User's Manual, V2.0, Mar. 2001

Instruction Set Manual

for the C166 Family of Infineon 16-Bit Single-Chip Microcontrollers

Microcontrollers

Infineon technologies

Never stop thinking.

Edition 2001-03

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

User's Manual, V2.0, Mar. 2001

Instruction Set Manual

for the C166 Family of Infineon 16-Bit Single-Chip Microcontrollers

Microcontrollers

Never stop thinking.

C166 Family Microcontroller Instruction Set Manual
Revision History: V2.0, 2001-03

Previous Version:	Version 1.2, 12.97 Version 1.1, 09-95 03.94

Page	Subjects (major changes since last revision)
all	Converted to new company layout
4 … 30	Overview- and summary-tables reformatted
2	List of derivatives updated
31	Description template added
34	PSW image added
38	Condition code table moved
40	Note for MUL/DIV added
42ff	Immediate data for byte instructions corrected to #data8
52f	Note improved
62	Description of operation corrected
72ff	Description of division instructions improved
85	Format description corrected
86	Description improved
101f	Description of multiplication instructions improved
128	Description of flags corrected
132	“bitoff” for ESFRs added
137	Section moved
139	Target address for “rel” corrected
141	General description improved
142ff	Timing examples converted to 25 MHz
143	Branch execution times corrected
148f	Keyword index introduced

Table of Contents

Page

1 Introduction

The Infineon C166 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 Infineon 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 basically the same instructions saves know-how 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 Infineon 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):

- C161K, C161O, C161PI
- C161CS, C161JC, C161JI
- C163
- C164CI, C164SI, C164CM, C164SM
- C165
- C167CR, C167SR
- C167CS

A few instructions (ATOMIC and EXTended instructions) have been added for these devices and are not recognized by the following devices from the first generation of 16-bit microcontrollers:

- SAB 80C166, SAB 80C166W
- SAB 83C166, SAB 83C166W

These differences are noted for each instruction, where applicable.

2 Overviews

The following compressed cross-reference tables quickly identify a specific instruction and provide basic information about it.

Two ordering schemes are included:

- **The hexadecimal opcode** of a specific instruction can be quickly identified with the respective mnemonic using the first compressed cross-reference table.
- **The mnemonics and addressing modes** of the various instructions are listed in the second table. The table shows which addressing modes may be used with a specific instruction and also 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.

Both ordering schemes (hexadecimal opcode and mnemonic) are provided in more detailed 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.*

Table 1 Instruction Overview ordered by Hex-Code (lower half)

	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

Table 2 Instruction Overview ordered by Hex-Code (upper half)

	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	CMPI1	CMPI2	CMPD1	CMPD2	MOVBZ	MOVBS	MOV	MOV
x1	NEG	CPL	NEGB	CPLB	–	***ATOMIC 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[R] EXTS[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[R] EXTS[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

Table 3 Instruction Overview ordered by Mnemonic

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

Mnemo-nic(s)	Addressing Modes		Bytes
CPL[B] NEG[B]	Rwn (Rbn)[1]		2
DIV DIVL DIVLU DIVU	Rwn		2
MUL MULU	Rwn,	Rwm	2
CMPD1 CMPD2 CMPI1 CMPI2	Rwn, Rwn, Rwn,	#data4 #data16 mem	2 4 4
CMP CMPB[1]	Rwn, Rwn, Rwn, Rwn, reg, reg,	Rwm [Rwi] [Rwi+] #data3 #data16[2] mem	2 2 2 2 4 4
CALLA JMPA	cc,	caddr	4
CALLI JMPI	cc,	[Rwn]	2
EXTP EXTPR	Rwm, #pag,	#irang2 [3] #irang2	2 4
SRST IDLE PWRDN SRVWDT DISWDT EINIT	–		4

Table 3 Instruction Overview ordered by Mnemonic (cont'd)

Mnemo-nic(s)	Addressing Modes		Bytes	Mnemo-nic(s)	Addressing Modes		Bytes
MOV MOVB[1]	Rwn,	Rwm	2	CALLS JMPS	seg,	caddr	4
	Rwn,	#data4	2				
	Rwn,	[Rwm]	2	CALLR	rel		2
	Rwn,	[Rwm+]	2	JMPR	cc,	rel	2
	[Rwm],	Rwn	2	JB JBC JNB JNBS	bitaddrQ.q,	rel	4
	[-Rwm],	Rwn	2				
	[Rwn],	[Rwm]	2				
	[Rwn+],	[Rwm]	2				
	[Rwn],	[Rwm+]	2				
	reg,	#data16[2]	4	PCALL	reg,	caddr	4
	Rwn,	[Rwm+#d16]	4	POP PUSH RETP	reg		2
	[Rwm+#d16],	Rwn	4				
	[Rwn],	mem	4				
	mem,	[Rwn]	4	SCXT	reg,	#data16	4
	reg,	mem	4		reg,	mem	4
	mem,	reg	4	PRIOR	Rwn,	Rwm	2
MOVBS MOVBZ	Rwn,	Rbm	2	TRAP	#trap7		2
	reg,	mem	4	ATOMIC EXTR	#irang2	[3]	2
	mem,	reg	4				

[1] Byte oriented instructions (suffix 'B') use byte registers (Rb instead of Rw), except for indirect addressing modes ([Rw] or [Rw+]).

[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 Summary

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

The following general explanations apply to this summary:

3.1 Data Addressing Modes

Rw:	Word GPR (R0, R1, …, R15)
Rb:	Byte GPR (RL0, RH0, …, RL7, RH7)
reg:	SFR/ESFR 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)
baddr:	Direct bit in the bit-addressable memory area
bitoff:	Direct word in the bit-addressable memory area
#datax:	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

3.2 Branch Target Addressing Modes

caddr:	Direct 16-bit jump target address (updates the Instruction Pointer)
Rb:	Byte GPR (RL0, RH0, …, RL7, RH7)
seg:	Direct 8-bit segment address[1)] (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

[1)] In the 8XC166(W) devices the segment is only a 2-bit number, due to the smaller address range.

3.3 Multiply and Divide Operations

The MDL and MDH registers are implicit source and/or destination operands of the multiply and divide instructions.

3.4 Extension Operations

#pag10:	Immediate 10-bit page address
#seg8:	Immediate 8-bit segment address
#irang2:	Immediate 2-bit instruction range

The extension instructions EXTP, EXTPR, EXTS, and EXTSR override the standard DPP addressing scheme, using immediate addresses instead.

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

3.5 Branch Condition Codes

cc:	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

*Note: Condition codes can be specified symbolically within an instruction.
A detailed description of the condition codes can be found in* ***Table 5****.*

Table 4 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

Table 4 Instruction Set Summary (cont'd)

Mnemonic	Description	Bytes
Arithmetic Operations (cont'd)		
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
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

Table 4 Instruction Set Summary (cont'd)

Mnemonic	Description	Bytes
Arithmetic Operations (cont'd)		
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
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-bit $\times$ 16-bit)	2
MULU Rw, Rw	Unsigned multiply direct GPR by direct GPR (16-bit $\times$ 16-bit)	2
DIV Rw	Signed divide register MDL by direct GPR (16-bit $\div$ 16-bit)	2
DIVL Rw	Signed long divide register MD by direct GPR (32-bit $\div$ 16-bit)	2
DIVLU Rw	Unsigned long divide register MD by direct GPR (32-bit $\div$ 16-bit)	2
DIVU Rw	Unsigned divide register MDL by direct GPR (16-bit $\div$ 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

Table 4 Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
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
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

Table 4 Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Logical Instructions (cont'd)			
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
Prioritize Instruction			
PRIOR	Rw, Rw	Determine number of shift cycles to normalize direct word GPR and store result in direct word GPR	2

Table 4 Instruction Set Summary (cont'd)

Mnemonic	Description	Bytes
Boolean Bit Manipulation Operations		
BCLR baddr	Clear direct bit	2
BSET baddr	Set direct bit	2
BMOV baddr, baddr	Move direct bit to direct bit	4
BMOVN baddr, baddr	Move negated direct bit to direct bit	4
BAND baddr, baddr	AND direct bit with direct bit	4
BOR baddr, baddr	OR direct bit with direct bit	4
BXOR baddr, baddr	XOR direct bit with direct bit	4
BCMP baddr, baddr	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

Table 4 **Instruction Set Summary** (cont'd)

Mnemonic	Description	Bytes
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
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
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

Table 4 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

Table 4 Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Data Movement (cont'd)			
MOV	Rw, [Rw + #d16]	Move indirect word memory by base plus constant to direct word GPR	4
MOV	[Rw + #d16], Rw	Move direct word GPR to indirect memory by base plus constant	4
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 + #d16]	Move indirect byte memory by base plus constant to direct byte GPR	4
MOVB	[Rw + #d16], 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

Table 4 Instruction Set Summary (cont'd)

Mnemonic	Description	Bytes
Data Movement (cont'd)		
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
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 Instructions		
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 baddr, rel	Jump relative if direct bit is set	4
JBC baddr, rel	Jump relative and clear bit if direct bit is set	4
JNB baddr, rel	Jump relative if direct bit is not set	4
JNBS baddr, 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

Table 4 Instruction Set Summary (cont'd)

Mnemonic	Description	Bytes
Return Instructions		
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[1)]		
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 $\overline{\text{RSTOUT}}$-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

Table 4 Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
System Stack Instructions			
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 and update register with immediate data	4
SCXT	reg, mem	Push direct word register onto system stack and update register with direct memory	4
Miscellaneous			
NOP		Null operation	2

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

4 Encoding

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, i.e. during the debugging phase.

The explanations below should help to read the tables on the following pages:

Extended Opcodes

1) These instructions (ADD[C][B], SUB[C][B], CMP[B], AND[B], [X]OR[B]) are encoded by means of additional bits (1/2) in the operand field of the instruction. For these instructions only the lowest four GPRs, R0 to R3, can be used as indirect address pointers.

$nnnn.0\#\#\#_B$:	Rw_n, #data3	or Rb_n, #data3
$nnnn.10ii_B$:	Rw_n, [Rw_i]	or Rb_n, [Rw_i]
$nnnn.11ii_B$:	Rw_n, [Rw_i+]	or Rb_n, [Rw_i+]

2) The following instructions are encoded by means of two additional bits in the operand field of the instruction.

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

$00xx.xxxx_B$:	EXTS	or ATOMIC
$01xx.xxxx_B$:	EXTP	
$10xx.xxxx_B$:	EXTSR	or EXTR
$11xx.xxxx_B$:	EXTPR	

Conditional 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 (condition codes are described in **Table 5**).

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.

Undefined Opcodes

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

Note: The 8XC166(W) devices also do not recognize ATOMIC and EXTended instructions, but rather decode an undefined opcode.

Hex	Bytes	Mnemonic	Operands	Hex	Bytes	Mnemonic	Operands
00	2	ADD	Rw, Rw	10	2	ADDC	Rw, Rw
01	2	ADDB	Rb, Rb	11	2	ADDCB	Rb, Rb
02	4	ADD	reg, mem	12	4	ADDC	reg, mem
03	4	ADDB	reg, mem	13	4	ADDCB	reg, mem
04	4	ADD	mem, reg	14	4	ADDC	mem, reg
05	4	ADDB	mem, reg	15	4	ADDCB	mem, reg
06	4	ADD	reg, #data16	16	4	ADDC	reg, #data16
07	4	ADDB	reg, #data8	17	4	ADDCB	reg, #data8
08	2	ADD1)	Rw, [Rw +] or Rw, [Rw] or Rw, #data3	18	2	ADDC1)	Rw, [Rw +] or Rw, [Rw] or Rw, #data3
09	2	ADDB1)	Rb, [Rw +] or Rb, [Rw] or Rb, #data3	19	2	ADDCB1)	Rb, [Rw +] or Rb, [Rw] or Rb, #data3
0A	4	BFLDL	bitoff, #mask8, #data8	1A	4	BFLDH	bitoff, #mask8, #data8
0B	2	MUL	Rw, Rw	1B	2	MULU	Rw, Rw
0C	2	ROL	Rw, Rw	1C	2	ROL	Rw, #data4
0D	2	JMPR	cc_UC, rel	1D	2	JMPR	cc_NET, rel
0E	2	BCLR	bitoff.0	1E	2	BCLR	bitoff.1
0F	2	BSET	bitoff.0	1F	2	BSET	bitoff.1

Hex	Bytes	Mnemonic	Operands	Hex	Bytes	Mnemonic	Operands
20	2	SUB	Rw, Rw	30	2	SUBC	Rw, Rw
21	2	SUBB	Rb, Rb	31	2	SUBCB	Rb, Rb
22	4	SUB	reg, mem	32	4	SUBC	reg, mem
23	4	SUBB	reg, mem	33	4	SUBCB	reg, mem
24	4	SUB	mem, reg	34	4	SUBC	mem, reg
25	4	SUBB	mem, reg	35	4	SUBCB	mem, reg
26	4	SUB	reg, #data16	36	4	SUBC	reg, #data16
27	4	SUBB	reg, #data8	37	4	SUBCB	reg, #data8
28	2	SUB1)	Rw, [Rw +] or Rw, [Rw] or Rw, #data3	38	2	SUBC1)	Rw, [Rw +] or Rw, [Rw] or Rw, #data3
29	2	SUBB1)	Rb, [Rw +] or Rb, [Rw] or Rb, #data3	39	2	SUBCB1)	Rb, [Rw +] or Rb, [Rw] or Rb, #data3
2A	4	BCMP	bitaddr, bitaddr	3A	4	BMOVN	bitaddr, bitaddr
2B	2	PRIOR	Rw, Rw	3B	–	–	–
2C	2	ROR	Rw, Rw	3C	2	ROR	Rw, #data4
2D	2	JMPR	cc_EQ, rel or cc_Z, rel	3D	2	JMPR	cc_NE, rel or cc_NZ, rel
2E	2	BCLR	bitoff.2	3E	2	BCLR	bitoff.3
2F	2	BSET	bitoff.2	3F	2	BSET	bitoff.3

Hex	Bytes	Mnemonic	Operands	Hex	Bytes	Mnemonic	Operands
40	2	CMP	Rw, Rw	50	2	XOR	Rw, Rw
41	2	CMPB	Rb, Rb	51	2	XORB	Rb, Rb
42	4	CMP	reg, mem	52	4	XOR	reg, mem
43	4	CMPB	reg, mem	53	4	XORB	reg, mem
44	–	–	–	54	4	XOR	mem, reg
45	–	–	–	55	4	XORB	mem, reg
46	4	CMP	reg, #data16	56	4	XOR	reg, #data16
47	4	CMPB	reg, #data8	57	4	XORB	reg, #data8
48	2	CMP1)	Rw, [Rw +] or Rw, [Rw] or Rw, #data3	58	2	XOR1)	Rw, [Rw +] or Rw, [Rw] or Rw, #data3
49	2	CMPB1)	Rb, [Rw +] or Rb, [Rw] or Rb, #data3	59	2	XORB1)	Rb, [Rw +] or Rb, [Rw] or Rb, #data3
4A	4	BMOV	bitaddr, bitaddr	5A	4	BOR	bitaddr, bitaddr
4B	2	DIV	Rw	5B	2	DIVU	Rw
4C	2	SHL	Rw, Rw	5C	2	SHL	Rw, #data4
4D	2	JMPR	cc_V, rel	5D	2	JMPR	cc_NV, rel
4E	2	BCLR	bitoff.4	5E	2	BCLR	bitoff.5
4F	2	BSET	bitoff.4	5F	2	BSET	bitoff.5

Hex	Bytes	Mnemonic	Operands	Hex	Bytes	Mnemonic	Operands
60	2	AND	Rw, Rw	70	2	OR	Rw, Rw
61	2	ANDB	Rb, Rb	71	2	ORB	Rb, Rb
62	4	AND	reg, mem	72	4	OR	reg, mem
63	4	ANDB	reg, mem	73	4	ORB	reg, mem
64	4	AND	mem, reg	74	4	OR	mem, reg
65	4	ANDB	mem, reg	75	4	ORB	mem, reg
66	4	AND	reg, #data16	76	4	OR	reg, #data16
67	4	ANDB	reg, #data8	77	4	ORB	reg, #data8
68	2	AND1)	Rw, [Rw +] or Rw, [Rw] or Rw, #data3	78	2	OR1)	Rw, [Rw +] or Rw, [Rw] or Rw, #data3
69	2	ANDB1)	Rb, [Rw +] or Rb, [Rw] or Rb, #data3	79	2	ORB1)	Rb, [Rw +] or Rb, [Rw] or Rb, #data3
6A	4	BAND	bitaddr, bitaddr	7A	4	BXOR	bitaddr, bitaddr
6B	2	DIVL	Rw	7B	2	DIVLU	Rw
6C	2	SHR	Rw, Rw	7C	2	SHR	Rw, #data4
6D	2	JMPR	cc_N, rel	7D	2	JMPR	cc_NN, rel
6E	2	BCLR	bitoff.6	7E	2	BCLR	bitoff.7
6F	2	BSET	bitoff.6	7F	2	BSET	bitoff.7

Hex	Bytes	Mnemonic	Operands	Hex	Bytes	Mnemonic	Operands
80	2	CMPI1	Rw, #data4	90	2	CMPI2	Rw, #data4
81	2	NEG	Rw	91	2	CPL	Rw
82	4	CMPI1	Rw, mem	92	4	CMPI2	Rw, mem
83	–	–	–	93	–	–	–
84	4	MOV	[Rw], mem	94	4	MOV	mem, [Rw]
85	–	–	–	95	–	–	–
86	4	CMPI1	Rw, #data16	96	4	CMPI2	Rw, #data16
87	4	IDLE	–	97	4	PWRDN	–
88	2	MOV	[-Rw], Rw	98	2	MOV	Rw, [Rw+]
89	2	MOVB	[-Rw], Rb	99	2	MOVB	Rb, [Rw+]
8A	4	JB	bitaddr, rel	9A	4	JNB	bitaddr, rel
8B	–	–	–	9B	2	TRAP	#trap7
8C	–	–	–	9C	2	JMPI	cc, [Rw]
8D	2	JMPR	cc_C, rel or cc_ULT, rel	9D	2	JMPR	cc_NC, rel or cc_UGE, rel
8E	2	BCLR	bitoff.8	9E	2	BCLR	bitoff.9
8F	2	BSET	bitoff.8	9F	2	BSET	bitoff.9

Hex	Bytes	Mnemonic	Operands	Hex	Bytes	Mnemonic	Operands
A0	2	CMPD1	Rw, #data4	B0	2	CMPD2	Rw, #data4
A1	2	NEGB	Rb	B1	2	CPLB	Rb
A2	4	CMPD1	Rw, mem	B2	4	CMPD2	Rw, mem
A3	–	–	–	B3	–	–	–
A4	4	MOVB	[Rw], mem	B4	4	MOVB	mem, [Rw]
A5	4	DISWDT	–	B5	4	EINIT	–
A6	4	CMPD1	Rw, #data16	B6	4	CMPD2	Rw, #data16
A7	4	SRVWDT	–	B7	4	SRST	–
A8	2	MOV	Rw, [Rw]	B8	2	MOV	[Rw], Rw
A9	2	MOVB	Rb, [Rw]	B9	2	MOVB	[Rw], Rb
AA	4	JBC	bitaddr, rel	BA	4	JNBS	bitaddr, rel
AB	2	CALLI	cc, [Rw]	BB	2	CALLR	rel
AC	2	ASHR	Rw, Rw	BC	2	ASHR	Rw, #data4
AD	2	JMPR	cc_SGT, rel	BD	2	JMPR	cc_SLE, rel
AE	2	BCLR	bitoff.10	BE	2	BCLR	bitoff.11
AF	2	BSET	bitoff.10	BF	2	BSET	bitoff.11

Hex	Bytes	Mnemonic	Operands	Hex	Bytes	Mnemonic	Operands
C0	2	MOVBZ	Rw, Rb	D0	2	MOVBS	Rw, Rb
C1	–	–	–	D1	2	ATOMIC[2)]or EXTR[2)]	#irang2
C2	4	MOVBZ	reg, mem	D2	4	MOVBS	reg, mem
C3	–	–	–	D3	–	–	–
C4	4	MOV	[Rw+#data16], Rw	D4	4	MOV	Rw, [Rw + #data16]
C5	4	MOVBZ	mem, reg	D5	4	MOVBS	mem, reg
C6	4	SCXT	reg, #data16	D6	4	SCXT	reg, mem
C7	–	–	–	D7	4	EXTP(R)[2)], EXTS(R)[2)]	#pag10,#irang2 #seg8, #irang2
C8	2	MOV	[Rw], [Rw]	D8	2	MOV	[Rw+], [Rw]
C9	2	MOVB	[Rw], [Rw]	D9	2	MOVB	[Rw+], [Rw]
CA	4	CALLA	cc, addr	DA	4	CALLS	seg, caddr
CB	2	RET	–	DB	2	RETS	–
CC	2	NOP	–	DC	2	EXTP(R)[2)], EXTS(R)[2)]	Rw, #irang2
CD	2	JMPR	cc_SLT, rel	DD	2	JMPR	cc_SGE, rel
CE	2	BCLR	bitoff.12	DE	2	BCLR	bitoff.13
CF	2	BSET	bitoff.12	DF	2	BSET	bitoff.13

Hex	Bytes	Mnemonic	Operands	Hex	Bytes	Mnemonic	Operands
E0	2	MOV	Rw, #data4	F0	2	MOV	Rw, Rw
E1	2	MOVB	Rb, #data4	F1	2	MOVB	Rb, Rb
E2	4	PCALL	reg, caddr	F2	4	MOV	reg, mem
E3	–	–	–	F3	4	MOVB	reg, mem
E4	4	MOVB	[Rw+#data16], Rb	F4	4	MOVB	Rb, [Rw + #data16]
E5	–	–	–	F5	–	–	–
E6	4	MOV	reg, #data16	F6	4	MOV	mem, reg
E7	4	MOVB	reg, #data8	F7	4	MOVB	mem, reg
E8	2	MOV	[Rw], [Rw+]	F8	–	–	–
E9	2	MOVB	[Rw], [Rw+]	F9	–	–	–
EA	4	JMPA	cc, caddr	FA	4	JMPS	seg, caddr
EB	2	RETP	reg	FB	2	RETI	–
EC	2	PUSH	reg	FC	2	POP	reg
ED	2	JMPR	cc_UGT, rel	FD	2	JMPR	cc_ULE, rel
EE	2	BCLR	bitoff.14	FE	2	BCLR	bitoff.15
EF	2	BSET	bitoff.14	FF	2	BSET	bitoff.15

5 Detailed Description

This chapter describes each instruction in detail. The example further down on this page lists the elements of a description and demonstrates how the information given for each instruction is arranged.

The next pages explain the elements of an instruction description (see example), and then all instructions are listed individually. The instructions are ordered alphabetically.

MNEM *Short Description* *MNEM*

Syntax	*MNEM operand(s)*
Operation	*definition in pseudo-code*
[**Data Types**	BIT ǀ BYTE ǀ WORD ǀ DOUBLEWORD]
Description	*Verbal description of the instruction's effect.*
[**Note**	*Additional hints.*]

Condition Flags

E	Z	V	C	N
?	*?*	*?*	*?*	*?*

E	*Effect of this instruction on flag E.*
Z	*Effect of this instruction on flag Z.*
V	*Effect of this instruction on flag V.*
C	*Effect of this instruction on flag C.*
N	*Effect of this instruction on flag N.*

Addressing Modes

Mnemonic	Format	Bytes
MNEM operand(s)	*encoding*	2 ǀ 4
[…]		

Instruction Name

MNEM	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. These mnemonics are also used by tools such as assemblers.
Short D.	Short description which is also used in the compact tables on the previous pages.

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.

Note: 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 (pseudo code).
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 **ANDed** with (opY)
∨	(opX) is	logically **ORed** with (opY)
⊕	(opX) is	logically **EXCLUSIVELY ORed** with (opY)
⇔	(opX) is	**COMPARED** against (opY)
mod	(opX) is	divided **MODULO** (opY)

Monadic Operations:		***operator*** **(opX)**
$\neg$	(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, i.e. 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
MD	Multiply/Divide register (32 bits wide), consists of (16-bit) registers MDH and MDL
MDL, MDH	Multiply/Divide Low and High registers (both 16 bits wide)
PSW	Program Status Word register
SP	System Stack Pointer register
SYSCON	System Configuration register
C	Carry condition flag in register PSW
V	Overflow condition flag in register PSW
SGTDIS	Segmentation Disable bit in register SYSCON
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. Also hints are given on using the instruction itself, its operands, and its flags.

Note

In some cases additional notes point out special circumstances. These notes shall help the user to avoid faulty operation of his/her software.
Conditional instructions refer here to the condition codes listed in **Table 5**.

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 condition flags are displayed in the way they appear in register PSW:

PSW
Processor Status Word **SFR (FF10$_H$/88$_H$)** **Reset Value: 0000$_H$**

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

The resulting state of the flags is represented by symbols as follows:

Symbolic Settings for Condition Flags

'*'	The flag value depends on the result of the instruction and is set/cleared according to the following standard rules:

N = 1:	MSB of the result is set
N = 0:	MSB of the result is not set
C = 1:	Carry occurred during operation
C = 0	No Carry occurred during operation
V = 1:	Arithmetic Overflow occurred during operation
V = 0	No Arithmetic Overflow occurred during operation
Z = 1:	Result equals zero
Z = 0:	Result does not equal zero
E = 1:	Source operand represents the…
E = 0:	Source operand does not represent the… …lowest negative number (8000_H/80_H for word/byte data)

'S'	The flag is set/cleared according to special 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 NOR of the two specified bit operands.
'AND'	The flag contains the logical AND of the two specified bit operands.
'OR'	The flag contains the logical OR of the two specified bit operands.
'XOR'	The flag contains the logical XOR 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, register PSW 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** specifies the format of the instruction (symbols are explained on **Page 37**) 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).
- **Number of Bytes** specifies the size of an instruction in bytes. All C166 Family instructions consist of 2 bytes or 4 bytes (single word or double word instruction).

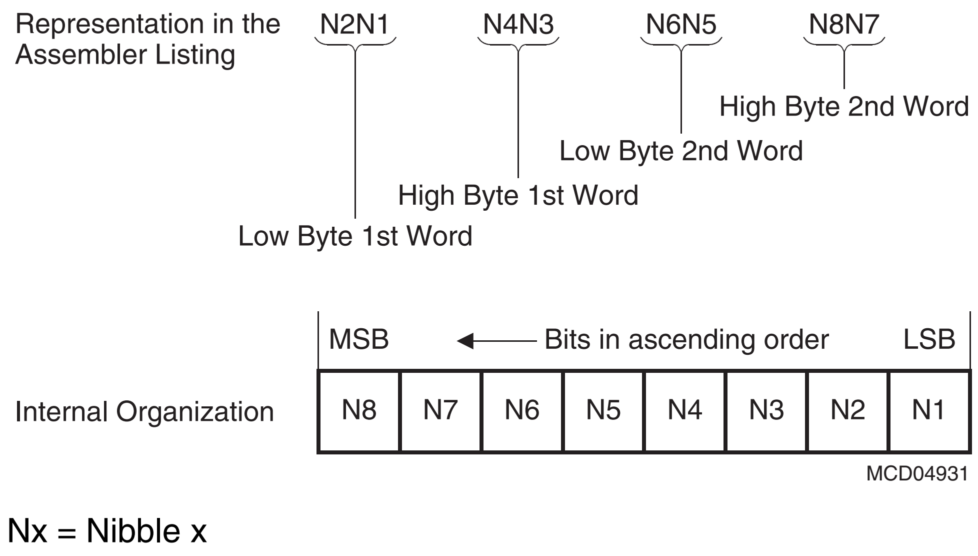

Figure 1 Instruction Format Representation

Symbols for the Instruction Format

00_H through FF_H	Instruction Opcodes (Hex)
0, 1	Constant Values (bits)
:....	Each of the 4 characters immediately following a colon represents a single bit
:..ii	2-bit short GPR address (Rw_i)
SS	Code segment number 'seg' (byte value)[1)]
:..##	2-bit immediate constant (#irang2)
:.###	3-bit immediate constant (#data3)
c	4-bit condition code specification (cc), see also **Table 5**
n	4-bit short GPR address (Rw_n or Rb_n)
m	4-bit short GPR address (Rw_m or Rb_m)
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, where 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)

1) For the SAB 8xC166 devices the segment number is a 2-bit value (:..ss) due to the smaller addressing range of 256 KByte (compared to 16 MByte).

Condition Code

Some instructions (JUMP, CALL, …) are executed only if a specific condition is true, and are skipped otherwise. The condition which has to be fulfilled for the execution of the respective instruction is specified in the so-called condition code. Table 5 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.

Table 5 Condition Code Encoding

Condition Code Mnemonic (cc)	Test	Description	Encoding (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	$(Z \vee C) = 1$	Unsigned less than or equal	F_H
cc_UGE	C = 0	Unsigned greater than or equal	9_H
cc_UGT	$(Z \vee C) = 0$	Unsigned greater than	E_H
cc_SLT	$(N \oplus V) = 1$	Signed less than	C_H
cc_SLE	$(Z \vee (N \oplus V)) = 1$	Signed less than or equal	B_H
cc_SGE	$(N \oplus V) = 0$	Signed greater than or equal	D_H
cc_SGT	$(Z \vee (N \oplus V)) = 0$	Signed greater than	A_H
cc_NET	$(Z \vee E) = 0$	Not equal AND not end of table	1_H

Peculiarities of 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 detailed instruction description).
The ATOMIC and EXTended instructions become active immediately, i.e. no interrupt/ PEC request or ClassA trap is accepted during the execution of the ATOMIC (EXTx) instruction and the following locked instructions (see #irang2). 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.

ATTENTION: When a ClassB trap interrupts 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!
Within a ClassA or ClassB trap service routine EXTend instructions do not work (i.e. override the DPP mechanism) as long as any of the ClassB trap flags is set.

ATTENTION: There is only ONE counter to control the length of an ATOMIC or EXTend sequence, i.e. issuing an ATOMIC or EXTend instruction within a sequence will reload the counter with the value of the new instruction. ATOMIC and EXTend instructions can be nested to generate longer locked sequences.
When using the ATOMIC and EXTended instructions with other system control or branch instructions, please note that the counter counts any executed instruction.

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

Peculiarities of Multiplication and Division Instructions

Multiplications and divisions are interruptible to optimize the interrupt response time. Bit MDC.MDRIU indicates that register MD is currently in use, bit PSW.MULIP indicates an interrupted multiplication. Chapter “System Programming” of the respective User’s Manual describes the handling of interrupted multiplications and divisions.

Bit MDRIU is set at the start of a MUL instruction (not when the instruction is resumed) or upon a write to register MDL or MDH. Bit MDRIU is cleared upon a read from register MDL. Bit MDRIU is affected by a write to register MDC, of course.

When the MUL instruction is interrupted, bit MULIP is set in the PSW of the interrupting routine, i.e. after pushing the previous PSW onto stack. When returning from the interrupt bit MULIP must be set/cleared according to the next executed instruction.

Note: For the first instruction after RETI bit MULIP = ‘1’ prevents the multiplicand from being reloaded (the intermediate result resides in MD).
This mechanism will disturb the operand fetching if another instruction (than the continued multiplication) is executed after RETI.

For standard interrupt handling (return to interrupted multiplication) this is done automatically. Task schedulers must keep track of interrupted multiplications in each task.

The following pages of this section contain a detailed 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, i.e. 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, i.e. 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, #data8	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 the previous Z flag was set. Cleared otherwise.

V Set if an arithmetic overflow occurred, i.e. 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 the previous Z flag was set. Cleared otherwise.

V Set if an arithmetic overflow occurred, i.e. 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, #data8	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) $\leftarrow$ (op1) $\wedge$ (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) $\leftarrow$ (op1) $\wedge$ (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, #data8	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 Please see additional notes on **Page 39**.
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) $\leftarrow$ (op1) $\wedge$ (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	$\text{bitaddr}_{Z.z}$, $\text{bitaddr}_{Q.q}$	6A QQ ZZ qz	4

BCLR Bit Clear BCLR

Syntax	BCLR op1
Operation	(op1) $\leftarrow$ 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) $\leftarrow$ (op1)
(high byte (tmp)) $\leftarrow$ ((high byte (tmp) $\wedge \neg$op2) $\vee$ op3)
(op1) $\leftarrow$ (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”).
If the target operand (op1) features bit-protection only the bits marked by a ‘1’ in the mask operand (op2) will be updated.

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, #mask8, #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”).
If the target operand (op1) features bit-protection only the bits marked by a ‘1’ in the mask operand (op2) will be updated.

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	$\text{bitaddr}_{Z.z}$, $\text{bitaddr}_{Q.q}$	4A QQ ZZ qz	4

BMOVN Bit to Bit Move and Negate BMOVN

Syntax BMOVN op1, op2

Operation (op1) $\leftarrow \neg$(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	$\text{bitaddr}_{Z.z}$, $\text{bitaddr}_{Q.q}$	5A QQ ZZ qz	4

BSET **Bit Set** BSET

Syntax BSET op1

Operation (op1) $\leftarrow$ 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	$\text{bitaddr}_{Q.q}$	qF QQ	2

BXOR Bit Logical XOR BXOR

Syntax BXOR op1, op2

Operation (op1) $\leftarrow$ (op1) $\oplus$ (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.

Note The condition codes for op1 are defined in **Table 5**.

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.

Note The condition codes for op1 are defined in **Table 5**.

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 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

$(SP) \leftarrow (SP) - 2$
$((SP)) \leftarrow (CSP)$
$(SP) \leftarrow (SP) - 2$
$((SP)) \leftarrow (IP)$
$(CSP) \leftarrow op1$
$(IP) \leftarrow op2$

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 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, i.e. 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, i.e. 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, #data8	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, i.e. 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, i.e. 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, i.e. 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, i.e. 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) $\leftarrow \neg$(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) $\leftarrow \neg$(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.

Note 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
MDRIU = 1
(MDL) $\leftarrow$ (MDL) / (op1)
(MDH) $\leftarrow$ (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).

Note DIV is interruptable.
Please see additional description on **Page 40**.

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, i.e. if the divisor (op1) was zero (the result in MDH and MDL is not valid in this case). 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

MDRIU = 1
(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).

Note DIVL is interruptable.
Please see additional description on **Page 40**.

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, i.e. the quotient cannot be represented in a word data type, or if the divisor (op1) was zero (the result in MDH and MDL is not valid in this case). 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	MDRIU = 1 (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).
Note	DIVLU is interruptable. Please see additional description on **Page 40**.

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, i.e. the quotient cannot be represented in a word data type, or if the divisor (op1) was zero (the result in MDH and MDL is not valid in this case). 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
MDRIU = 1
(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).

Note DIVU is interruptable.
Please see additional description on **Page 40**.

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, i.e. if the divisor (op1) was zero (the result in MDH and MDL is not valid in this case). 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.

Note 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/PEC interrupts and class A hardware traps are locked. The value of op1 defines the length of the effected instruction sequence.

Note Please see additional notes on **Page 39**.
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/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 Please see additional notes on **Page 39**.
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.

EXTP continued … EXTP

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/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

Please see additional notes on **Page 39**.
The EXTPR instruction is not available in the SAB 8XC166(W) devices.

EXTPR continued … EXTPR

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/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

Please see additional notes on **Page 39**.
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.

EXTS continued … EXTS

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/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 Please see additional notes on **Page 39**.
The EXTSR instruction is not available in the SAB 8XC166(W) devices.

EXTSR continued … EXTSR

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 device to enter idle mode or sleep mode (if provided by the device). In both modes the CPU is powered down. In idle mode the peripherals remain running, while in sleep mode also the peripherals are powered down. The device remains powered down until a peripheral interrupt (only possible in Idle mode) or an external interrupt occurs.
Sleep mode must be selected before executing the IDLE instruction.

Note 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.

Note The condition codes for op1 are defined in **Table 5**.

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.

Note The condition codes for op1 are defined in **Table 5**.

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.

Note The condition codes for op1 are defined in **Table 5**.

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	$\text{bitaddr}_{Q.q}$, 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.

MOV continued … MOV

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.

MOVB continued … MOVB

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 MDRIU = 1
(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.

Note MUL is interruptable.
Please see additional description on **Page 40**.

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 MDRIU = 1
(MD) $\leftarrow$ (op1) $\times$ (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.

Note MULU is interruptable.
Please see additional description on **Page 40**.

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) $\leftarrow$ 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, i.e. 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) $\leftarrow$ 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, i.e. 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) $\leftarrow$ (op1) $\vee$ (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 $(op1) \leftarrow (op1) \vee (op2)$

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, #data8	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

$(tmp) \leftarrow ((SP))$
$(SP) \leftarrow (SP) + 2$
$(op1) \leftarrow (tmp)$

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 $(tmp_{15}) \neq 1$ AND $(count) \neq 15$ AND $(op2) \neq 0$
  $(tmp_n) \leftarrow (tmp_{n-1})$
  (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 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.

Note 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
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

```
$(count) \leftarrow (op2)$
$(C) \leftarrow 0$
$(V) \leftarrow 0$
DO WHILE $(count) \neq 0$
 $(V) \leftarrow (V) \vee (C)$
 $(C) \leftarrow (op1_0)$
 $(op1_n) \leftarrow (op1_{n+1})$ [n = 0 … 14]
 $(op1_{15}) \leftarrow (C)$
 $(count) \leftarrow (count) - 1$
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

```
$(count) \leftarrow (op2)$
$(C) \leftarrow 0$
$(V) \leftarrow 0$
DO WHILE $(count) \neq 0$
    $(V) \leftarrow (C) \vee (V)$
    $(C) \leftarrow (op1_0)$
    $(op1_n) \leftarrow (op1_{n+1})$ [n = 0 … 14]
    $(op1_{15}) \leftarrow 0$
    $(count) \leftarrow (count) - 1$
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.

SHR continued … SHR

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 a similar effect on the microcontroller as an externally applied hardware reset.

Note 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	Not affected.
Z	Not affected.
V	Not affected.
C	Not affected.
N	Not affected.

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.

Note 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) $\leftarrow$ (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, i.e. 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) $\leftarrow$ (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, i.e. 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, #data8	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) $\leftarrow$ (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, i.e. 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) $\leftarrow$ (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	*	S	*

E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.

Z Set if result equals zero and the previous Z flag was set. Cleared otherwise.

V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the 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, #data8	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) $\leftarrow$ (op1) $\oplus$ (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, #data8	57 RR ## xx	4
XORB	reg, mem	53 RR MM MM	4
XORB	mem, reg	55 RR MM MM	4

6 Addressing Modes

The Infineon 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.

6.1 Short Addressing Modes

All of these addressing modes use an implicit base offset address to specify a 24-bit physical address (18-bit address for the SAB 8XC166(W) devices).
Short addressing modes permit access to the GPR, SFR/ESFR or bit-addressable memory space by specifying just 8 bits within an instruction:

Physical Address = Base Address + $\Delta \times$ Short Address

Note: Δ is 1 for byte GPRs, Δ is 2 for word GPRs and SFRs/ESFRs.

Table 6 Short Addressing

Mnemo-nic	Physical Address	Short Address Range	Scope of Access
Rw	(CP) $+ 2 \times$ Rw	Rw $= 0 \dots 15$	GPRs (word)
Rb	(CP) $+ 1 \times$ Rb	Rb $= 0 \dots 15$	GPRs (byte)
reg	00’FE00$_H$ $+ 2 \times$ reg	reg $= 00_H \dots EF_H$	SFRs (word, low byte)
	00’F000$_H$ $+ 2 \times$ reg	reg $= 00_H \dots EF_H$	ESFRs (word, low byte)[1)]
	(CP) $+ 2 \times$ (reg$\wedge$0F$_H$)	reg $= F0_H \dots FF_H$	GPRs (word)
	(CP) $+ 1 \times$ (reg$\wedge$0F$_H$)	reg $= F0_H \dots FF_H$	GPRs (byte)
bitoff	00’FD00$_H$ $+ 2 \times$ bitoff	bitoff $= 00_H \dots 7F_H$	RAM bit word offset
	00’FF00$_H$ $+ 2 \times$ (bitoff$\wedge$7F$_H$)	bitoff $= 80_H \dots EF_H$	SFR bit word offset
	00’F100$_H$ $+ 2 \times$ (bitoff$\wedge$7F$_H$)	bitoff $= 80_H \dots EF_H$	ESFR bit word offset
	(CP) $+ 2 \times$ (bitoff$\wedge$0F$_H$)	bitoff $= F0_H \dots FF_H$	GPR bit word offset
bitaddr	Word offset as with bitoff.	bitoff $= 00_H \dots FF_H$	Any single bit
	Immediate bit position.	bitpos $= 0 \dots 15$	

1) 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 ($00'FD00_H$ 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.

6.2 Long Addressing Mode

This addressing mode uses one of the four DPP registers to specify a physical 24-bit address (18-bit for the SAB 8XC166(W) devices). Any word or byte data within the entire address space can be accessed with this mode. An override mechanism for the DPP addressing scheme is also supported (see **Section 6.4**).

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 24-bit (or 18-bit) address (see **Figure 2**).

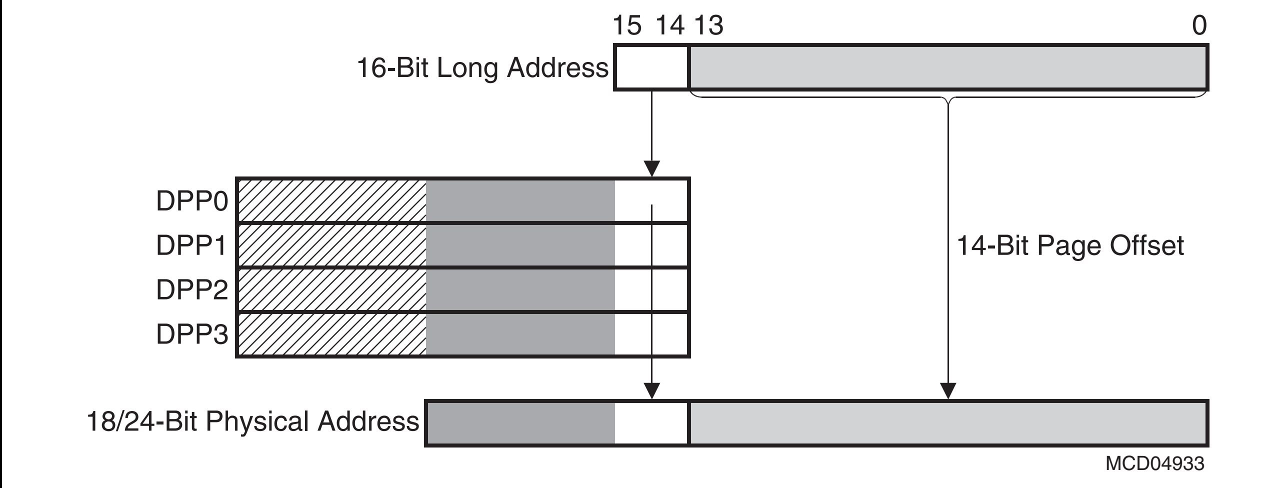

Figure 2 Interpretation of a 16-bit Long Address

The supported address space is up to 16 MByte (256 KByte for the SAB 8XC166(W) devices), so only the lower ten bits (two 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’.

Table 7 Long Addressing

Mnemonic	Physical Address	Long Address Range	Scope of Access
mem	(DPP0) ‖ mem$\wedge$3FFF$_H$ (DPP1) ‖ mem$\wedge$3FFF$_H$ (DPP2) ‖ mem$\wedge$3FFF$_H$ (DPP3) ‖ mem$\wedge$3FFF$_H$	0000$_H$ … 3FFF$_H$ 4000$_H$ … 7FFF$_H$ 8000$_H$ … BFFF$_H$ C000$_H$ … FFFF$_H$	Any Word or Byte
mem	pag ‖ mem$\wedge$3FFF$_H$	0000$_H$ … FFFF$_H$ (14-bit)	Any Word or Byte
mem	seg ‖ mem	0000$_H$ … FFFF$_H$ (16-bit)	Any Word or Byte

6.3 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 24-bit addresses (18-bit for the SAB 8XC166(W) devices). Any word or byte data within the entire memory space can be addressed indirectly.

Note: The exceptions for instructions EXTP(R) and EXTS(R), i.e. 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 (R_i = 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 $\times$ 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 24-bit (or 18-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:

Table 8 Indirect Addressing

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.

6.4 DPP Override Mechanism

The DPP override mechanism temporarily bypasses the standard DPP addressing scheme. The EXTP(R) and EXTS(R) instructions override this addressing mechanism. Instruction EXTP(R) replaces the content of the respective DPP register (i.e. the data page number) with a direct page number, while instruction EXTS(R) concatenates the complete 16-bit long address with the specified segment base address.

The overriding page or segment may be specified directly as a constant (#pag, #seg) or indirectly via a word GPR (Rw).

The override mechanism is valid for the number of instructions specified in the #irang parameter of the respective EXTend instruction.

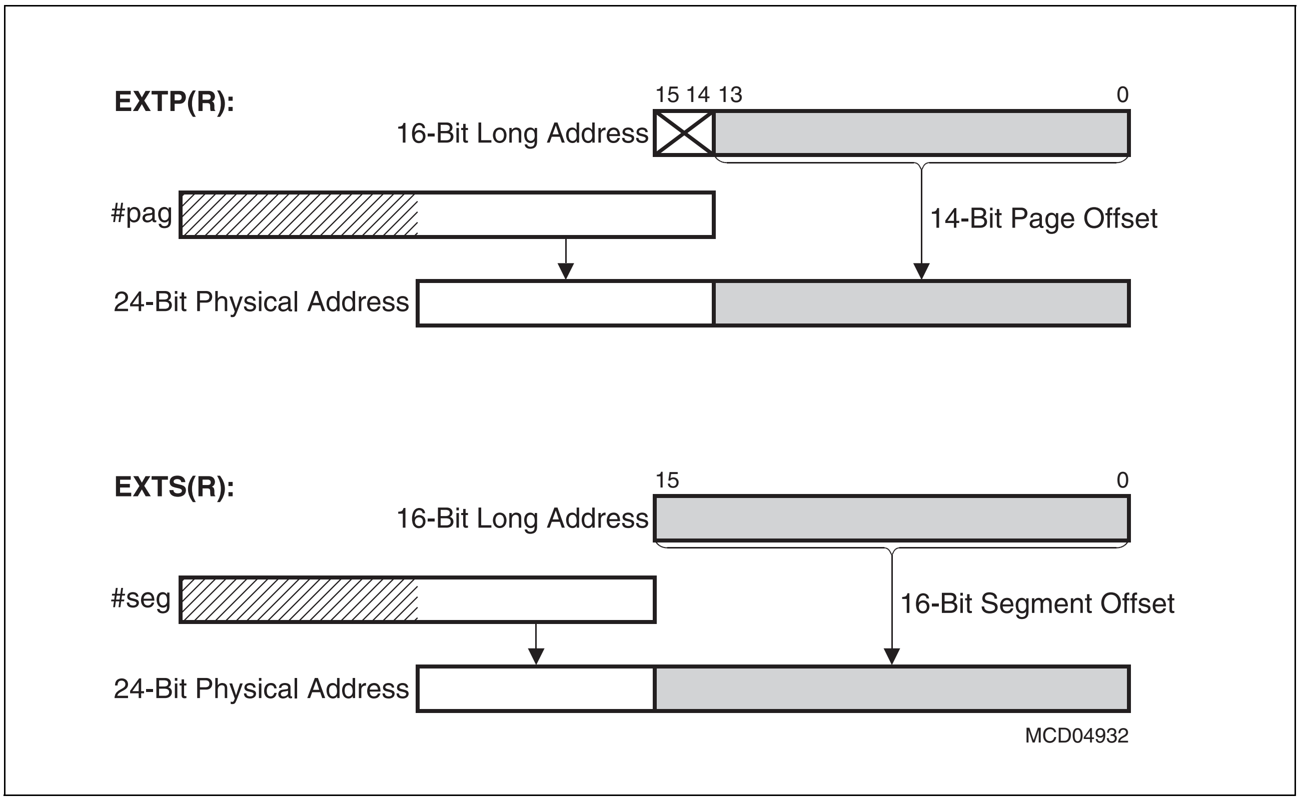

Figure 3 Overriding the DPP Mechanism

Note: The EXTend instruction (and hence the override mechanism) are not available in the SAB 8XC166(X) devices.

6.5 Constants within Instructions

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 9**):

Table 9 Constants

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 '#'.

6.6 Instruction Range (#irang2)

The effect of the ATOMIC and EXTend instructions is valid only for a limited scope and 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.

6.7 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.

Table 10 Branch Addressing

Mnemonic	Target Address	Target Segment	Valid Address Range
caddr	(IP) = caddr	-current-	caddr = 0000_H … $FFFE_H$

rel	(IP) = (IP) + 2 × rel
	(IP) = (IP) - 2 × ($\overline{\text{rel}}$+1)
		-current-
		-current-
			rel = 00_H … $7F_H$
			rel = 80_H … FF_H

[Rw]	(IP) = ((CP) + 2 × Rw)	-current-	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 (e.g. 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. 256 different code segments are supported, where the eight bits of the ‘seg’ operand value are used for updating the lower half of register CSP.

Note: The SAB 8XC166(W) devices support only 4 different code segments, where only the two lower bits 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 (i.e. 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” in the respective User’s Manual.

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 program memory (ROM, OTP, Flash). 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 execution from the internal program memory, 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 (e.g. for loading temporary programs into the internal RAM via the chip's serial interface, or end-of-line programming via the bootstrap loader). Execution from the on-chip extension RAM (XRAM) is faster than execution from the internal RAM (IRAM).

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.

In general the execution time of an instruction is composed of several additive units:

- **The minimum instruction state time** represents the number of clock cycles required to step through the instruction pipeline or to execute the instruction (MUL, DIV).
- **Operand reads** can increase the instruction's execution time if the operand …
  - is read from the on-chip program memory space.
  - is read from the IRAM immediately after a preceeding write to the IRAM.
  - is read from external resources (or from the XRAM) via the EBC.
- **Operand writes** can increase the instruction's execution time if the target is in the external memory and the write cycle conflicts with another external memory operation.
- **Jumps to the on-chip program memory** can increase the instruction's execution time if the jump target is a non-aligned doubleword-instruction.
- **Testing branch conditions** can increase the instruction's execution time if the previous instruction has written to register PSW.

7.1 Time Unit Definitions

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.

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

f_{CPU}: CPU operating frequency, may vary depending on the employed device type and on the actual operating mode of the device.

State: One state time is specified as the duration of 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]
	= 40 ns for f_{CPU} = 25 MHz

ACT: The 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
	= 80 ns … 720 ns for f_{CPU} = 25 MHz

In case of **multiplexed** external bus modes:

1 ACT	= (3 + (15 - MCTC) + (1 - MTTC)) States
	= 120 ns … 760 ns for f_{CPU} = 25 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:

$T_{tot} = T_{I1} + T_{I2} + \ldots + T_{IN} + 6$ States

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

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

7.2 Minimum Execution Time

The minimum number of state times to process an instruction is required if the instruction is fetched from the internal program memory (T_{Imin}(PM)). 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 be easily calculated by adding the indicated additional times.

Most of the 16-bit microcontroller instructions require a minimum of two state times - except some of the branches, the multiplication, the division, and a special move instruction. In case of execution from internal program memory 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 program memory, regardless of which memory area the rest of the current program is really fetched from.

For some of the branch instructions **Table 11** represents two execution times:

- standard execution time for a taken branch
- reduced execution time for a branch,
  - which is not taken, because the specified condition is not met
  - which can be serviced by the jump cache

The respective longer execution time result from the fact that after a taken branch the current instruction stream is broken and the pipeline has to be refilled.

Table 11 Minimum Instruction State Times [Unit = States / ns]

Instruction(s)	**T_{Imin}(PM) [States]**	**T_{Imin}(PM) [ns](@ 25 MHz)**
CALLI, CALLA	4 or 2	160 or 80
CALLS, CALLR, PCALL	4	160
JB, JBC, JNB, JNBS	4 or 2	160 or 80
JMPS	4	160
JMPA, JMPI, JMPR	4 or 2	160 or 80
MUL, MULU	10	400
DIV, DIVL, DIVU, DIVLU	20	800
RET, RETI, RETP, RETS	4	160
MOV[B] Rn, [Rm+#data16]	4	160
TRAP	4	160
All other instructions	2	80

Instructions executed from the internal RAM

The minimum instruction time (see **Table 11**) must be extended by an instruction-length dependent number of state times, as follows:

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

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

Instructions executed from the External Memory

The minimum instruction time (see **Table 11**) must be extended by an instruction-length dependent number of ALE Cycle Times. This number depends on the instruction length (2-byte or 4-byte) and on the data bus width (8-bit or 16-bit).
Accesses to the on-chip XRAM are controlled by the EBC and therefore also must be considered as external accesses.

For 2-byte instructions: T_{Imin}(ext) = T_{Imin}(PM) + 1 ACT

For 4-byte instructions: T_{Imin}(ext) = T_{Imin}(PM) + 2 ACT

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

7.3 Additional State Times

In most cases the given execution time also includes the handling of the involved operands (if any). Some operand accesses, however, 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.

Operand Reads from Internal Program Memory

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

– T_{Iadd} = 2 States

Operand Reads from Internal RAM via Indirect Addressing Modes

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:

```
MOV    R1, [R0+]      ;auto-increment R0
MOV    [R3], [R2]     ;if R2 points into IRAM space: Tadd = 1 State
```

In this case, the additional time can easily be avoided by putting another suitable instruction before the instruction indirectly reading the internal RAM.

– T_{Iadd} = 0 or 1 State

Operand Reads from an SFR

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:

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

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

```
ADD    R0, #1000h     ;implicit modification of PSW flags
BAND   C, Z           ;read from PSW:                 Tadd = 2 States
```

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

```
MOV    SP, #0FB00h     ;explicit update of the stack pointer
SCXT   R1, #1000h      ;implicit decrement of SP:     Tadd = 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.

– T_{Iadd} = 0 or 1 or 2 State(s)

Operand Reads from External Memory

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.

– T_{Iadd} = 1 or 2 ACT

Operand Writes to External Memory

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.

– T_{Iadd} = 0 or 1 or 2 ACT

Testing Branch Conditions

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 register PSW, as shown in the following example:

```
BSET   USR0                ;Explicit write to PSW
JMPR   cc_Z, JumpTarget    ;Test condition flag in PSW: Tadd = 1 State
```

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

– T_{Iadd} = 0 or 1 State

Jumps into the Internal Program Memory

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 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:

```
ORG     #0FFEh                    ;Any non-aligned double-word location
LABEL   JumpTarget
...     ...                       ;Any double-word instruction
...
JMPA    cc_UC, JumpTarget         ;If standard branch is taken: Tadd = 2 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:

```
ORG     #12FAh                    ;Any non-aligned double-word location
LABEL   JumpTarget
...     ...                       ;Any double-word instruction
...     ...                       ;Any double-word instruction
JMPR    cc_UC, JumpTarget         ;If cache jump is taken: Tadd = 2 States
```

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

– T_{Iadd} = 0 or 2 States

8 Keyword Index

This section lists a number of keywords which refer to specific details of the C166 Family instruction set. This helps to quickly find the answer to specific questions about the C166 Family instruction set, e.g. addressing, encoding, etc.

Infineon goes for Business Excellence

"Business excellence means intelligent approaches and clearly defined processes, which are both constantly under review and ultimately lead to good operating results.
Better operating results and business excellence mean less idleness and wastefulness for all of us, more professional success, more accurate information, a better overview and, thereby, less frustration and more satisfaction."

Dr. Ulrich Schumacher

http://www.infineon.com

Published by Infineon Technologies AG