OR of the two specified bits.
C Contains the logical AND of the two specified bits.
N Contains the logical XOR of the two specified bits.

Encoding

Mnemonic		Format	Bytes
BCMP	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	2A QQ ZZ qz	4

BFLDH Bit Field High Byte BFLDH

Group Boolean Bit Manipulation Instructions

Syntax **BFLDH op1, op2, op3**

Source Operand(s) op1 → WORD
op2, op3 → BYTE

Destination Operand(s) op1 → WORD

Operation

```
(count) ← 0
DO WHILE ((count) <8)
        IF (op2[(count)] = 1)
            (op1[(count) + 8]) ← op3[(count)]
        ENDIF
(count) ← (count) + 1
END WHILE
```

Description
Replaces those bits in the high byte of the destination word operand op1 which are selected by an ’1’ in the mask specified by op2 with the bits at the corresponding positions in "op3".

CPU Flags

E	Z	V	C	N
0	*	0	0	*

E Always cleared.
Z Set if result equals zero. Cleared otherwise.
V Always cleared.
C Always cleared.
N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
BFLDH	$bitoff_Q$, #mask8 , #data8	1A QQ ## @@	4

BFLDL Bit Field Low Byte BFLDL

Group Boolean Bit Manipulation Instructions

Syntax **BFLDL op1, op2, op3**

Source Operand(s) op1 → WORD
op2, op3 → BYTE

Destination Operand(s) op1 → WORD

Operation

```
(count) ← 0
DO WHILE ((count) <8)
        IF op2[(count)] = 1
            (op1[(count)]) ← op3[(count)]
        ENDIF
(count) ← (count) + 1
END WHILE
```

Description
Replaces those bits in the low byte of the destination word operand op1 which are selected by an ’1’ in the mask specified by op2 with the bits at the corresponding positions in "op3".

CPU Flags

E	Z	V	C	N
0	*	0	0	*

E Always cleared.
Z Set if result equals zero. Cleared otherwise.
V Always cleared.
C Always cleared.
N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
BFLDL	$bitoff_Q$, #mask8 , #data8	0A QQ @@ ##	4

BMOV Bit to Bit Move BMOV

Group Boolean Bit Manipulation Instructions

Syntax **BMOV op1, op2**

Source Operand(s) op2 → BIT

Destination Operand(s) op1 → BIT

Operation

(op1) ← (op2)

Description

Moves a single bit from the source operand specified by op2 into the destination operand specified by op1. The source bit is examined and the flags are updated accordingly.

CPU Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

E Always cleared.
Z Contains the logical negation of the source bit.
V Always cleared.
C Always cleared.
N Contains the state of the source bit.

Encoding

Mnemonic		Format	Bytes
BMOV	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	4A QQ ZZ qz	4

BMOVN Bit to Bit Move and Negate BMOVN

Group Boolean Bit Manipulation Instructions

Syntax **BMOVN op1, op2**

Source Operand(s) op2 → BIT

Destination Operand(s) op1 → BIT

Operation

(op1) ← ¬(op2)

Description

Moves the complement of a single bit from the source operand specified by op2 into the destination operand specified by op1. The source bit is examined and the flags are updated accordingly.

CPU Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

E Always cleared.
Z Contains the logical negation of the source bit.
V Always cleared.
C Always cleared.
N Contains the state of the source bit.

Encoding

Mnemonic		Format	Bytes
BMOVN	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	3A QQ ZZ qz	4

BOR Bit Logical OR BOR

Group Boolean Bit Manipulation Instructions

Syntax **BOR op1, op2**

Source Operand(s) op1, op2 → BIT

Destination Operand(s) op1 → BIT

Operation

(op1) ← (op1) ∨ (op2)

Description

Performs a single bit logical OR of the source bit specified by op2 and the destination bit specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
0	NOR	OR	AND	XOR

- E Always cleared.
- Z Contains the logical NOR of the two specified bits.
- V Contains the logical OR of the two specified bits.
- C Contains the logical AND of the two specified bits.
- N Contains the logical XOR of the two specified bits.

Encoding

Mnemonic		Format	Bytes
BOR	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	5A QQ ZZ qz	4

BSET Bit Set BSET

Group Boolean Bit Manipulation Instructions

Syntax **BSET op1**

Source Operand(s) none

Destination Operand(s) op1 → BIT

Operation

(op1) ← 1

Description

Sets the bit specified by op1.

CPU Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

E Always cleared.

Z Contains the logical negation of the previous state of the specified bit.

V Always cleared.

C Always cleared.

N Contains the previous state of the specified bit.

Encoding

Mnemonic		Format	Bytes
BSET	$bitaddr_{Q.q}$	qF QQ	2

BXOR Bit Logical XOR BXOR

Group Boolean Bit Manipulation Instructions

Syntax **BXOR op1, op2**

Source Operand(s) op1, op2 → BIT

Destination Operand(s) op1 → BIT

Operation

(op1) ← (op1) ⊕ (op2)

Description

Performs a single bit logical EXCLUSIVE OR of the source bit specified by op2 and the destination bit specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
0	NOR	OR	AND	XOR

- E Always cleared.
- Z Contains the logical NOR of the two specified bits.
- V Contains the logical OR of the two specified bits.
- C Contains the logical AND of the two specified bits.
- N Contains the logical XOR of the two specified bits.

Encoding

Mnemonic		Format	Bytes
BXOR	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	7A QQ ZZ qz	4

CALLA Call Subroutine Absolute CALLA

Group Call Instructions

Syntax **CALLA op1, op2**

Alternative Syntax CALLA+ op1, op2
CALLA- op1, op2

Source Operand(s) op1 → extended condition code
op2 → 16-bit address offset

Destination Operand(s) none

Operation

```
IF (op1) THEN
        (SP) ← (SP) - 2
        ((SP)) ← (IP)
        (IP) ← op2
ELSE
        next instruction
END IF
```

Description

If the condition specified by op1 is met, a branch to the absolute memory location specified by the second operand op2 is taken. The value of the instruction pointer IP is placed into the system stack. Because the IP always points to the instruction following the branch instruction, the value stored in the system stack represents the return address of the calling routine. A static prediction scheme is used: if the bit 'a' of the instruction long word is cleared then CALLA is assumed 'taken' and if this bit is set to 1, CALLA is assumed 'not taken'. CALLA+ and CALLA- instructions are converted into CALLA assumed 'taken' (prediction bit cleared) and 'not taken' (prediction bit set) respectively. For regular CALLA instructions, the assembler assumes them 'taken'.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.

N	Not affected.

Encoding

Mnemonic		Format	Bytes
CALLA	xcc , caddr	CA d00a MM MM	4

CALLI Call Subroutine Indirect CALLI

Group Call Instructions

Syntax **CALLI op1, op2**

Source Operand(s) op1 → condition code
op2 → 16-bit address offset

Destination Operand(s) none

Operation

```
IF (op1) THEN
        (SP) ← (SP) - 2
        ((SP)) ← (IP)
        (IP) ← op2
ELSE
        next instruction
END IF
```

Description

If the condition specified by op1 is met, a branch to the location specified indirectly by the second operand op2 is taken. The value of the instruction pointer IP is placed onto the system stack. Because the IP always points to the instruction following the branch instruction, the value stored in the system stack represents the return address of the calling routine. If the condition is not met, no action is taken and the next instruction is executed normally.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
CALLI	cc , [Rw_n]	AB cn	2

CALLR Call Subroutine Relative CALLR

Group Call Instructions

Syntax **CALLR op1**

Source Operand(s) op1 → 8-bit signed displacement

Destination Operand(s) none

Operation

(SP) ← (SP) - 2
((SP)) ← (IP)
(IP) ← (IP) + 2*sign_extend(op1)

Description

A branch is taken to the location specified by the instruction pointer IP plus the relative displacement op1. The displacement is a two's complement number which is sign extended and counts the relative distance in **words**. The value of the instruction pointer (IP) is placed into the system stack. Because the IP always points to the instruction following the branch instruction, the value stored in the system stack represents the return address of the calling routine. The value of the IP used in the target address calculation is the address of the instruction following the CALLR instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
CALLR	rel	BB rr	2

CALLS Call Inter-Segment Subroutine CALLS

Group Call Instructions

Syntax **CALLS op1, op2**

Source Operand(s) op1 → segment number
op2 → 16-bit address offset

Destination Operand(s) none

Operation

```
(SP) ← (SP) - 2
((SP)) ← (CSP)
(SP) ← (SP) - 2
((SP)) ← (IP)
IF (CPUCON1.SGTDIS = 0) THEN
        (CSP) ←  op1
END IF
(IP) ← op2
```

Description

A branch is taken to the absolute location specified by op2 within the segment specified by op1. The previous value of the CSP is placed into the system stack to ensure correct return to the calling segment. The value of the instruction pointer (IP) is also placed into the system stack. Because the IP always points to the instruction following the branch instruction, the value stored on the system stack represents the return address to the calling routine.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
CALLS	seg , caddr	DA SS MM MM	4

CMP Integer Compare CMP

Group Boolean Bit Manipulation Instructions

Syntax **CMP op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) none

Operation

(op1) ⇔ (op2)

Description

The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2s complement binary subtraction of op2 from op1. The flags are set according to the rules of subtraction. The operands remain unchanged.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CMP	Rw_n , #data3	48 n:0###	2
CMP	Rw_n , Rw_m	40 nm	2
CMP	Rw_n , [Rw_i+]	48 n:11ii	2
CMP	Rw_n , [Rw_i]	48 n:10ii	2
CMP	reg , #data16	46 RR ## ##	4
CMP	reg , mem	42 RR MM MM	4

CMPB Integer Compare CMPB

Group Boolean Bit Manipulation Instructions

Syntax **CMPB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) none

Operation

(op1) ⇔ (op2)

Description

The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2s complement binary subtraction of op2 from op1. The flags are set according to the rules of subtraction. The operands remain unchanged.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the byte data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CMPB	Rb_n , #data3	49 n:0###	2
CMPB	Rb_n , Rb_m	41 nm	2
CMPB	Rb_n , [Rw_i+]	49 n:11ii	2
CMPB	Rb_n , [Rw_i]	49 n:10ii	2
CMPB	reg , #data8	47 RR ## xx	4
CMPB	reg , mem	43 RR MM MM	4

CMPD1 Integer Compare and Decrement by 1 CMPD1

Group Compare and Loop Control Instructions

Syntax **CMPD1 op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ⇔ (op2)
(op1) ← (op1) - 1

Description

This instruction is used to enhance the performance and flexibility of loops. The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2s complement binary subtraction of op2 from op1. Operand op1 may specify ONLY GPR registers. Once the subtraction has completed, the operand op1 is decremented by one. Using the set flags, a branch instruction can then be used in conjunction with this instruction to form common high level language FOR loops of any range.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CMPD1	Rw_n , #data16	A6 Fn ## ##	4
CMPD1	Rw_n , #data4	A0 #n	2
CMPD1	Rw_n , mem	A2 Fn MM MM	4

CMPD2 Integer Compare and Decrement by 2 CMPD2

Group Compare and Loop Control Instructions

Syntax **CMPD2 op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ⇔ (op2)
(op1) ← (op1) - 2

Description

This instruction is used to enhance the performance and flexibility of loops. The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2s complement binary subtraction of op2 from op1. Operand op1 may specify ONLY GPR registers. Once the subtraction has completed, the operand op1 is decremented by two. Using the set flags, a branch instruction can then be used in conjunction with this instruction to form common high level language FOR loops of any range.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CMPD2	Rw_n , #data16	B6 Fn ## ##	4
CMPD2	Rw_n , #data4	B0 #n	2
CMPD2	Rw_n , mem	B2 Fn MM MM	4

CMPI1 Integer Compare and Increment by 1 CMPI1

Group Compare and Loop Control Instructions

Syntax **CMPI1 op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ⇔ (op2)
(op1) ← (op1) + 1

Description

This instruction is used to enhance the performance and flexibility of loops. The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2s complement binary subtraction of op2 from op1. Operand op1 may specify ONLY GPR registers. Once the subtraction has completed, the operand op1 is incremented by one. Using the set flags, a branch instruction can then be used in conjunction with this instruction to form common high level language FOR loops of any range.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CMPI1	Rw_n , #data16	86 Fn ## ##	4
CMPI1	Rw_n , #data4	80 #n	2
CMPI1	Rw_n , mem	82 Fn MM MM	4

CMPI2 Integer Compare and Increment by 2 CMPI2

Group Compare and Loop Control Instructions

Syntax **CMPI2 op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ⇔ (op2)
(op1) ← (op1) + 2

Description

This instruction is used to enhance the performance and flexibility of loops. The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2s complement binary subtraction of op2 from op1. Operand op1 may specify ONLY GPR registers. Once the subtraction has completed, the operand op1 is incremented by two. Using the set flags, a branch instruction can then be used in conjunction with this instruction to form common high level language FOR loops of any range.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CMPI2	Rw_n , #data16	96 Fn ## ##	4
CMPI2	Rw_n , #data4	90 #n	2
CMPI2	Rw_n , mem	92 Fn MM MM	4

CPL Integer One's Complement CPL

Group Arithmetic Instructions

Syntax **CPL op1**

Source Operand(s) op1 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← ¬(op1)

Description

Performs a 1s complement of the source operand specified by op1. The result is stored back into op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CPL	Rw_n	91 n0	2

CPLB Integer One's Complement CPLB

Group Arithmetic Instructions

Syntax **CPLB op1**

Source Operand(s) op1 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← ¬(op1)

Description

Performs a 1s complement of the source operand specified by op1. The result is stored back into op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CPLB	Rb_n	B1 n0	2

DISWDT Disable Watchdog Timer DISWDT

Group System Control Instructions

Syntax **DISWDT**

Source Operand(s) none

Destination Operand(s) none

Operation

Disable the watchdog timer

Description

This instruction disables the Watchdog Timer. If the WDTCTL bit is cleared, the DISWDT instruction can be executed at any time between the Reset and the first execution of either EINIT or SRVWDT. After execution of either an EINIT or a SRVWDT, the DISWDT instruction will have no effect. If the WDTCTL bit is set, the DISWDT instruction can always be executed regardless of the execution of EINIT or SRVWDT. To ensure that this instruction is not accidentally executed, it is implemented as a protected instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
DISWDT	A5 5A A5 A5	4

DIV 16-by-16 Signed Division DIV

Group Arithmetic Instructions

Syntax **DIV op1**

Source Operand(s) op1 → WORD
MDL → WORD

Destination Operand(s) MD → DOUBLEWORD

Operation

(MDL) ← (MDL) / (op1)
(MDH) ← (MDL) mod (op1)

Description

Performs a signed 16-bit by 16-bit division of the low order word stored in the MD register by the source word operand op1. The signed quotient is then stored in the low order word of the MD register (MDL) and the remainder is stored in the high order word of the MD register (MDH).

CPU Flags

E	Z	V	C	N
0	*	*	0	*

- E Always cleared.
- Z Set if quotient, stored in the MDL register, equals zero. Cleared otherwise. Undefined if the V flag is set.
- V Set if an arithmetic overflow occurred, i.e. the quotient cannot be represented in a word data type (only in case of $8000_H/FFFE_H$), or if the divisor op1 was zero. Cleared otherwise.
- C Always cleared.
- N Set if the most significant bit of the quotient, stored in the MDL register, is set. Cleared otherwise. Undefined if the V flag is set.

Encoding

Mnemonic		Format	Bytes
DIV	Rw_n	4B nn	2

DIVL 32-by-16 Signed Division DIVL

Group Arithmetic Instructions

Syntax **DIVL op1**

Source Operand(s) op1 → WORD
MD → DOUBLEWORD

Destination Operand(s) MD → DOUBLEWORD

Operation

(MDL) ← (MD) / (op1)
(MDH) ← (MD) mod (op1)

Description

Performs an extended signed 32-bit by 16-bit division of the two words stored in the MD register by the source word operand op1. The signed quotient is then stored in the low order word of the MD register (MDL) and the remainder is stored in the high order word of the MD register (MDH).

CPU Flags

E	Z	V	C	N
0	*	*	0	*

- E Always cleared.
- Z Set if quotient, stored in the MDL register, equals zero. Cleared otherwise. Undefined if the V flag is set.
- V Set if an arithmetic overflow occurred, i.e. the quotient cannot be represented in a word data type, or if the divisor op1 was zero. Cleared otherwise.
- C Always cleared.
- N Set if the most significant bit of the quotient, stored in the MDL register, is set. Cleared otherwise. Undefined if the V flag is set.

Encoding

Mnemonic		Format	Bytes
DIVL	Rw_n	6B nn	2

DIVLU 32-by-16 Unsigned Division DIVLU

Group Arithmetic Instructions

Syntax **DIVLU op1**

Source Operand(s) op1 → WORD
MD → DOUBLEWORD

Destination Operand(s) MD → DOUBLEWORD

Operation

(MDL) ← (MD) / op1
(MDH) ← (MD) mod (op1)

Description

Performs an extended unsigned 32-bit by 16-bit division of the two words stored in the MD register by the source word operand op1. The unsigned quotient is then stored in the low order word of the MD register (MDL) and the remainder is stored in the high order word of the MD register (MDH).

CPU Flags

E	Z	V	C	N
0	*	*	0	*

E Always cleared.

Z Set if quotient, stored in the MDL register, equals zero. Cleared otherwise. Undefined if the V flag is set.

V Set if an arithmetic overflow occurred, i.e. the quotient cannot be represented in a word data type, or if the divisor op1 was zero. Cleared otherwise.

C Always cleared.

N Set if the most significant bit of the quotient, stored in the MDL register, is set. Cleared otherwise. Undefined if the V flag is set.

Encoding

Mnemonic		Format	Bytes
DIVLU	Rw_n	7B nn	2

DIVU 16-by-16 Unsigned Division DIVU

Group Arithmetic Instructions

Syntax **DIVU op1**

Source Operand(s) op1 → WORD
MDL → WORD

Destination Operand(s) MD → DOUBLEWORD

Operation

(MDL) ← (MDL) / (op1)
(MDH) ← (MDL) mod (op1)

Description

Performs an unsigned 16-bit by 16-bit division of the low order word stored in the MD register by the source word operand op1. The unsigned quotient is then stored in the low order word of the MD register (MDL) and the remainder is stored in the high order word of the MD register (MDH).

CPU Flags

E	Z	V	C	N
0	*	*	0	*

- E Always cleared.
- Z Set if quotient, stored in the MDL register, equals zero. Cleared otherwise. Undefined if the V flag is set.
- V Set if the divisor op1 was zero.
- C Always cleared.
- N Set if the most significant bit of the quotient, stored in the MDL register, is set. Cleared otherwise. Undefined if the V flag is set.

Encoding

Mnemonic		Format	Bytes
DIVU	Rw_n	5B nn	2

EINIT End of Initialization EINIT

Group System Control Instructions

Syntax **EINIT**

Source Operand(s) none

Destination Operand(s) none

Operation

End of Initialization

Description

After a reset, the reset output pin RSTOUT is pulled low. It remains low until the EINIT instruction has been executed at which time it goes high. This enables the software to signal the external circuitry that it has successfully initialized the microcontroller. After EINIT execution, registers can be locked until reset. The DISWDT instruction executed after the first EINIT instruction has effect only if the WDTCTL bit was cleared before the EINIT instruction. To ensure that this instruction is not accidentally executed, it is implemented as a protected instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
EINIT	B5 4A B5 B5	4

ENWDT Enable Watchdog Timer ENWDT

Group System Control Instructions

Syntax **ENWDT**

Source Operand(s) none

Destination Operand(s) none

Operation

Enable Watchdog Timer

Description

If the WDTCTL bit of the CPUCON1 register is cleared, this instruction has no effect. If the WDTCTL bit is set, this instruction enables the Watchdog Timer. Specifically, it allows the Watchdog Timer to be re-enabled after it has been previously disabled by a DISWDT instruction. To ensure that this instruction is not accidentally executed, it is implemented as a protected instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
ENWDT	85 7A 85 85	4

EXTP Begin EXTended Page Sequence EXTP

Group System Control Instructions

Syntax **EXTP op1, op2**

Source Operand(s) op1 → 10-bit page number
op2 → 2-bit instruction counter

Destination Operand(s) none

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Page ← (op1)
DO WHILE ((count) ≠ 0 AND Class_B_Trap_Condition ≠ TRUE)
Next Instruction
(count) ← (count) - 1
END WHILE
(count) ← 0
Data_Page ← (DPPx)
Enable interrupts and traps

Description

Overrides the standard DPP addressing scheme of the long and indirect addressing modes for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. The EXTP instruction becomes active immediately such that no additional NOPs are required. For any long (’mem’) or indirect ([...]) address in the EXTP instruction sequence, the 10-bit page number (address bits A23-A14) is not determined by the contents of a DPP register, but by the value of op1 itself. The 14-bit page offset (address bits A13-A0) is derived from the long or indirect address as usual. The value of op2 defines the length of the affected instruction sequence.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
EXTP	#pag , #irang2	D7 :01##-0 pp 0:00pp	4
EXTP	Rw_m , #irang2	DC :01##-m	2

EXTPR Begin EXTended Page and Register Sequence EXTPR

Group System Control Instructions

Syntax **EXTPR op1, op2**

Source Operand(s) op1 → 10-bit page number
op2 → 2-bit instruction counter

Destination Operand(s) none

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Page ← (op1)
SFR_range ← Extended
DO WHILE ((count) ≠ 0 AND Class_B_Trap_Condition ≠ TRUE)
    Next Instruction
    (count) ← (count) - 1
END WHILE
(count) ← 0
Data_Page ← (DPPx)
SFR_range ← Standard
Enable interrupts and traps

Description

Overrides the standard DPP addressing scheme of the long and indirect addressing modes and causes all SFR or SFR bit accesses via the 'reg', 'bitoff' or 'bitaddr' addressing modes being made to the Extended SFR space for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. For any long ('mem') or indirect ([...]) address in the EXTP instruction sequence, the 10-bit page number (address bits A23-A14) is not determined by the contents of a DPP register, but by the value of op1 itself. The 14-bit page offset (address bits A13-A0) is derived from the long or indirect address as usual. The value of op2 defines the length of the affected instruction sequence.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.

C	Not affected.
N	Not affected.

Encoding

Mnemonic		Format	Bytes
EXTPR	#pag , #irang2	D7 :11##-0 pp 0:00pp	4
EXTPR	Rw_m , #irang2	DC :11##-m	2

EXTR Begin EXTended Register Sequence EXTR

Group System Control Instructions

Syntax **EXTR op1**

Source Operand(s) op1 → 2-bit instruction counter

Destination Operand(s) none

Operation

(count) ← (op1) [1 ≤ op1 ≤ 4]
Disable interrupts and Class A traps
SFR_range ← Extended
DO WHILE ((count) ≠ 0 AND Class_B_Trap_Condition ≠ TRUE)
    Next Instruction
    (count) ← (count) - 1
END WHILE
(count) ← 0
SFR_range ← Standard
Enable interrupts and traps

Description

Causes all SFR or SFR bit accesses via the 'reg', 'bitoff' or 'bitaddr' addressing modes being made to the Extended SFR space for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. The value of op1 defines the length of the affected instruction sequence.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
EXTR	#irang2	D1 :10##-0	2

EXTS Begin EXTended Segment Sequence EXTS

Group System Control Instructions

Syntax **EXTS op1, op2**

Source Operand(s) op1 → segment number
op2 → 2-bit instruction counter

Destination Operand(s) none

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Segment ← (op1)
DO WHILE ((count) ≠ 0 AND Class_B_Trap_Condition ≠ TRUE)
    Next Instruction
    (count) ← (count) - 1
END WHILE
(count) ← 0
Data_Page ← (DPPx)
Enable interrupts and traps

Description

Overrides the standard DPP addressing scheme of the long and indirect addressing modes for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. The EXTS instruction becomes immediately active such that no additional NOPs are required. For any long (’mem’) or indirect ([...]) address in an EXTS instruction sequence, the value of op1 determines the 8-bit segment (address bits A23-A16) valid for the corresponding data access. The long or indirect address itself represents the 16-bit segment offset (address bits A15-A0). The value of op2 defines the length of the affected instruction sequence.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
EXTS	#seg , #irang2	D7 :00##-0 ss 00	4
EXTS	Rw_m , #irang2	DC :00##-m	2

EXTSR Begin EXTended Segment and Register Sequence EXTSR

Group System Control Instructions

Syntax **EXTSR op1, op2**

Source Operand(s) op1 → segment number
op2 → 2-bit instruction counter

Destination Operand(s) none

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Segment ← (op1)
SFR_range ← Extended
DO WHILE ((count) ≠ 0 AND Class_B_Trap_Condition ≠ TRUE)
Next Instruction
(count) ← (count) - 1
END WHILE
(count) ← 0
Data_Page ← (DPPx)
SFR_range ← Standard
Enable interrupts and traps

Description

Overrides the standard DPP addressing scheme of the long and indirect addressing modes and causes all SFR or SFR bit accesses via the 'reg', 'bitoff' or 'bitaddr' addressing modes being made to the Extended SFR space for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. The EXTSR instruction becomes immediately active such that no additional NOPs are required. For any long ('mem') or indirect ([...]) address in an EXTSR instruction sequence, the value of op1 determines the 8-bit segment (address bits A23-A16) valid for the corresponding data access. The long or indirect address itself represents the 16-bit segment offset (address bits A15-A0). The value of op2 defines the length of the affected instruction sequence.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V	Not affected.
C	Not affected.
N	Not affected.

Encoding

Mnemonic		Format	Bytes
EXTSR	#seg , #irang2	D7 :10##-0 ss 00	4
EXTSR	Rw_m , #irang2	DC :10##-m	2

IDLE Enter Idle Mode IDLE

Group System Control Instructions

Syntax **IDLE**

Source Operand(s) none

Destination Operand(s) none

Operation

Enter Idle Mode

Description

This instruction causes the part to enter the idle mode. In this mode, the CPU is powered down while the peripherals remain running. It remains powered down until a peripheral interrupt or external interrupt occurs. To ensure that this instruction is not accidentally executed, it is implemented as a protected instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
IDLE	87 78 87 87	4

JB Relative Jump if Bit Set JB

Group Jump Instructions

Syntax **JB op1, op2**

Source Operand(s) op1 → BIT
op2 → 8-bit signed displacement

Destination Operand(s) none

Operation

```
IF ((op1) = 1) THEN
        (IP) ← (IP) + 2*sign_extend(op2)
ELSE
        Next Instruction
END IF
```

Description

If the bit specified by op1 is set, program execution continues at the location of the instruction pointer IP, plus the specified displacement op2. The displacement is a 2s complement number which is sign extended and counts the relative distance in **words**. The value of the IP used in the target address calculation is the address of the instruction following the JB instruction. If the specified bit is cleared, program execution continues normally with the instruction following the JB instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
JB	$bitaddr_{Q.q}$, rel	8A QQ rr q0	4

JBC Relative Jump if Bit Set and Clear Bit JBC

Group Jump Instructions

Syntax **JBC op1, op2**

Source Operand(s) op1 → BIT
op2 → 8-bit signed displacement

Destination Operand(s) none

Operation

```
IF ((op1) = 1) THEN
        (op1) ← 0
        (IP) ← (IP) + 2*sign_extend(op2)
ELSE
        Next Instruction
END IF
```

Description

If the bit specified by op1 is set, program execution continues at the location of the instruction pointer IP, plus the specified displacement op2. The bit specified by op1 is cleared, allowing implementation of semaphore operations. The displacement is a 2s complement number which is sign extended and counts the relative distance in **words**. The value of the IP used in the target address calculation is the address of the instruction following the JBC instruction. If the specified bit was clear, program execution continues normally with the instruction following the JBC instruction.

Note: Flags are updated by this instruction even if the branch is not executed. An explicit write operation to the PSW register supersedes the condition flag values which are implicitly generated by the CPU.

CPU Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

E Always cleared.
Z Contains the logical negation of the previous state of the specified bit.
V Always cleared.
C Always cleared.
N Contains the previous state of the specified bit.

Encoding

Mnemonic		Format	Bytes
JBC	$bitaddr_{Q.q}$, rel	AA QQ rr q0	4

JMPA Absolute Conditional Jump JMPA

Group Jump Instructions

Syntax **JMPA op1, op2**

Alternative Syntax JMPA+ op1, op2
JMPA- op1, op2

Source Operand(s) op1 → extended condition code
op2 → 16-bit address offset

Destination Operand(s) none

Operation

```
IF ((op1) = 1) THEN
        (IP) ← op2
ELSE
        Next Instruction
END IF
```

Description

If the condition specified by op1 is met, a branch to the absolute address specified by op2 is taken. If the condition is not met, no action is taken, and the instruction following the JMPA instruction is executed normally. A static prediction scheme is used: if the prediction bit 'a' of the instruction long word is cleared then JMPA is assumed 'taken' and if this bit is set to 1 JMPA is assumed 'not taken'. JMPA+ and JMPA- instructions are converted into JMPA assumed 'taken' (bit 'a' cleared) and 'not taken' (bit 'a' set) respectively. For regular JMPA instructions, the assembler applies the following rule: cc_z is predicted 'not taken' meanwhile all other conditions are predicted 'taken'. A prefetch hint bit is also used. This bit is the instruction long word bit 'l' and is required by the fetch unit to deal efficiently with short backward loops. It must be set only if (0 < IP_jmpa - IP_target ≤ 32), cleared otherwise. IP_jmpa is the address of the JMPA instruction and IP_target is the target address of JMPA.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.

C	Not affected.
N	Not affected.

Encoding

Mnemonic		Format	Bytes
JMPA	xcc , caddr	EA d0la MM MM	4

JMPI Indirect Conditional Jump JMPI

Group Jump Instructions

Syntax **JMPI op1, op2**

Source Operand(s) op1 → condition code
op2 → 16-bit address offset

Destination Operand(s) none

Operation

```
IF ((op1) = 1) THEN
        (IP) ← (op2)
ELSE
        Next Instruction
END IF
```

Description

If the condition specified by op1 is met, a branch to the absolute address specified by op2 is taken. If the condition is not met, no action is taken, and program execution continues normally with the instruction following the JMPI instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
JMPI	cc , [Rw_n]	9C cn	2

JMPR Relative Conditional Jump JMPR

Group Jump Instructions

Syntax **JMPR op1, op2**

Source Operand(s) op1 → condition code
op2 → 8-bit signed displacement

Destination Operand(s) none

Operation

```
IF ((op1) = 1) THEN
        (IP) ← (IP) + 2*sign_extend(op2)
ELSE
        Next Instruction
END IF
```

Description

If the extended condition specified by op1 is met, program execution continues at the location of the instruction pointer, IP, plus the specified displacement, op2. The displacement is a 2s complement number which is sign-extended and counts the relative distance in words. The value of the IP used in the target address calculation is the address of the instruction following the JMPR instruction. If the specified condition is not met, program execution continues normally with the instruction following the JMPR instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
JMPR	cc , rel	cD rr	2

JMPS Absolute Inter-Segment Jump JMPS

Group Jump Instructions

Syntax **JMPS op1, op2**

Source Operand(s) op1 → segment number
op2 → 16-bit address offset

Destination Operand(s) none

Operation

```
IF (CPUCON1.SGTDIS = 0) THEN
        (CSP) ← op1
END IF
(IP) ← op2
```

Description
Branches unconditionally to the absolute address specified by op2 within the segment specified by op1.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
JMPS	seg , caddr	FA SS MM MM	4

JNB Relative Jump if Bit Clear JNB

Group Jump Instructions

Syntax **JNB op1, op2**

Source Operand(s) op1 → BIT
op2 → 8-bit signed displacement

Destination Operand(s) none

Operation

```
IF ((op1) = 0) THEN
        (IP) ← (IP) + 2*sign_extend(op2)
ELSE
        Next Instruction
END IF
```

Description

If the bit specified by op1 is clear, program execution continues at the location of the instruction pointer IP, plus the specified displacement op2. The displacement is a 2s complement number which is sign-extended and counts the relative distance in words. The value of the IP used in the target address calculation is the address of the instruction following the JNB instruction. If the specified bit is set, program execution continues normally with the instruction following the JNB instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
JNB	$bitaddr_{Q.q}$, rel	9A QQ rr q0	4

JNBS Relative Jump if Bit Clear and Set Bit JNBS

Group Jump Instructions

Syntax **JNBS op1, op2**

Source Operand(s) op1 → BIT
op2 → 8-bit signed displacement

Destination Operand(s) none

Operation

```
IF ((op1) = 0) THEN
        (op1) ← 1
        (IP) ← (IP) + 2*sign_extend(op2)
ELSE
        Next Instruction
END IF
```

Description

If the bit specified by op1 is clear, program execution continues at the location of the instruction pointer IP, plus the specified displacement op2. The bit specified by op1 is set, allowing implementation of semaphore operations. The displacement is a 2s complement number which is sign-extended and counts the relative distance in words. The value of the IP used in the target address calculation is the address of the instruction following the JNBS instruction. If the specified bit was set, program execution continues normally with the instruction following the JNBS instruction.

Note: Flags are updated by this instruction even if the branch is not executed. An explicit write operation to the PSW register supersedes the condition flag values which are implicitly generated by the CPU.

CPU Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

E Always cleared.
Z Contains the logical negation of the previous state of the specified bit.
V Always cleared.
C Always cleared.
N Contains the previous state of the specified bit.

Encoding

Mnemonic		Format	Bytes
JNBS	$bitaddr_{Q.q}$, rel	BA QQ rr q0	4

MOV Move Data MOV

Group Data Movement Instructions

Syntax **MOV op1, op2**

Source Operand(s) op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op2)

Description

Moves the contents of the source operand specified by op2 to the location specified by the destination operand op1. The contents of the moved data are examined, and the flags are updated accordingly.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if the value of the source operand op2 equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the source operand op2 is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
MOV	Rw_n , #data4	E0 #n	2
MOV	Rw_n , Rw_m	F0 nm	2
MOV	Rw_n , [Rw_m+#data16]	D4 nm ## ##	4
MOV	Rw_n , [Rw_m+]	98 nm	2
MOV	Rw_n , [Rw_m]	A8 nm	2
MOV	[-Rw_m] , Rw_n	88 nm	2
MOV	[Rw_m+#data16] , Rw_n	C4 nm ## ##	4

MOV	[Rw_m] , Rw_n	B8 nm	2
MOV	[Rw_n+] , [Rw_m]	D8 nm	2
MOV	[Rw_n] , [Rw_m+]	E8 nm	2
MOV	[Rw_n] , [Rw_m]	C8 nm	2
MOV	[Rw_n] , mem	84 0n MM MM	4
MOV	mem , [Rw_n]	94 0n MM MM	4
MOV	mem , reg	F6 RR MM MM	4
MOV	reg , #data16	E6 RR ## ##	4
MOV	reg , mem	F2 RR MM MM	4

MOVB Move Data MOVB

Group Data Movement Instructions

Syntax **MOVB op1, op2**

Source Operand(s) op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op2)

Description

Moves the contents of the source operand specified by op2 to the location specified by the destination operand op1. The contents of the moved data are examined, and the flags are updated accordingly.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if the value of the source operand op2 equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the source operand op2 is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
MOVB	Rb_n , #data4	E1 #n	2
MOVB	Rb_n , Rb_m	F1 nm	2
MOVB	Rb_n , [Rw_m + #data16]	F4 nm ## ##	4
MOVB	Rb_n , [Rw_m+]	99 nm	2
MOVB	Rb_n , [Rw_m]	A9 nm	2
MOVB	[-Rw_m] , Rb_n	89 nm	2
MOVB	[Rw_m + #data16] , Rb_n	E4 nm ## ##	4

MOVB	[Rw_m] , Rb_n	B9 nm	2
MOVB	[Rw_n+] , [Rw_m]	D9 nm	2
MOVB	[Rw_n] , [Rw_m+]	E9 nm	2
MOVB	[Rw_n] , [Rw_m]	C9 nm	2
MOVB	[Rw_n] , mem	A4 0n MM MM	4
MOVB	mem , [Rw_n]	B4 0n MM MM	4
MOVB	mem , reg	F7 RR MM MM	4
MOVB	reg , #data8	E7 RR ## xx	4
MOVB	reg , mem	F3 RR MM MM	4

MOVBS Move Byte Sign Extend MOVBS

Group Data Movement Instructions

Syntax **MOVBS op1, op2**

Source Operand(s) op2 → BYTE

Destination Operand(s) op1 → WORD

Operation

```
(low byte op1) ← (op2)
IF ((op2[7]) = 1) THEN
        (high byte op1) ← FFH
ELSE
        (high byte op1) ← 00H
END IF
```

Description

Moves and sign-extends the contents of the source byte operand specified by op2 to the word location specified by the destination operand op1. The contents of the moved data are examined, and the flags are updated accordingly.

CPU Flags

E	Z	V	C	N
0	*	-	-	*

- E Always cleared.
- Z Set if the value of the source byte operand op2 equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the source operand op2 is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
MOVBS	Rw_n , Rb_m	D0 mn	2
MOVBS	mem , reg	D5 RR MM MM	4
MOVBS	reg , mem	D2 RR MM MM	4

MOVBZ Move Byte Zero Extend MOVBZ

Group Data Movement Instructions

Syntax **MOVBZ op1, op2**

Source Operand(s) op2 → BYTE

Destination Operand(s) op1 → WORD

Operation

(low byte op1) ← (op2)
(high byte op1) ← 00H

Description

Moves and zero-extends the contents of the source byte operand specified by op2 to the word location specified by the destination operand op1. The contents of the moved data are examined, and the flags are updated accordingly.

CPU Flags

E	Z	V	C	N
0	*	-	-	0

- E Always cleared.
- Z Set if the value of the source byte operand op2 equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Always cleared.

Encoding

Mnemonic		Format	Bytes
MOVBZ	Rw_n , Rb_m	C0 mn	2
MOVBZ	mem , reg	C5 RR MM MM	4
MOVBZ	reg , mem	C2 RR MM MM	4

MUL Signed Multiplication MUL

Group Arithmetic Instructions

Syntax **MUL op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) MD → DOUBLEWORD

Operation

(MD) ← (op1) * (op2)

Description

Performs a 16-bit by 16-bit signed multiplication using the two words specified by operands op1 and op2 respectively. The signed 32-bit result is placed in the MD register.

CPU Flags

E	Z	V	C	N
0	*	*	0	*

- E Always cleared.
- Z Set if result equals zero. Cleared otherwise.
- V This bit is set if the result cannot be represented in a word data type. Cleared otherwise.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
MUL	Rw_n , Rw_m	0B nm	2

MULU Unsigned Multiplication MULU

Group Arithmetic Instructions

Syntax **MULU op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) MD → DOUBLEWORD

Operation

(MD) ← (op1) * (op2)

Description

Performs a 16-bit by 16-bit unsigned multiplication using the two words specified by operands op1 and op2 respectively. The unsigned 32-bit result is placed in the MD register.

CPU Flags

E	Z	V	C	N
0	*	*	0	*

- E Always cleared.
- Z Set if result equals zero. Cleared otherwise.
- V This bit is set if the result cannot be represented in a word data type. Cleared otherwise.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
MULU	Rw_n , Rw_m	1B nm	2

NEG Integer Two's Complement NEG

Group Arithmetic Instructions

Syntax **NEG op1**

Source Operand(s) op1 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← 0 - (op1)

Description

Performs a binary 2s complement of the source operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	*	*	*

- E Set if the value of op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
NEG	Rw_n	81 n0	2

NEGB Integer Two's Complement NEGB

Group Arithmetic Instructions

Syntax **NEGB op1**

Source Operand(s) op1 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← 0 - (op1)

Description

Performs a binary 2s complement of the source operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	*	*	*

- E Set if the value of op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the byte data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
NEGB	Rb_n	A1 n0	2

NOP No Operation NOP

Group Null operation

Syntax **NOP**

Source Operand(s) none

Destination Operand(s) none

Operation

No Operation

Description

This instruction causes a null operation to be performed. A null operation causes no change in the status of the flags.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
NOP	CC 00	2

OR — Logical OR — OR

Group Logical Instructions

Syntax **OR op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) ∨ (op2)

Description

Performs a bitwise logical OR of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
OR	Rw_n , #data3	78 n:0###	2
OR	Rw_n , Rw_m	70 nm	2
OR	Rw_n , [Rw_i+]	78 n:11ii	2
OR	Rw_n , [Rw_i]	78 n:10ii	2
OR	mem , reg	74 RR MM MM	4
OR	reg , #data16	76 RR ## ##	4
OR	reg , mem	72 RR MM MM	4

ORB Logical OR ORB

Group Logical Instructions

Syntax **ORB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) ∨ (op2)

Description

Performs a bitwise logical OR of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ORB	Rb_n , #data3	79 n:0###	2
ORB	Rb_n , Rb_m	71 nm	2
ORB	Rb_n , [Rw_i+]	79 n:11ii	2
ORB	Rb_n , [Rw_i]	79 n:10ii	2
ORB	mem , reg	75 RR MM MM	4
ORB	reg , #data8	77 RR ## xx	4
ORB	reg , mem	73 RR MM MM	4

PCALL Push Word and Call Subroutine Absolute PCALL

Group Call Instructions

Syntax **PCALL op1, op2**

Source Operand(s) op1 → WORD
op2 → 16-bit address offset

Destination Operand(s) none

Operation

(tmp) ← (op1)
(SP) ← (SP) - 2
((SP)) ← (tmp)
(SP) ← (SP) - 2
((SP)) ← (IP)
(IP) ← op2

Description

Pushes the word specified by operand op1 and the value of the instruction pointer, IP, onto the system stack, and branches to the absolute memory location specified by the second operand op2. Because IP always points to the instruction following the branch instruction, the value stored on the system stack represents the return address of the calling routine.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

E Set if the value of the pushed operand op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.

Z Set if the value of the pushed operand op1 equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Set if the most significant bit of the pushed operand op1 is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
PCALL	reg , caddr	E2 RR MM MM	4

POP Pop Word from System Stack POP

Group System Stack Instructions

Syntax **POP op1**

Source Operand(s) none

Destination Operand(s) op1 → WORD

Operation

(tmp) ← ((SP))
(SP) ← (SP) + 2
(op1) ← (tmp)

Description

Pops one word from the system stack specified by the Stack Pointer into the operand specified by op1. The Stack Pointer is then incremented by two.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

- E Set if the value of the popped word represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if the value of the popped word equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the popped word is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
POP	reg	FC RR	2

PRIOR Prioritize Register PRIOR

Group Prioritize Instruction

Syntax **PRIOR op1, op2**

Source Operand(s) op2 → WORD

Destination Operand(s) op1 → WORD

Operation

```
(tmp) ← (op2)
(count) ← 0
DO WHILE (((tmp[15]) ≠ 1) AND ((op2) ≠ 0)))
        (tmp[n]) ← (tmp[n-1]) [n=15...1]
        (count) ← (count) + 1
END WHILE
(op1) ← (count)
```

Description

This instruction stores a count value in the word operand specified by op1. This count value indicates the number of single bit shifts required to normalize the word operand op2 so that its most significant bit is equal to one. If the source operand op2 equals zero, a zero is written to operand op1 and the zero flag is set. Otherwise, the zero flag is cleared.

CPU Flags

E	Z	V	C	N
0	*	0	0	0

- E Always cleared.
- Z Set if the value of the source operand op2 equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Always cleared.

Encoding

Mnemonic		Format	Bytes
PRIOR	Rw_n , Rw_m	2B nm	2

PUSH Push Word on System Stack PUSH

Group System Stack Instructions

Syntax **PUSH op1**

Source Operand(s) op1 → WORD

Destination Operand(s) none

Operation

(tmp) ← (op1)
(SP) ← (SP) - 2
((SP)) ← (tmp)

Description

Moves the word specified by operand op1 to the location in the system stack specified by the Stack Pointer, after the Stack Pointer has been decremented by two.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

- E Set if the value of the pushed operand op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if the value of the pushed operand op1 equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the pushed operand op1 is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
PUSH	reg	EC RR	2

PWRDN Enter Power Down Mode PWRDN

Group System Control Instructions

Syntax **PWRDN**

Source Operand(s) none

Destination Operand(s) none

Operation

Enter Power Down Mode

Description

This instruction causes the device to enter the power down mode. In this mode, all peripherals and the CPU are powered down until the device is externally reset. To ensure that this instruction is not accidentally executed, it is implemented as a protected instruction. To further control the action of this instruction, the PWRDN instruction is only enabled when the non-maskable interrupt pin (NMI) is in the low state. Otherwise, this instruction has no effect.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
PWRDN	97 68 97 97	4

RET Return from Subroutine RET

Group Return Instructions

Syntax **RET**

Source Operand(s) none

Destination Operand(s) none

Operation

(IP) ← ((SP))
(SP) ← (SP) + 2

Description

Returns from a subroutine. The IP is popped from the system stack.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
RET	CB 00	2

RETI Return from Interrupt Subroutine RETI

Group Return Instructions

Syntax **RETI**

Source Operand(s) none

Destination Operand(s) none

Operation

```
(IP) ← ((SP))
(SP) ← (SP) + 2
IF (CPUCON1.SGTDIS = 0) THEN
        (CSP) ← ((SP))
        (SP) ← (SP) + 2
END IF
(PSW) ← ((SP))
(SP) ← (SP) + 2
```

Description

Returns from an interrupt routine. The IP, CSP, and PSW are popped off the system stack. The CSP is only popped if segmentation is enabled. This is indicated by the SGTDIS bit in the CPUCON1 register.

CPU Flags

E	Z	V	C	N
*	*	*	*	*

E Restored from the PSW popped from stack.
Z Restored from the PSW popped from stack.
V Restored from the PSW popped from stack.
C Restored from the PSW popped from stack.
N Restored from the PSW popped from stack.

Encoding

Mnemonic	Format	Bytes
RETI	FB 88	2

RETP Return from Subroutine and Pop Word RETP

Group Return Instructions

Syntax **RETP op1**

Source Operand(s) none

Destination Operand(s) op1 → WORD

Operation

(IP) ← ((SP))
(SP) ← (SP) + 2
(tmp) ← ((SP))
(SP) ← (SP) + 2
(op1) ← (tmp)

Description

Returns from a subroutine. First the IP is popped from the system stack and then the next word is popped from the system stack into the operand specified by op1.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

- E Set if the value of the popped word represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if the value of the popped word equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the popped word is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
RETP	reg	EB RR	2

RETS Return from Inter-Segment Subroutine RETS

Group Return Instructions

Syntax **RETS**

Source Operand(s) none

Destination Operand(s) none

Operation

```
(IP) ← ((SP))
(SP) ← (SP) + 2
IF (CPUCON1.SGTDIS = 0) THEN
        (CSP) ← ((SP))
END IF
(SP) ← (SP) + 2
```

Description
Returns from an inter-segment subroutine. The IP and CSP are popped from the system stack.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
RETS	DB 00	2

ROL Rotate Left ROL

Group Shift and Rotate Instructions

Syntax **ROL op1, op2**

Source Operand(s) op1 → WORD
op2 → shift counter

Destination Operand(s) op1 → WORD

Operation

```
(count) ← (op2)
(C) ← 0
DO WHILE ((count) ≠ 0)
        (C) ← (op1[15])
        (op1[n]) ← (op1[n-1]) [n=15...1]
        (op1[0]) ← (C)
        (count) ← (count) - 1
END WHILE
```

Description
Rotates the destination word operand op1 the number of times as specified by the source operand op2. Bit 15 is rotated into Bit 0 and into the Carry. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant four bits are used.

CPU Flags

E	Z	V	C	N
0	*	0	S	*

E Always cleared.
Z Set if result equals zero. Cleared otherwise.
V Always cleared.
C The carry flag is set according to the last most significant bit shifted out of op1. Cleared for a shift count of zero.
N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ROL	Rw_n , #data4	1C #n	2
ROL	Rw_n , Rw_m	0C nm	2

ROR Rotate Right ROR

Group Shift and Rotate Instructions

Syntax **ROR op1, op2**

Source Operand(s) op1 → WORD
op2 → shift counter

Destination Operand(s) op1 → WORD

Operation

```
(count) ← (op2)
(C) ← 0
(V) ← 0
DO WHILE ((count) ≠ 0)
        (V) ← (V) ∨ (C)
        (C) ← (op1[0])
        (op1[n]) ← (op1[n+1]) [n=0...14]
        (op1[15]) ← (C)
        (count) ← (count) - 1
END WHILE
```

Description
Rotates the destination word operand op1 right by the number of times as specified by the source operand op2. Bit 0 is rotated into Bit 15 and into the Carry. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant four bits are used.

CPU Flags

E	Z	V	C	N
0	*	S	S	*

- E Always cleared.
- Z Set if result equals zero. Cleared otherwise.
- V Set if in any cycle of the rotate operation a 1 is shifted out of the carry flag. Cleared for a rotate count of zero.
- C The carry flag is set according to the last least significant bit shifted out of op1. Cleared for a shift count of zero.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ROR	Rw_n , #data4	3C #n	2
ROR	Rw_n , Rw_m	2C nm	2

SBRK Software Break SBRK

Group System Control Instructions

Syntax **SBRK**

Source Operand(s) none

Destination Operand(s) none

Operation

Software Break

Description

If the SBRK instruction is enabled by the One Chip Emulator (OCE), then the break mode is activated. If SBRK is not enabled by the OCE, then the hardware trap "soft break" (Class A, Vector 8) is activated. For more details about this instruction, see the OCE specifications.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
SBRK	8C 00	2

SCXT Switch Context SCXT

Group System Stack Instructions

Syntax **SCXT op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(tmp1) ← (op1)
(tmp2) ← (op2)
(SP) ← (SP) - 2
((SP)) ← (tmp1)
(op1) ← (tmp2)

Description

Switches contexts of any register. Switching context is a push and load operation. The contents of the register specified by the first operand op1, are pushed onto the stack. That register is then loaded with the value specified by the second operand, op2.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
SCXT	reg , #data16	C6 RR ## ##	4
SCXT	reg , mem	D6 RR MM MM	4

SHL Shift Left SHL

Group Shift and Rotate Instructions

Syntax **SHL op1, op2**

Source Operand(s) op1 → WORD
op2 → shift counter

Destination Operand(s) op1 → WORD

Operation

```
(count) ← (op2)
(C) ← 0
DO WHILE ((count) ≠ 0)
        (C) ← (op1[15])
        (op1[n]) ← (op1[n-1]) [n=15...1]
        (op1[0]) ← 0
        (count) ← (count) - 1
END WHILE
```

Description

Shifts the destination word operand op1 the number of times as specified by the source operand op2. The least significant bits of the result are filled with zeros accordingly. The most significant bit is shifted into the Carry. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant four bits are used.

CPU Flags

E	Z	V	C	N
0	*	0	S	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V Always cleared.

C The carry flag is set according to the last most significant bit shifted out of op1. Cleared for a shift count of zero.

N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
SHL	Rw_n , #data4	5C #n	2
SHL	Rw_n , Rw_m	4C nm	2

SHR Shift Right SHR

Group Shift and Rotate Instructions

Syntax **SHR op1, op2**

Source Operand(s) op1 → WORD
op2 → shift counter

Destination Operand(s) op1 → WORD

Operation

```
(count) ← (op2)
(C) ← 0
(V) ← 0
DO WHILE ((count) ≠ 0)
        (V) ← (C) ∨ (V)
        (C) ← (op1[0])
        (op1[n]) ← (op1[n+1]) [n=0...14]
        (op1[15]) ← 0
        (count) ← (count) - 1
END WHILE
```

Description

Shifts the destination word operand op1 right by the number of times as specified by the source operand op2. The most significant bits of the result are filled with zeros accordingly. Since the bits shifted out effectively represent the remainder, the Overflow flag is used instead as a Rounding flag. A shift right is a division by a power of two. The overflow flag with the carry flag allows determination of whether the fractional part of the division result is greater than, less than, or equal to one half (0.5 in decimal base). This allows rounding of the division result accordingly. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant four bits are used.

CPU Flags

E	Z	V	C	N
0	*	S	S	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V Set if in any cycle of the shift operation a 1 is shifted out of the carry flag. Cleared in case of a shift count equal 0.

C	The carry flag is set according to the last least significant bit shifted out of op1. Cleared for a shift count of zero.
N	Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
SHR	Rw_n , #data4	7C #n	2
SHR	Rw_n , Rw_m	6C nm	2

SRST Software Reset SRST

Group System Control Instructions

Syntax **SRST**

Source Operand(s) none

Destination Operand(s) none

Operation

Software Reset

Description

This instruction is used to perform a software reset. A software reset has the same effect on the microcontroller as an externally applied hardware reset. To ensure that this instruction is not accidentally executed, it is implemented as a protected instruction.

CPU Flags

E	Z	V	C	N
0	0	0	0	0

E Always cleared.
Z Always cleared.
V Always cleared.
C Always cleared.
N Always cleared.

Encoding

Mnemonic	Format	Bytes
SRST	B7 48 B7 B7	4

SRVWDT Service Watchdog Timer SRVWDT

Group System Control Instructions

Syntax **SRVWDT**

Source Operand(s) none

Destination Operand(s) none

Operation

Service Watchdog Timer

Description

This instruction reloads the high order byte of the Watchdog Timer with a preset value and clears the low byte. After this instruction has been executed and if the WDTCTL bit of the CPUCON1 register is cleared, the Watchdog Timer cannot be disabled regardless of the execution of SRVWDT. If the WDTCTL bit is set, the Watchdog Timer can still be disabled. To ensure that this instruction is not accidentally executed, it is implemented as a protected instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
SRVWDT	A7 58 A7 A7	4

SUB Integer Subtraction SUB

Group Arithmetic Instructions

Syntax **SUB op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) - (op2)

Description

Performs a 2s complement binary subtraction of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
SUB	Rw_n , #data3	28 n:0###	2
SUB	Rw_n , Rw_m	20 nm	2
SUB	Rw_n , [Rw_i+]	28 n:11ii	2
SUB	Rw_n , [Rw_i]	28 n:10ii	2
SUB	mem , reg	24 RR MM MM	4
SUB	reg , #data16	26 RR ## ##	4
SUB	reg , mem	22 RR MM MM	4

SUBB Integer Subtraction SUBB

Group Arithmetic Instructions

Syntax **SUBB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) - (op2)

Description

Performs a 2s complement binary subtraction of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
SUBB	Rb_n , #data3	29 n:0###	2
SUBB	Rb_n , Rb_m	21 nm	2
SUBB	Rb_n , [Rw_i+]	29 n:11ii	2
SUBB	Rb_n , [Rw_i]	29 n:10ii	2
SUBB	mem , reg	25 RR MM MM	4
SUBB	reg , #data8	27 RR ## xx	4
SUBB	reg , mem	23 RR MM MM	4

SUBC Integer Subtraction with Carry SUBC

Group Arithmetic Instructions

Syntax **SUBC op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) - (op2) - (C)

Description

Performs a 2s complement binary subtraction of the source operand specified by op2 and the previously generated carry bit from the destination operand specified by op1. The result is then stored in op1. This instruction can be used to perform multiple precision arithmetic.

CPU Flags

E	Z	V	C	N
*	S	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero and previous Z flag was set. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
SUBC	Rw_n , #data3	38 n:0###	2
SUBC	Rw_n , Rw_m	30 nm	2
SUBC	Rw_n , [Rw_i+]	38 n:11ii	2
SUBC	Rw_n , [Rw_i]	38 n:10ii	2
SUBC	mem , reg	34 RR MM MM	4
SUBC	reg , #data16	36 RR ## ##	4
SUBC	reg , mem	32 RR MM MM	4

SUBCB Integer Subtraction with Carry SUBCB

Group Arithmetic Instructions

Syntax **SUBCB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) - (op2) - (C)

Description

Performs a 2s complement binary subtraction of the source operand specified by op2 and the previously generated carry bit from the destination operand specified by op1. The result is then stored in op1. This instruction can be used to perform multiple precision arithmetic.

CPU Flags

E	Z	V	C	N
*	S	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero and the previous Z flag was set. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
SUBCB	Rb_n , #data3	39 n:0###	2
SUBCB	Rb_n , Rb_m	31 nm	2
SUBCB	Rb_n , [Rw_i+]	39 n:11ii	2
SUBCB	Rb_n , [Rw_i]	39 n:10ii	2
SUBCB	mem , reg	35 RR MM MM	4
SUBCB	reg , #data8	37 RR ## xx	4
SUBCB	reg , mem	33 RR MM MM	4

TRAP Software Trap TRAP

Group Call Instructions

Syntax **TRAP op1**

Source Operand(s) op1 → 7-bit trap number

Destination Operand(s) none

Operation

```
(SP) ← (SP) - 2
((SP) ← (PSW)
IF (CPUCON1.SGTDIS = 0) THEN
        (SP) ← (SP) - 2
        ((SP)) ← (CSP)
END IF
(CSP) ← (VSEG)
(SP) ← (SP) - 2
((SP)) ← (IP)
(IP) ← ((op1) * 4) <<CPUCON1.SCINT
```

Description

Invokes a trap or interrupt routine based on the specified operand op1. The invoked routine is determined by branching to the specified vector table entry point. This routine has no indication of whether it was called by software or hardware. System state is preserved identically to hardware interrupt entry except that the CPU priority level is not affected. The RETI, Return from Interrupt instruction is used to resume execution after the completion of the trap or interrupt routine. The CSP is pushed if the segmentation is enabled. This is indicated by the SGTDIS bit of the CPUCON1 register.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
TRAP	#trap7	9B t:ttt0	2

XOR Logical Exclusive OR XOR

Group Logical Instructions

Syntax **XOR op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) ⊕ (op2)

Description

Performs a bitwise logical EXCLUSIVE OR of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
XOR	Rw_n , #data3	58 n:0###	2
XOR	Rw_n , Rw_m	50 nm	2
XOR	Rw_n , [Rw_i+]	58 n:11ii	2
XOR	Rw_n , [Rw_i]	58 n:10ii	2
XOR	mem , reg	54 RR MM MM	4
XOR	reg , #data16	56 RR ## ##	4
XOR	reg , mem	52 RR MM MM	4

XORB Logical Exclusive OR XORB

Group Logical Instructions

Syntax **XORB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) ⊕ (op2)

Description

Performs a bitwise logical EXCLUSIVE OR of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
XORB	Rb_n , #data3	59 n:0###	2
XORB	Rb_n , Rb_m	51 nm	2
XORB	Rb_n , [Rw_i+]	59 n:11ii	2
XORB	Rb_n , [Rw_i]	59 n:10ii	2
XORB	mem , reg	55 RR MM MM	4
XORB	reg , #data8	57 RR ## xx	4
XORB	reg , mem	53 RR MM MM	4

8.2 DSP Instruction Set

CoABS Absolute Value CoABS

Group Arithmetic Instructions

Syntax **CoABS**

Source Operand(s) ACC → 40-bit signed value

Destination Operand(s) ACC → 40-bit signed value

Operation

(ACC) ← Abs(ACC)

Description

Computes the absolute value of the 40-bit ACC contents.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	0	*	*	yes

MV Set if the ACC contents was 80 0000 0000H. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if the ACC contents was 80 0000 0000H. Not affected otherwise.

MC Always cleared.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic	Format	Bytes
CoABS	A3 00 1A rrr0:0000	4

CoABS Absolute Value CoABS

Group Arithmetic Instructions

Syntax **CoABS op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(ACC) ← Abs((op2) || (op1))

Description

Computes the absolute value of a 40-bit source operand and loads the result in the 40-bit ACC register. The 40-bit operand is a sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW).

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	*	*	-	0	*	*	yes

MV Always cleared.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Not affected.

MC Always cleared.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoABS	Rw_n , Rw_m	A3 nm CA rrr0:0000	4
CoABS	Rw_n , [Rw_m*]	83 nm CA rrr0:0qqq	4
CoABS	[IDXi*] , [Rw_m*]	93 Xm CA rrr0:0qqq	4

CoADD Add CoADD

Group Arithmetic Instructions

Syntax **CoADD op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op2) || (op1)
(ACC) ← (ACC) + (tmp)

Description

Adds a 40-bit operand to the 40-bit ACC register contents and stores the result in the ACC register. The 40-bit operand is a sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW).

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic overflow occurred. Not affected otherwise.
- MC Set if a carry is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoADD	Rw_n , Rw_m	A3 nm 02 rrr0:0000	4
CoADD	Rw_n , [Rw_m*]	83 nm 02 rrr0:0qqq	4
CoADD	[IDXi*] , [Rw_m*]	93 Xm 02 rrr0:0qqq	4

CoADD2 Add CoADD2

Group Arithmetic Instructions

Syntax **CoADD2 op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← 2 * ((op2) || (op1))
(ACC) ← (ACC) + (tmp)

Description

Adds a 40-bit operand to the 40-bit ACC register contents and stores the result in the ACC register. The 40-bit operand is a sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW). The 40-bit operand is then multiplied by two before being added to ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoADD2	Rw_n , Rw_m	A3 nm 42 rrr0:0000	4
CoADD2	Rw_n , [Rw_m*]	83 nm 42 rrr0:0qqq	4
CoADD2	[IDXi*] , [Rw_m*]	93 Xm 42 rrr0:0qqq	4

CoASHR Accumulator Arithmetic Shift Right with Round CoASHR

Group Shift Instructions

Syntax **CoASHR op1, rnd**

Source Operand(s) op1 → shift counter

Destination Operand(s) ACC → 40-bit signed value

Operation

(count) ← (op1)
(C) ← 0
DO WHILE (count) ≠ 0
(ACC[n]) ← (ACC[n+1]) [n=0...38]
(count) ← (count) -1
END WHILE
(ACC) ← (ACC) + 0000 8000h
(MAL) ← 0

Description

Arithmetically shifts the ACC register right by the number of times as specified by the operand op1. Then, the result is 2s complement rounded before being stored in the 40-bit ACC register. To preserve the sign of the ACC register, the most significant bits of the result are filled with sign 0 if the original most significant bit was a 0 or with sign 1 if the original most significant bit was 1. Only shift values from 0 to 16 (inclusive) are allowed. op1 can be either a 5-bit unsigned immediate data (the shift range is from 0 to 16 in this case) or the four least significant bits (the shift range is from 0 to 15 in that case) of any register directly or indirectly addressed operand.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred. Cleared otherwise.
MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
ME Set if the MAE is used. Cleared otherwise.
MSV Set if an arithmetic overflow occurred. Not affected otherwise.
MC Set if a carry is generated when rounding. Cleared otherwise.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoASHR	#data5 , rnd	A3 00 B2 rrr#:#	4
CoASHR	Rw_n , rnd	A3 nn BA rrr0:0000	4
CoASHR	[Rw_m*] , rnd	83 mm BA rrr0:0qqq	4

CoASHR Accumulator Arithmetic Shift Right CoASHR

Group Shift Instructions

Syntax **CoASHR op1**

Source Operand(s) op1 → shift counter

Destination Operand(s) ACC → 40-bit signed value

Operation

```
(count) ← (op1)
(C) ← 0
DO WHILE (count) ≠ 0
        (ACC[n]) ← (ACC[n+1]) [n=0...38]
        (count) ← (count) -1
END WHILE
```

Description

Arithmetically shifts the ACC register right by the number of times as specified by the operand op1. To preserve the sign of the ACC register, the most significant bits of the result are filled with sign 0 if the original most significant bit was a 0 or with sign 1 if the original most significant bit was 1. Only shift values from 0 to 16 (inclusive) are allowed. op1 can be either a 5-bit unsigned immediate data (the shift range is from 0 to 16 in this case) or the four least significant bits (the shift range is from 0 to 15 in that case) of any register directly or indirectly addressed operand. The MS bit of the MCW register does not affect the result.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	-	*	-	0	*	*	no

MV Always cleared.
MSL Not affected.
ME Set if the MAE is used. Cleared otherwise.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoASHR	#data5	A3 00 A2 rrr#:#	4
CoASHR	Rw_n	A3 nn AA rrr0:0000	4
CoASHR	[Rw_m*]	83 mm AA rrr0:0qqq	4

CoCMP Compare CoCMP

Group Compare Instructions

Syntax **CoCMP op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) none

Operation

tmp ← (op2) || (op1)
(ACC) ⇔ (tmp)

Description

Subtracts a 40-bit signed operand from the 40-bit ACC contents and updates the N, Z and C flags of the MSW register leaving the ACC register unchanged. The 40-bit operand is a sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW). The MS bit of the MCW register does not affect the result.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	-	-	-	*	*	*	no

MV Set if the ACC contents are strictly less than the 40-bit operand. Cleared otherwise.
MSL Not affected.
ME Not affected.
MSV Not affected.
MC Set if a borrow is generated. Cleared otherwise.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoCMP	Rw_n , Rw_m	A3 nm C2 rrr0:0000	4
CoCMP	Rw_n , [Rw_m*]	83 nm C2 rrr0:0qqq	4
CoCMP	[IDXi*] , [Rw_m*]	93 Xm C2 rrr0:0qqq	4

CoLOAD Load Accumulator CoLOAD

Group Arithmetic Instructions

Syntax **CoLOAD op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op2) || (op1)
(ACC) ← 0 + (tmp)

Description

Loads the 40-bit ACC register with a 40-bit source operand. The 40-bit source operand is the sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW).

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	-	0	-	0	*	*	no

MV Always cleared.
MSL Not affected.
ME Always cleared.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoLOAD	Rw_n , Rw_m	A3 nm 22 rrr0:0000	4
CoLOAD	Rw_n , [Rw_m*]	83 nm 22 rrr0:0qqq	4
CoLOAD	[IDXi*] , [Rw_m*]	93 Xm 22 rrr0:0qqq	4

CoLOAD- Load Accumulator CoLOAD-

Group Arithmetic Instructions

Syntax **CoLOAD- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op2) || (op1)
(ACC) ← 0 - (tmp)

Description

Loads the 40-bit ACC register with a 40-bit source operand. The 40-bit source operand is a sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW). The 40-bit source operand is 2s complemented, before being stored in the ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	*	*	-	*	*	*	yes

MV Always cleared.
MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
ME Set if the MAE is used. Cleared otherwise.
MSV Not affected.
MC Set if a borrow is generated. Cleared otherwise.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoLOAD-	Rw_n , Rw_m	A3 nm 2A rrr0:0000	4
CoLOAD-	Rw_n , [Rw_m*]	83 nm 2A rrr0:0qqq	4
CoLOAD-	[IDXi*] , [Rw_m*]	93 Xm 2A rrr0:0qqq	4

CoLOAD2 Load Accumulator CoLOAD2

Group Arithmetic Instructions

Syntax **CoLOAD2 op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← 2 * ((op2) || (op1))
(ACC) ← 0 + (tmp)

Description

Loads the 40-bit ACC register with a 40-bit source operand. The 40-bit source operand is a sign-extended results of the concatenation of the two source operands op1 (LSW) and op2 (MSW). The 40-bit operand is also multiplied by two, before being stored in the ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	*	*	-	0	*	*	yes

MV Always cleared.
MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
ME Set if the MAE is used. Cleared otherwise.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoLOAD2	Rw_n , Rw_m	A3 nm 62 rrr0:0000	4
CoLOAD2	Rw_n , [Rw_m*]	83 nm 62 rrr0:0qqq	4
CoLOAD2	[IDXi*] , [Rw_m*]	93 Xm 62 rrr0:0qqq	4

CoLOAD2- Load Accumulator CoLOAD2-

Group Arithmetic Instructions

Syntax **CoLOAD2- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← 2 * ((op2) || (op1))
(ACC) ← 0 - (tmp)

Description

Loads the 40-bit ACC register with a 40-bit source operand. The 40-bit source operand is a sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW). The 40-bit operand is also multiplied by two and negated, before being stored in the ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	*	*	-	*	*	*	yes

MV Always cleared.
MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
ME Set if the MAE is used. Cleared otherwise.
MSV Not affected.
MC Set if a borrow is generated. Cleared otherwise.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoLOAD2-	Rw_n , Rw_m	A3 nm 6A rrr0:0000	4
CoLOAD2-	Rw_n , [Rw_m*]	83 nm 6A rrr0:0qqq	4
CoLOAD2-	[IDXi*] , [Rw_m*]	93 Xm 6A rrr0:0qqq	4

CoMAC Multiply-Accumulate with Round CoMAC

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMAC op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (tmp) ← ((op1) * (op2)) <<1
        (ACC) ← (ACC) + (tmp) + 00 0000 8000h
ELSE
        (tmp) ← (op1) * (op2)
        (ACC) ← (ACC) + (tmp) + 00 0000 8000h
END IF
(MAL) ← 0
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then, if the MP flag is set, it is one-bit left shifted; then, it is added to the 40-bit ACC register contents. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMAC	Rw_n , Rw_m , rnd	A3 nm D1 rrr0:0000	4
CoMAC	Rw_n , [Rw_m*] , rnd	83 nm D1 rrr0:0qqq	4
CoMAC	[IDXi*] , [Rw_m*] , rnd	93 Xm D1 rrr0:0qqq	4

CoMAC Multiply-Accumulate CoMAC

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMAC op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (tmp) ← ((op1) * (op2)) <<1
        (ACC) ← (ACC) + (tmp)
ELSE
        (tmp) ← (op1) * (op2)
        (ACC) ← (ACC) + (tmp)
END IF
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then, if the MP flag is set, it is one-bit left shifted; then, it is added to the 40-bit ACC register contents before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMAC	Rw_n , Rw_m	A3 nm D0 rrr0:0000	4
CoMAC	Rw_n , [Rw_m*]	83 nm D0 rrr0:0qqq	4
CoMAC	[IDXi*] , [Rw_m*]	93 Xm D0 rrr0:0qqq	4

CoMAC- Multiply-Accumulate CoMAC-

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMAC- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (tmp) ← ((op1) * (op2)) <<1
        (ACC) ← (ACC) - (tmp)
ELSE
        (tmp) ← (op1) * (op2)
        (ACC) ← (ACC) - (tmp)
END IF
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then, if the MP flag is set, it is one-bit left shifted; then, it is subtracted from the 40-bit ACC register contents before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMAC-	Rw_n , Rw_m	A3 nm E0 rrr0:0000	4
CoMAC-	Rw_n , [Rw_m*]	83 nm E0 rrr0:0qqq	4
CoMAC-	[IDXi*] , [Rw_m*]	93 Xm E0 rrr0:0qqq	4

CoMACM Multiply-Accumulate & Move & Round CoMACM

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACM op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (tmp) ← (((op1)) * ((op2))) <<1
        (ACC) ← (ACC) + (tmp) + 00 0000 8000h
ELSE
        (tmp) ← ((op1))*((op2))
        (ACC) ← (ACC) + (tmp) + 00 0000 8000h
END IF
(MAL) ← 0
((IDXi(-*))) ← ((IDXi))
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then, if the MP flag is set, it is one-bit left shifted; and next, it is added to the 40-bit ACC register contents. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic overflow occurred. Not affected otherwise.
- MC Set if a carry is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACM	[IDXi*], [Rw_m*] , rnd	93 Xm D9 rrr0:0qqq	4

CoMACM Multiply-Accumulate & Move CoMACM

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACM op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (tmp) ← (((op1)) * ((op2))) <<1
        (ACC) ← (ACC) + (tmp)
ELSE
        (tmp) ← ((op1)) * ((op2))
        (ACC) ← (ACC) + (tmp)
END IF
((IDXi(-*))) ← ((IDXi))
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then if the MP flag is set, it is one-bit left shifted; and next it is added to the 40-bit ACC register contents before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACM	[IDXi*], [Rw_m*]	93 Xm D8 rrr0:0qqq	4

CoMACM- Multiply-Accumulate & Move CoMACM-

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACM- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (tmp) ← (((op1)) * ((op2))) <<1
        (ACC) ← (ACC) - (tmp)
ELSE
        (tmp) ← ((op1)) * ((op2))
        (ACC) ← (ACC) - (tmp)
END IF
((IDXi(-*))) ← ((IDXi))
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then, if the MP flag is set, it is one-bit left shifted; and next, it is subtracted from the 40-bit ACC register contents before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACM-	[IDXi*], [Rw_m*]	93 Xm E8 rrr0:0qqq	4

CoMACMR Multiply-Accumulate & Move & Round CoMACMR

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMR op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (tmp) ← (((op1)) * ((op2))) <<1
        (ACC) ← (tmp) - (ACC) + 00 0000 8000h
ELSE
        (tmp) ← ((op1))*((op2))
        (ACC) ← (tmp) - (ACC) + 00 0000 8000h
END IF
(MAL) ← 0
((IDXi(-*))) ← ((IDXi))
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then, if the MP flag is set, it is one-bit left shifted; and next, the 40-bit ACC register contents are subtracted from the result Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACMR	[IDXi*], [Rw_m*] , rnd	93 Xm F9 rrr0:0qqq	4

CoMACMR Multiply-Accumulate & Move CoMACMR

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMR op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (tmp) ← (((op1)) * ((op2))) <<1
        (ACC) ← (tmp) - (ACC)
ELSE
        (tmp) ← ((op1)) * ((op2))
        (ACC) ← (tmp) - (ACC)
END IF
((IDXi(-*))) ← ((IDXi))
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then, if the MP flag is set, it is one-bit left shifted; and next, the 40-bit ACC register contents are subtracted from the result before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACMR	[IDXi*], [Rw_m*]	93 Xm F8 rrr0:0qqq	4

CoMACMRsu Multiply-Accumulate & Move & Round CoMACMRsu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMRsu op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1)) * ((op2))
(ACC) ← (tmp) - (ACC) + 00 0000 8000h
(MAL) ← 0
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, the 40-bit ACC register contents are subtracted from the result before being stored in the 40-bit ACC register. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic	Format	Bytes
CoMACMRsu [IDXi*], [Rw_m*] , rnd	93 Xm 79 rrr0:0qqq	4

CoMACMRsu Multiply-Accumulate & Move CoMACMRsu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMRsu op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1)) * ((op2))
(ACC) ← (tmp) - (ACC)
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, the 40-bit ACC register contents are subtracted from the result before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic	Format	Bytes
CoMACMRsu [IDXi*], [Rw_m*]	93 Xm 78 rrr0:0qqq	4

CoMACMRu Multiply-Accumulate & Move & Round CoMACMRu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMRu op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1))*((op2))
(ACC) ← (tmp) - (ACC) + 00 0000 8000h
(MAL) ← 0
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is first zero-extended; then, the 40-bit ACC register contents are subtracted from the result. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACMRu	[IDXi*], [Rw_m*] , rnd	93 Xm 39 rrr0:0qqq	4

CoMACMRu Multiply-Accumulate & Move CoMACMRu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMRu op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1))*((op2))
(ACC) ← (tmp) - (ACC)
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is first zero-extended; then, the 40-bit ACC register contents are subtracted from the result before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACMRu	[IDXi*], [Rw_m*]	93 Xm 38 rrr0:0qqq	4

CoMACMRus Multiply-Accumulate & Move & Round CoMACMRus

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMRus op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1)) * ((op2))
(ACC) ← (tmp) - (ACC) + 00 0000 8000h
(MAL) ← 0
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, the 40-bit ACC register contents are subtracted from the result. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic	Format	Bytes
CoMACMRus [IDXi*], [Rw_m*] , rnd	93 Xm B9 rrr0:0qqq	4

CoMACMRus Multiply-Accumulate & Move CoMACMRus

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMRus op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1)) * ((op2))
(ACC) ← (tmp) - (ACC)
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, the 40-bit ACC register contents are subtracted from the result before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic	Format	Bytes
CoMACMRus [IDXi*], [Rw_m*]	93 Xm B8 rrr0:0qqq	4

CoMACMsu Multiply-Accumulate & Move & Round CoMACMsu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMsu op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1)) * ((op2))
(ACC) ← (ACC) + (tmp) + 00 0000 8000h
(MAL) ← 0
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, it is added to the 40-bit ACC register contents. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACMsu	[IDXi*], [Rw_m*] , rnd	93 Xm 59 rrr0:0qqq	4

CoMACMsu Multiply-Accumulate & Move CoMACMsu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMsu op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1)) * ((op2))
(ACC) ← (ACC) + (tmp)
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, it is added to the 40-bit ACC register contents before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACMsu	[IDXi*], [Rw_m*]	93 Xm 58 rrr0:0qqq	4

CoMACMsu- Multiply-Accumulate & Move CoMACMsu-

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMsu- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1)) * ((op2))
(ACC) ← (ACC) - (tmp)
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, it is subtracted from the 40-bit ACC register contents before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic	Format	Bytes
CoMACMsu- [IDXi*], [Rw_m*]	93 Xm 68 rrr0:0qqq	4

CoMACMu Multiply-Accumulate & Move & Round CoMACMu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMu op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1))*((op2))
(ACC) ← (ACC) + (tmp) + 00 0000 8000h
(MAL) ← 0
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is first zero-extended; then, it is added to the 40-bit ACC register contents. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACMu	[IDXi*], [Rw_m*] , rnd	93 Xm 19 rrr0:0qqq	4

CoMACMu Multiply-Accumulate & Move CoMACMu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMu op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1))*((op2))
(ACC) ← (ACC) + (tmp)
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is first zero-extended; then, it is added to the 40-bit ACC register contents before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic overflow occurred. Not affected otherwise.
- MC Set if a carry is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACMu	[IDXi*], [Rw_m*]	93 Xm 18 rrr0:0qqq	4

CoMACMu- Multiply-Accumulate & Move CoMACMu-

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMu- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1))*((op2))
(ACC) ← (ACC) - (tmp)
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is first zero-extended; then, it is subtracted from the 40-bit ACC register contents before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACMu-	[IDXi*], [Rw_m*]	93 Xm 28 rrr0:0qqq	4

CoMACMus Multiply-Accumulate & Move & Round CoMACMus

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMus op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1)) * ((op2))
(ACC) ← (ACC) + (tmp) + 00 0000 8000h
(MAL) ← 0
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, it is added to the 40-bit ACC register contents. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACMus	[IDXi*], [Rw_m*] , rnd	93 Xm 99 rrr0:0qqq	4

CoMACMus Multiply-Accumulate & Move CoMACMus

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMus op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1)) * ((op2))
(ACC) ← (ACC) + (tmp)
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, it is added to the 40-bit ACC register contents before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACMus	[IDXi*], [Rw_m*]	93 Xm 98 rrr0:0qqq	4

CoMACMus- Multiply-Accumulate & Move CoMACMus-

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACMus- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← ((op1)) * ((op2))
(ACC) ← (ACC) - (tmp)
((IDXi(-*))) ← ((IDXi))

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, it is subtracted from the 40-bit ACC register contents before being stored in the 40-bit ACC register. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDXi overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDXi.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic	Format	Bytes
CoMACMus- [IDXi*], [Rw_m*]	93 Xm A8 rrr0:0qqq	4

CoMACR Multiply-Accumulate & Round CoMACR

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACR op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (tmp) ← ((op1) * (op2)) <<1
        (ACC) ← (tmp) - (ACC) + 00 0000 8000h
ELSE
        (tmp) ← (op1) * (op2)
        (ACC) ← (tmp) - (ACC) + 00 0000 8000h
END IF
(MAL) ← 0
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then, if the MP flag is set, it is one-bit left shifted; then, the 40-bit ACC register contents are subtracted from the result. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACR	Rw_n , Rw_m , rnd	A3 nm F1 rrr0:0000	4
CoMACR	Rw_n , [Rw_m*] , rnd	83 nm F1 rrr0:0qqq	4
CoMACR	[IDXi*] , [Rw_m*] , rnd	93 Xm F1 rrr0:0qqq	4

CoMACR Multiply-Accumulate CoMACR

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACR op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (tmp) ← ((op1) * (op2)) <<1
        (ACC) ← (tmp) - (ACC)
ELSE
        (tmp) ← (op1) * (op2)
        (ACC) ← (tmp) - (ACC)
END IF
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then, if the MP flag is set, it is one-bit left shifted; then, the 40-bit ACC register contents are subtracted from the result before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACR	Rw_n , Rw_m	A3 nm F0 rrr0:0000	4
CoMACR	Rw_n , [Rw_m*]	83 nm F0 rrr0:0qqq	4
CoMACR	[IDXi*] , [Rw_m*]	93 Xm F0 rrr0:0qqq	4

CoMACRsu Mixed Multiply-Accumulate & Round CoMACRsu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACRsu op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (tmp) - (ACC) + 00 0000 8000h
(MAL) ← 0

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended and then the 40-bit ACC register contents are subtracted from the result. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACRsu	Rw_n , Rw_m , rnd	A3 nm 71 rrr0:0000	4
CoMACRsu	Rw_n , [Rw_m*] , rnd	83 nm 71 rrr0:0qqq	4
CoMACRsu	[IDXi*] , [Rw_m*] , rnd	93 Xm 71 rrr0:0qqq	4

CoMACRsu Mixed Multiply-Accumulate CoMACRsu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACRsu op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (tmp) - (ACC)

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended and then the 40-bit ACC register contents are subtracted from the result before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN
*	*	*	*	*	*	*

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACRsu	Rw_n , Rw_m	A3 nm 70 rrr0:0000	4
CoMACRsu	Rw_n , [Rw_m*]	83 nm 70 rrr0:0qqq	4
CoMACRsu	[IDXi*] , [Rw_m*]	93 Xm 70 rrr0:0qqq	4

CoMACRu Unsigned Multiply-Accumulate & Round CoMACRu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACRu op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (tmp) - (ACC) + 00 0000 8000h
(MAL) ← 0

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is first zero-extended and then the 40-bit ACC register contents are subtracted from the result. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACRu	Rw_n , Rw_m , rnd	A3 nm 31 rrr0:0000	4
CoMACRu	Rw_n , [Rw_m*] , rnd	83 nm 31 rrr0:0qqq	4
CoMACRu	[IDXi*] , [Rw_m*] , rnd	93 Xm 31 rrr0:0qqq	4

CoMACRu Unsigned Multiply-Accumulate CoMACRu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACRu op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (tmp) - (ACC)

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is first zero-extended and then the 40-bit ACC register contents are subtracted from the result before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACRu	Rw_n , Rw_m	A3 nm 30 rrr0:0000	4
CoMACRu	Rw_n , [Rw_m*]	83 nm 30 rrr0:0qqq	4
CoMACRu	[IDXi*] , [Rw_m*]	93 Xm 30 rrr0:0qqq	4

CoMACRus Mixed Multiply-Accumulate & Round CoMACRus

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACRus op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (tmp) - (ACC) + 00 0000 8000h
(MAL) ← 0

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended and then the 40-bit ACC register contents are subtracted from the result. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACRus	Rw_n , Rw_m , rnd	A3 nm B1 rrr0:0000	4
CoMACRus	Rw_n , [Rw_m*] , rnd	83 nm B1 rrr0:0qqq	4
CoMACRus	[IDXi*] , [Rw_m*] , rnd	93 Xm B1 rrr0:0qqq	4

CoMACRus Mixed Multiply-Accumulate CoMACRus

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACRus op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (tmp) - (ACC)

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended and then the 40-bit ACC register contents are subtracted from the result before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACRus	Rw_n , Rw_m	A3 nm B0 rrr0:0000	4
CoMACRus	Rw_n , [Rw_m*]	83 nm B0 rrr0:0qqq	4
CoMACRus	[IDXi*] , [Rw_m*]	93 Xm B0 rrr0:0qqq	4

CoMACsu Mixed Multiply-Accumulate & Round CoMACsu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACsu op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (ACC) + (tmp) + 00 0000 8000h
(MAL) ← 0

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended and then added to the 40-bit ACC register contents. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACsu	Rw_n , Rw_m , rnd	A3 nm 51 rrr0:0000	4
CoMACsu	Rw_n , [Rw_m*] , rnd	83 nm 51 rrr0:0qqq	4
CoMACsu	[IDXi*] , [Rw_m*] , rnd	93 Xm 51 rrr0:0qqq	4

CoMACsu Mixed Multiply-Accumulate CoMACsu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACsu op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (ACC) + (tmp)

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended and then added to the 40-bit ACC register contents before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic overflow occurred. Not affected otherwise.
- MC Set if a carry is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACsu	Rw_n , Rw_m	A3 nm 50 rrr0:0000	4
CoMACsu	Rw_n , [Rw_m*]	83 nm 50 rrr0:0qqq	4
CoMACsu	[IDXi*] , [Rw_m*]	93 Xm 50 rrr0:0qqq	4

CoMACsu- Mixed Multiply-Accumulate CoMACsu-

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACsu- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (ACC) - (tmp)

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended and then subtracted from the 40-bit ACC register contents before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACsu-	Rw_n , Rw_m	A3 nm 60 rrr0:0000	4
CoMACsu-	Rw_n , [Rw_m*]	83 nm 60 rrr0:0qqq	4
CoMACsu-	[IDXi*] , [Rw_m*]	93 Xm 60 rrr0:0qqq	4

CoMACu Unsigned Multiply-Accumulate & Round CoMACu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACu op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (ACC) + (tmp) + 00 0000 8000h
(MAL) ← 0

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is first zero-extended and then added to the 40-bit ACC register contents. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACu	Rw_n , Rw_m , rnd	A3 nm 11 rrr0:0000	4
CoMACu	Rw_n , [Rw_m*] , rnd	83 nm 11 rrr0:0qqq	4
CoMACu	[IDXi*] , [Rw_m*] , rnd	93 Xm 11 rrr0:0qqq	4

CoMACu Unsigned Multiply-Accumulate CoMACu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACu op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (ACC) + (tmp)

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is first zero-extended and then added to the 40-bit ACC register contents before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic overflow occurred. Not affected otherwise.
- MC Set if a carry is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACu	Rw_n , Rw_m	A3 nm 10 rrr0:0000	4
CoMACu	Rw_n , [Rw_m*]	83 nm 10 rrr0:0qqq	4
CoMACu	[IDXi*] , [Rw_m*]	93 Xm 10 rrr0:0qqq	4

CoMACu- Unsigned Multiply-Accumulate CoMACu-

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACu- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (ACC) - (tmp)

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is first zero-extended and then subtracted from the 40-bit ACC register contents before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACu-	Rw_n , Rw_m	A3 nm 20 rrr0:0000	4
CoMACu-	Rw_n , [Rw_m*]	83 nm 20 rrr0:0qqq	4
CoMACu-	[IDXi*] , [Rw_m*]	93 Xm 20 rrr0:0qqq	4

CoMACus Mixed Multiply-Accumulate with Round CoMACus

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACus op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (ACC) + (tmp) + 00 0000 8000h
(MAL) ← 0

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended and then added to the 40-bit ACC register contents. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACus	Rw_n , Rw_m , rnd	A3 nm 91 rrr0:0000	4
CoMACus	Rw_n , [Rw_m*] , rnd	83 nm 91 rrr0:0qqq	4
CoMACus	[IDXi*] , [Rw_m*] , rnd	93 Xm 91 rrr0:0qqq	4

CoMACus Mixed Multiply-Accumulate CoMACus

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACus op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (ACC) + (tmp)

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended and then added to the 40-bit ACC register contents before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic overflow occurred. Not affected otherwise.
- MC Set if a carry is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACus	Rw_n , Rw_m	A3 nm 90 rrr0:0000	4
CoMACus	Rw_n , [Rw_m*]	83 nm 90 rrr0:0qqq	4
CoMACus	[IDXi*] , [Rw_m*]	93 Xm 90 rrr0:0qqq	4

CoMACus- Mixed Multiply-Accumulate CoMACus-

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMACus- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op1) * (op2)
(ACC) ← (ACC) - (tmp)

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended and then subtracted from the 40-bit ACC register contents before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMACus-	Rw_n , Rw_m	A3 nm A0 rrr0:0000	4
CoMACus-	Rw_n , [Rw_m*]	83 nm A0 rrr0:0qqq	4
CoMACus-	[IDXi*] , [Rw_m*]	93 Xm A0 rrr0:0qqq	4

CoMAX Maximum CoMAX

Group Compare Instructions

Syntax **CoMAX op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op2) || (op1)
(ACC) ← max((ACC),(tmp))

Description

Compares a signed 40-bit operand against the 40-bit ACC register contents. The 40-bit operand is a sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW) which is then sign-extended. If the contents of the 40-bit ACC register are smaller than the 40-bit operand, then the ACC register is loaded with it. Otherwise, the ACC register remains unchanged. The MS bit of the MCW register does not affect the result.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	*	*	-	0	*	*	no

MV Always cleared.
MSL Set if the contents of ACC is changed. Not affected otherwise.
ME Set if the MAE is used. Cleared otherwise.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMAX	Rw_n , Rw_m	A3 nm 3A rrr0:0000	4
CoMAX	Rw_n , [Rw_m*]	83 nm 3A rrr0:0qqq	4
CoMAX	[IDXi*] , [Rw_m*]	93 Xm 3A rrr0:0qqq	4

CoMIN Minimum CoMIN

Group Compare Instructions

Syntax **CoMIN op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op2) || (op1)
(ACC) ← min((ACC),(tmp))

Description

Compares a signed 40-bit operand against the 40-bit ACC register contents. The 40-bit operand is a sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW). If the contents of the ACC register are greater than the 40-bit operand, then the ACC register is loaded with it. Otherwise, the ACC register remains unchanged. The MS bit of the MCW register does not affect the result.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	*	*	-	0	*	*	no

MV Always cleared.
MSL Set if the contents of ACC is changed. Not affected otherwise.
ME Set if the MAE is used. Cleared otherwise.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMIN	Rw_n , Rw_m	A3 nm 7A rrr0:0000	4
CoMIN	Rw_n , [Rw_m*]	83 nm 7A rrr0:0qqq	4
CoMIN	[IDXi*] , [Rw_m*]	93 Xm 7A rrr0:0qqq	4

CoMOV Memory to Memory Move CoMOV

Group Data Movement Instructions

Syntax **CoMOV op1, op2**

Source Operand(s) op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op2)

Description

Moves the contents of the memory location specified by the source operand op2 to the memory location specified by the destination operand op1. Note that in this case, unlike for the other instructions, IDXi can address the entire memory. This instruction does not affect the Mac Flags, but modifies the CPU Flags as any other MOV instruction.

Note: CoMOV is the only MAC instruction which affects the CPU flags. MAC Flags are not affected.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if the value of the source operand op2 equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the source operand op2 is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMOV	[IDXi*] , [Rw_m*]	D3 Xm 00 rrr0:0qqq	4

CoMUL Signed Multiply with Round CoMUL

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMUL op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (ACC) ← ((op1) * (op2)) <<1 + 00 0000 8000h
ELSE
        (ACC) ← (op1) * (op2) + 00 0000 8000h
END IF
(MAL) ← 0
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then, if the MP flag is set, it is one-bit left shifted. Finally, the result is 2s complement rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	*	*	-	0	*	*	yes

MV Always cleared.

MSL Not affected when MP or MS are cleared, otherwise, only set in case of 8000h by 8000h multiplication.

ME Set when MP is set and MS is cleared and in case of 8000h by 8000h multiplication. Cleared otherwise.

MSV Not affected.

MC Always cleared.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMUL	Rw_n , Rw_m , rnd	A3 nm C1 rrr0:0000	4
CoMUL	Rw_n , [Rw_m*] , rnd	83 nm C1 rrr0:0qqq	4
CoMUL	[IDXi*] , [Rw_m*] , rnd	93 Xm C1 rrr0:0qqq	4

CoMUL Signed Multiply CoMUL

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMUL op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (ACC) ← ((op1) * (op2)) <<1
ELSE
        (ACC) ← (op1) * (op2)
END IF
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then, if the MP flag is set, it is one-bit left shifted before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	*	*	-	0	*	*	yes

MV Always cleared.

MSL Not affected when MP or MS are cleared, otherwise, only set in case of 8000h by 8000h multiplication.

ME Set when MP is set and MS is cleared and in case of 8000h by 8000h multiplication. Cleared otherwise.

MSV Not affected.

MC Always cleared.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMUL	Rw_n , Rw_m	A3 nm C0 rrr0:0000	4
CoMUL	Rw_n , [Rw_m*]	83 nm C0 rrr0:0qqq	4
CoMUL	[IDXi*] , [Rw_m*]	93 Xm C0 rrr0:0qqq	4

CoMUL- Signed Multiply CoMUL-

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMUL- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

```
IF (MP = 1) THEN
        (ACC) ← - ((op1) * (op2)) <<1
ELSE
        (ACC) ← - ((op1) * (op2))
END IF
```

Description

Multiplies the two signed 16-bit source operands op1 and op2. The resulting signed 32-bit product is first sign-extended; then, if the MP flag is set, it is one-bit left shifted; and, finally, it is negated before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	0	0	-	0	*	*	no

MV Always cleared.
MSL Always cleared.
ME Always cleared.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMUL-	Rw_n , Rw_m	A3 nm C8 rrr0:0000	4
CoMUL-	Rw_n , [Rw_m*]	83 nm C8 rrr0:0qqq	4
CoMUL-	[IDXi*] , [Rw_m*]	93 Xm C8 rrr0:0qqq	4

CoMULsu Mixed Multiply & Round CoMULsu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMULsu op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(ACC) ← (op1) * (op2) + 00 0000 8000h
(MAL) ← 0

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, it is rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	-	0	-	0	*	*	no

MV Always cleared.
MSL Not affected.
ME Always cleared.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMULsu	Rw_n , Rw_m , rnd	A3 nm 41 rrr0:0000	4
CoMULsu	Rw_n , [Rw_m*] , rnd	83 nm 41 rrr0:0qqq	4
CoMULsu	[IDXi*] , [Rw_m*] , rnd	93 Xm 41 rrr0:0qqq	4

CoMULsu Mixed Multiply CoMULsu

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMULsu op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(ACC) ← (op1) * (op2)

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	-	0	-	0	*	*	no

MV Always cleared.
MSL Not affected.
ME Always cleared.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMULsu	Rw_n , Rw_m	A3 nm 40 rrr0:0000	4
CoMULsu	Rw_n , [Rw_m*]	83 nm 40 rrr0:0qqq	4
CoMULsu	[IDXi*] , [Rw_m*]	93 Xm 40 rrr0:0qqq	4

CoMULsu- Mixed Multiply CoMULsu-

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMULsu- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(ACC) ← - ((op1) * (op2))

Description

Multiplies the two signed and unsigned 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, is negated before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	-	0	-	0	*	*	no

MV Always cleared.
MSL Not affected.
ME Always cleared.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMULsu-	Rw_n , Rw_m	A3 nm 48 rrr0:0000	4
CoMULsu-	Rw_n , [Rw_m*]	83 nm 48 rrr0:0qqq	4
CoMULsu-	[IDXi*] , [Rw_m*]	93 Xm 48 rrr0:0qqq	4

CoMULu — Unsigned Multiply with Round — CoMULu

Group: Multiply/Multiply-Accumulate Instructions

Syntax **CoMULu op1, op2, rnd**

Source Operand(s): op1, op2 → WORD

Destination Operand(s): ACC → 40-bit signed value

Operation

(ACC) ← (op1) * (op2) + 00 0000 8000h
(MAL) ← 0

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is first zero-extended; then, it is rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	*	*	-	0	*	0	yes

- MV Always cleared.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Not affected.
- MC Always cleared.
- MZ Set if result equals zero. Cleared otherwise.
- MN Always cleared.

Encoding

Mnemonic		Format	Bytes
CoMULu	Rw_n , Rw_m , rnd	A3 nm 01 rrr0:0000	4
CoMULu	Rw_n , [Rw_m*] , rnd	83 nm 01 rrr0:0qqq	4
CoMULu	[IDXi*] , [Rw_m*] , rnd	93 Xm 01 rrr0:0qqq	4

CoMULu — Unsigned Multiply — CoMULu

Group: Multiply/Multiply-Accumulate Instructions

Syntax **CoMULu op1, op2**

Source Operand(s): op1, op2 → WORD

Destination Operand(s): ACC → 40-bit signed value

Operation

(ACC) ← (op1) * (op2)

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is zero-extended before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	*	*	-	0	*	0	yes

- MV Always cleared.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Not affected.
- MC Always cleared.
- MZ Set if result equals zero. Cleared otherwise.
- MN Always cleared.

Encoding

Mnemonic		Format	Bytes
CoMULu	Rw_n , Rw_m	A3 nm 00 rrr0:0000	4
CoMULu	Rw_n , [Rw_m*]	83 nm 00 rrr0:0qqq	4
CoMULu	[IDXi*] , [Rw_m*]	93 Xm 00 rrr0:0qqq	4

CoMULu- Unsigned Multiply CoMULu-

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMULu- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(ACC) ← - ((op1) * (op2))

Description

Multiplies the two unsigned 16-bit source operands op1 and op2. The resulting unsigned 32-bit product is first zero-extended; then, it is negated before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	*	*	-	0	*	*	yes

MV Always cleared.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Not affected.

MC Always cleared.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMULu-	Rw_n , Rw_m	A3 nm 08 rrr0:0000	4
CoMULu-	Rw_n , [Rw_m*]	83 nm 08 rrr0:0qqq	4
CoMULu-	[IDXi*] , [Rw_m*]	93 Xm 08 rrr0:0qqq	4

CoMULus Mixed Multiply with Round CoMULus

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMULus op1, op2, rnd**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(ACC) ← (op1) * (op2) + 00 0000 8000h
(MAL) ← 0

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, it is rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	-	0	-	0	*	*	no

MV Always cleared.
MSL Not affected.
ME Always cleared.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMULus	Rw_n , Rw_m , rnd	A3 nm 81 rrr0:0000	4
CoMULus	Rw_n , [Rw_m*] , rnd	83 nm 81 rrr0:0qqq	4
CoMULus	[IDXi*] , [Rw_m*] , rnd	93 Xm 81 rrr0:0qqq	4

CoMULus Mixed Multiply CoMULus

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMULus op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(ACC) ← (op1) * (op2)

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	-	0	-	0	*	*	no

MV Always cleared.
MSL Not affected.
ME Always cleared.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMULus	Rw_n , Rw_m	A3 nm 80 rrr0:0000	4
CoMULus	Rw_n , [Rw_m*]	83 nm 80 rrr0:0qqq	4
CoMULus	[IDXi*] , [Rw_m*]	93 Xm 80 rrr0:0qqq	4

CoMULus- Mixed Multiply CoMULus-

Group Multiply/Multiply-Accumulate Instructions

Syntax **CoMULus- op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(ACC) ← - ((op1) * (op2))

Description

Multiplies the two unsigned and signed 16-bit source operands op1 and op2, respectively. The resulting signed 32-bit product is first sign-extended; then, it is negated before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	-	0	-	0	*	*	no

MV Always cleared.
MSL Not affected.
ME Always cleared.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoMULus-	Rw_n , Rw_m	A3 nm 88 rrr0:0000	4
CoMULus-	Rw_n , [Rw_m*]	83 nm 88 rrr0:0qqq	4
CoMULus-	[IDXi*] , [Rw_m*]	93 Xm 88 rrr0:0qqq	4

CoNEG Negate Accumulator CoNEG

Group Arithmetic Instructions

Syntax **CoNEG**

Source Operand(s) ACC → 40-bit signed value

Destination Operand(s) ACC → 40-bit signed value

Operation

(ACC) ← 0 - (ACC)

Description

The ACC register contents are subtracted from zero before being stored in the 40-bit ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic	Format	Bytes
CoNEG	A3 00 32 rrr0:0000	4

CoNEG Negate Accumulator with Round CoNEG

Group Arithmetic Instructions

Syntax **CoNEG rnd**

Source Operand(s) ACC → 40-bit signed value

Destination Operand(s) ACC → 40-bit signed value

Operation

(ACC) ← 0 - (ACC) + 00 0000 8000h
(MAL) ← 0

Description

The ACC register contents are subtracted from zero and the result is rounded before being stored in the 40-bit ACC register. The MAL register is cleared.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoNEG	rnd	A3 00 72 rrr0:0000	4

CoNOP No-Operation CoNOP

Group Arithmetic Instructions

Syntax **CoNOP**

Source Operand(s) none

Destination Operand(s) none

Operation

No Operation

Description

Modifies the address pointers.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
-	-	-	-	-	-	-	no

MV Not affected.
MSL Not affected.
ME Not affected.
MSV Not affected.
MC Not affected.
MZ Not affected.
MN Not affected.

Encoding

Mnemonic		Format	Bytes
CoNOP	[IDXi*] , [Rw_m*]	93 Xm 5A rrr0:0qqq	4
CoNOP	[IDXi*]	93 X0 5A rrr0:0001	4
CoNOP	[Rw_m*]	93 1m 5A rrr0:0qqq	4

CoRND Round Accumulator CoRND

Group Shift Instructions

Syntax **CoRND**

Source Operand(s) ACC → 40-bit signed value

Destination Operand(s) ACC → 40-bit signed value signed value

Operation

(ACC) ← (ACC) + 00 0000 8000h
(MAL) ← 0

Description

Rounds the ACC register contents by adding 00 0000 8000h and stores the result in the ACC register and the lower part of the ACC register. MAL, is cleared.

Note: CoRND is a shortname for CoASHR #0, rnd

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic overflow occurred. Not affected otherwise.

MC Set if a carry is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic	Format	Bytes
CoRND	A3 00 B2 rrr0:0000	4

CoSHL Accumulator Logical Shift Left CoSHL

Group Shift Instructions

Syntax **CoSHL op1**

Source Operand(s) op1 → 5-bit unsigned data

Destination Operand(s) ACC → 40-bit signed value

Operation

```
(count) ← (op1)
(C) <- (ACC[39])
DO WHILE ((count) ≠ 0)
        (C) ← (ACC[39])
        (ACC[n]) ← (ACC[n-1]) [n=39...1]
        (ACC[0]) ← 0
        (count) ← (count) -1
END WHILE
```

Description

Shifts the 40-bit ACC register contents left by the number of times specified by the operand op1. The least significant bits of the result are filled with zeros accordingly. Only shift values from 0 to 16 (inclusive) are allowed. op1 can be either a 5-bit unsigned immediate data (the shift range is from 0 to 16 in this case) or the four least significant bits (the shift range is from 0 to 15 in that case) of any register directly or indirectly addressed operand.

Note: For this instruction only, the saturation is computed using the 40-bit result. So a sign shifted over the 40 bit result is disregarded.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	*	*	-	*	*	*	yes

MV Always cleared.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Not affected.

MC Carry flag is set according to the last most significant bit shifted out of ACC or according to the sign of ACC.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoSHL	#data5	A3 00 82 rrr#:#	4
CoSHL	Rw_n	A3 nn 8A rrr0:0000	4
CoSHL	[Rw_m*]	83 mm 8A rrr0:0qqq	4

CoSHR Accumulator Logical Shift Right CoSHR

Group Shift Instructions

Syntax **CoSHR op1**

Source Operand(s) op1 → 5-bit unsigned data

Destination Operand(s) ACC → 40-bit signed value

Operation

```
(count) ← (op1)
(C) ← 0
DO WHILE (count) ≠ 0
        ((ACC[n]) ← (ACC[n+1]) [n=0...38]
        (ACC[39]) ← 0
        (count) ← (count) -1
END WHILE
```

Description

Shifts the 40-bit ACC register contents right the number of times as specified by the operand op1. The most significant bits of the result are filled with zeros accordingly. Only shift values from 0 to 16 (inclusive) are allowed. op1 can be either a 5-bit unsigned immediate data (the shift range is from 0 to 16 in this case) or the four least significant bits (the shift range is from 0 to 15 in that case) of any register directly or indirectly addressed operand. The MS bit of the MCW register does not affect the result.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
0	-	*	-	0	*	*	no

MV Always cleared.
MSL Not affected.
ME Set if the MAE is used. Cleared otherwise.
MSV Not affected.
MC Always cleared.
MZ Set if result equals zero. Cleared otherwise.
MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoSHR	#data5	A3 00 92 rrr#:#	4
CoSHR	Rw_n	A3 nn 9A rrr0:0000	4
CoSHR	[Rw_m*]	83 mm 9A rrr0:0qqq	4

CoSTORE Store a MAC-Unit Register CoSTORE

Group Data Movement Instructions

Syntax **CoSTORE op1, op2**

Source Operand(s) op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op2)

Description

Moves the contents of a MAC-Unit register specified by the source operand op2 to the location specified by the destination operand op1.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
-	-	-	-	-	-	-	no

MV Not affected.
MSL Not affected.
ME Not affected.
MSV Not affected.
MC Not affected.
MZ Not affected.
MN Not affected.

Encoding

Mnemonic		Format	Bytes
CoSTORE	Rw_n , CoReg	C3 nn wwww:w000 rrr0:0000	4
CoSTORE	[Rw_n*] , CoReg	B3 nn wwww:w000 rrr0:0qqq	4

CoSUB Subtract CoSUB

Group Arithmetic Instructions

Syntax **CoSUB op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op2) || (op1)
(ACC) ← (ACC) - (tmp)

Description

Subtracts a 40-bit operand from the 40-bit ACC contents and stores the result in the ACC register. The 40-bit operand is a sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW).

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoSUB	Rw_n , Rw_m	A3 nm 0A rrr0:0000	4
CoSUB	Rw_n , [Rw_m*]	83 nm 0A rrr0:0qqq	4
CoSUB	[IDXi*] , [Rw_m*]	93 Xm 0A rrr0:0qqq	4

CoSUB2 Subtract CoSUB2

Group Arithmetic Instructions

Syntax **CoSUB2 op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← 2 * (op2) || (op1)
(ACC) ← (ACC) - (tmp)

Description

Subtracts a 40-bit operand from the 40-bit ACC contents and stores the result in the ACC register. The 40-bit operand is a sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW). The 40-bit operand is then multiplied by two before being subtracted from the ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoSUB2	Rw_n , Rw_m	A3 nm 4A rrr0:0000	4
CoSUB2	Rw_n , [Rw_m*]	83 nm 4A rrr0:0qqq	4
CoSUB2	[IDXi*] , [Rw_m*]	93 Xm 4A rrr0:0qqq	4

CoSUB2R Subtract CoSUB2R

Group Arithmetic Instructions

Syntax **CoSUB2R op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← 2 * (op2) || (op1)
(ACC) ← (tmp) - (ACC)

Description

Subtracts the 40-bit ACC contents from a 40-bit operand and stores the result in the ACC register. The 40-bit operand is a sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW). The 40-bit operand is then multiplied by two before being subtracted from the ACC register.

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.

MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.

ME Set if the MAE is used. Cleared otherwise.

MSV Set if an arithmetic underflow occurred. Not affected otherwise.

MC Set if a borrow is generated. Cleared otherwise.

MZ Set if result equals zero. Cleared otherwise.

MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoSUB2R	Rw_n , Rw_m	A3 nm 52 rrr0:0000	4
CoSUB2R	Rw_n , [Rw_m*]	83 nm 52 rrr0:0qqq	4
CoSUB2R	[IDXi*] , [Rw_m*]	93 Xm 52 rrr0:0qqq	4

CoSUBR Subtract CoSUBR

Group Arithmetic Instructions

Syntax **CoSUBR op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) ACC → 40-bit signed value

Operation

(tmp) ← (op2) || (op1)
(ACC) ← (tmp) - (ACC)

Description

Subtracts the 40-bit ACC contents from a 40-bit operand and stores the result in the ACC register. The 40-bit operand is a sign-extended result of the concatenation of the two source operands op1 (LSW) and op2 (MSW).

MAC Flags

MV	MSL	ME	MSV	MC	MZ	MN	Sat.
*	*	*	*	*	*	*	yes

- MV Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 40-bit data type. Cleared otherwise.
- MSL Set if the contents of ACC is automatically saturated. Not affected otherwise.
- ME Set if the MAE is used. Cleared otherwise.
- MSV Set if an arithmetic underflow occurred. Not affected otherwise.
- MC Set if a borrow is generated. Cleared otherwise.
- MZ Set if result equals zero. Cleared otherwise.
- MN Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CoSUBR	Rw_n , Rw_m	A3 nm 12 rrr0:0000	4
CoSUBR	Rw_n , [Rw_m*]	83 nm 12 rrr0:0qqq	4
CoSUBR	[IDXi*] , [Rw_m*]	93 Xm 12 rrr0:0qqq	4

8.3 Instructions for OCDS/ITC injection and System Control

The following table gives a brief overview of the instructions that are defined especially for injections via the Interrupt and PEC controller and for debugging reasons by the OCDS. All instruction are 32 bit wide and overlap the existing instruction set. All these instructuions are not modifying the PSW except direct writes to the PSW and the ITRAP/ ITRAPS instruction that adjust the level inside the PSW. All these instructions are only available for injection.

operand	symbol	size	comment
mem24	MM_2 MM_0 MM_1	24	direct 24 bit address for memory access. The format MM_2 MM_0 MM_1 means that the 24 bit address (byte2,byte1,byte0) has to be presented in the order byte2, byte0, byte1.
#addr23	aa aa a:aaa	23	direct 23bit (to be LSB extended by zero) for program access.
#banksel2	ss	2	selection of local/global banks 00 global register bank 10 local register bank 1 11 local register bank 2 01 reserved
#data23	dd dd d:ddd	23	direct 23bit (to be LSB extended by zero) data to be written to CSP/IP.
Rx	x	4	word GPR address
Rbx	x	4	byte GPR address

Table 8-1 Used shortcuts

Mnemonic	Operands	Opcode	Cycle	Comment
OLOAD	mem24	0D MM_2 MM_0 MM_1	1	reads word from memory and writes to OCDS
OSTORE	mem24	1D MM_2 MM_0 MM_1	1	reads word from OCDS and writes to memory

Table 8-2 Instructions for Injection only

Mnemonic	Operands	Opcode	Cycle	Comment
OLOADB	mem24	2D MM_2 MM_0 MM_1	1	reads byte from memory and writes to OCDS
OSTOREB	mem24	3D MM_2 MM_0 MM_1	1	reads byte from OCDS and writes to memory
OLOAD	Rx, #banksel2	4D ss00:x 00 00	1	reads word from GPR and writes to OCDS
OSTORE	Rx, #banksel2	5D ss00:x 00 00	1	reads word from OCDS and writes to GPR
OLOADB	Rbx, #banksel2	6D ss00:x 00 00	1	reads byte from GPR and writes to OCDS
OSTOREB	Rbx, #banksel2	7D ss00:x 00 00	1	reads byte from OCDS and writes to GPR
MOVCSIP	#data23	9D dd dd d:ddd0	1	writes CSP/IP register to force a program brnach
OLOADIP		8D 00 00 00	1	reads the current instruction pointer and writes it to OCDS
ITRAP	#addr23, #banksel2	10ss:B aa aa a:aaa0	4	Interrupt Trap with absolut address
ITRAPS	#trap10, #banksel2	11ss:B 00 0t t:tt00“	4	Short Interrupt Trap with10 bit trap number using VECSEG

Table 8-2 Instructions for Injection only

Mnemonic	Operands	Opcode	Cycle	Comment
PEC[1)]	mem24	CD MM_2 MM_0 MM_1	1	word PEC transfer started by ITC
DPEC[1)]	mem24	DD MM_2 MM_0 MM_1	1	word PEC transfer started by OCDS
PECB[1)]	mem24	AD MM_2 MM_0 MM_1	1	byte PEC transfer started by ITC
DPECB[1)]	mem24	BD MM_2 MM_0 MM_1	1	byte PEC transfer started by OCDS
TLOAD	mem24	8A MM_2 MM_0 MM_1	1	reads memory and writes result to trace bus
TLOAD	Rx, #banksel2	CA ss00:x 00 00	1	reads GPR and writes result to trace bus
INOP		9A 00 00 00 AA 00 00 00 BA 00 00 00 DA 00 00 00 EA 00 00 00 FA 00 00 00	1	injected NOP (reserved for later use)
CXLOAD		ED 00 00 00	11	internal instruction used for switch context
CXSW		FD 00 00 00	19	internal instruction used for switch context

Table 8-2 Instructions for Injection only

1) The shown operand specifies the source address for the PEC operation. For the destination address a dedicated CPU input is provided.

9 Summary of CPU/Subsystem Registers

This chapter summarizes all registers implemented in the C166S V2 CPU. There are two register types: the General Purpose Registers (GPR) and the CPU-Special Function Registers (CSFR). GPRs are the working registers of the arithmetic and logic operations and may be also used as address pointers indirect addressing modes. CSFRs are the control registers of the C166S V2 CPU. The register set for the PEC and Interrupt Controller is listed. For easy reference, the SFRs are ordered in two different ways:

- Sorted by the address, to identify a register at a given address.
- Sorted by the register name, to find an address of a specific register.

9.1 General Purpose Registers (GPRs)

The General Purpose Registers **(**GPRs) are the working registers of the C166S V2 CPU. All GPRs are bit addressable.

Table 9-1 Addressing Modes to Access Word–GPRs

Name	Physical Address 1)	8-Bit Address	4-Bit Address	Description	Reset Value
R0	(CP)+0	$F0_H$	0h	General Purpose Word Register R0	$UUUU_H$
R1	(CP)+2	$F1_H$	1h	General Purpose Word Register R1	$UUUU_H$
R2	(CP)+4	$F2_H$	2h	General Purpose Word Register R2	$UUUU_H$
R3	(CP)+6	$F3_H$	3h	General Purpose Word Register R3	$UUUU_H$
R4	(CP)+8	$F4_H$	4h	General Purpose Word Register R4	$UUUU_H$
R5	(CP)+10	$F5_H$	5h	General Purpose Word Register R5	$UUUU_H$
R6	(CP)+12	$F6_H$	6h	General Purpose Word Register R6	$UUUU_H$
R7	(CP)+14	$F7_H$	7h	General Purpose Word Register R7	$UUUU_H$
R8	(CP)+16	$F8_H$	8h	General Purpose Word Register R8	$UUUU_H$
R9	(CP)+18	$F9_H$	9h	General Purpose Word Register R9	$UUUU_H$
R10	(CP)+20	FA_H	Ah	General Purpose Word Register R10	$UUUU_H$
R11	(CP)+22	FB_H	Bh	General Purpose Word Register R11	$UUUU_H$
R12	(CP)+24	FC_H	Ch	General Purpose Word Register R12	$UUUU_H$
R13	(CP)+26	FD_H	Dh	General Purpose Word Register R13	$UUUU_H$
R14	(CP)+28	FE_H	Eh	General Purpose Word Register R14	$UUUU_H$
R15	(CP)+30	FF_H	Fh	General Purpose Word Register R15	$UUUU_H$

1) Addressing mode only usable if the GPR bank is memory mapped.

The first 8 GPRs (R7...R0) may be also accessed bytewise. Unlike SFRs, writing to a GPR byte does not affect another byte of the GPR.

The following byte-accessible registers have special names.

Table 9-2 Addressing Modes to Access Byte–GPRs

Name	Physical Address 1)	8-Bit Address	4-Bit Address	Description	Reset Value
RL0	(CP)+0	$F0_H$	0h	General Purpose Byte Register RL0	UU_H
RH0	(CP)+1	$F1_H$	1h	General Purpose Byte Register RL1	UU_H
RL1	(CP)+2	$F2_H$	2h	General Purpose Byte Register RL2	UU_H
RH1	(CP)+3	$F3_H$	3h	General Purpose Byte Register RL3	UU_H
RL2	(CP)+4	$F4_H$	4h	General Purpose Byte Register RL4	UU_H
RH2	(CP)+5	$F5_H$	5h	General Purpose Byte Register RL5	UU_H
RL3	(CP)+6	$F6_H$	6h	General Purpose Byte Register RL6	UU_H
RH3	(CP)+7	$F7_H$	7h	General Purpose Byte Register RL7	UU_H
RL4	(CP)+8	$F8_H$	8h	General Purpose Byte Register RL8	UU_H
RH4	(CP)+9	$F9_H$	9h	General Purpose Byte Register RL9	UU_H
RL5	(CP)+10	FA_H	Ah	General Purpose Byte Register RL10	UU_H
RH5	(CP)+11	FB_H	Bh	General Purpose Byte Register RL11	UU_H
RL6	(CP)+12	FC_H	Ch	General Purpose Byte Register RL12	UU_H
RH6	(CP)+13	FD_H	Dh	General Purpose Byte Register RL13	UU_H
RL7	(CP)+14	FE_H	Eh	General Purpose Byte Register RL14	UU_H
RH7	(CP)+15	FF_H	Fh	General Purpose Byte Register RL15	UU_H

1) Addressing mode only usable if the GPR bank is memory mapped.

The 8-bit short addresses $F0_H$...FE_H within the ESFR area are reserved and provide access to the current register bank via short register addressing modes. The GPRs are mirrored to the ESFR area which allows access to the current register bank even after switching register spaces (see example below).

```
MOV    R5, DP3   ;GPR access via SFR area
EXTR   #1
MOV    R5, ODP3  ;GPR access via ESFR area
```

9.2 Core Special Function Registers

9.2.1 Ordered by Name

Table 9-3 lists all CSFRs implemented in the C166S V2 CPU, in alphabetical order.
Bit addressable CSFRs are marked with the letter “b” in the “Name” column.
CSFRs within the **Extended CSFR-Space** (ECSFRs) are marked with the letter “E” in the “8-Bit Address” column.

Table 9-3 Addressing Modes to Access Core-SFRs: Ordered by Name

Name		Physical Address	8-Bit Address	Description	Reset Value
CP		$FE10_H$	08_H	Context Pointer	$FC00_H$
CPUCON1		$FE18_H$	$0C_H$	Core Control Register	0000_H
CPUCON2		$FE1A_H$	$0D_H$	Core Control Register	0000_H
CPUID		$F00C_H$	$E\text{-}06_H$	CPU Identification Register	$03??_H$ [1]
CSP		$FE08_H$	04_H	Code Segment Pointer (8 bits, not directly writable)	0000_H
DPP0		$FE00_H$	00_H	Data Page Pointer 0 (10 bits)	0000_H
DPP1		$FE02_H$	01_H	Data Page Pointer 1 (10 bits)	0001_H
DPP2		$FE04_H$	02_H	Data Page Pointer 2 (10 bits)	0002_H
DPP3		$FE06_H$	03_H	Data Page Pointer 3 (10 bits)	0003_H
IDX0	b	$FF08_H$	84_H	MAC Address Pointer 0	0000_H
IDX1	b	$FF0A_H$	85_H	MAC Address Pointer 1	0000_H
MAL		$FE5C_H$	$2E_H$	MAC Accumulator – Low Word	0000_H
MAH		$FE5E_H$	$2F_H$	MAC Accumulator – High Word	0000_H
MCW	b	$FFDC_H$	EE_H	MAC Control Word	0000_H
MDC	b	$FF0E_H$	87_H	Multiply Divide Control Register	0000_H
MDH		$FE0C_H$	06_H	Multiply Divide Register – High Word	0000_H
MDL		$FE0E_H$	07_H	Multiply Divide Register – Low Word	0000_H
MRW	b	$FFDA_H$	ED_H	MAC Repeat Word	0000_H
MSW	b	$FFDE_H$	EF_H	MAC Status Word	0200_H
ONES	b	$FF1E_H$	$8F_H$	Constant Value 1’s Register (read only)	$FFFF_H$
PSW	b	$FF10_H$	88_H	Program Status Word	0000_H
QX0		$F000_H$	$E\text{-}00_H$	MAC Offset Register X0	0000_H

Table 9-3 Addressing Modes to Access Core-SFRs: Ordered by Name (cont'd)

Name	Physical Address	8-Bit Address	Description	Reset Value
QX1	$F002_H$	$E\text{-}01_H$	MAC Offset Register X1	0000_H
QR0	$F004_H$	$E\text{-}02_H$	MAC Offset Register R0	0000_H
QR1	$F006_H$	$E\text{-}03_H$	MAC Offset Register R1	0000_H
SP	$FE12_H$	09_H	Stack Pointer	$FC00_H$
SPSEG b	$FF0C_H$	86_H	Stack Pointer Segment Register	0000_H
STKOV	$FE14_H$	$0A_H$	Stack Overflow Register	$FA00_H$
STKUN	$FE16_H$	$0B_H$	Stack Underflow Register	$FC00_H$
TFR b	$FFAC_H$	$D6_H$	Trap Flag Register	0000_H
VECSEG b	$FF12_H$	89_H	Vector Table Segment Register	$????_H$ [2]
ZEROS b	$FF1C_H$	$8E_H$	Constant Value 0's Register (read only)	0000_H

[1] '??': defined by reset configuration

[2] '????': defined by reset configuration

9.2.2 Ordered by Address

Table 9-4 lists all CSFRs implemented in the C166S V2 ordered by physical address.

Bit addressable CSFRs are marked with the letter "b" in the "Name" column.
CSFRs within the **Extended SFR-Space** (ESFRs) are marked with the letter "E" in the "8-Bit Address" column.

Table 9-4 Addressing Modes to Access Core-SFRs: Ordered by Address

Name	Physical Address	8-Bit Address	Description	Reset Value
QX0	$F000_H$	$E\text{-}00_H$	MAC Offset Register X0	0000_H
QX1	$F002_H$	$E\text{-}01_H$	MAC Offset Register X1	0000_H
QR0	$F004_H$	$E\text{-}02_H$	MAC Offset Register R0	0000_H
QR1	$F006_H$	$E\text{-}03_H$	MAC Offset Register R1	0000_H
CPUID	$F00C_H$	$E\text{-}06_H$	CPU Identification Register	$03??_H$ [1]
DPP0	$FE00_H$	00_H	Data Page Pointer 0 (10 bits)	0000_H
DPP1	$FE02_H$	01_H	Data Page Pointer 1 (10 bits)	0001_H
DPP2	$FE04_H$	02_H	Data Page Pointer 2 (10 bits)	0002_H
DPP3	$FE06_H$	03_H	Data Page Pointer 3 (10 bits)	0003_H

Table 9-4 Addressing Modes to Access Core-SFRs: Ordered by Address

Name	Physical Address	8-Bit Address	Description	Reset Value
CSP	$FE08_H$	04_H	Code Segment Pointer (8 bits, not directly writable)	0000_H
MDH	$FE0C_H$	06_H	Multiply Divide Register – High Word	0000_H
MDL	$FE0E_H$	07_H	Multiply Divide Register – Low Word	0000_H
CP	$FE10_H$	08_H	Context Pointer	$FC00_H$
SP	$FE12_H$	09_H	Stack Pointer	$FC00_H$
STKOV	$FE14_H$	$0A_H$	Stack Overflow Register	$FA00_H$
STKUN	$FE16_H$	$0B_H$	Stack Underflow Register	$FC00_H$
CPUCON1	$FE18_H$	$0C_H$	Core Control Register	0000_H
CPUCON2	$FE1A_H$	$0D_H$	Core Control Register	0000_H
MAL	$FE5C_H$	$2E_H$	MAC Accumulator – Low Word	0000_H
MAH	$FE5E_H$	$2F_H$	MAC Accumulator – High Word	0000_H
IDX0 b	$FF08_H$	84_H	MAC Address Pointer 0	0000_H
IDX1 b	$FF0A_H$	85_H	MAC Address Pointer 1	0000_H
SPSEG b	$FF0C_H$	86_H	Stack Pointer Segment Register	0000_H
MDC b	$FF0E_H$	87_H	Multiply Divide Control Register	0000_H
PSW b	$FF10_H$	88_H	Program Status Word	0000_H
VECSEG b	$FF12_H$	89_H	Vector Table Segment Register	$????_H$ [2]
ZEROS b	$FF1C_H$	$8E_H$	Constant Value 0s Register (read only)	0000_H
ONES b	$FF1E_H$	$8F_H$	Constant Value 1s Register (read only)	$FFFF_H$
TFR b	$FFAC_H$	$D6_H$	Trap Flag Register	0000_H
MRW b	$FFDA_H$	ED_H	MAC Repeat Word	0000_H
MCW b	$FFDC_H$	EE_H	MAC Control Word	0000_H
MSW b	$FFDE_H$	EF_H	MAC Status Word	0200_H

1) '??': defined by reset configuration

2) '????': defined by reset configuration

9.3 Register Overview Interrupt and Peripheral Event Controller

9.3.1 Ordered by Name

Table 9-5 lists all xSFRs that are implemented in the C166S V2 Interrupt and Peripheral Event Controller, ordered by name.

Bit addressable SFRs are marked with the letter “b” in the “Name” column.
SFRs within the **Extended SFR-Space** (ESFRs) are marked with the letter “E” in the “8-Bit Address” column.

Table 9-5 Register Overview Interrupt and PEC: Ordered by Name

Name	Physical Address	8-bit Address	Description	Reset Value
BNKSEL0	$EC20_H$	--	Bank Selection Register 0	0000_H
BNKSEL1	$EC22_H$	--	Bank Selection Register 1	0000_H
BNKSEL2	$EC24_H$	--	Bank Selection Register 2	0000_H
BNKSEL3	$EC26_H$	--	Bank Selection Register 3	0000_H
DSTP0	$EC42_H$	--	PEC Channel 0 Destination Pointer	0000_H
DSTP1	$EC46_H$	--	PEC Channel 1 Destination Pointer	0000_H
DSTP2	$EC4A_H$	--	PEC Channel 2 Destination Pointer	0000_H
DSTP3	$EC4E_H$	--	PEC Channel 3 Destination Pointer	0000_H
DSTP4	$EC52_H$	--	PEC Channel 4 Destination Pointer	0000_H
DSTP5	$EC56_H$	--	PEC Channel 5 Destination Pointer	0000_H
DSTP6	$EC5A_H$	--	PEC Channel 6 Destination Pointer	0000_H
DSTP7	$EC5E_H$	--	PEC Channel 7 Destination Pointer	0000_H
EOPIC[1] b	$F180_H$	$E\text{-}C0_H$	End of PEC Interrupt Control Reg.	0000_H
FINT0ADDR	$EC02_H$	--	Fast Interrupt 0 Address Register	0000_H
FINT0CSP	$EC00_H$	--	Fast Interrupt 0 CSP Register	0000_H
FINT1ADDR	$EC06_H$	--	Fast Interrupt 1 Address Register	0000_H
FINT1CSP	$EC04_H$	--	Fast Interrupt 1 CSP Register	0000_H
IRQxIC[1]	$xxxx_H$	xx_H	Interrupt x Control Register	0000_H
PECC0	$FEC0_H$	60_H	PEC Channel 0 Control Register	0000_H
PECC1	$FEC2_H$	61_H	PEC Channel 1 Control Register	0000_H
PECC2	$FEC4_H$	62_H	PEC Channel 2 Control Register	0000_H
PECC3	$FEC6_H$	63_H	PEC Channel 3 Control Register	0000_H
PECC4	$FEC8_H$	64_H	PEC Channel 4 Control Register	0000_H

Table 9-5 Register Overview Interrupt and PEC: Ordered by Name (cont'd)

Name	Physical Address	8-bit Address	Description	Reset Value
PECC5	$FECA_H$	65_H	PEC Channel 5 Control Register	0000_H
PECC6	$FECC_H$	66_H	PEC Channel 6 Control Register	0000_H
PECC7	$FECE_H$	67_H	PEC Channel 7 Control Register	0000_H
PECISNC b	$FFA8_H$	$D4_H$	PEC Interrupt Subnode Control Reg.	0000_H
PECSEG0	$EC80_H$	--	PEC Pointer 0 Segment Address Reg.	0000_H
PECSEG1	$EC82_H$	--	PEC Pointer 1 Segment Address Reg.	0000_H
PECSEG2	$EC84_H$	--	PEC Pointer 2 Segment Address Reg.	0000_H
PECSEG3	$EC86_H$	--	PEC Pointer 3 Segment Address Reg.	0000_H
PECSEG4	$EC88_H$	--	PEC Pointer 4 Segment Address Reg.	0000_H
PECSEG5	$EC8A_H$	--	PEC Pointer 5 Segment Address Reg.	0000_H
PECSEG6	EC8C	--	PEC Pointer 6 Segment Address Reg.	0000
PECSEG7	EC8E	--	PEC Pointer 7 Segment Address Reg.	0000
SRCP0	EC40	--	PEC Channel 0 Source Pointer	0000
SRCP1	EC44	--	PEC Channel 1 Source Pointer	0000
SRCP2	EC48	--	PEC Channel 2 Source Pointer	0000
SRCP3	EC4C	--	PEC Channel 3 Source Pointer	0000
SRCP4	EC50	--	PEC Channel 4 Source Pointer	0000
SRCP5	EC54	--	PEC Channel 5 Source Pointer	0000
SRCP6	EC58	--	PEC Channel 6 Source Pointer	0000
SRCP7	EC5C	--	PEC Channel 7 Source Pointer	0000

1) The implementation and assignment of these Interrupt Control Registers are product specific.

9.3.2 Ordered by Address

Table 9-6 lists all xSFRs that are implemented in the C166S V2 Interrupt and Peripheral Event Controller ordered by address.

Bit addressable SFRs are marked with the letter "b" in the "Name" column.
SFRs within the **Extended SFR-Space** (ESFRs) are marked with the letter "E" in the "8-Bit Address" column.

Table 9-6 Register Overview Interrupt and PEC: Ordered by Address

Name	Physical Address	8-bit Address	Description	Reset Value
FINT0CSP	$EC00_H$	--	Fast Interrupt 0 CSP Register	0000_H
FINT0ADDR	$EC02_H$	--	Fast Interrupt 0 Address Register	0000_H
FINT1CSP	$EC04_H$	--	Fast Interrupt 1 CSP Register	0000_H
FINT1ADDR	$EC06_H$	--	Fast Interrupt 1 Address Register	0000_H
BNKSEL0	$EC20_H$	--	Bank Selection Register 0	0000_H
BNKSEL1	$EC22_H$	--	Bank Selection Register 1	0000_H
BNKSEL2	$EC24_H$	--	Bank Selection Register 2	0000_H
BNKSEL3	$EC26_H$	--	Bank Selection Register 3	0000_H
SRCP0	$EC40_H$	--	PEC Channel 0 Source Pointer	0000_H
DSTP0	$EC42_H$	--	PEC Channel 0 Destination Pointer	0000_H
SRCP1	$EC44_H$	--	PEC Channel 1 Source Pointer	0000_H
DSTP1	$EC46_H$	--	PEC Channel 1 Destination Pointer	0000_H
SRCP2	$EC48_H$	--	PEC Channel 2 Source Pointer	0000_H
DSTP2	$EC4A_H$	--	PEC Channel 2 Destination Pointer	0000_H
SRCP3	$EC4C_H$	--	PEC Channel 3 Source Pointer	0000_H
DSTP3	$EC4E_H$	--	PEC Channel 3 Destination Pointer	0000_H
SRCP4	$EC50_H$	--	PEC Channel 4 Source Pointer	0000_H
DSTP4	$EC52_H$	--	PEC Channel 4 Destination Pointer	0000_H
SRCP5	$EC54_H$	--	PEC Channel 5 Source Pointer	0000_H
DSTP5	$EC56_H$	--	PEC Channel 5 Destination Pointer	0000_H
SRCP6	$EC58_H$	--	PEC Channel 6 Source Pointer	0000_H
DSTP6	$EC5A_H$	--	PEC Channel 6 Destination Pointer	0000_H
SRCP7	$EC5C_H$	--	PEC Channel 7 Source Pointer	0000_H
DSTP7	$EC5E_H$	--	PEC Channel 7 Destination Pointer	0000_H
PECSEG0	$EC80_H$	--	PEC Pointer 0 Segment Address Reg.	0000_H
PECSEG1	$EC82_H$	--	PEC Pointer 1 Segment Address Reg.	0000_H
PECSEG2	$EC84_H$	--	PEC Pointer 2 Segment Address Reg.	0000_H
PECSEG3	$EC86_H$	--	PEC Pointer 3 Segment Address Reg.	0000_H
PECSEG4	$EC88_H$	--	PEC Pointer 4 Segment Address Reg.	0000_H

Table 9-6 Register Overview Interrupt and PEC: Ordered by Address (cont'd)

Name	Physical Address	8-bit Address	Description	Reset Value
PECSEG5	$EC8A_H$	--	PEC Pointer 5 Segment Address Reg.	0000_H
PECSEG6	$EC8C_H$	--	PEC Pointer 6 Segment Address Reg.	0000_H
PECSEG7	$EC8E_H$	--	PEC Pointer 7 Segment Address Reg.	0000_H
EOPIC[1] b	$F180_H$	$E\text{-}C0_H$	End of PEC Interrupt Control Reg.	0000_H
PECC0	$FEC0_H$	60_H	PEC Channel 0 Control Register	0000_H
PECC1	$FEC2_H$	61_H	PEC Channel 1 Control Register	0000_H
PECC2	$FEC4_H$	62_H	PEC Channel 2 Control Register	0000_H
PECC3	$FEC6_H$	63_H	PEC Channel 3 Control Register	0000_H
PECC4	$FEC8_H$	64_H	PEC Channel 4 Control Register	0000_H
PECC5	$FECA_H$	65_H	PEC Channel 5 Control Register	0000_H
PECC6	$FECC_H$	66_H	PEC Channel 6 Control Register	0000_H
PECC7	$FECE_H$	67_H	PEC Channel 7 Control Register	0000_H
IRQxIC[1]	$xxxx_H$	xx_H	Interrupt x Control Register	0000_H
PECISNC b	$FFA8_H$	$D0_H$	PEC Interrupt Subnode Control Reg.	0000_H

[1] The implementation and assignment of theses Interrupt Control Registers are product specific.

9.4 Register Overview External Bus Controller

9.4.1 Ordered by Name

Table 9-7 Register Overview EBC: Ordered by Name

Name	Physical Address	8-Bit Address	Description	Reset Value
ADDRSEL1	$EE1E_H$	--	Address Window Selection for $\overline{CS1}$	0000_H
ADDRSEL2	$EE26_H$	--	Address Window Selection for $\overline{CS2}$	0000_H
ADDRSEL3	$EE\ 2E_H$	--	Address Window Selection for $\overline{CS3}$	0000_H
ADDRSEL4	$EE36_H$	--	Address Window Selection for $\overline{CS4}$	0000_H
ADDRSEL5	$EE\ 3E_H$	--	Address Window Selection for $\overline{CS5}$	0000_H
ADDRSEL6	$EE46_H$	--	Address Window Selection for $\overline{CS6}$	0000_H
ADDRSEL7	$EE\ 4E_H$	--	Address Window Selection for $\overline{CS7}$	0000_H
EBCMOD0	$EE00_H$	--	Alternate Function of EBC Pins	$00F0_H$
EBCMOD1	$EE02_H$	--	Global Behavior of EBC	0000_H
FCONCS0	$EE12_H$	--	Function Control for $\overline{CS0}$	0021_H
FCONCS1	$EE1A_H$	--	Function Control for $\overline{CS1}$	0000_H
FCONCS2	$EE22_H$	--	Function Control for $\overline{CS2}$	0000_H
FCONCS3	$EE2A_H$	--	Function Control for $\overline{CS3}$	0000_H
FCONCS4	$EE32_H$	--	Function Control for $\overline{CS4}$	0000_H
FCONCS5	$EE\ 3A_H$	--	Function Control for $\overline{CS5}$	0000_H
FCONCS6	$EE\ 42_H$	--	Function Control for $\overline{CS6}$	0000_H
FCONCS7	$EE4A_H$	--	Function Control for $\overline{CS7}$	0000_H
TCONCS0	$EE10_H$	--	Timing Control for $\overline{CS0}$	6243_H
TCONCS1	$EE18_H$	--	Timing Control for $\overline{CS1}$	0000_H
TCONCS2	$EE20_H$	--	Timing Control for $\overline{CS2}$	0000_H
TCONCS3	$EE28_H$	--	Timing Control for $\overline{CS3}$	0000_H
TCONCS4	$EE\ 30_H$	--	Timing Control for $\overline{CS4}$	0000_H
TCONCS5	$EE38_H$	--	Timing Control for $\overline{CS5}$	0000_H
TCONCS6	$EE40_H$	--	Timing Control for $\overline{CS6}$	0000_H
TCONCS7	$EE48_H$	--	Timing Control for $\overline{CS7}$	0000_H
TCONCSMM	$EE0C_H$	--	Timing Control for Monitor Memory	6243_H
TCONCSSM	$EE0E_H$	--	Timing Control for Startup Memory	6243_H

9.4.2 Ordered by Address

Table 9-8 Register Overview EBC: Ordered by Name

Name	Physical Address	8-Bit Address	Description	Reset Value
EBCMOD0	$EE00_H$	--	Alternate Function of EBC Pins	$00F0_H$
EBCMOD1	$EE02_H$	--	Global Behavior of EBC	0000_H
TCONCSMM	$EE0C_H$	--	Timing Control for Monitor Memory	6243_H
TCONCSSM	$EE0E_H$	--	Timing Control for Startup Memory	6243_H
TCONCS0	$EE10_H$	--	Timing Control for $\overline{CS0}$	6243_H
FCONCS0	$EE12_H$	--	Function Control for $\overline{CS0}$	0021_H
TCONCS1	$EE18_H$	--	Timing Control for $\overline{CS1}$	0000_H
FCONCS1	$EE1A_H$	--	Function Control for $\overline{CS1}$	0000_H
ADDRSEL1	$EE1E_H$	--	Address Window Selection for $\overline{CS1}$	0000_H
TCONCS2	$EE20_H$	--	Timing Control for $\overline{CS2}$	0000_H
FCONCS2	$EE22_H$	--	Function Control for $\overline{CS2}$	0000_H
ADDRSEL2	$EE26_H$	--	Address Window Selection for $\overline{CS2}$	0000_H
TCONCS3	$EE28_H$	--	Timing Control for $\overline{CS3}$	0000_H
FCONCS3	$EE2A_H$	--	Function Control for $\overline{CS3}$	0000_H
ADDRSEL3	$EE2E_H$	--	Address Window Selection for $\overline{CS3}$	0000_H
TCONCS4	$EE30_H$	--	Timing Control for $\overline{CS4}$	0000_H
FCONCS4	$EE32_H$	--	Function Control for $\overline{CS4}$	0000_H
ADDRSEL4	$EE36_H$	--	address window selection for $\overline{CS4}$	0000_H
TCONCS5	$EE38_H$	--	Timing Control for $\overline{CS5}$	0000_H
FCONCS5	$EE3A_H$	--	Function Control for $\overline{CS5}$	0000_H
ADDRSEL5	$EE3E_H$	--	Address Window Selection for $\overline{CS5}$	0000_H
TCONCS6	$EE40_H$	--	Timing Control for $\overline{CS6}$	0000_H
FCONCS6	$EE42_H$	--	Function Control for $\overline{CS6}$	0000_H
ADDRSEL6	$EE46_H$	--	Address Window Selection for $\overline{CS6}$	0000_H
TCONCS7	$EE48_H$	--	Timing Control for $\overline{CS7}$	0000_H
FCONCS7	$EE4A_H$	--	Function Control for $\overline{CS7}$	0000_H
ADDRSEL7	$EE\ 4E_H$	--	Address Window Selection for $\overline{CS7}$	0000_H

10 Keyword Index

This section lists a number of keywords which refer to specific details of the C166S V2 in terms of its architecture, its functional units or functions. This helps to quickly find the answer to specific questions about the C166S V2.

A

Address Boundaries 99
Addressing Modes
- CoREG Addressing Mode 63
- IDX Indirect Addressing Mode 56
- Indirect Addressing Mode 53
- Long Adressing Mode 52
- Short Addressing Modes 46

B

Bit Protection 71
Block Diagram ITC / PEC 119
BWT Bit 140

C

Central System Control 14
CGU 14
Clock Generation Unit 14
Context Pointer
- Updating 43

Context Switch 40
Continuous PEC Transfers 140
COUNT Bit 140
CP Register 423, 425
CPUCON1 Register 423, 425
CPUCON2 Register 423, 425
CPUID Register 423, 424
CSP Register 423, 425
Cycle counts 194

D

Data Page Boundaries 99
Data Page Pointer 49
Data Types 68
DMU 12

E

EBC 13
End of PEC Interrupt Sub Node 143
External Bus Controller 13
External Bus Idle State 169
External Interrupt 14

F

Fast Bank Switching 131

G

General Purpose Register 100
GPR 100

I

ID Control 14
IDX0 Register 423, 425
IDX1 Register 423, 425
INC Bit 144
Instructions
- ADD 212
- ADDB 213
- ADDC 214
- ADDCB 215
- AND 216
- ANDB 217
- ASHR 218
- ATOMIC 220
- BAND 221
- BCLR 222
- BCMP 223
- BFLDH 224
- BFLDL 225
- BMOV 226
- BMOVN 227

BOR 228
BSET 229
BXOR 230
CALLA 231
CALLI 233
CALLR 234
CALLS 235
CMP 236
CMPB 237
CMPD1 238
CMPD2 239
CMPI1 240
CMPI2 241
CoABS 316, 317
CoADD 318
CoADD2 319
CoASHR 320, 322
CoCMP 324
CoLOAD 325
CoLOAD- 326
CoLOAD2 327
CoLOAD2- 327, 328
CoMAC 329, 331
CoMAC- 333
CoMACM 335, 337
CoMACM- 339
CoMACMR 341, 343
CoMACMRsu 345, 347
CoMACMRu 348, 350
CoMACMRus 351, 353
CoMACMsu 354, 356
CoMACMsu- 357
CoMACMu 358, 360
CoMACMu- 361
CoMACMus 362, 364
CoMACMus- 365
CoMACR 366, 368
CoMACRsu 370, 372
CoMACRu 373, 374
CoMACRus 375, 377
CoMACsu 378, 379
CoMACsu- 380
CoMACu 381, 382
CoMACu- 383
CoMACus 384, 385
CoMACus- 386
CoMAX 387
CoMIN 388
CoMOV 389
CoMUL 390, 392
CoMUL- 394
CoMULsu 395, 396
CoMULsu- 397
CoMULu 398, 399
CoMULu- 400
CoMULus 401, 402
CoMULus- 403
CoNEG 404, 405
CoNOP 406
CoRND 407
CoSHL 408
CoSHR 410
CoSTORE 412
CoSUB 413
CoSUB2 414
CoSUB2R 415
CoSUBR 416
CPL 242
CPLB 243
DISWDT 244
DIV 245
DIVL 246
DIVLU 247
DIVU 248
EINIT 249
ENWDT 250
EXTP 251
EXTPR 253
EXTR 255
EXTS 256
EXTSR 258
IDLE 260
JB 261
JBC 262
JMPA 264
JMPI 266

JMPR 267
JMPS 268
JNB 269
JNBS 270
MOV 272
MOVB 274
MOVBS 276
MOVBZ 277
MUL 278
MULU 279
NEG 280
NEGB 281
NOP 282
OR 283
ORB 284
PCALL 285
POP 287
PRIOR 288
PUSH 289
PWRDN 290
RET 291
RETI 292
RETP 293
RETS 294
ROL 295
ROR 297
SBRK 299
SCXT 300
SHL 301
SHR 303
SRST 305
SRVWDT 306
SUB 307
SUBB 308
SUBC 309
SUBCB 310
TRAP 311
XOR 313
XORB 314
Interrrupt Control Register 140
Interrupt Jump Table Cache 125
Interrupt System 118

J

JTAG 13

M

MAH Register 423, 425
MAL Register 423, 425
MCW Register 423, 425
MDC Register 423, 425
MDH Register 423, 425
MDL Register 423, 425
Memory
External 98
ROM 93
MRW Register 423, 425
MSW Register 423, 425

N

NMI 117

O

OCDS 13
ONES Register 423, 425

P

PEC 138
Channel Actions 144
Control Register 139
Pointer Address Handling 146
Transfer Count 140
Peripheral Event Controller 138
PMU 12
Power Saving Control 14
Program Memory Unit 12
Protected Bits 71
PSW Register 423, 425

Q

QR0 Register 424
QR1 Register 424
QX0 Register 423, 424
QX1 Register 424

R

Register
- CP 423, 425
- CPUCON1 423, 425
- CPUCON2 423, 425
- CPUID 423, 424
- CSP 423, 425
- DPP0 423, 424
- DPP1 423, 424
- DPP2 423, 424
- DPP3 423, 424
- IDX0 423, 425
- IDX1 423, 425
- MAH 423, 425
- MAL 423, 425
- MCW 423, 425
- MDC 423, 425
- MDH 423, 425
- MDL 423, 425
- MRW 423, 425
- MSW 423, 425
- ONES 423, 425
- PSW 423, 425
- QR0 424
- QR1 424
- QX0 423, 424
- QX1 424
- SP 424, 425
- SPSEG 424, 425
- STKOV 424, 425
- STKUN 424, 425
- TFR 424, 425
- VECSEG 424, 425
- xxIC 140
- ZEROS 424, 425

Reset Control 13

S

SCU 13
Segment Boundaries 99
SFR 96
Sleep mode 14
SP Register 424, 425
SPSEG Register 424, 425
STKOV Register 424, 425
STKUN Register 424, 425
System Control Unit 13

T

TFR Register 424, 425
Traps 135

V

VECSEG Register 424, 425

W

Watchdog Timer 14
WDT 14

Z

ZEROS Register 424, 425

Infineon goes for Business Excellence

“Business excellence means intelligent approaches and clearly defined processes, which are both constantly under review and ultimately lead to good operating results.
Better operating results and business excellence mean less idleness and wastefulness for all of us, more professional success, more accurate information, a better overview and, thereby, less frustration and more satisfaction.”

Dr. Ulrich Schumacher

http://www.infineon.com

Published by Infineon Technologies AG