SIEMENS



Instruction Set Manual

for the C16x Family of
Siemens 16-Bit CMOS Single-Chip Microcontrollers

Instruction Set Manual Version 1.2, 12.97

http://www.siemens.de/Semiconductor/

Version 1.2, 12.97

**Published by Siemens AG,
Bereich Halbleiter, Marketing-
Kommunikation, Balanstraße 73,
81541 München**

© Siemens AG 1997.
All Rights Reserved.

Attention please!

As far as patents or other rights of third parties are concerned, liability is only assumed for components, not for applications, processes and circuits implemented within components or assemblies.

The information describes the type of component and shall not be considered as assured characteristics.

Terms of delivery and rights to change design reserved.

For questions on technology, delivery and prices please contact the Semiconductor Group Offices in Germany or the Siemens Companies and Representatives worldwide (see address list).

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Siemens Office, Semiconductor Group.

Siemens AG is an approved CECC manufacturer.

Packing

Please use the recycling operators known to you. We can also help you – get in touch with your nearest sales office. By agreement we will take packing material back, if it is sorted. You must bear the costs of transport.

For packing material that is returned to us unsorted or which we are not obliged to accept, we shall have to invoice you for any costs incurred.

Components used in life-support devices or systems must be expressly authorized for such purpose!

Critical components[1] of the Semiconductor Group of Siemens AG, may only be used in life-support devices or systems[2] with the express written approval of the Semiconductor Group of Siemens AG.

1 A critical component is a component used in a life-support device or system whose failure can reasonably be expected to cause the failure of that life-support device or system, or to affect its safety or effectiveness of that device or system.

2 Life support devices or systems are intended (a) to be implanted in the human body, or (b) to support and/or maintain and sustain human life. If they fail, it is reasonable to assume that the health of the user may be endangered.

C166 Family Microcontroller Instruction Set Manual Revision History:	Version 1.2, 12.97
Previous Releases:	Version 1.1, 09.95 03.94

Page	Subjects
8	BFLD* code size corrected
35	ADDCB: spelling corrected
38	ASHR: "operation" corrected
43, 44	BFLD*: Note improved, format corrected
51	CALLI: "operation" corrected
67	EINIT: Syntax corrected
75	JBC: Condition flags corrected
77	JMPI: "operation" corrected
81	JNBS: Condition flags corrected
86, 87	MUL(U): Flag N corrected
95	PRIOR: "Operation" corrected
104	SCXT: Data Type added
108	SRVWDT: Syntax corrected

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:

mcdocu.comments@hl.siemens.de

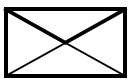


Table of Contents **Page**

1 Introduction 5

2 Short Instruction Summary 6

3 Instruction Set Summary 9

4 Instruction Opcodes 21

5 Instruction Description 26

6 Addressing Modes 116

7 Instruction State Times 123

1 Introduction

The Siemens family of 16-bit microcontrollers offers devices that provide various levels of peripheral performance and programmability. This allows to equip each specific application with the microcontroller that fits best to the required functionality and performance.

Still the Siemens family concept provides an easy path to upgrade existing applications or to climb the next level of performance in order to realize a subsequent more sophisticated design. Two major characteristics enable this upgrade path to save and reuse almost all of the engineering efforts that have been made for previous designs:

• All family members are based on the same basic architecture

• All family members execute the same instructions (except for upgrades for new members)

The fact that all members execute the same instructions (almost) saves knowhow with respect to the understanding of the controller itself and also with respect to the used tools (assembler, disassembler, compiler, etc.).

This instruction set manual provides an easy and direct access to the instructions of the Siemens 16-bit microcontrollers by listing them according to different criteria, and also unloads the technical manuals for the different devices from redundant information.

This manual also describes the different addressing mechanisms and the relation between the logical addresses used in a program and the resulting physical addresses.
There is also information provided to calculate the execution time for specific instructions depending on the used address locations and also specific exceptions to the standard rules.

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

• **The Instruction Description** describes each instruction in full detail

All instructions listed in this manual are executed by the following devices (new derivatives will be added to this list):

C161V, C161K, C161O, C161RI, C161SI, C161CI, C163, C163F, C164CI, C165, C167, C167CR, C167SR, C167S, C167CS.

A few instructions (ATOMIC and EXTended instructions) have been added for these devices and are not recognized by the following devices:

SAB 80C166, SAB 80C166W, SAB 83C166, SAB 83C166W, SAB 88C166, SAB 88C166W.

These differences are noted for each instruction, where applicable.

2 Short Instruction Summary

The following compressed cross-reference tables quickly identify a specific instruction and provide basic information about it. Two ordering schemes are included:

The first table (two pages) is a compressed cross-reference table that quickly identifies a specific hexadecimal opcode with the respective mnemonic.

The second table lists the instructions by their mnemonic and identifies the addressing modes that may be used with a specific instruction and the instruction length depending on the selected addressing mode. This reference helps to optimize instruction sequences in terms of code size and/ or execution time.

•	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

Note: Both ordering schemes (hexadecimal opcode and mnemonic) are provided in more detailled lists in the following sections of this manual.

Note: The ATOMIC and EXTended instructions are not available in the SAB 8XC166(W) devices. They are ***marked*** in the cross-reference table.

	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	-	-	-	-	-	-	-	MOVB
x4	MOV	MOV	MOVB	MOVB	MOV	MOV	MOVB	MOVB
x5	-	-	DISWDT	EINIT	MOVBZ	MOVBS	-	-
x6	CMPI1	CMPI2	CMPD1	CMPD2	SCXT	SCXT	MOV	MOV
x7	IDLE	PWRDN	SRVWDT	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	-	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

Mnemonic	Addressing Modes			Bytes
ADD[B]	Rwn	Rwm	1)	2
ADDC[B]	Rwn	[Rwi]	1)	2
AND[B]	Rwn	[Rwi+]	1)	2
OR[B]	Rwn	#data3	1)	2
SUB[B]				
SUBC[B]	reg	#data16	2)	4
XOR[B]	reg	mem		4
	mem	reg		4
ASHR	Rwn	Rwm		2
ROL / ROR	Rwn	#data4		2
SHL / SHR				
BAND	bitaddrZ.z	bitaddrQ.q		4
BCMP				
BMOV				
BMOVN				
BOR /				
BXOR				
BCLR	bitaddrQ.q			2
BSET				
BFLDH	bitoffQ	#mask8	#data8	4
BFLDL				
MOV[B]	Rwn	Rwm	1)	2
	Rwn	#data4	1)	2
	Rwn	[Rwm]	1)	2
	Rwn	[Rwm+]	1)	2
	[Rwm]	Rwn	1)	2
	[-Rwm]	Rwn	1)	2
	[Rwn]	[Rwm]		2
	[Rwn+]	[Rwm]		2
	[Rwn]	[Rwm+]		2
	reg	#data16	2)	4
	Rwn	[Rwm+#d16]	1)	4
	[Rwm+#d16]	Rwn	1)	4
	[Rwn]	mem		4
	mem	[Rwn]		4
	reg	mem		4
	mem	reg		4
MOVBS	Rwn	Rbm		2
MOVBZ	reg	mem		4
	mem	reg		4
EXTS	Rwm	#irang2	3)	2
EXTSR	#seg	#irang2		4
NOP	-			2
RET				
RETI				
RETS				

Mnemonic	Addressing Modes			Bytes
CPL[B]	Rwn		1)	2
NEG[B]				
DIV	Rwn			2
DIVL				
DIVLU				
DIVU				
MUL	Rwn	Rwm		2
MULU				
CMPD1/2	Rwn	#data4		2
CMPI1/2	Rwn	#data16		4
	Rwn	mem		4
CMP[B]	Rwn	Rwm	1)	2
	Rwn	[Rwi]	1)	2
	Rwn	[Rwi+]	1)	2
	Rwn	#data3	1)	2
	reg	#data16	2)	4
	reg	mem		4
CALLA	cc	caddr		4
JMPA				
CALLI	cc	[Rwn]		2
JMPI				
CALLS	seg	caddr		4
JMPS				
CALLR	rel			2
JMPR	cc	rel		2
JB	bitaddrQ.q	rel		4
JBC				
JNB				
JNBS				
PCALL	reg	caddr		4
POP	reg			2
PUSH				
RETP				
SCXT	reg	#data16		4
	reg	mem		4
PRIOR	Rwn	Rwm		2
TRAP	#trap7			2
ATOMIC	#irang2		3)	2
EXTR				
EXTP	Rwm	#irang2	3)	2
EXTPR	#pag	#irang2		4
SRST/IDLE	-			4
PWRDN				
SRVWDT				
DISWDT				
EINIT				

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.
3) The ATOMIC and EXTended instructions are not available in the SAB 8XC166(W) devices.

3 Instruction Set Summary

This chapter summarizes the instructions by listing them according to their functional class. This allows to identify the right instruction(s) for a specific required function.

The following notes apply to this summary:

Data Addressing Modes

Rw: – Word GPR (R0, R1, … , R15)

Rb: – Byte GPR (RL0, RH0, …, RL7, RH7)

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

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.

Note: The EXTended instructions are not available in the SAB 8XC166(W) devices.

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

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
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
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

*) The EXTended instructions are not available in the SAB 8XC166(W) devices.

4 Instruction Opcodes

The following pages list the instructions of the 16-bit microcontrollers ordered by their hexadecimal opcodes. This helps to identify specific instructions when reading executable code, ie. during the debugging phase.

Notes for Opcode Lists

1) 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.

2) 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		

The ATOMIC and EXTended instructions are not available in the SAB 8XC166(W) devices.

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 BCLR and BSET Instructions

The position of the bit to be set or to be 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 the Undefined Opcodes

A hardware trap occurs when one of the undefined opcodes signified by ‘----’ is decoded by the CPU.

Hex-code	Number of Bytes	Mnemonic	Operands		Hex-code	Number of Bytes	Mnemonic	Operands
00	2	ADD	Rw, Rw		20	2	SUB	Rw, Rw
01	2	ADDB	Rb, Rb		21	2	SUBB	Rb, Rb
02	4	ADD	reg, mem		22	4	SUB	reg, mem
03	4	ADDB	reg, mem		23	4	SUBB	reg, mem
04	4	ADD	mem, reg		24	4	SUB	mem, reg
05	4	ADDB	mem, reg		25	4	SUBB	mem, reg
06	4	ADD	reg, #data16		26	4	SUB	reg, #data16
07	4	ADDB	reg, #data8		27	4	SUBB	reg, #data8
08	2	ADD	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 [1)]		28	2	SUB	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 [1)]
09	2	ADDB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]		29	2	SUBB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]
0A	4	BFLDL	bitoff, #mask8, #data8		2A	4	BCMP	bitaddr, bitaddr
0B	2	MUL	Rw, Rw		2B	2	PRIOR	Rw, Rw
0C	2	ROL	Rw, Rw		2C	2	ROR	Rw, Rw
0D	2	JMPR	cc_UC, rel		2D	2	JMPR	cc_EQ, rel or cc_Z, rel
0E	2	BCLR	bitoff.0		2E	2	BCLR	bitoff.2
0F	2	BSET	bitoff.0		2F	2	BSET	bitoff.2
10	2	ADDC	Rw, Rw		30	2	SUBC	Rw, Rw
11	2	ADDCB	Rb, Rb		31	2	SUBCB	Rb, Rb
12	4	ADDC	reg, mem		32	4	SUBC	reg, mem
13	4	ADDCB	reg, mem		33	4	SUBCB	reg, mem
14	4	ADDC	mem, reg		34	4	SUBC	mem, reg
15	4	ADDCB	mem, reg		35	4	SUBCB	mem, reg
16	4	ADDC	reg, #data16		36	4	SUBC	reg, #data16
17	4	ADDCB	reg, #data8		37	4	SUBCB	reg, #data8
18	2	ADDC	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 [1)]		38	2	SUBC	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 [1)]
19	2	ADDCB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]		39	2	SUBCB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]
1A	4	BFLDH	bitoff, #mask8, #data8		3A	4	BMOVN	bitaddr, bitaddr
1B	2	MULU	Rw, Rw		3B	-	-	-
1C	2	ROL	Rw, #data4		3C	2	ROR	Rw, #data4
1D	2	JMPR	cc_NET, rel		3D	2	JMPR	cc_NE, rel or cc_NZ, rel
1E	2	BCLR	bitoff.1		3E	2	BCLR	bitoff.3
1F	2	BSET	bitoff.1		3F	2	BSET	bitoff.3

Hex-code	Number of Bytes	Mnemonic	Operands		Hex-code	Number of Bytes	Mnemonic	Operands
40	2	CMP	Rw, Rw		60	2	AND	Rw, Rw
41	2	CMPB	Rb, Rb		61	2	ANDB	Rb, Rb
42	4	CMP	reg, mem		62	4	AND	reg, mem
43	4	CMPB	reg, mem		63	4	ANDB	reg, mem
44	-	-	-		64	4	AND	mem, reg
45	-	-	-		65	4	ANDB	mem, reg
46	4	CMP	reg, #data16		66	4	AND	reg, #data16
47	4	CMPB	reg, #data8		67	4	ANDB	reg, #data8
48	2	CMP	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 [1)]		68	2	AND	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 [1)]
49	2	CMPB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]		69	2	ANDB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]
4A	4	BMOV	bitaddr, bitaddr		6A	4	BAND	bitaddr, bitaddr
4B	2	DIV	Rw		6B	2	DIVL	Rw
4C	2	SHL	Rw, Rw		6C	2	SHR	Rw, Rw
4D	2	JMPR	cc_V, rel		6D	2	JMPR	cc_N, rel
4E	2	BCLR	bitoff.4		6E	2	BCLR	bitoff.6
4F	2	BSET	bitoff.4		6F	2	BSET	bitoff.6
50	2	XOR	Rw, Rw		70	2	OR	Rw, Rw
51	2	XORB	Rb, Rb		71	2	ORB	Rb, Rb
52	4	XOR	reg, mem		72	4	OR	reg, mem
53	4	XORB	reg, mem		73	4	ORB	reg, mem
54	4	XOR	mem, reg		74	4	OR	mem, reg
55	4	XORB	mem, reg		75	4	ORB	mem, reg
56	4	XOR	reg, #data16		76	4	OR	reg, #data16
57	4	XORB	reg, #data8		77	4	ORB	reg, #data8
58	2	XOR	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 [1)]		78	2	OR	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 [1)]
59	2	XORB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]		79	2	ORB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]
5A	4	BOR	bitaddr, bitaddr		7A	4	BXOR	bitaddr, bitaddr
5B	2	DIVU	Rw		7B	2	DIVLU	Rw
5C	2	SHL	Rw, #data4		7C	2	SHR	Rw, #data4
5D	2	JMPR	cc_NV, rel		7D	2	JMPR	cc_NN, rel
5E	2	BCLR	bitoff.5		7E	2	BCLR	bitoff.7
5F	2	BSET	bitoff.5		7F	2	BSET	bitoff.7

Hex-code	Number of Bytes	Mnemonic	Operands		Hex-code	Number of Bytes	Mnemonic	Operands
80	2	CMPI1	Rw, #data4		A0	2	CMPD1	Rw, #data4
81	2	NEG	Rw		A1	2	NEGB	Rb
82	4	CMPI1	Rw, mem		A2	4	CMPD1	Rw, mem
83	-	-	-		A3	-	-	-
84	4	MOV	[Rw], mem		A4	4	MOVB	[Rw], mem
85	-	-	-		A5	4	DISWDT	
86	4	CMPI1	Rw, #data16		A6	4	CMPD1	Rw, #data16
87	4	IDLE			A7	4	SRVWDT	
88	2	MOV	[-Rw], Rw		A8	2	MOV	Rw, [Rw]
89	2	MOVB	[-Rw], Rb		A9	2	MOVB	Rb, [Rw]
8A	4	JB	bitaddr, rel		AA	4	JBC	bitaddr, rel
8B	-	-	-		AB	2	CALLI	cc, [Rw]
8C	-	-	-		AC	2	ASHR	Rw, Rw
8D	2	JMPR	cc_C, rel or cc_ULT, rel		AD	2	JMPR	cc_SGT, rel
8E	2	BCLR	bitoff.8		AE	2	BCLR	bitoff.10
8F	2	BSET	bitoff.8		AF	2	BSET	bitoff.10
90	2	CMPI2	Rw, #data4		B0	2	CMPD2	Rw, #data4
91	2	CPL	Rw		B1	2	CPLB	Rb
92	4	CMPI2	Rw, mem		B2	4	CMPD2	Rw, mem
93	-	-	-		B3	-	-	-
94	4	MOV	mem, [Rw]		B4	4	MOVB	mem, [Rw]
95	-	-	-		B5	4	EINIT	
96	4	CMPI2	Rw, #data16		B6	4	CMPD2	Rw, #data16
97	4	PWRDN			B7	4	SRST	
98	2	MOV	Rw, [Rw+]		B8	2	MOV	[Rw], Rw
99	2	MOVB	Rb, [Rw+]		B9	2	MOVB	[Rw], Rb
9A	4	JNB	bitaddr, rel		BA	4	JNBS	bitaddr, rel
9B	2	TRAP	#trap7		BB	2	CALLR	rel
9C	2	JMPI	cc, [Rw]		BC	2	ASHR	Rw, #data4
9D	2	JMPR	cc_NC, rel or cc_UGE, rel		BD	2	JMPR	cc_SLE, rel
9E	2	BCLR	bitoff.9		BE	2	BCLR	bitoff.11
9F	2	BSET	bitoff.9		BF	2	BSET	bitoff.11

Hex-code	Number of Bytes	Mnemonic	Operands		Hex-code	Number of Bytes	Mnemonic	Operands
C0	2	MOVBZ	Rw, Rb		E0	2	MOV	Rw, #data4
C1	-	-	-		E1	2	MOVB	Rb, #data4
C2	4	MOVBZ	reg, mem		E2	4	PCALL	reg, caddr
C3	-	-	-		E3	-	-	-
C4	4	MOV	[Rw+#data16], Rw		E4	4	MOVB	[Rw+#data16], Rb
C5	4	MOVBZ	mem, reg		E5	-	-	-
C6	4	SCXT	reg, #data16		E6	4	MOV	reg, #data16
C7	-	-	-		E7	4	MOVB	reg, #data8
C8	2	MOV	[Rw], [Rw]		E8	2	MOV	[Rw], [Rw+]
C9	2	MOVB	[Rw], [Rw]		E9	2	MOVB	[Rw], [Rw+]
CA	4	CALLA	cc, addr		EA	4	JMPA	cc, caddr
CB	2	RET			EB	2	RETP	reg
CC	2	NOP			EC	2	PUSH	reg
CD	2	JMPR	cc_SLT, rel		ED	2	JMPR	cc_UGT, rel
CE	2	BCLR	bitoff.12		EE	2	BCLR	bitoff.14
CF	2	BSET	bitoff.12		EF	2	BSET	bitoff.14
D0	2	MOVBS	Rw, Rb		F0	2	MOV	Rw, Rw
D1	2	ATOMIC or EXTR	#irang2 [2)]		F1	2	MOVB	Rb, Rb
D2	4	MOVBS	reg, mem		F2	4	MOV	reg, mem
D3	-	-	-		F3	4	MOVB	reg, mem
D4	4	MOV	Rw, [Rw + #data16]		F4	4	MOVB	Rb, [Rw + #data16]
D5	4	MOVBS	mem, reg		F5	-	-	-
D6	4	SCXT	reg, mem		F6	4	MOV	mem, reg
D7	4	EXTP(R), EXTS(R)	#pag10,#irang2 #seg8, #irang2 [2)]		F7	4	MOVB	mem, reg
D8	2	MOV	[Rw+], [Rw]		F8	-	-	-
D9	2	MOVB	[Rw+], [Rw]		F9	-	-	-
DA	4	CALLS	seg, caddr		FA	4	JMPS	seg, caddr
DB	2	RETS			FB	2	RETI	
DC	2	EXTP(R), EXTS(R)	Rw, #irang2 [2)]		FC	2	POP	reg
DD	2	JMPR	cc_SGE, rel		FD	2	JMPR	cc_ULE, rel
DE	2	BCLR	bitoff.13		FE	2	BCLR	bitoff.15
DF	2	BSET	bitoff.13		FF	2	BSET	bitoff.15

5 Instruction Description

This chapter describes each instruction in detail. The instructions are ordered 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 which is performed by the specified 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 addressing modes available are summarized at the end of each single instruction description. In contrast to the syntax for the instructions described in the following, the assembler provides much more flexibility in writing C166 Family programs (e.g. by generic instructions and by automatically selecting appropriate addressing modes whenever possible), and thus it eases the use of the instruction set. For more information about this item please refer to the Assembler manual.

•Operation• This part presents a logical description of the operation performed by an instruction by means of 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)
∧	(opX)	is	logically **AND**ed with (opY)
∨	(opX)	is	logically **OR**ed with (opY)
⊕	(opX)	is	logically **EXCLUSIVELY OR**ed with (opY)
⇔	(opX)	is	**COMPARED** against (opY)
mod	(opX)	is	divided **MODULO** (opY)

Monadic operations:			**operator (opX)**
¬	(opX)	is	logically **COMPLEMENTED**

Missing or existing parentheses signify whether the used operand specifies an immediate constant value, an address or a pointer to an address 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 will also be used in the operational description:

CP	Context Pointer register
CSP	Code Segment Pointer register
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)
PSW	Program Status Word register
SP	System Stack Pointer register
SYSCON	System 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

Except for those instructions which extend byte data to word data, all instructions have only one particular data type. Note that the data types mentioned in this subsection do not consider accesses to indirect address pointers or to the system stack which are always performed with word data. Moreover, no data type is specified for System Control Instructions and for those of the branch instructions which do not access any explicitly addressed data.

•Description• This part provides a brief verbal description of the action that is executed by the respective instruction.

•Condition Code• This notifies that the respective instruction contains a condition code, so it is executed, if the specified condition is true, and is skipped, if it is false. The table below summarizes the 16 possible condition codes that can be used within Call and Branch instructions. The table shows the mnemonic abbreviations, the test that is executed for a specific condition and the internal representation by a 4-bit number.

Condition Code Mnemonic cc	Test	Description	Condition Code Number c
cc_UC	1 = 1	Unconditional	0_H
cc_Z	Z = 1	Zero	2_H
cc_NZ	Z = 0	Not zero	3_H
cc_V	V = 1	Overflow	4_H
cc_NV	V = 0	No overflow	5_H
cc_N	N = 1	Negative	6_H
cc_NN	N = 0	Not negative	7_H
cc_C	C = 1	Carry	8_H
cc_NC	C = 0	No carry	9_H
cc_EQ	Z = 1	Equal	2_H
cc_NE	Z = 0	Not equal	3_H
cc_ULT	C = 1	Unsigned less than	8_H
cc_ULE	(Z∨C) = 1	Unsigned less than or equal	F_H
cc_UGE	C = 0	Unsigned greater than or equal	9_H
cc_UGT	(Z∨C) = 0	Unsigned greater than	E_H
cc_SLT	(N⊕V) = 1	Signed less than	C_H
cc_SLE	(Z∨(N⊕V)) = 1	Signed less than or equal	B_H
cc_SGE	(N⊕V) = 0	Signed greater than or equal	D_H
cc_SGT	(Z∨(N⊕V)) = 0	Signed greater than	A_H
cc_NET	(Z∨E) = 0	Not equal AND not end of table	1_H

•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 occured during operation
- C = 0 : No Carry occured during operation
- V = 1 : Arithmetic Overflow occured during operation
- V = 0 : No Arithmetic Overflow occured 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. Mostly, the selected addressing mode combination is specified by the opcode of the corresponding instruction. However, there are some arithmetic and logical instructions where 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 an example of what operands the respective instruction will accept.

Format This part specifies the format of the instructions as it is represented in the assembler listing. The figure below shows the reference 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 (seg). 8-bit for C165/7, 2-bit (:..ss) for SAB8xC166
:..##	: 2-bit immediate constant (#irang2)
:.###	: 3-bit immediate constant (#data3)
c	: 4-bit condition code specification (cc)
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
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)
RR	: 8-bit word address reg
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)

Number of Bytes Specifies the size of an instruction in bytes. All C166 Family instructions consist of either 2 or 4 bytes. Regarding the instruction size, all instructions can be classified as either single word or double word instructions.

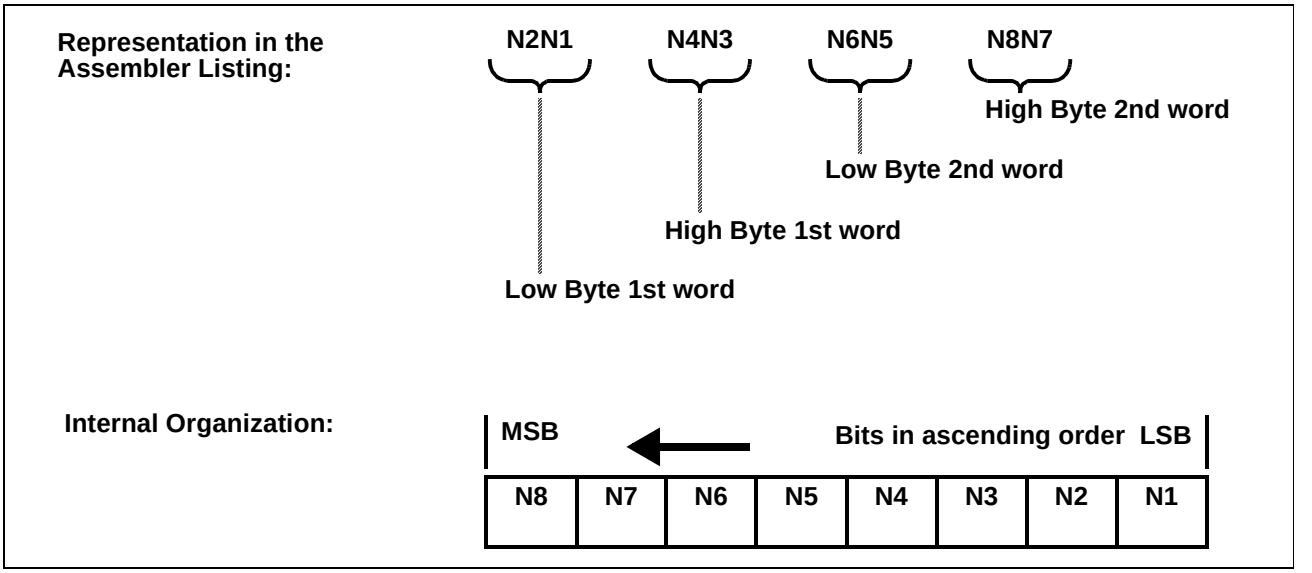


Figure 5-1: Instruction Format Representation

Notes on the ATOMIC and EXTended Instructions

These instructions (ATOMIC, EXTR, EXTP, EXTS, EXTPR, EXTSR) disable standard and PEC interrupts and class A traps during a sequence of the following 1...4 instructions. The length of the sequence is determined by an operand (op1 or op2, depending on the instruction). The EXTended instruction additionally change the addressing mechanism during this sequence (see detailled instruction description).
The ATOMIC and EXTended instructions become active immediately, so no additional NOPs are required. 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 and EXTended instructions.

CAUTION: When a Class B trap interupts an ATOMIC or EXTended sequence, this sequence is terminated, the interrupt lock is removed and the standard condition is restored, before the trap routine is executed! The remaining instructions of the terminated sequence that are executed after returning from the trap routine will run under standard conditions!

CAUTION: Be careful, when using the ATOMIC and EXTended instructions with other system control or branch instructions.

CAUTION: Be careful, when using nested ATOMIC and EXTended instructions. There is ONE counter to control the length of such a sequence, ie. issuing an ATOMIC or EXTended instruction within a sequence will reload the counter with value of the new instruction.

Note: The ATOMIC and EXTended instructions are not available in the SAB 8XC166(W) devices.

The following pages of this section contain a detailled description of each instruction of the C166 Family in alphabetical order.

ADD Integer Addition ADD

Syntax ADD op1, op2

Operation (op1) ← (op1) + (op2)

Data Types WORD

Description Performs a 2's complement binary addition of the source operand specified by op2 and the destination operand specified by op1. The sum is then stored in op1.

Condition 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, ie. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a carry is generated from the most significant bit of the specified data type. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ADD	Rw_n, Rw_m	00 nm	2
ADD	Rw_n, [Rw_i]	08 n:10ii	2
ADD	Rw_n, [Rw_i+]	08 n:11ii	2
ADD	Rw_n, #data3	08 n:0###	2
ADD	reg, #data16	06 RR ## ##	4
ADD	reg, mem	02 RR MM MM	4
ADD	mem, reg	04 RR MM MM	4

ADDB Integer Addition ADDB

Syntax ADDB op1, op2

Operation (op1) ← (op1) + (op2)

Data Types BYTE

Description Performs a 2's complement binary addition of the source operand specified by op2 and the destination operand specified by op1. The sum is then stored in op1.

Condition 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, ie. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a carry is generated from the most significant bit of the specified data type. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ADDB	Rb_n, Rb_m	01 nm	2
ADDB	Rb_n, [Rw_i]	09 n:10ii	2
ADDB	Rb_n, [Rw_i+]	09 n:11ii	2
ADDB	Rb_n, #data3	09 n:0###	2
ADDB	reg, #data16	07 RR ## xx	4
ADDB	reg, mem	03 RR MM MM	4
ADDB	mem, reg	05 RR MM MM	4

ADDC Integer Addition with Carry ADDC

Syntax ADDC op1, op2

Operation (op1) ← (op1) + (op2) + (C)

Data Types WORD

Description Performs a 2's 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.

Condition 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, ie. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a carry is generated from the most significant bit of the specified data type. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ADDC	Rw_n, Rw_m	10 nm	2
ADDC	Rw_n, [Rw_i]	18 n:10ii	2
ADDC	Rw_n, [Rw_i+]	18 n:11ii	2
ADDC	Rw_n, #data3	18 n:0###	2
ADDC	reg, #data16	16 RR ## ##	4
ADDC	reg, mem	12 RR MM MM	4
ADDC	mem, reg	14 RR MM MM	4

ADDCB Integer Addition with Carry ADDCB

Syntax ADDCB op1, op2

Operation (op1) ← (op1) + (op2) + (C)

Data Types BYTE

Description Performs a 2's 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.

Condition 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, ie. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a carry is generated from the most significant bit of the specified data type. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ADDCB	Rb_n, Rb_m	11 nm	2
ADDCB	Rb_n, [Rw_i]	19 n:10ii	2
ADDCB	Rb_n, [Rw_i+]	19 n:11ii	2
ADDCB	Rb_n, #data3	19 n:0###	2
ADDCB	reg, #data16	17 RR ## xx	4
ADDCB	reg, mem	13 RR MM MM	4
ADDCB	mem, reg	15 RR MM MM	4

AND — Logical AND — AND

Syntax AND op1, op2

Operation (op1) ← (op1) ∧ (op2)

Data Types WORD

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.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
AND	Rw_n, Rw_m	60 nm	2
AND	Rw_n, [Rw_i]	68 n:10ii	2
AND	Rw_n, [Rw_i+]	68 n:11ii	2
AND	Rw_n, #data3	68 n:0###	2
AND	reg, #data16	66 RR ## ##	4
AND	reg, mem	62 RR MM MM	4
AND	mem, reg	64 RR MM MM	4

ANDB — Logical AND — ANDB

Syntax ANDB op1, op2

Operation (op1) ← (op1) ∧ (op2)

Data Types BYTE

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.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
ANDB	Rb_n, Rb_m	61 nm	2
ANDB	Rb_n, [Rw_i]	69 n:10ii	2
ANDB	Rb_n, [Rw_i+]	69 n:11ii	2
ANDB	Rb_n, #data3	69 n:0###	2
ANDB	reg, #data16	67 RR ## xx	4
ANDB	reg, mem	63 RR MM MM	4
ANDB	mem, reg	65 RR MM MM	4

ASHR Arithmetic Shift Right ASHR

Syntax ASHR op1, op2

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

Data Types WORD

Description Arithmetically shifts the destination word operand op1 right by as many times as specified in 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 MSB was a 0 or with ones if the original MSB was a 1. The Overflow flag is used as a Rounding flag. The LSB 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.

Condition 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 for a shift count of zero.

C The carry flag is set according to the last LSB 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.

Addressing Modes

Mnemonic		Format	Bytes
ASHR	Rw_n, Rw_m	AC nm	2
ASHR	Rw_n, #data4	BC #n	2

ATOMIC — Begin ATOMIC Sequence — ATOMIC

Syntax

ATOMIC op1

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 such that no additional NOPs are required. Depending on the value of op1, the period of validity of the ATOMIC sequence extends over the sequence of the next 1 to 4 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.

Note

The ATOMIC instruction must be used carefully (see introductory note). The ATOMIC instruction is not available in the SAB 8XC166(W) devices.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
ATOMIC	#irang2	D1 :00##-0	2

BAND — Bit Logical AND — BAND

Syntax BAND op1, op2

Operation (op1) ← (op1) ∧ (op2)

Data Types BIT

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.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
BAND	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	6A QQ ZZ qz	4

BCLR Bit Clear BCLR

Syntax BCLR op1

Operation (op1) ← 0

Data Types BIT

Description CLears the bit specified by op1. This instruction is primarily used for peripheral and system control.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
BCLR	$bitaddr_{Q.q}$	qE QQ	2

BCMP Bit to Bit Compare BCMP

Syntax BCMP op1, op2

Operation (op1) ⇔ (op2)

Data Types BIT

Description Performs a single bit comparison of the source bit specified by operand op1 to the source bit specified by operand op2. No result is written by this instruction. Only the condition codes are updated.

Note: The meaning of the condition flags for the BCMP instruction is different from the meaning of the flags for the other compare instructions.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
BCMP	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	2A QQ ZZ qz	4

BFLDH — Bit Field High Byte — BFLDH

Syntax BFLDH op1, op2, op3

Operation

(tmp) ← (op1)

(high byte (tmp)) ← ((high byte (tmp) ∧ ¬op2) ∨ op3)

(op1) ← (tmp)

Data Types WORD

Description Replaces those bits in the high byte of the destination word operand op1 which are selected by a '1' in the AND mask op2 with the bits at the corresponding positions in the OR mask specified by op3.

Note: op1 bits which shall remain unchanged must have a '0' in the respective bit of both the AND mask op2 and the OR mask op3.
Otherwise a '1' in op3 will set the corresponding op1 bit (see „Operation").

Condition Flags

E	Z	V	C	N
0	*	0	0	*

E Always cleared.

Z Set if the word result equals zero. Cleared otherwise.

V Always cleared.

C Always cleared.

N Set if the most significant bit of the word result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
BFLDH	$bitoff_Q$, #$mask_8$, #data8	1A QQ ## @@	4

BFLDL — Bit Field Low Byte — BFLDL

Syntax BFLDL op1, op2, op3

Operation

(tmp) ← (op1)

(low byte (tmp)) ← ((low byte (tmp) ∧ ¬op2) ∨ op3)

(op1) ← (tmp)

Data Types WORD

Description Replaces those bits in the low byte of the destination word operand op1 which are selected by a '1' in the AND mask op2 with the bits at the corresponding positions in the OR mask specified by op3.

Note: op1 bits which shall remain unchanged must have a '0' in the respective bit of both the AND mask op2 and the OR mask op3.
Otherwise a '1' in op3 will set the corresponding op1 bit (see „Operation").

Condition Flags

E	Z	V	C	N
0	*	0	0	*

E Always cleared.

Z Set if the word result equals zero. Cleared otherwise.

V Always cleared.

C Always cleared.

N Set if the most significant bit of the word result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
BFLDL	$bitoff_Q$, #$mask_8$, #data8	0A QQ @@ ##	4

BMOV — Bit to Bit Move — BMOV

Syntax BMOV op1, op2

Operation (op1) ← (op2)

Data Types BIT

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.

Condition 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 source bit.

V Always cleared.

C Always cleared.

N Contains the previous state of the source bit.

Addressing Modes

Mnemonic		Format	Bytes
BMOV	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	4A QQ ZZ qz	4

BMOVN Bit to Bit Move and Negate BMOVN

Syntax BMOVN op1, op2

Operation (op1) ← ¬(op2)

Data Types BIT

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.

Condition 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 source bit.

V Always cleared.

C Always cleared.

N Contains the previous state of the source bit.

Addressing Modes

Mnemonic		Format	Bytes
BMOVN	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	3A QQ ZZ qz	4

BOR — Bit Logical OR — BOR

Syntax BOR op1, op2

Operation (op1) ← (op1) ∨ (op2)

Data Types BIT

Description Performs a single bit logical OR of the source bit specified by operand op2 with the destination bit specified by operand op1. The ORed result is then stored in op1.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
BOR	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	5A QQ ZZ qz	4

BSET Bit Set BSET

Syntax BSET op1

Operation (op1) ← 1

Data Types BIT

Description Sets the bit specified by op1. This instruction is primarily used for peripheral and system control.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
BSET	$bitaddr_{Q.q}$	qF QQ	2

BXOR — Bit Logical XOR — BXOR

Syntax BXOR op1, op2

Operation (op1) ← (op1) ⊕ (op2)

Data Types BIT

Description Performs a single bit logical EXCLUSIVE OR of the source bit specified by operand op2 with the destination bit specified by operand op1. The XORed result is then stored in op1.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
BXOR	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	7A QQ ZZ qz	4

CALLA Call Subroutine Absolute CALLA

Syntax CALLA op1, op2

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 onto 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 of the calling routine. If the condition is not met, no action is taken and the next instruction is executed normally.

Condition Codes See condition code table.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
CALLA	cc, caddr	CA c0 MM MM	4

CALLI — Call Subroutine Indirect — CALLI

Syntax CALLI op1, op2

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 on 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.

Condition Codes See condition code table.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
CALLI	cc, [Rw_n]	AB cn	2

CALLR Call Subroutine Relative CALLR

Syntax CALLR op1

Operation

(SP) ← (SP) - 2

((SP)) ← (IP)

(IP) ← (IP) + 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 onto 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 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.

Condition Codes See condition code table.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
CALLR	rel	BB rr	2

CALLS Call Inter-Segment Subroutine CALLS

Syntax CALLS op1, op2

Operation

(SP) ← (SP) - 2
((SP)) ← (CSP)
(SP) ← (SP) - 2
((SP)) ← (IP)
(CSP) ← op1
(IP) ← op1

Description A branch is taken to the absolute location specified by op2 within the segment specified by op1. 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 on the system stack represents the return address to the calling routine. The previous value of the CSP is also placed on the system stack to insure correct return to the calling segment.

Condition Codes See condition code table.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
CALLS	seg, caddr	DA SS MM MM	4

CMP Integer Compare CMP

Syntax CMP op1, op2

Operation (op1) ⇔ (op2)

Data Types WORD

Description The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. The flags are set according to the rules of subtraction. The operands remain unchanged.

Condition 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, ie. the result cannot be represented in the specified 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.

Addressing Modes

Mnemonic		Format	Bytes
CMP	Rw_n, Rw_m	40 nm	2
CMP	Rw_n, [Rw_i]	48 n:10ii	2
CMP	Rw_n, [Rw_i+]	48 n:11ii	2
CMP	Rw_n, #data3	48 n:0###	2
CMP	reg, #data16	46 RR ## ##	4
CMP	reg, mem	42 RR MM MM	4

CMPB Integer Compare CMPB

Syntax CMPB op1, op2

Operation (op1) ⇔ (op2)

Data Types BYTE

Description The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. The flags are set according to the rules of subtraction. The operands remain unchanged.

Condition 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, ie. the result cannot be represented in the specified 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.

Addressing Modes

Mnemonic		Format	Bytes
CMPB	Rb_n, Rb_m	41 nm	2
CMPB	Rb_n, [Rw_i]	49 n:10ii	2
CMPB	Rb_n, [Rw_i+]	49 n:11ii	2
CMPB	Rb_n, #data3	49 n:0###	2
CMPB	reg, #data16	47 RR ## xx	4
CMPB	reg, mem	43 RR MM MM	4

CMPD1 Integer Compare and Decrement by 1 CMPD1

Syntax CMPD1 op1, op2

Operation (op1) ⇔ (op2)
(op1) ← (op1) - 1

Data Types WORD

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 2's 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.

Condition 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, ie. the result cannot be represented in the specified 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.

Addressing Modes

Mnemonic		Format	Bytes
CMPD1	Rw_n, #data4	A0 #n	2
CMPD1	Rw_n, #data16	A6 Fn ## ##	4
CMPD1	Rw_n, mem	A2 Fn MM MM	4

CMPD2 Integer Compare and Decrement by 2 CMPD2

Syntax CMPD2 op1, op2

Operation

(op1) ⇔ (op2)

(op1) ← (op1) - 2

Data Types WORD

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 2's 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.

Condition 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, ie. the result cannot be represented in the specified 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.

Addressing Modes

Mnemonic		Format	Bytes
CMPD2	Rw_n, #data4	B0 #n	2
CMPD2	Rw_n, #data16	B6 Fn ## ##	4
CMPD2	Rw_n, mem	B2 Fn MM MM	4

CMPI1 Integer Compare and Increment by 1 CMPI1

Syntax CMPI1 op1, op2

Operation
(op1) ⇔ (op2)
(op1) ← (op1) + 1

Data Types WORD

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 2's 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.

Condition 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, ie. the result cannot be represented in the specified 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.

Addressing Modes

Mnemonic		Format	Bytes
CMPI1	Rw_n, #data4	80 #n	2
CMPI1	Rw_n, #data16	86 Fn ## ##	4
CMPI1	Rw_n, mem	82 Fn MM MM	4

CMPI2 Integer Compare and Increment by 2 CMPI2

Syntax CMPI2 op1, op2

Operation
(op1) ⇔ (op2)
(op1) ← (op1) + 2

Data Types WORD

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 2's 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.

Condition 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, ie. the result cannot be represented in the specified 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.

Addressing Modes

Mnemonic		Format	Bytes
CMPI2	Rw_n, #data4	90 #n	2
CMPI2	Rw_n, #data16	96 Fn ## ##	4
CMPI2	Rw_n, mem	92 Fn MM MM	4

CPL Integer One's Complement CPL

Syntax CPL op1

Operation (op1) ← ¬(op1)

Data Types WORD

Description Performs a 1's complement of the source operand specified by op1. The result is stored back into op1.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
CPL	Rw_n	91 n0	2

CPLB Integer One's Complement CPLB

Syntax CPL op1

Operation (op1) ← ¬(op1)

Data Types BYTE

Description Performs a 1's complement of the source operand specified by op1. The result is stored back into op1.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
CPLB	Rb_n	B1 n0	2

DISWDT Disable Watchdog Timer DISWDT

Syntax DISWDT

Operation Disable the watchdog timer

Description This instruction disables the watchdog timer. The watchdog timer is enabled by a reset. The DISWDT instruction allows the watchdog timer to be disabled for applications which do not require a watchdog function. Following a reset, this instruction can be executed at any time until either a Service Watchdog Timer instruction (SRVWDT) or an End of Initialization instruction (EINIT) are executed. Once one of these instructions has been executed, the DISWDT instruction will have no effect. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
DISWDT	A5 5A A5 A5	4

DIV 16-by-16 Signed Division DIV

Syntax DIV op1

Operation
(MDL) ← (MDL) / (op1)
(MDH) ← (MDL) mod (op1)

Data Types WORD

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).

Condition Flags

E	Z	V	C	N
0	*	S	0	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V Set if an arithmetic overflow occurred, ie. the result 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 result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
DIV	Rw_n	4B nn	2

DIVL 32-by-16 Signed Division DIVL

Syntax DIVL op1

Operation
(MDL) ← (MD) / (op1)
(MDH) ← (MD) mod (op1)

Data Types WORD, DOUBLEWORD

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).

Condition Flags

E	Z	V	C	N
0	*	S	0	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V Set if an arithmetic overflow occurred, ie. the result 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 result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
DIVL	Rw_n	6B nn	2

DIVLU 32-by-16 Unsigned Division DIVLU

Syntax DIVLU op1

Operation
(MDL) ← (MD) / (op1)
(MDH) ← (MD) mod (op1)

Data Types WORD, DOUBLEWORD

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).

Condition Flags

E	Z	V	C	N
0	*	S	0	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V Set if an arithmetic overflow occurred, ie. the result 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 result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
DIVLU	Rw_n	7B nn	2

DIVU 16-by-16 Unsigned Division DIVU

Syntax DIVU op1

Operation
(MDL) ← (MDL) / (op1)
(MDH) ← (MDL) mod (op1)

Data Types WORD

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 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).

Condition Flags

E	Z	V	C	N
0	*	S	0	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V Set if an arithmetic overflow occurred, ie. the result 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 result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
DIVU	Rw_n	5B nn	2

EINIT End of Initialization EINIT

Syntax EINIT

Operation End of Initialization

Description This instruction is used to signal the end of the initialization portion of a program. After a reset, the reset output pin $\overline{\text{RSTOUT}}$ is pulled low. It remains low until the EINIT instruction has been executed at which time it goes high. This enables the program to signal the external circuitry that it has successfully initialized the microcontroller. After the EINIT instruction has been executed, execution of the Disable Watchdog Timer instruction (DISWDT) has no effect. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
EINIT	B5 4A B5 B5	4

EXTR Begin EXTended Register Sequence EXTR

Syntax EXTR op1

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 effected instruction sequence.

Note

The EXTR instruction must be used carefully (see introductory note).
The EXTR instruction is not available in the SAB 8XC166(W) devices.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
EXTR	#irang2	D1 :10##-0	2

EXTP — Begin EXTended Page Sequence — EXTP

Syntax

EXTP op1, op2

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 immediately active 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 effected instruction sequence.

Note

The EXTP instruction must be used carefully (see introductory note).
The EXTP instruction is not available in the SAB 8XC166(W) devices.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
EXTP	Rwm, #irang2	DC :01##-m	2
EXTP	#pag, #irang2	D7 :01##-0 pp 0:00pp	4

EXTPR Begin EXTended Page and Register Sequence EXTPR

Syntax EXTPR op1, op2

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Page = (op1) AND 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) AND 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 effected instruction sequence.

Note

The EXTPR instruction must be used carefully (see introductory note).
The EXTPR instruction is not available in the SAB 8XC166(W) devices.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
EXTPR	Rwm, #irang2	DC :11##-m	2
EXTPR	#pag, #irang2	D7 :11##-0 pp 0:00pp	4

EXTS Begin EXTended Segment Sequence EXTS

Syntax

EXTS op1, op2

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 effected instruction sequence.

Note

The EXTS instruction must be used carefully (see introductory note).
The EXTS instruction is not available in the SAB 8XC166(W) devices.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
EXTS	Rwm, #irang2	DC :00##-m	2
EXTS	#seg, #irang2	D7 :00##-0 ss 00	4

EXTSR Begin EXTended Segment and Register Sequence EXTSR

Syntax EXTSR op1, op2

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Segment = (op1) AND 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) AND 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 effected instruction sequence.

Note

The EXTSR instruction must be used carefully (see introductory note).
The EXTSR instruction is not available in the SAB 8XC166(W) devices.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
EXTSR	Rwm, #irang2	DC :10##-m	2
EXTSR	#seg, #irang2	D7 :10##-0 ss 00	4

IDLE — Enter Idle Mode — IDLE

Syntax IDLE

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 insure that this instruction is not accidentally executed, it is implemented as a protected instruction.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
IDLE	87 78 87 87	4

JB — Relative Jump if Bit Set — JB

Syntax JB op1, op2

Operation

IF (op1) = 1 THEN
(IP) ← (IP) + sign_extend (op2)
ELSE
Next Instruction
END IF

Data Types BIT

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 two's 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 clear, the instruction following the JB instruction is executed.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
JB	$bitaddr_{Q.q}$, rel	8A QQ rr q0	4

JBC Relative Jump if Bit Set and Clear Bit JBC

Syntax JBC op1, op2

Operation

IF (op1) = 1 THEN
(op1) = 0
(IP) ← (IP) + sign_extend (op2)
ELSE
Next Instruction
END IF

Data Types BIT

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 two's 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, the instruction following the JBC instruction is executed.

Condition Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

E Always cleared.

Z Contains 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.

Addressing Modes

Mnemonic		Format	Bytes
JBC	$bitaddr_{Q.q}$, rel	AA QQ rr q0	4

JMPA Absolute Conditional Jump JMPA

Syntax JMPA op1, op2

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.

Condition Codes See condition code table.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
JMPA	cc, caddr	EA c0 MM MM	4

JMPI — Indirect Conditional Jump — JMPI

Syntax JMPI op1, op2

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 JMPI instruction is executed normally.

Condition Codes See condition code table.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
JMPI	cc, [Rw_n]	9C cn	2

JMPR — Relative Conditional Jump — JMPR

Syntax

JMPR op1, op2

Operation

IF (op1) = 1 THEN
(IP) ← (IP) + sign_extend (op2)
ELSE
Next Instruction
END IF

Description

If the 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 two's 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.

Condition Codes

See condition code table.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
JMPR	cc, rel	cD rr	2

JMPS Absolute Inter-Segment Jump JMPS

Syntax JMPS op1, op2

Operation
(CSP) ← op1
(IP) ← op2

Description Branches unconditionally to the absolute address specified by op2 within the segment specified by op1.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
JMPS	seg, caddr	FA SS MM MM	4

JNB Relative Jump if Bit Clear JNB

Syntax JNB op1, op2

Operation

IF (op1) = 0 THEN
(IP) ← (IP) + sign_extend (op2)
ELSE
Next Instruction
END IF

Data Types BIT

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 two's 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, the instruction following the JNB instruction is executed.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
JNB	$bitaddr_{Q.q}$, rel	9A QQ rr q0	4

JNBS Relative Jump if Bit Clear and Set Bit JNBS

Syntax JNBS op1, op2

Operation

IF (op1) = 0 THEN
(op1) = 1
(IP) ← (IP) + sign_extend (op2)
ELSE
Next Instruction
END IF

Data Types BIT

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 two's 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, the instruction following the JNBS instruction is executed.

Condition Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

E Always cleared.

Z Contains 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.

Addressing Modes

Mnemonic		Format	Bytes
JNBS	$bitaddr_{Q.q}$, rel	BA QQ rr q0	4

MOV — Move Data — MOV

Syntax MOV op1, op2

Operation (op1) ← (op2)

Data Types WORD

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 is examined, and the condition codes are updated accordingly.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
MOV	Rw_n, Rw_m	F0 nm	2
MOV	Rw_n, #data4	E0 #n	2
MOV	reg, #data16	E6 RR ## ##	4
MOV	Rw_n, [Rw_m]	A8 nm	2
MOV	Rw_n, [Rw_m+]	98 nm	2
MOV	[Rw_m], Rw_n	B8 nm	2
MOV	[-Rw_m], Rw_n	88 nm	2
MOV	[Rw_n], [Rw_m]	C8 nm	2
MOV	[Rw_n+], [Rw_m]	D8 nm	2
MOV	[Rw_n], [Rw_m+]	E8 nm	2
MOV	Rw_n, [Rw_m+#data16]	D4 nm ## ##	4
MOV	[Rw_m+#data16], Rw_n	C4 nm ## ##	4
MOV	[Rw_n], mem	84 0n MM MM	4
MOV	mem, [Rw_n]	94 0n MM MM	4
MOV	reg, mem	F2 RR MM MM	4
MOV	mem, reg	F6 RR MM MM	4

MOVB Move Data MOVB

Syntax MOVB op1, op2

Operation (op1) ← (op2)

Data Types BYTE

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 is examined, and the condition codes are updated accordingly.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
MOVB	Rb_n, Rb_m	F1 nm	2
MOVB	Rb_n, #data4	E1 #n	2
MOVB	reg, #data8	E7 RR ## xx	4
MOVB	Rb_n, [Rw_m]	A9 nm	2
MOVB	Rb_n, [Rw_m+]	99 nm	2
MOVB	[Rw_m], Rb_n	B9 nm	2
MOVB	[-Rw_m], Rb_n	89 nm	2
MOVB	[Rw_n], [Rw_m]	C9 nm	2
MOVB	[Rw_n+], [Rw_m]	D9 nm	2
MOVB	[Rw_n], [Rw_m+]	E9 nm	2
MOVB	Rb_n, [Rw_m+#data16]	F4 nm ## ##	4
MOVB	[Rw_m+#data16], Rb_n	E4 nm ## ##	4
MOVB	[Rw_n], mem	A4 0n MM MM	4
MOVB	mem, [Rw_n]	B4 0n MM MM	4
MOVB	reg, mem	F3 RR MM MM	4
MOVB	mem, reg	F7 RR MM MM	4

MOVBS Move Byte Sign Extend MOVBS

Syntax MOVBS op1, op2

Operation

(low byte op1) ← (op2)
IF ($op2_7$) = 1 THEN
(high byte op1) ← FF_H
ELSE
(high byte op1) ← 00_H
END IF

Data Types WORD, BYTE

Description Moves and sign extends the contents of the source byte specified by op2 to the word location specified by the destination operand op1. The contents of the moved data is examined, and the condition codes are updated accordingly.

Condition Flags

E	Z	V	C	N
0	*	-	-	*

E Always cleared.

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.

Addressing Modes

Mnemonic		Format	Bytes
MOVBS	Rw_n, Rb_m	D0 mn	2
MOVBS	reg, mem	D2 RR MM MM	4
MOVBS	mem, reg	D5 RR MM MM	4

MOVBZ — Move Byte Zero Extend — MOVBZ

Syntax MOVBZ op1, op2

Operation
(low byte op1) ← (op2)
(high byte op1) ← 00_H

Data Types WORD, BYTE

Description Moves and zero extends the contents of the source byte specified by op2 to the word location specified by the destination operand op1. The contents of the moved data is examined, and the condition codes are updated accordingly.

Condition Flags

E	Z	V	C	N
0	*	-	-	0

E Always cleared.

Z Set if the value of the source operand op2 equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Always cleared.

Addressing Modes

Mnemonic		Format	Bytes
MOVBZ	Rw_n, Rb_m	C0 mn	2
MOVBZ	reg, mem	C2 RR MM MM	4
MOVBZ	mem, reg	C5 RR MM MM	4

MUL — Signed Multiplication — MUL

Syntax MUL op1, op2

Operation (MD) ← (op1) * (op2)

Data Types WORD

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.

Condition Flags

E	Z	V	C	N
0	*	S	0	*

E Always cleared.

Z Set if the 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.

Addressing Modes

Mnemonic		Format	Bytes
MUL	Rw_n, Rw_m	0B nm	2

MULU — Unsigned Multiplication — MULU

Syntax MULU op1, op2

Operation (MD) ← (op1) * (op2)

Data Types WORD

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.

Condition Flags

E	Z	V	C	N
0	*	S	0	*

E Always cleared.

Z Set if the 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.

Addressing Modes

Mnemonic		Format	Bytes
MULU	Rw_n, Rw_m	1B nm	2

NEG Integer Two's Complement NEG

Syntax NEG op1

Operation (op1) ← 0 - (op1)

Data Types WORD

Description Performs a binary 2's complement of the source operand specified by op1. The result is then stored in op1.

Condition Flags

E	Z	V	C	N
*	*	*	S	*

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, ie. the result cannot be represented in the specified 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.

Addressing Modes

Mnemonic		Format	Bytes
NEG	Rw_n	81 n0	2

NEGB Integer Two's Complement NEGB

Syntax NEGB op1

Operation (op1) ← 0 - (op1)

Data Types BYTE

Description Performs a binary 2's complement of the source operand specified by op1. The result is then stored in op1.

Condition Flags

E	Z	V	C	N
*	*	*	S	*

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, ie. the result cannot be represented in the specified 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.

Addressing Modes

Mnemonic		Format	Bytes
NEGB	Rb_n	A1 n0	2

NOP — No Operation — NOP

Syntax NOP

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.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
NOP	CC 00	2

OR — Logical OR — OR

Syntax OR op1, op2

Operation (op1) ← (op1) ∨ (op2)

Data Types WORD

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.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
OR	Rw_n, Rw_m	70 nm	2
OR	Rw_n, [Rw_i]	78 n:10ii	2
OR	Rw_n, [Rw_i+]	78 n:11ii	2
OR	Rw_n, #data3	78 n:0###	2
OR	reg, #data16	76 RR ## ##	4
OR	reg, mem	72 RR MM MM	4
OR	mem, reg	74 RR MM MM	4

ORB — Logical OR — ORB

Syntax ORB op1, op2

Operation (op1) ← (op1) ∨ (op2)

Data Types BYTE

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.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
ORB	Rb_n, Rb_m	71 nm	2
ORB	Rb_n, [Rw_i]	79 n:10ii	2
ORB	Rb_n, [Rw_i+]	79 n:11ii	2
ORB	Rb_n, #data3	79 n:0###	2
ORB	reg, #data16	77 RR ## xx	4
ORB	reg, mem	73 RR MM MM	4
ORB	mem, reg	75 RR MM MM	4

PCALL Push Word and Call Subroutine Absolute PCALL

Syntax PCALL op1, op2

Operation

(tmp) ← (op1)
(SP) ← (SP) - 2
((SP)) ← (tmp)
(SP) ← (SP) - 2
((SP)) ← (IP)
(IP) ← op2

Data Types WORD

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.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
PCALL	reg, caddr	E2 RR MM MM	4

POP — Pop Word from System Stack — POP

Syntax

POP op1

Operation

(tmp) ← ((SP))
(SP) ← (SP) + 2
(op1) ← (tmp)

Data Types

WORD

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.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
POP	reg	FC RR	2

PRIOR — Prioritize Register — PRIOR

Syntax PRIOR op1, op2

Operation

(tmp) ← (op2)
(count) ← 0
DO WHILE (tmp_{15}) ≠ 1 AND (count) ≠ 15 AND (op2) ≠ 0
(tmp_n) ← (tmp_{n-1})
(count) ← (count) + 1
END WHILE
(op1) ← (count)

Data Types WORD

Description This instruction stores a count value in the word operand specified by op1 indicating the number of single bit shifts required to normalize the operand op2 so that its MSB 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.

Condition Flags

E	Z	V	C	N
0	*	0	0	0

E Always cleared.

Z Set if the source operand op2 equals zero. Cleared otherwise.

V Always cleared.

C Always cleared.

N Always cleared.

Addressing Modes

Mnemonic		Format	Bytes
PRIOR	Rw_n, Rw_m	2B nm	2

PUSH — Push Word on System Stack — PUSH

Syntax

PUSH op1

Operation

(tmp) ← (op1)

(SP) ← (SP) - 2

((SP)) ← (tmp)

Data Types

WORD

Description

Moves the word specified by operand op1 to the location in the internal system stack specified by the Stack Pointer, after the Stack Pointer has been decremented by two.

Condition Flags

E	Z	V	C	N
*	*	-	-	*

E Set if the value of the pushed word represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.

Z Set if the value of the pushed word equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Set if the most significant bit of the pushed word is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
PUSH	reg	EC RR	2

PWRDN — Enter Power Down Mode — PWRDN

Syntax PWRDN

Operation Enter Power Down Mode

Description This instruction causes the part to enter the power down mode. In this mode, all peripherals and the CPU are powered down until the part is externally reset. To insure 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 ($\overline{\text{NMI}}$) is in the low state. Otherwise, this instruction has no effect.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
PWRDN	97 68 97 97	4

RET — Return from Subroutine — RET

Syntax RET

Operation

(IP) ← ((SP))

(SP) ← (SP) + 2

Description

Returns from a subroutine. The IP is popped from the system stack. Execution resumes at the instruction following the CALL instruction in the calling routine.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
RET	CB 00	2

RETI — Return from Interrupt Routine — RETI

Syntax

RETI

Operation

(IP) ← ((SP))
(SP) ← (SP) + 2
IF (SYSCON.SGTDIS=0) THEN
(CSP) ← ((SP))
(SP) ← (SP) + 2
END IF
(PSW) ← ((SP))
(SP) ← (SP) + 2

Description

Returns from an interrupt routine. The PSW, IP, and CSP are popped off the system stack. Execution resumes at the instruction which had been interrupted. The previous system state is restored after the PSW has been popped. The CSP is only popped if segmentation is enabled. This is indicated by the SGTDIS bit in the SYSCON register.

Condition Flags

E	Z	V	C	N
S	S	S	S	S

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.

Addressing Modes

Mnemonic	Format	Bytes
RETI	FB 88	2

RETP Return from Subroutine and Pop Word RETP

Syntax RETP op1

Operation

(IP) ← ((SP))
(SP) ← (SP) + 2
(tmp) ← ((SP))
(SP) ← (SP) + 2
(op1) ← (tmp)

Data Types WORD

Description Returns from a subroutine. The IP is first popped from the system stack and then the next word is popped from the system stack into the operand specified by op1. Execution resumes at the instruction following the CALL instruction in the calling routine.

Condition Flags

E	Z	V	C	N
*	*	-	-	*

E Set if the value of the word popped into 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 word popped into operand op1 equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Set if the most significant bit of the word popped into operand op1 is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
RETP	reg	EB RR	2

RETS Return from Inter-Segment Subroutine RETS

Syntax

RETS

Operation

(IP) ← ((SP))
(SP) ← (SP) + 2
(CSP) ← ((SP))
(SP) ← (SP) + 2

Description

Returns from an inter-segment subroutine. The IP and CSP are popped from the system stack. Execution resumes at the instruction following the CALLS instruction in the calling routine.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
RETS	DB 00	2

ROL — Rotate Left — ROL

Syntax ROL op1, op2

Operation

(count) ← (op2)
(C) ← 0
DO WHILE (count) ≠ 0
 (C) ← ($op1_{15}$)
 ($op1_n$) ← ($op1_{n-1}$) [n=1...15]
 ($op1_0$) ← (C)
 (count) ← (count) - 1
END WHILE

Data Types WORD

Description Rotates the destination word operand op1 left by as many 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 4 bits are used.

Condition 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 MSB shifted out of op1. Cleared for a rotate count of zero.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ROL	Rw_n, Rw_m	0C nm	2
ROL	Rw_n, #data4	1C #n	2

ROR — Rotate Right — ROR

Syntax ROR op1, op2

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

Data Types WORD

Description Rotates the destination word operand op1 right by as many 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 4 bits are used.

Condition 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 LSB shifted out of op1. Cleared for a rotate count of zero.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ROR	Rw_n, Rw_m	2C nm	2
ROR	Rw_n, #data4	3C #n	2

SCXT Switch Context SCXT

Syntax SCXT op1, op2

Operation

(tmp1) ← (op1)
(tmp2) ← (op2)
(SP) ← (SP) - 2
((SP)) ← (tmp1)
(op1) ← (tmp2)

Data Types WORD

Description Used to switch contexts for 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.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
SCXT	reg, #data16	C6 RR ## ##	4
SCXT	reg, mem	D6 RR MM MM	4

SHL — Shift Left — SHL

Syntax SHL op1, op2

Operation

(count) ← (op2)
(C) ← 0
DO WHILE (count) ≠ 0
 (C) ← ($op1_{15}$)
 ($op1_n$) ← ($op1_{n-1}$) [n=1...15]
 ($op1_0$) ← 0
 (count) ← (count) - 1
END WHILE

Data Types WORD

Description Shifts the destination word operand op1 left by as many times as specified by the source operand op2. The least significant bits of the result are filled with zeros accordingly. The MSB 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.

Condition 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 MSB 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.

Addressing Modes

Mnemonic		Format	Bytes
SHL	Rw_n, Rw_m	4C nm	2
SHL	Rw_n, #data4	5C #n	2

SHR — Shift Right — SHR

Syntax SHR op1, op2

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

Data Types WORD

Description

Shifts the destination word operand op1 right by as many 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. This flag together with the Carry flag helps the user to determine whether the remainder bits lost were greater than, less than or equal to one half an LSB. 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.

Condition 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 for a shift count of zero.

C The carry flag is set according to the last LSB 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.

Addressing Modes

Mnemonic		Format	Bytes
SHR	Rw_n, Rw_m	6C nm	2
SHR	Rw_n, #data4	7C #n	2

SRST — Software Reset — SRST

Syntax SRST

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 insure that this instruction is not accidentally executed, it is implemented as a protected instruction.

Condition 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.

Addressing Modes

Mnemonic	Format	Bytes
SRST	B7 48 B7 B7	4

SRVWDT Service Watchdog Timer SRVWDT

Syntax SRVWDT

Operation Service Watchdog Timer

Description This instruction services the Watchdog Timer. It reloads the high order byte of the Watchdog Timer with a preset value and clears the low byte on every occurrence. Once this instruction has been executed, the watchdog timer cannot be disabled. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
SRVWDT	A7 58 A7 A7	4

SUB Integer Subtraction SUB

Syntax SUB op1, op2

Operation (op1) ← (op1) - (op2)

Data Types WORD

Description Performs a 2's complement binary subtraction of the source operand specified by op2 from the destination operand specified by op1. The result is then stored in op1.

Condition 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, ie. the result cannot be represented in the specified 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.

Addressing Modes

Mnemonic		Format	Bytes
SUB	Rw_n, Rw_m	20 nm	2
SUB	Rw_n, [Rw_i]	28 n:10ii	2
SUB	Rw_n, [Rw_i+]	28 n:11ii	2
SUB	Rw_n, #data3	28 n:0###	2
SUB	reg, #data16	26 RR ## ##	4
SUB	reg, mem	22 RR MM MM	4
SUB	mem, reg	24 RR MM MM	4

SUBB Integer Subtraction SUBB

Syntax SUBB op1, op2

Operation (op1) ← (op1) - (op2)

Data Types BYTE

Description Performs a 2's complement binary subtraction of the source operand specified by op2 from the destination operand specified by op1. The result is then stored in op1.

Condition 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, ie. the result cannot be represented in the specified 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.

Addressing Modes

Mnemonic		Format	Bytes
SUBB	Rb_n, Rb_m	21 nm	2
SUBB	Rb_n, [Rw_i]	29 n:10ii	2
SUBB	Rb_n, [Rw_i+]	29 n:11ii	2
SUBB	Rb_n, #data3	29 n:0###	2
SUBB	reg, #data16	27 RR ## xx	4
SUBB	reg, mem	23 RR MM MM	4
SUBB	mem, reg	25 RR MM MM	4

SUBC Integer Subtraction with Carry SUBC

Syntax SUBC op1, op2

Operation (op1) ← (op1) - (op2) - (C)

Data Types WORD

Description Performs a 2's 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.

Condition 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, ie. the result cannot be represented in the specified 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.

Addressing Modes

Mnemonic		Format	Bytes
SUBC	Rw_n, Rw_m	30 nm	2
SUBC	Rw_n, [Rw_i]	38 n:10ii	2
SUBC	Rw_n, [Rw_i+]	38 n:11ii	2
SUBC	Rw_n, #data3	38 n:0###	2
SUBC	reg, #data16	36 RR ## ##	4
SUBC	reg, mem	32 RR MM MM	4
SUBC	mem, reg	34 RR MM MM	4

SUBCB Integer Subtraction with Carry SUBCB

Syntax SUBCB op1, op2

Operation (op1) ← (op1) - (op2) - (C)

Data Types BYTE

Description Performs a 2's 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.

Condition 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, ie. the result cannot be represented in the specified 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.

Addressing Modes

Mnemonic		Format	Bytes
SUBCB	Rb_n, Rb_m	31 nm	2
SUBCB	Rb_n, [Rw_i]	39 n:10ii	2
SUBCB	Rb_n, [Rw_i+]	39 n:11ii	2
SUBCB	Rb_n, #data3	39 n:0###	2
SUBCB	reg, #data16	37 RR ## xx	4
SUBCB	reg, mem	33 RR MM MM	4
SUBCB	mem, reg	35 RR MM MM	4

TRAP Software Trap TRAP

Syntax TRAP op1

Operation

(SP) ← (SP) - 2
((SP)) ← (PSW)
IF (SYSCON.SGTDIS=0) THEN
(SP) ← (SP) - 2
((SP)) ← (CSP)
(CSP) ← 0
END IF
(SP) ← (SP) - 2
((SP)) ← (IP)
(IP) ← zero_extend (op1*4)

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 trap or interrupt routine has completed. The CSP is pushed if segmentation is enabled. This is indicated by the SGTDIS bit in the SYSCON register.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
TRAP	#trap7	9B t:ttt0	2

XOR — Logical Exclusive OR — XOR

Syntax XOR op1, op2

Operation (op1) ← (op1) ⊕ (op2)

Data Types WORD

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.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
XOR	Rw_n, Rw_m	50 nm	2
XOR	Rw_n, [Rw_i]	58 n:10ii	2
XOR	Rw_n, [Rw_i+]	58 n:11ii	2
XOR	Rw_n, #data3	58 n:0###	2
XOR	reg, #data16	56 RR ## ##	4
XOR	reg, mem	52 RR MM MM	4
XOR	mem, reg	54 RR MM MM	4

XORB Logical Exclusive OR XORB

Syntax XORB op1, op2

Operation (op1) ← (op1) ⊕ (op2)

Data Types BYTE

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.

Condition 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.

Addressing Modes

Mnemonic		Format	Bytes
XORB	Rb_n, Rb_m	51 nm	2
XORB	Rb_n, [Rw_i]	59 n:10ii	2
XORB	Rb_n, [Rw_i+]	59 n:11ii	2
XORB	Rb_n, #data3	59 n:0###	2
XORB	reg, #data16	57 RR ## xx	4
XORB	reg, mem	53 RR MM MM	4
XORB	mem, reg	55 RR MM MM	4

6 Addressing Modes

The Siemens 16-bit microcontrollers provide a lot of powerful addressing modes for access to word, byte and bit data (short, long, indirect), or to specify the target address of a branch instruction (absolute, relative, indirect). The different addressing modes use different formats and cover different scopes.

Short Addressing Modes

All of these addressing modes use an implicit base offset address to specify an 18-bit or 24-bit physical address (SAB 80C166 group or C167/5 group, respectively).
Short addressing modes allow to access 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.

Mnemonic	Physical Address	Short Address Range	Scope of Access
Rw	(CP) + 2*Rw	Rw = 0...15	GPRs (Word)
Rb	(CP) + 1*Rb	Rb = 0...15	GPRs (Byte)
reg	$00'FE00_H$ + 2*reg $00'F000_H$ + 2*reg *) (CP) + 2*(reg$\wedge 0F_H$) (CP) + 1*(reg$\wedge 0F_H$)	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$\wedge FF_H$) (CP) + 2*(bitoff$\wedge 0F_H$)	bitoff = 00_H...$7F_H$ bitoff = 80_H...EF_H bitoff = $F0_H$...FF_H	RAM Bit word offset SFR 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

*) The Extended Special Function Register (ESFR) area is not available in the SAB 8XC166(W) devices.

Rw, Rb: Specifies direct access to any GPR in the currently active context (register bank). Both 'Rw' and 'Rb' require four bits in the instruction format. The base address of the current register bank is determined by the content 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).

reg: Specifies direct access to any (E)SFR or GPR in the currently active context (register bank). 'reg' requires eight bits in the instruction format. Short 'reg' addresses from 00_H to EF_H always specify (E)SFRs. In that case, the factor 'Δ' equates 2 and the base address is 00'$FE00_H$ for the standard SFR area or 00'$F000_H$ for the extended ESFR area. 'reg' accesses to the ESFR area require a preceding EXT*R instruction to switch the base address (not available in the SAB 8XC166(W) devices). Depending on the opcode of an instruction, 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 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 thus it can be regarded as being 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. 'bitoff' 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 from 00_H to $7F_H$ use 00'$FD00_H$ as a base address, and thus they specify the 128 highest internal RAM word locations (00'$FD00_H$h to 00'$FDFE_H$). Short 'bitoff' addresses from 80_H to EF_H use 00'$FF00_H$ as a base address to specify the highest internal SFR word locations (00'$FF00_H$ to 00'$FFDE_H$) or use 00'$F100_H$ as a base address to specify the highest internal ESFR word locations (00'$F100_H$ to 00'$F1DE_H$). 'bitoff' accesses to the ESFR area require a preceding EXT*R instruction to switch the base address (not available in the SAB 8XC166(W) devices). For short 'bitoff' addresses from $F0_H$ to FF_H, only the lowest four bits and the contents of the CP register are used to generate the physical 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. Thus, 'bitaddr' requires twelve bits in the instruction format.

Long Addressing Mode

This addressing mode uses one of the four DPP registers to specify a physical 18-bit or 24-bit address. Any word or byte data within the entire address space can be accessed with this mode. The C167/5 devices also support an override mechanism for the DPP adressing scheme.

Note: Word accesses on odd byte addresses are not executed, but rather trigger a hardware trap. After reset, the DPP registers are initialized in a way that all long addresses are directly mapped onto the identical physical addresses.

Any long 16-bit address consists of two portions, which are interpreted in different ways. Bits 13...0 specify a 14-bit data page offset, while bits 15...14 specify the Data Page Pointer (1 of 4), which is to be used to generate the physical 18-bit or 24-bit address (see figure below).

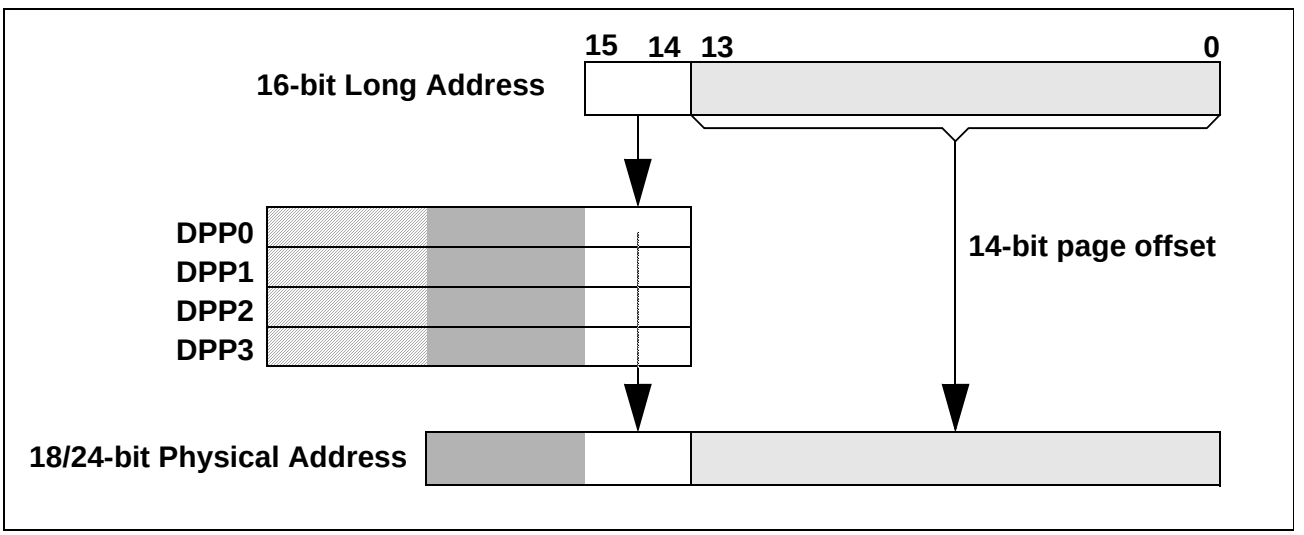


Figure 6-1: Interpretation of a 16-bit Long Address

The SAB 8XC166(W) devices support an address space of up to 256 KByte, while the C167/5 devices support an address space of up to 16 MByte, so only the lower two or ten bits (respectively) of the selected DPP register content are concatenated with the 14-bit data page offset to build the physical address.

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

Mnemonic	Physical Address	Long Address Range	Scope of Access
mem	(DPP0) || mem∧$3FFF_H$ (DPP1) || mem∧$3FFF_H$ (DPP2) || mem∧$3FFF_H$ (DPP3) || mem∧$3FFF_H$	0000_H...$3FFF_H$ 4000_H...$7FFF_H$ 8000_H...$BFFF_H$ $C000_H$...$FFFF_H$	Any Word or Byte
mem	pag || mem∧$3FFF_H$	0000_H...$FFFF_H$ (14-bit)	Any Word or Byte
mem	seg || mem	0000_H...$FFFF_H$ (16-bit)	Any Word or Byte

DPP Override Mechansim in the C167/5

Other than the older devices from the SAB 80C166 group the C167 and C165 devices provide an override mechanism that allows to bypass the DPP addressing scheme temporarily.

The EXTP(R) and EXTS(R) instructions override this addressing mechanism. Instruction EXTP(R) replaces the content of the respective DPP register, while instruction EXTS(R) concatenates th 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).

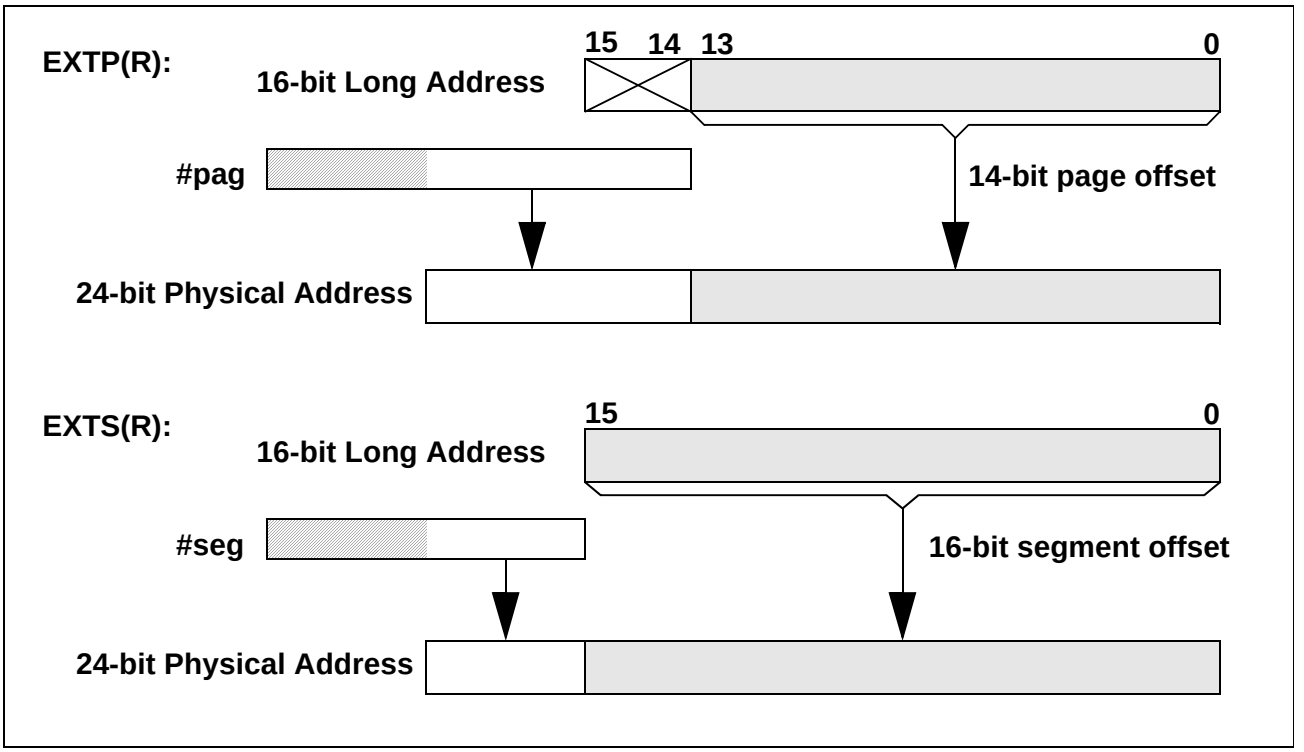


Figure 6-2: Overriding the DPP Mechanism

Indirect Addressing Modes

These addressing modes can be regarded as a combination of short and long addressing modes. This means that long 16-bit addresses are specified 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 allow decrementing or incrementing the indirect address pointers (GPR content) by 2 or 1 (referring to words or bytes).

In each case, one of the four DPP registers is used to specify physical 18-bit or 24-bit addresses. Any word or byte data within the entire memory space can be addressed indirectly.

Note: The exceptions for instructions EXTP(R) and EXTS(R), ie. overriding the DPP mechanism, apply in the same way as described for the long addressing modes.

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.

Note: Word accesses on odd byte addresses are not executed, but rather trigger a hardware trap. After reset, the DPP registers are initialized in a way that all indirect long addresses are directly mapped onto the identical physical addresses.

Physical addresses are generated from indirect address pointers via 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 register bank base address (CP).

GPR Address = (CP) + 2 * Short Address

2) Pre-decremented indirect address pointers ('-Rw') are decremented by a data-type-dependent value (Δ=1 for byte operations, Δ=2 for word operations), before the long 16-bit address is generated:

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

3) Calculate the long 16-bit address by adding a constant value (if selected) to the content of the indirect address pointer:

Long Address = (GPR Pointer) + Constant

4) Calculate the physical 18-bit or 24-bit address using the resulting long address and the corresponding DPP register content (see long 'mem' addressing modes).

Physical Address = (DPPi) + Page offset

5) Post-Incremented indirect address pointers ('Rw+') are incremented by a data-type-dependent value (Δ=1 for byte operations, Δ=2 for word operations):

(GPR Pointer) = (GPR Pointer) + Δ ; [optional step!]

The following indirect addressing modes are provided:

Mnemonic	Particularities
[Rw]	Most instructions accept any GPR (R15...R0) as indirect address pointer. Some instructions, however, only accept 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.

Constants

The C166 Family instruction set also supports the use of wordwide or bytewide immediate constants. For an optimum utilization of the available code storage, these constants are represented in the instruction formats by either 3, 4, 8 or 16 bits. Thus, short constants are always zero-extended while long constants are truncated if necessary to match the data format required for the particular operation (see table below):

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 number sign '#'.

Instruction Range (#irang2)

The effect of the ATOMIC and EXTended instructions can be defined for the following 1...4 instructions. This instruction range (1...4) is coded in the 2-bit constant #irang2 and is represented by the values 0...3.

Branch Target Addressing Modes

Different addressing modes are provided to specify the target address and segment of jump or call instructions. Relative, absolute and indirect modes can be used to update the Instruction Pointer register (IP), while the Code Segment Pointer register (CSP) can only be updated with an absolute value. A special mode is provided to address the interrupt and trap jump vector table, which resides in the lowest portion of code segment 0.

Mnemonic	Target Address	Target Segment	Valid Address Range
caddr	(IP) = caddr	-	caddr = 0000_H...$FFFE_H$
rel	(IP) = (IP) + 2*rel (IP) = (IP) + 2*($\overline{rel}$+1)	- -	rel = 00_H...$7F_H$ rel = 80_H...FF_H
[Rw]	(IP) = ((CP) + 2*Rw)	-	Rw = 0...15
seg	-	(CSP) = seg	seg = 0...255(3)
#trap7	(IP) = 0000_H + 4*trap7	(CSP) = 0000_H	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' must always contain a '0', otherwise a hardware trap would occur.

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, either forward ('rel'= 00_H to $7F_H$) or 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 content 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 the address pointer GPR must always contain a '0', otherwise a hardware trap would occur.

seg: Specifies an absolute code segment number. The devices of the SAB 80C166 group support 4 different code segments, while the devices of the C167/5 group support 256 different code segments, so only the two or eight lower bits (respectively) of the 'seg' operand value are used for updating 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, which allow to access any double word code location within the address range $00'0000_H$...$00'01FC_H$ in code segment 0 (ie. the interrupt jump vector table). For the association of trap numbers with the corresponding interrupt or trap sources please refer to chapter “Interrupt and Trap Functions”.

7 Instruction State Times

Basically, the time to execute an instruction depends on where the instruction is fetched from, and where possible operands are read from or written to. The fastest processing mode is to execute a program fetched from the internal ROM. In that case most of the instructions can be processed within just one machine cycle, which is also the general minimum execution time.

All external memory accesses are performed by the on-chip External Bus Controller (EBC), which works in parallel with the CPU. Mostly, instructions from external memory cannot be processed as fast as instructions from the internal ROM, because some data transfers, which internally can be performed in parallel, have to be performed sequentially via the external interface. In contrast to internal ROM program execution, the time required to process an external program additionally depends on the length of the instructions and operands, on the selected bus mode, and on the duration of an external memory cycle, which is partly selectable by the user.

Processing a program from the internal RAM space is not as fast as execution from the internal ROM area, but it offers a lot of flexibility (ie. for loading temporary programs into the internal RAM via the chip's serial interface, or end-of-line programming via the bootstrap loader).

The following description allows evaluating the minimum and maximum program execution times. This will be sufficient for most requirements. For an exact determination of the instructions' state times it is recommended to use the facilities provided by simulators or emulators.

This section defines the subsequently used time units, summarizes the minimum (standard) state times of the 16-bit microcontroller instructions, and describes the exceptions from that standard timing.

Time Unit Definitions

The following time units are used to describe the instructions' processing times:

[f_{CPU}]: CPU operating frequency (may vary from 1 MHz to 20 MHz).

[State]: One state time is specified by one CPU clock period. Henceforth, one State is used as the basic time unit, because it represents the shortest period of time which has to be considered for instruction timing evaluations.

1 [State] = $1/f_{CPU}$ [s] ; for f_{CPU} = variable
= 50 [ns] ; for f_{CPU} = 20 MHz

[ACT]: This ALE (Address Latch Enable) Cycle Time specifies the time required to perform one external memory access. One ALE Cycle Time consists of either two (for demultiplexed external bus modes) or three (for multiplexed external bus modes) state times plus a number of state times, which is determined by the number of waitstates programmed in the MCTC (Memory Cycle Time Control) and MTTC (Memory Tristate Time Control) bit fields of the SYSCON/BUSCONx registers.

In case of demultiplexed external bus modes:
1*ACT = (2 + (15 – MCTC) + (1 – MTTC)) * States
= 100 ns ... 900 ns ; for f_{CPU} = 20 MHz

In case of multiplexed external bus modes:
1*ACT = 3 + (15 – MCTC) + (1 – MTTC) * States
= 150 ns ... 950 ns ; for f_{CPU} = 20 MHz

The total time (T_{tot}), which a particular part of a program takes to be processed, can be calculated by the sum of the single instruction processing times (T_{In}) of the considered instructions plus an offset value of 6 state times which considers the solitary filling of the pipeline, as follows:

$T_{tot} = T_{I1} + T_{I2} + ... + T_{In} + 6 * States$

The time T_{In}, which a single instruction takes to be processed, consists of a minimum number (T_{Imin}) plus an additional number (T_{Iadd}) of instruction state times and/or ALE Cycle Times, as follows:

$T_{In} = T_{Imin} + T_{Iadd}$

Minimum State Times

The table below shows the minimum number of state times required to process an instruction fetched from the internal ROM (T_{Imin} (ROM)). The minimum number of state times for instructions fetched from the internal RAM (T_{Imin} (RAM)), or of ALE Cycle Times for instructions fetched from the external memory (T_{Imin} (ext)), can also be easily calculated by means of this table.

Most of the 16-bit microcontroller instructions - except some of the branches, the multiplication, the division and a special move instruction - require a minimum of two state times. In case of internal ROM program execution there is no execution time dependency on the instruction length except for some special branch situations. The injected target instruction of a cache jump instruction can be considered for timing evaluations as if being executed from the internal ROM, regardless of which memory area the rest of the current program is really fetched from.

For some of the branch instructions the table below represents both the standard number of state times (ie. the corresponding branch is taken) and an additional T_{Imin} value in parentheses, which refers to the case that either the branch condition is not met or a cache jump is taken.

Minimum Instruction State Times [Unit = ns]

Instruction	**T_{Imin} (ROM) [States]**		**T_{Imin} (ROM) (@ 20 MHz CPU clock)**	
CALLI, CALLA	4	(+2)	200	(+100)
CALLS, CALLR, PCALL	4		200	
JB, JBC, JNB, JNBS	4	(+2)	200	(+100)
JMPS	4		200	
JMPA, JMPI, JMPR	4	(+2)	200	(+100)
MUL, MULU	10		500	
DIV, DIVL, DIVU, DIVLU	20		1000	
MOV[B] Rn, [Rm+#data16]	4		200	
RET, RETI, RETP, RETS	4		200	
TRAP	4		200	
All other instructions	2		100	

Instructions executed from the internal RAM require the same minimum time as if being fetched from the internal ROM plus an instruction-length dependent number of state times, as follows:

For 2-byte instructions: $T_{Imin}(RAM) = T_{Imin}(ROM) + 4 * States$

For 4-byte instructions: $T_{Imin}(RAM) = T_{Imin}(ROM) + 6 * States$

In contrast to the internal ROM program execution, the minimum time $T_{Imin}(ext)$ to process an external instruction additionally depends on the instruction length. $T_{Imin}(ext)$ is either 1 ALE Cycle Time for most of the 2-byte instructions, or 2 ALE Cycle Times for most of the 4-byte instructions. The following formula represents the minimum execution time of instructions fetched from an external memory via a 16-bit wide data bus:

For 2-byte instructions: $T_{Imin}(ext) = 1*ACT + (T_{Imin}(ROM) - 2) * States$
For 4-byte instructions: $T_{Imin}(ext) = 2*ACTs + (T_{Imin}(ROM) - 2) * States$

Note: For instructions fetched from an external memory via an 8-bit wide data bus, the minimum number of required ALE Cycle Times is twice the number for a 16-bit wide bus.

Additional State Times

Some operand accesses can extend the execution time of an instruction T_{In}. Since the additional time T_{Iadd} is mostly caused by internal instruction pipelining, it often will be possible to evade these timing effects in time-critical program modules by means of a suitable rearrangement of the corresponding instruction sequences. Simulators and emulators offer a lot of facilities, which support the user in optimizing his program whenever required.

• Internal ROM operand reads: $T_{Iadd} = 2 * States$

Both byte and word operand reads always require 2 additional state times.

• Internal RAM operand reads via indirect addressing modes: $T_{Iadd} = 0$ or $1 * State$

Reading a GPR or any other directly addressed operand within the internal RAM space does NOT cause additional state times. However, reading an indirectly addressed internal RAM operand will extend the processing time by 1 state time, if the preceding instruction auto-increments or auto-decrements a GPR as shown in the following example:

```
I_n     : MOV R1 , [R0+]     ; auto-increment R0
I_n+1   : MOV [R3], [R2]     ; if R2 points into the internal RAM space:
                             ; T_Iadd = 1 * State
```

In this case, the additional time can simply be avoided by putting another suitable instruction before the instruction I_{n+1} indirectly reading the internal RAM.

• **Internal SFR operand reads:** T_{Iadd} = 0, 1 * State or 2 * States

Mostly, SFR read accesses do NOT require additional processing time. In some rare cases, however, either one or two additional state times will be caused by particular SFR operations, as follows:

– Reading an SFR immediately after an instruction, which writes to the internal SFR space, as shown in the following example:

```
In      : MOV   T0, #1000h      ; write to Timer 0
In+1    : ADD   R3, T1          ; read from Timer 1: TIadd = 1 * State
```

– Reading the PSW register immediately after an instruction, which implicitly updates the condition flags, as shown in the following example:

```
In      : ADD   R0, #1000h      ; implicit modification of PSW flags
In+1    : BAND   C, Z           ; read from PSW: TIadd = 2 * States
```

– Implicitly incrementing or decrementing the SP register immediately after an instruction, which explicitly writes to the SP register, as shown in the following example:

```
In      : MOV   SP, #0FB00h     ; explicit update of the stack pointer
In+1    : SCXT  R1, #1000h      ; implicit decrement of the stack pointer:
                                : TIadd = 2 * States
```

In these cases, the extra state times can be avoided by putting other suitable instructions before the instruction I_{n+1} reading the SFR.

• **External operand reads:** T_{Iadd} = 1 * ACT

Any external operand reading via a 16-bit wide data bus requires one additional ALE Cycle Time. Reading word operands via an 8-bit wide data bus takes twice as much time (2 ALE Cycle Times) as the reading of byte operands.

• **External operand writes:** T_{Iadd} = 0 * State ... 1 * ACT

Writing an external operand via a 16-bit wide data bus takes one additional ALE Cycle Time. For timing calculations of external program parts, this extra time must always be considered. The value of T_{Iadd} which must be considered for timing evaluations of internal program parts, may fluctuate between 0 state times and 1 ALE Cycle Time. This is because external writes are normally performed in parallel to other CPU operations. Thus, T_{Iadd} could already have been considered in the standard processing time of another instruction. Writing a word operand via an 8-bit wide data bus requires twice as much time (2 ALE Cycle Times) as the writing of a byte operand.

• Jumps into the internal ROM space: T_{Iadd} = 0 or 2 * States

The minimum time of 4 state times for standard jumps into the internal ROM space will be extended by 2 additional state times, if the branch target instruction is a double word instruction at a non-aligned double word location ($xxx2_H$, $xxx6_H$, $xxxA_H$, $xxxE_H$), as shown in the following example:

```
label     : ....                ; any non-aligned double word instruction
                                : (eg. at location 0FFE_H)
....      : ....
I_n+1     : JMPA cc–UC, label   ; if a standard branch is taken:
                                : T_Iadd = 2 * States (T_In = 6 * States)
```

A cache jump, which normally requires just 2 state times, will be extended by 2 additional state times, if both the cached jump target instruction and its successor instruction are non-aligned double word instructions, as shown in the following example:

```
label     : ....                ; any non-aligned double word instruction
                                : (eg. at location 12FA_H)
I_t+1     : ....                ; any non-aligned double word instruction
                                : (eg. at location 12FE_H)
I_n+1     :JMPR cc–UC, label    ; provided that a cache jump is taken:
                                : T_Iadd = 2 * States (T_In = 4 * States)
```

If required, these extra state times can be avoided by allocating double word jump target instructions to aligned double word addresses ($xxx0_H$, $xxx4_H$, $xxx8_H$, $xxxC_H$).

• Testing Branch Conditions: T_{Iadd} = 0 or 1 * States

Mostly, NO extra time is required for conditional branch instructions to decide whether a branch condition is met or not. However, an additional state time is required, if the preceding instruction writes to the PSW register, as shown in the following example:

```
I_n       : BSET  USR0          ; write to PSW
I_n+1     :JMPR  cc–Z, label    ; test condition flag in PSW: T_Iadd = 1 * State
```

In this case, the extra state time can simply be intercepted by putting another suitable instruction before the conditional branch instruction.