7 INSTRUCTION SET

OVERVIEW

GLOSSARY

This chapter describes the CalmRISC instruction set and the details of each instruction are listed in alphabetical order. The following notations are used for the description.

Table 7-1. Instruction Notation Conventions

Notation	Interpretation
<opN>	Operand N. N can be omitted if there is only one operand. Typically, <op1> is the destination (and source) operand and <op2> is a source operand.
GPR	General Purpose Register
SPR	Special Purpose Register (IDL0, IDL1, IDH, SR0, ILX, ILH, ILL, SR1)
adr:N	N-bit address specifier
@idm	Content of memory location pointed by ID0 or ID1
(adr:N)	Content of memory location specified by adr:N
cc:4	4-bit condition code. Table 7-6 describes cc:4.
imm:N	N-bit immediate number
&	Bit-wise AND
|	Bit-wise OR
~	Bit-wise NOT
^	Bit-wise XOR
N**M	Mth power of N
$(N)_M$	M-based number N

As additional note, only the affected flags are described in the tables in this section. That is, if a flag is not affected by an operation, it is NOT specified.

INSTRUCTION SET MAP

Table 7-2.Overall Instruction Set Map

IR	[12:10]000	001	010	011	100	101	110	111
[15:13,7:2] 000 xxxxxx	ADD GPR, #imm:8	SUB GPR, #imm:8	CP GPR, #imm8	LD GPR, #imm:8	TM GPR, #imm:8	AND GPR, #imm:8	OR GPR, #imm:8	XOR GPR, #imm:8
001 xxxxxx	ADD GPR, @idm	SUB GPR, @idm	CP GPR, @idm	LD GPR, @idm	LD @idm, GPR	AND GPR, @idm	OR GPR, @idm	XOR GPR, @idm
010 xxxxxx	ADD GPR, adr:8	SUB GPR, adr:8	CP GPR, adr:8	LD GPR, adr:8	BITT adr:8.bs		BITS adr:8.bs	
011 xxxxxx	ADC GPR, adr:8	SBC GPR, adr:8	CPC GPR, adr:8	LD adr:8, GPR	BITR adr:8.bs		BITC adr:8.bs	
100 000000	ADD GPR, GPR	SUB GPR, GPR	CP GPR, GPR	BMS/BMC	LD SPR0, #imm:8	AND GPR, adr:8	OR GPR, adr:8	XOR GPR, adr:8
100 000001	ADC GPR, GPR	SBC GPR, GPR	CPC GPR, GPR	*invalid*				
100 000010	*invalid*	*invalid*	*invalid*	*invalid*				
100 000011	AND GPR, GPR	OR GPR, GPR	XOR GPR, GPR	*invalid*				
100 00010x	SLA/SL/ RLC/RL/ SRA/SR/ RRC/RR/ GPR	INC/INCC /DEC/ DECC/ COM/ COM2/ COMC GPR	*invalid*	*invalid*				
100 00011x	LD SPR, GPR	LD GPR, SPR	SWAP GPR, SPR	LD TBH/TBL, GPR				
100 00100x	PUSH SPR	POP SPR	*invalid*	*invalid*				
100 001010	PUSH GPR	POP GPR	LD GPR, GPR	LD GPR, TBH/TBL				

Table 7-2. Overall Instruction Set Map (Continued)

IR	[12:10]000	001	010	011	100	101	110	111
100 001011	POP	*invalid*	LDC	*invalid*	LD SPR0, #imm:8	AND GPR, adr:8	OR GPR, adr:8	XOR GPR, adr:8
100 00110x	RET/LRET/ IRET/NOP/ BREAK	*invalid*	*invalid*	*invalid*				
100 00111x	*invalid*	*invalid*	*invalid*	*invalid*				
100 01xxxx	LD GPR:bank, GPR:bank	AND SR0, #imm:8	OR SR0, #imm:8	BANK #imm:2				
100 100000 100 110011	*invalid*	*invalid*	*invalid*	*invalid*				
100 1101xx	LCALL cc:4, imm:20 (2-word instruction)							
100 1110xx	LLNK cc:4, imm:20 (2-word instruction)							
100 1111xx	LJP cc:4, imm:20 (2-word instruction)							
[15:10] 101 xxx	JR cc:4, imm:9							
110 0xx	CALLS imm:12							
110 1xx	LNKS imm:12							
111 xxx	CLD GPR, imm:8 / CLD imm:8, GPR / JNZD GPR, imm:8 / SYS #imm:8 / COP #imm:12							

NOTE: "*invalid*" - invalid instruction.

Table 7-3. Instruction Encoding

Instruction	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD GPR, #imm:8	000			000			GPR		imm[7:0]							
SUB GPR, #imm:8				001												
CP GPR, #imm:8				010												
LD GPR, #imm:8				011												
TM GPR, #imm:8				100												
AND GPR, #imm:8				101												
OR GPR, #imm:8				110												
XOR GPR, #imm:8				111												
ADD GPR, @idm	001			000			GPR		idx	mod		offset[4:0]				
SUB GPR, @idm				001												
CP GPR, @idm				010												
LD GPR, @idm				011												
LD @idm, GPR				100												
AND GPR, @idm				101												
OR GPR, @idm				110												
XOR GPR, @idm				111												
ADD GPR, adr:8	010			000			GPR		adr[7:0]							
SUB GPR, adr:8				001												
CP GPR, adr:8				010												
LD GPR, adr:8				011												
BITT adr:8.bs				10		bs										
BITS adr:8.bs				11												
ADC GPR, adr:8	011			000			GPR		adr[7:0]							
SBC GPR, adr:8				001												
CPC GPR, adr:8				010												
LD adr:8, GPR				011												
BITR adr:8.bs				10		bs										
BITC adr:8.bs				11												

Table 7-3. Instruction Encoding (Continued)

Instruction	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD GPRd, GPRs	100			000			GPRd		000000						GPRs	
SUB GPRd, GPRs				001												
CP GPRd, GPRs				010												
BMS/BMC				011												
ADC GPRd, GPRs				000					000001							
SBC GPRd, GPRs				001												
CPC GPRd, GPRs				010												
invalid				011												
invalid				ddd					000010							
AND GPRd, GPRs				000					000011							
OR GPRd, GPRs				001												
XOR GPRd, GPRs				010												
invalid				011												
ALUop1				000			GPR		00010					ALUop1		
ALUop2				001			GPR							ALUop2		
invalid				010–011			xx							xxx		
LD SPR, GPR				000			GPR		00011					SPR		
LD GPR, SPR				001			GPR							SPR		
SWAP GPR, SPR				010			GPR							SPR		
LD TBL, GPR				011			GPR							x	0	x
LD TBH, GPR														x	1	x
PUSH SPR				000			xx		00100					SPR		
POP SPR				001			xx							SPR		
invalid				010–011			xx							xxx		
PUSH GPR				000			GPR		001010						GPR	
POP GPR				001			GPR								GPR	
LD GPRd, GPRs				010			GPRd								GPRs	
LD GPR, TBL				011			GPR								0	x
LD GPR, TBH															1	x
POP				000			xx		001011						xx	
LDC @IL				010											0	x
LDC @IL+															1	x
Invalid				001, 011											xx	

NOTE: "x" means not applicable.

Table 7-3. Instruction Encoding (Concluded)

Instruction	15-13	12	11	10	9	8	7	6	5	4	3	2	1	0	2nd word
MODop1	100	000			xx		00110					MODop1			–
Invalid		001–011			xx							xxx			
Invalid		000			xx		01		xxxxxx						
AND SR0, #imm:8		001			imm[7:6]				imm[5:0]						
OR SR0, #imm:8		010			imm[7:6]										
BANK #imm:2		011			xx				x	imm [1:0]		xxx			
Invalid		0	xxxx				10000000-11001111								
LCALL cc, imm:20			cc				1101				imm[19:16]				imm[15:0]
LLNK cc, imm:20															
LJP cc, imm:20															
LD SPR0, #imm:8		1	00		SPR0		IMM[7:0]								–
AND GPR, adr:8			01		GPR		ADR[7:0]								
OR GPR, adr:8			10												
XOR GPR, adr:8			11												
JR cc, imm:9	101	imm [8]	cc				imm[7:0]								
CALLS imm:12	110	0	imm[11:0]												
LNKS imm:12		1													
CLD GPR, imm:8	111	0	00		GPR		imm[7:0]								
CLD imm:8, GPR			01		GPR										
JNZD GPR, imm:8			10		GPR										
SYS #imm:8			11		xx										
COP #imm:12		1	imm[11:0]												

NOTES:

1. "x" means not applicable.
2. There are several MODop1 codes that can be used, as described in table 7-9.
3. The operand 1(GPR) of the instruction JNZD is Bank 3's register.

Table 7-4. Index Code Information (“idx”)

Symbol	Code	Description
ID0	0	Index 0 IDH:IDL0
ID1	1	Index 1 IDH:IDL1

Table 7-5. Index Modification Code Information (“mod”)

Symbol	Code	Function
@IDx + offset:5	00	DM[IDx], IDx ← IDx + offset
@[IDx - offset:5]	01	DM[IDx + (2’s complement of offset:5)], IDx ← IDx + (2’s complement of offset:5)
@[IDx + offset:5]!	10	DM[IDx + offset], IDx ← IDx
@[IDx - offset:5]!	11	DM[IDx + (2’s complement of offset:5)], IDx ← IDx

NOTE: Carry from IDL is propagated to IDH. In case of @[IDx - offset:5] or @[IDx - offset:5]!, the assembler should convert offset:5 to the 2’s complement format to fill the operand field (offset[4:0]).
Furthermore, @[IDx - 0] and @[IDx - 0]! are converted to @[IDx + 0] and @[IDx + 0]!, respectively.

Table 7-6. Condition Code Information (“cc”)

Symbol (cc:4)	Code	Function
Blank	0000	always
NC or ULT	0001	C = 0, unsigned less than
C or UGE	0010	C = 1, unsigned greater than or equal to
Z or EQ	0011	Z = 1, equal to
NZ or NE	0100	Z = 0, not equal to
OV	0101	V = 1, overflow - signed value
ULE	0110	~C | Z, unsigned less than or equal to
UGT	0111	C & ~Z, unsigned greater than
ZP	1000	N = 0, signed zero or positive
MI	1001	N = 1, signed negative
PL	1010	~N & ~Z, signed positive
ZN	1011	Z | N, signed zero or negative
SF	1100	Stack Full
EC0-EC2	1101-1111	EC[0] = 1/EC[1] = 1/EC[2] = 1

NOTE: EC[2:0] is an external input (CalmRISC core’s point of view) and used as a condition.

Table 7-7. "ALUop1" Code Information

Symbol	Code	Function
SLA	000	arithmetic shift left
SL	001	shift left
RLC	010	rotate left with carry
RL	011	rotate left
SRA	100	arithmetic shift right
SR	101	shift right
RRC	110	rotate right with carry
RR	111	rotate right

Table 7-8. "ALUop2" Code Information

Symbol	Code	Function
INC	000	increment
INCC	001	increment with carry
DEC	010	decrement
DECC	011	decrement with carry
COM	100	1's complement
COM2	101	2's complement
COMC	110	1's complement with carry
–	111	reserved

Table 7-9. "MODop1" Code Information

Symbol	Code	Function
LRET	000	return by IL
RET	001	return by HS
IRET	010	return from interrupt (by HS)
NOP	011	no operation
BREAK	100	reserved for debugger use only
–	101	reserved
–	110	reserved
–	111	reserved

QUICK REFERENCE

Operation	op1	op2	Function	Flag	# of word / cycle
AND	GPR	adr:8	op1 ← op1 & op2	z,n	1W1C
OR		#imm:8	op1 ← op1 | op2	z,n	
XOR		GPR	op1 ← op1 ^ op2	z,n	
ADD		@idm	op1 ← op1 + op2	c,z,v,n	
SUB			op1 ← op1 + ~op2 + 1	c,z,v,n	
CP			op1 + ~op2 + 1	c,z,v,n	
ADC	GPR	GPR	op1 ← op1 + op2 + c	c,z,v,n	
SBC		adr:8	op1 ← op1 + ~op2 + c	c,z,v,n	
CPC			op1 + ~op2 + c	c,z,v,n	
TM	GPR	#imm:8	op1 & op2	z,n	
BITS	R3	adr:8.bs	op1 ← (op2[bit] ← 1)	z	
BITR			op1 ← (op2[bit] ← 0)	z	
BITC			op1 ← ~(op2[bit])	z	
BITT			z ← ~(op2[bit])	z	
BMS/BMC	–	–	TF ← 1 / 0	–	
PUSH	GPR	–	HS[sptr] ← GPR, (sptr ← sptr + 1)	–	
POP			GPR ← HS[sptr - 1], (sptr ← sptr - 1)	z,n	
PUSH	SPR	–	HS[sptr] ← SPR, (sptr ← sptr + 1)	–	
POP			SPR ← HS[sptr - 1], (sptr ← sptr - 1)		
POP	–	–	sptr ← sptr – 2	–	
SLA	GPR	–	c ← op1[7], op1 ← {op1[6:0], 0}	c,z,v,n	
SL			c ← op1[7], op1 ← {op1[6:0], 0}	c,z,n	
RLC			c ← op1[7], op1 ← {op1[6:0], c}	c,z,n	
RL			c ← op[7], op1 ← {op1[6:0], op1[7]}	c,z,n	
SRA			c ← op[0], op1 ← {op1[7],op1[7:1]}	c,z,n	
SR			c ← op1[0], op1 ← {0, op1[7:1]}	c,z,n	
RRC			c ← op1[0], op1 ← {c, op1[7:1]}	c,z,n	
RR			c ← op1[0], op1 ← {op1[0], op1[7:1]}	c,z,n	
INC			op1 ← op1 + 1	c,z,v,n	
INCC			op1 ← op1 + c	c,z,v,n	
DEC			op1 ← op1 + 0FFh	c,z,v,n	
DECC			op1 ← op1 + 0FFh + c	c,z,v,n	
COM			op1 ← ~op1	z,n	
COM2			op1 ← ~op1 + 1	c,z,v,n	
COMC			op1 ← ~op1 + c	c,z,v,n	

QUICK REFERENCE (Continued)

Operation	op1	op2	Function	Flag	# of word / cycle
LD	GPR :bank	GPR :bank	op1 ← op2	z,n	1W1C
LD	SPR0	#imm:8	op1 ← op2	–	
LD	GPR	GPR SPR adr:8 @idm #imm:8 TBH/TBL	op1 ← op2	z,n	
LD	SPR TBH/TBL	GPR	op1 ← op2	–	
LD	adr:8	GPR	op1 ← op2	–	
LD	@idm	GPR	op1 ← op2	–	
LDC	@IL @IL+	–	(TBH:TBL) ← PM[(ILX:ILH:ILL)], ILL++ if @IL+	–	1W2C
AND OR	SR0	#imm:8	SR0 ← SR0 & op2 SR0 ← SR0 | op2	–	1W1C
BANK	#imm:2	–	SR0[4:3] ← op2	–	
SWAP	GPR	SPR	op1 ← op2, op2 ← op1 (excluding SR0/SR1)	–	
LCALL cc	imm:20	–	If branch taken, push XSTACK, HS[15:0] ← {PC[15:12],PC[11:0] + 2} and PC ← op1 else PC[11:0] ← PC[11:0] + 2	–	2W2C
LLNK cc	imm:20	–	If branch taken, IL[19:0] ← {PC[19:12], PC[11:0] + 2} and PC ← op1 else PC[11:0] ← PC[11:0] + 2	–	
CALLS	imm:12	–	push XSTACK, HS[15:0] ← {PC[15:12], PC[11:0] + 1} and PC[11:0] ← op1	–	1W2C
LNKS	imm:12	–	IL[19:0] ← {PC[19:12], PC[11:0] + 1} and PC[11:0] ← op1	–	
JNZD	Rn	imm:8	if (Rn == 0) PC ← PC[delay slot] - 2's complement of imm:8, Rn-- else PC ← PC[delay slot]++, Rn--	–	
LJP cc	imm:20	–	If branch taken, PC ← op1 else PC[11:0] < PC[11:0] + 2	–	2W2C
JR cc	imm:9	–	If branch taken, PC[11:0] ← PC[11:0] + op1 else PC[11:0] ← PC[11:0] + 1	–	1W2C

NOTE: op1 - operand1, op2 - operand2, 1W1C - 1-Word 1-Cycle instruction, 1W2C - 1-Word 2-Cycle instruction, 2W2C - 2-Word 2-Cycle instruction. The Rn of instruction JNZD is Bank 3's GPR.

QUICK REFERENCE (Concluded)

Operation	op1	op2	Function	Flag	# of word / cycle
LRET	–	–	PC ← IL[19:0]	–	1W2C
RET			PC ← HS[sptr - 2], (sptr ← sptr - 2)		1W2C
IRET			PC ← HS[sptr - 2], (sptr ← sptr - 2)		1W2C
NOP			no operation		1W1C
BREAK			no operation and hold PC		1W1C
SYS	#imm:8	–	no operation but generates SYSCP[7:0] and nSYSID	–	1W1C
CLD	imm:8	GPR	op1 ← op2, generates SYSCP[7:0], nCLDID, and CLDWR	–	
CLD	GPR	imm:8	op1 ← op2, generates SYSCP[7:0], nCLDID, and CLDWR	z,n	
COP	#imm:12	–	generates SYSCP[11:0] and nCOPID	–	

NOTES:

1. op1 - operand1, op2 - operand2, sptr - stack pointer register, 1W1C - 1-Word 1-Cycle instruction, 1W2C - 1-Word 2-Cycle instruction
2. Pseudo instructions
   - — SCF/RCF
     Carry flag set or reset instruction
   - — STOP/IDLE
     MCU power saving instructions
   - — EI/DI
     Exception enable and disable instructions
   - — JP/LNK/CALL
     If JR/LNKS/CALLS commands (1 word instructions) can access the target address, there is no conditional code in the case of CALL/LNK, and the JP/LNK/CALL commands are assembled to JR/LNKS/CALLS in linking time, or else the JP/LNK/CALL commands are assembled to LJP/LLNK/LCALL (2 word instructions) instructions.

INSTRUCTION GROUP SUMMARY

ALU INSTRUCTIONS

"ALU instructions" refer to the operations that use ALU to generate results. ALU instructions update the values in Status Register 1 (SR1), namely carry (C), zero (Z), overflow (V), and negative (N), depending on the operation type and the result.

ALUop GPR, adr:8

Performs an ALU operation on the value in GPR and the value in DM[adr:8] and stores the result into GPR.
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPR+(not DM[adr:8])+1 is performed.
adr:8 is the offset in a specific data memory page.

The data memory page is 0 or the value of IDH (Index of Data Memory Higher Byte Register), depending on the value of eid in Status Register 0 (SR0).

Operation

GPR ← GPR ALUop DM[00h:adr:8] if eid = 0
GPR ← GPR ALUop DM[IDH:adr8] if eid = 1
Note that this is an 8-bit operation.

Example

```
ADD R0, 80h          // Assume eid = 1 and IDH = 01H
                     // R0 ← R0 + DM[0180h]
```

ALUop GPR, #imm:8

Stores the result of an ALU operation on GPR and an 8-bit immediate value into GPR.
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPR+(not #imm:8)+1 is performed.
#imm:8 is an 8-bit immediate value.

Operation

GPR ← GPR ALUop #imm:8

Example

```
ADD R0, #7Ah         // R0 ← R0 + 7Ah
```

ALUop GPRd, GPRs

Store the result of ALUop on GPRs and GPRd into GPRd.
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPRd + (not GPRs) + 1 is performed.
GPRs and GPRd need not be distinct.

Operation

GPRd ← GPRd ALUop GPRs
GPRd - GPRs when ALUop = CP (comparison only)

Example

```
ADD R0, R1          // R0 ← R0 + R1
```

ALUop GPR, @idm

Performs ALUop on the value in GPR and DM[ID] and stores the result into GPR. Index register ID is IDH:IDL (IDH:IDL0 or IDH:IDL1).
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPR+(not DM[idm])+1 is performed.
idm = IDx+off:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]!
(IDx = ID0 or ID1)

Operation

GPR - DM[idm] when ALUop = CP (comparison only)
GPR ← GPR ALUop DM[IDx], IDx ← IDx + offset:5 when idm = IDx + offset:5
GPR ← GPR ALUop DM[IDx - offset:5], IDx ← IDx - offset:5 when idm = [IDx - offset:5]
GPR ← GPR ALUop DM[IDx + offset:5] when idm = [IDx + offset:5]!
GPR ← GPR ALUop DM[IDx - offset:5] when idm = [IDx - offset:5]!

When carry is generated from IDL (on a post-increment or pre-decrement), it is propagated to IDH.

Example

```
ADD R0, @ID0+2      // assume ID0 = 02FFh
                    // R0 ← R0 + DM[02FFh], IDH ← 03h and IDL0 ← 01h
ADD R0, @[ID0-2]    // assume ID0 = 0201h
                    // R0 ← R0 + DM[01FFh], IDH ← 01h and IDL0 ← FFh
ADD R0, @[ID1+2]!   // assume ID1 = 02FFh
                    // R0 ← R0 + DM[0301], IDH ← 02h and IDL1 ← FFh
ADD R0, @[ID1-2]!   // assume ID1 = 0200h
                    // R0 ← R0 + DM[01FEh], IDH ← 02h and IDL1 ← 00h
```

ALUopc GPRd, GPRs

Performs ALUop with carry on GPRd and GPRs and stores the result into GPRd.
ALUopc = ADC, SBC, CPC
GPRd and GPRs need not be distinct.

Operation

GPRd ← GPRd + GPRs + C when ALUopc = ADC
GPRd ← GPRd + (not GPRs) + C when ALUopc = SBC
GPRd + (not GPRs) + C when ALUopc = CPC (comparison only)

Example

```
ADD R0, R2      // assume R1:R0 and R3:R2 are 16-bit signed or unsigned numbers.
ADC R1, R3      // to add two 16-bit numbers, use ADD and ADC.

SUB R0, R2      // assume R1:R0 and R3:R2 are 16-bit signed or unsigned numbers.
SBC R1, R3      // to subtract two 16-bit numbers, use SUB and SBC.

CP R0, R2       // assume both R1:R0 and R3:R2 are 16-bit unsigned numbers.
CPC R1, R3      // to compare two 16-bit unsigned numbers, use CP and CPC.
```

ALUopc GPR, adr:8

Performs ALUop with carry on GPR and DM[adr:8].

Operation

GPR ← GPR + DM[adr:8] + C when ALUopc = ADC
GPR ← GPR + (not DM[adr:8]) + C when ALUopc = SBC
GPR + (not DM[adr:8]) + C when ALUopc = CPC (comparison only)

CPLop GPR (Complement Operations)

CPLop = COM, COM2, COMC

Operation

COM GPR	not GPR (logical complement)
COM2 GPR	not GPR + 1 (2's complement of GPR)
COMC GPR	not GPR + C (logical complement of GPR with carry)

Example

```
COM2 R0     // assume R1:R0 is a 16-bit signed number.
COMC R1     // COM2 and COMC can be used to get the 2's complement of it.
```

IncDec GPR (Increment/Decrement Operations)

IncDec = INC, INCC, DEC, DECC

Operation

INC GPR	Increase GPR, i.e., GPR ← GPR + 1
INCC GPR	Increase GPR if carry = 1, i.e., GPR ← GPR + C
DEC GPR	Decrease GPR, i.e., GPR ← GPR + FFh
DECC GPR	Decrease GPR if carry = 0, i.e., GPR ← GPR + FFh + C

Example

```
INC R0      // assume R1:R0 is a 16-bit number
INCC R1     // to increase R1:R0, use INC and INCC.

DEC R0      // assume R1:R0 is a 16-bit number
DECC R1     // to decrease R1:R0, use DEC and DECC.
```

SHIFT/ROTATE INSTRUCTIONS

Shift (Rotate) instructions shift (rotate) the given operand by 1 bit. Depending on the operation performed, a number of Status Register 1 (SR1) bits, namely Carry (C), Zero (Z), Overflow (V), and Negative (N), are set.

SL GPR

Operation

Carry (C) is the MSB of GPR before shifting, Negative (N) is the MSB of GPR after shifting.
Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

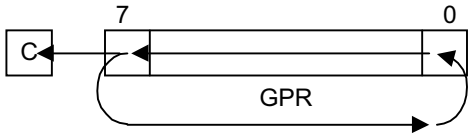
SLA GPR

Operation

Carry (C) is the MSB of GPR before shifting, Negative (N) is the MSB of GPR after shifting.
Overflow (V) will be 1 if the MSB of the result is different from C. Z will be 1 if the result is 0.

RL GPR

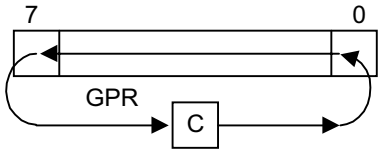
Operation



Carry (C) is the MSB of GPR before rotating. Negative (N) is the MSB of GPR after rotatin/g.
Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

RLC GPR

Operation



Carry (C) is the MSB of GPR before rotating, Negative (N) is the MSB of GPR after rotating.
Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

SR GPR

Operation

Carry (C) is the LSB of GPR before shifting, Negative (N) is the MSB of GPR after shifting. Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

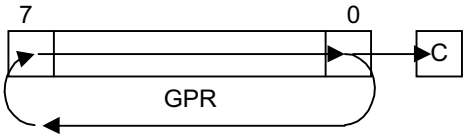
SRA GPR

Operation

Carry (C) is the LSB of GPR before shifting, Negative (N) is the MSB of GPR after shifting. Overflow (V) is not affected. Z will be 1 if the result is 0.

RR GPR

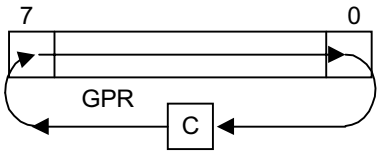
Operation



Carry (C) is the LSB of GPR before rotating. Negative (N) is the MSB of GPR after rotating. Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

RRC GPR

Operation



Carry (C) is the LSB of GPR before rotating, Negative (N) is the MSB of GPR after rotating. Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

LOAD INSTRUCTIONS

Load instructions transfer data from data memory to a register or from a register to data memory, or assigns an immediate value into a register. As a side effect, a load instruction placing a value into a register sets the Zero (Z) and Negative (N) bits in Status Register 1 (SR1), if the placed data is 00h and the MSB of the data is 1, respectively.

LD GPR, adr:8

Loads the value of DM[adr:8] into GPR. Adr:8 is offset in the page specified by the value of eid in Status Register 0 (SR0).

Operation

GPR ← DM[00h:adr:8] if eid = 0
GPR ← DM[IDH:adr:8] if eid = 1

Note that this is an 8-bit operation.

Example

```
LD R0, 80h          // assume eid = 1 and IDH= 01H
                    // R0 ← DM[0180h]
```

LD GPR, @idm

Loads a value from the data memory location specified by @idm into GPR.
idm = IDx+off:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]!
(IDx = ID0 or ID1)

Operation

GPR ← DM[IDx], IDx ← IDx + offset:5 when idm = IDx + offset:5
GPR ← DM[IDx - offset:5], IDx ← IDx - offset:5 when idm = [IDx - offset:5]
GPR ← DM[IDx + offset:5] when idm = [IDx + offset:5]!
GPR ← DM[IDx - offset:5] when idm = [IDx - offset:5]!

When carry is generated from IDL (on a post-increment or pre-decrement), it is propagated to IDH.

Example

```
LD R0, @[ID0 + 03h]!    // assume IDH:IDL0 = 0270h
                        // R0 ← DM[0273h], IDH:IDL0 ← 0270h
```

LD REG, #imm:8

Loads an 8-bit immediate value into REG. REG can be either GPR or an SPR0 group register - IDH (Index of Data Memory Higher Byte Register), IDL0 (Index of Data Memory Lower Byte Register)/ IDL1, and Status Register 0 (SR0). #imm:8 is an 8-bit immediate value.

Operation

REG ← #imm:8

Example

```
LD R0 #7Ah          // R0 ← 7Ah
LD IDH, #03h        // IDH ← 03h
```

LD GPR:bs:2, GPR:bs:2

Loads a value of a register from a specified bank into another register in a specified bank.

Example

```
LD R0:1, R2:3       // R0 in bank 1, R2 in bank 3
```

LD GPR, TBH/TBL

Loads the value of TBH or TBL into GPR. TBH and TBL are 8-bit long registers used exclusively for LDC instructions that access program memory. Therefore, after an LDC instruction, LD GPR, TBH/TBL instruction will usually move the data into GPRs, to be used for other operations.

Operation

GPR ← TBH (or TBL)

Example

```
LDC @IL             // gets a program memory item residing @ ILX:ILH:ILL
LD R0, TBH
LD R1, TBL
```

LD TBH/TBL, GPR

Loads the value of GPR into TBH or TBL. These instructions are used in pair in interrupt service routines to save and restore the values in TBH/TBL as needed.

Operation

TBH (or TBL) ← GPR

LD GPR, SPR

Loads the value of SPR into GPR.

Operation

GPR ← SPR

Example

```
LD R0, IDH          // R0 ← IDH
```

LD SPR, GPR

Loads the value of GPR into SPR.

Operation

SPR ← GPR

Example

```
LD IDH, R0          // IDH ← R0
```

LD adr:8, GPR

Stores the value of GPR into data memory (DM). adr:8 is offset in the page specified by the value of eid in Status Register 0 (SR0).

Operation

DM[00h:adr:8] ← GPR if eid = 0
DM[IDH:adr:8] ← GPR if eid = 1

Note that this is an 8-bit operation.

Example

```
LD 7Ah, R0          // assume eid = 1 and IDH = 02h.
                    // DM[027Ah] ← R0
```

LD @idm, GPR

Loads a value into the data memory location specified by @idm from GPR.
idm = IDx+off:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]!
(IDx = ID0 or ID1)

Operation

DM[IDx] ← GPR, IDx ← IDx + offset:5 when idm = IDx + offset:5
DM[IDx - offset:5] ← GPR, IDx ← IDx - offset:5 when idm = [IDx - offset:5]
DM[IDx + offset:5] ← GPR when idm = [IDx + offset:5]!
DM[IDx - offset:5] ← GPR when idm = [IDx - offset:5]!

When carry is generated from IDL (on a post-increment or pre-decrement), it is propagated to IDH.

Example

```
LD @[ID0 + 03h]!, R0    // assume IDH:IDL0 = 0170h
                        // DM[0173h] ← R0, IDH:IDL0 ← 0170h
```

BRANCH INSTRUCTIONS

Branch instructions can be categorized into jump instruction, link instruction, and call instruction. A jump instruction does not save the current PC, whereas a call instruction saves ("pushes") the current PC onto the stack and a link instruction saves the PC in the link register IL. Status registers are not affected. Each instruction type has a 2-word format that supports a 20-bit long jump.

JR cc:4, imm:9

imm:9 is a signed number (2's complement), an offset to be added to the current PC to compute the target (PC[19:12]:(PC[11:0] + imm:9)).

Operation

PC[11:0] ← PC[11:0] + imm:9 if branch taken (i.e., cc:4 resolves to be true)
PC[11:0] ← PC[11:0] + 1 otherwise

Example

```
L18411:                 // assume current PC = 18411h.
        JR Z, 107h      // next PC is 18518 (18411h + 107h) if Zero (Z) bit is set.
```

LJP cc:4, imm:20

Jumps to the program address specified by imm:20. If program size is less than 64K word, PC[19:16] is not affected.

Operation

PC[15:0] ← imm[15:0] if branch taken and program size is less than 64K word
PC[19:0] ← imm[19:0] if branch taken and program size is equal to 64K word or more
PC [11:0] ← PC[11:0] + 1 otherwise

Example

```
L18411:                 // assume current PC = 18411h.
        LJP Z, 10107h   // next instruction's PC is 10107h If Zero (Z) bit is set
```

JNZD Rn, imm:8

Jumps to the program address specified by imm:8 if the value of the bank 3 register Rn is not zero. JNZD performs only backward jumps, with the value of Rn automatically decreased. There is one delay slot following the JNZD instruction that is always executed, regardless of whether JNZD is taken or not.

Operation

If (Rn == 0) PC ← PC[delay slot] (-) 2's complement of imm:8, Rn ← Rn - 1
else PC ← PC[delay slot] + 1, Rn ← Rn - 1.

Example

```
LOOP_A:                 // start of loop body
        •
        •
        •
        JNZD R0, LOOP_A     // jump back to LOOP_A if R0 is not zero
        ADD R1, #2          // delay slot, always executed (you must use one cycle instruction only)
```

CALLS imm:12

Saves the current PC on the stack (“pushes” PC) and jumps to the program address specified by imm:12. The current page number PC[19:12] is not changed. Since this is a 1-word instruction, the return address pushed onto the stack is (PC + 1). If nP64KW is low when PC is saved, PC[19:16] is not saved in the stack.

Operation

```
HS[sptr][15:0] ← current PC + 1 and sptr ← sptr + 2 (push stack)    if nP64KW = 0
HS[sptr][19:0] ← current PC + 1 and sptr ← sptr + 2 (push stack)    if nP64KW = 1
PC[11:0] ← imm:12
```

Example

```
L18411:                 // assume current PC = 18411h.
        CALLS 107h      // call the subroutine at 18107h, with the current PC pushed
                        // onto the stack (HS ← 18412h) if nP64KW = 1.
```

LCALL cc:4, imm:20

Saves the current PC onto the stack (pushes PC) and jumps to the program address specified by imm:20. Since this is a 2-word instruction, the return address saved in the stack is (PC + 2). If nP64KW, a core input signal is low when PC is saved, 0000111111PC[19:16] is not saved in the stack and PC[19:16] is not set to imm[19:16].

Operation

```
HS[sptr][15:0] ← current PC + 2 and sptr + 2 (push stack)   if branch taken and nP64KW = 0
HS[sptr][19:0] ← current PC + 2 and sptr + 2 (push stack)   if branch taken and nP64KW = 1
PC[15:0] ← imm[15:0]    if branch taken and nP64KW = 0
PC[19:0] ← imm[19:0]    if branch taken and nP64KW = 1
PC[11:0] ← PC[11:0] + 2   otherwise
```

Example

```
L18411:                 // assume current PC = 18411h.
        LCALL NZ, 10h:107h  // call the subroutine at 10107h with the current PC pushed
                            // onto the stack (HS ← 18413h)
```

LNKS imm:12

Saves the current PC in IL and jumps to the program address specified by imm:12. The current page number PC[19:12] is not changed. Since this is a 1-word instruction, the return address saved in IL is (PC + 1). If the program size is less than 64K word when PC is saved, PC[19:16] is not saved in ILX.

Operation

IL[15:0] ← current PC + 1 if program size is less than 64K word
IL[19:0] ← current PC + 1 if program size is equal to 64K word or more
PC[11:0] ← imm:12

Example

```
L18411:                 // assume current PC = 18411h.
        LNKS 107h       // call the subroutine at 18107h, with the current PC saved
                        // in IL (IL[19:0] ← 18412h) if program size is 64K word or more.
```

LLNK cc:4, imm:20

Saves the current PC in IL and jumps to the program address specified by imm:20. Since this is a 2-word instruction, the return address saved in IL is (PC + 2). If the program size is less than 64K word when PC is saved, PC[19:16] is not saved in ILX.

Operation

IL[15:0] ← current PC + 2 if branch taken and program size is less than 64K word
IL[19:0] ← current PC + 2 if branch taken and program size is 64K word or more
PC[15:0] ← imm[15:0] if branch taken and program size is less than 64K word
PC[19:0] ← imm[19:0] if branch taken and program size is 64K word or more
PC[11:0] ← PC[11:0] + 2 otherwise

Example

```
L18411:                 // assume current PC = 18411h.
        LLNK NZ, 10h:107h   // call the subroutine at 10107h with the current PC saved
                            // in IL (IL[19:0] ← 18413h) if program size is 64K word or more
```

RET, IRET

Returns from the current subroutine. IRET sets ie (SR0[1]) in addition. If the program size is less than 64K word, PC[19:16] is not loaded from HS[19:16].

Operation

PC[15:0] ← HS[sptr - 2] and sptr ← sptr - 2 (pop stack) if program size is less than 64K word
PC[19:0] ← HS[sptr - 2] and sptr ← sptr - 2 (pop stack) if program size is 64K word or more

Example

```
        RET             // assume sptr = 3h and HS[1] = 18407h.
                        // the next PC will be 18407h and sptr is set to 1h
```

LRET

Returns from the current subroutine, using the link register IL. If the program size is less than 64K word, PC[19:16] is not loaded from ILX.

Operation

PC[15:0] ← IL[15:0] if program size is less than 64K word
PC[19:0] ← IL[19:0] if program size is 64K word or more

Example

```
LRET        // assume IL = 18407h.
            // the next instruction to execute is at PC = 18407h
                        // if program size is 64K word or more
```

JP/LNK/CALL

JP/LNK/CALL instructions are pseudo instructions. If JR/LNKS/CALLS commands (1 word instructions) can access the target address, there is no conditional code in the case of CALL/LNK and the JP/LNK/CALL commands are assembled to JR/LNKS/CALLS in linking time or else the JP/LNK/CALL commands are assembled to LJP/LLNK/LCALL (2 word instructions) instructions.

BIT MANIPULATION INSTRUCTIONS

BITop adr:8.bs

Performs a bit operation specified by op on the value in the data memory pointed by adr:8 and stores the result into R3 of current GPR bank or back into memory depending on the value of TF bit.

BITop = BITS, BITR, BITC, BITT
BITS: bit set
BITR: bit reset
BITC: bit complement
BITT: bit test (R3 is not touched in this case)
bs: bit location specifier, 0 - 7.

Operation

R3 ← DM[00h:adr:8] BITop bs if eid = 0
R3 ← DM[IDH:adr:8] BITop bs if eid = 1 (no register transfer for BITT)
Set the Zero (Z) bit if the result is 0.

Example

```
BITS 25h.3          // assume eid = 0. set bit 3 of DM[00h:25h] and store the result in R3.
BITT 25h.3          // check bit 3 of DM[00h:25h] if eid = 0.
```

BMC/BMS

Clears or sets the TF bit, which is used to determine the destination of BITop instructions. When TF bit is clear, the result of BITop instructions will be stored into R3 (fixed); if the TF bit is set, the result will be written back to memory.

Operation

TF ← 0 (BMC)
TF ← 1 (BMS)

TM GPR, #imm:8

Performs AND operation on GPR and imm:8 and sets the Zero (Z) and Negative (N) bits. No change in GPR.

Operation

Z, N flag ← GPR & #imm:8

BITop GPR.bs

Performs a bit operation on GPR and stores the result in GPR.
Since the equivalent functionality can be achieved using OR GPR, #imm:8, AND GPR, #imm:8, and XOR GPR, #imm:8, this instruction type doesn't have separate op codes.

AND SR0, #imm:8/OR SR0, #imm:8

Sets/resets bits in SR0 and stores the result back into SR0.

Operation

SR0 ← SR0 & #imm:8
SR0 ← SR0 | #imm:8

BANK #imm:2

Loads SR0[4:3] with #imm[1:0].

Operation

SR0[4:3] ← #imm[1:0]

MISCELLANEOUS INSTRUCTION

SWAP GPR, SPR

Swaps the values in GPR and SPR. SR0 and SR1 can NOT be used for this instruction.
No flag is updated, even though the destination is GPR.

Operation

temp ← SPR
SPR ← GPR
GPR ← temp

Example

```
SWAP R0, IDH        // assume IDH = 00h and R0 = 08h.
                    // after this, IDH = 08h and R0 = 00h.
```

PUSH REG

Saves REG in the stack (Pushes REG into stack).
REG = GPR, SPR

Operation

HS[sptr][7:0] ← REG and sptr ← sptr + 1

Example

```
PUSH R0             // assume R0 = 08h and sptr = 2h
                    // then HS[2][7:0] ← 08h and sptr ← 3h
```

POP REG

Pops stack into REG.
REG = GPR, SPR

Operation

REG ← HS[sptr-1][7:0] and sptr ← sptr – 1

Example

```
POP R0          // assume sptr = 3h and HS[2] = 18407h
                // R0 ← 07h and sptr ← 2h
```

POP

Pops 2 bytes from the stack and discards the popped data.

NOP

Does no work but increase PC by 1.

BREAK

Does nothing and does NOT increment PC. This instruction is for the debugger only. When this instruction is executed, the processor is locked since PC is not incremented. Therefore, this instruction should not be used under any mode other than the debug mode.

SYS #imm:8

Does nothing but increase PC by 1 and generates SYSCP[7:0] and nSYSID signals.

CLD GPR, imm:8

GPR ← (imm:8) and generates SYSCP[7:0], nCLDID, and nCLDWR signals.

CLD imm:8, GPR

(imm:8) ← GPR and generates SYSCP[7:0], nCLDID, and nCLDWR signals.

COP #imm:12

Generates SYSCP[11:0] and nCOPID signals.

LDC

Loads program memory item into register.

Operation

[TBH:TBL] ← PM[ILX:ILH:ILL] (LDC @IL)
[TBH:TBL] ← PM[ILX:ILH:ILL], ILL++ (LDC @IL+)

TBH and TBL are temporary registers to hold the transferred program memory items. These can be accessed only by LD GPR and TBL/TBH instruction.

Example

```
LD ILX, R1          // assume R1:R2:R3 has the program address to access
LD ILH, R2
LD ILL, R3
LDC @IL             // get the program data @(ILX:ILH:ILL) into TBH:TBL
```

PSEUDO INSTRUCTIONS

EI/DI

Exceptions enable and disable instruction.

Operation

SR0 ← OR SR0,#00000010b (EI)
SR0 ← AND SR0,#11111101b (DI)

Exceptions are enabled or disabled through this instruction. If there is an EI instruction, the SR0.1 is set and reset, when DI instruction.

Example

```
DI
•
•
•
EI
```

SCF/RCF

Carry flag set and reset instruction.

Operation

```
CP R0,R0        (SCF)
AND R0,R0       (RCF)
```

Carry flag is set or reset through this instruction. If there is an SCF instruction, the SR1.0 is set and reset, when RCF instruction.

Example

```
SCF
RCF
```

STOP/IDLE

MCU power saving instruction.

Operation

```
SYS #0Ah        (STOP)
SYS #05h        (IDLE)
```

The STOP instruction stops the both CPU clock and system clock and causes the microcontroller to enter STOP mode. The IDLE instruction stops the CPU clock while allowing system clock oscillation to continue.

Example

```
STOP(or IDLE)
NOP
NOP
NOP
•
•
•
```

ADC — Add with Carry

Format: ADC <op1>, <op2>
<op1>: GPR
<op2>: adr:8, GPR

Operation: <op1> ← <op1> + <op2> + C
ADC adds the values of <op1> and <op2> and carry (C) and stores the result back into <op1>

Flags: **C:** set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
. **N:** exclusive OR of V and MSB of result.

Example:

```
ADC     R0, 80h          //  If eid = 0, R0 ← R0 + DM[0080h] + C
                         //  If eid = 1, R0 ← R0 + DM[IDH:80h] + C

ADC     R0, R1           //  R0 ← R0 + R1 + C

ADD     R0, R2
ADC     R1, R3
```

In the last two instructions, assuming that register pair R1:R0 and R3:R2 are 16-bit signed or unsigned numbers. Even if the result of “ADD R0, R2” is not zero, Z flag can be set to ‘1’ if the result of “ADC R1,R3” is zero. Note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit addition, take care of the change of Z flag.

ADD — Add

Format: ADD <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> + <op2>

ADD adds the values of <op1> and <op2> and stores the result back into <op1>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
. **N:** exclusive OR of V and MSB of result.

Example: Given: IDH:IDL0 = 80FFh, eid = 1

```
ADD     R0, 80h          // R0 ← R0 + DM[8080h]

ADD     R0, #12h         // R0 ← R0 + 12h

ADD     R1, R2           // R1 ← R1 + R2

ADD     R0, @ID0 + 2     // R0 ← R0 + DM[80FFh], IDH ← 81h, IDL0 ← 01h
ADD     R0, @[ID0 – 3]   // R0 ← R0 + DM[80FCh], IDH ← 80h, IDL0 ← FCh
ADD     R0, @[ID0 + 2]!  // R0 ← R0 + DM[8101h], IDH ← 80h, IDL0 ← FFh
ADD     R0, @[ID0 – 2]!  // R0 ← R0 + DM[80FDh], IDH ← 80h, IDL0 ← FFh
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

AND — Bit-wise AND

Format: AND <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> & <op2>

AND performs bit-wise AND on the values in <op1> and <op2> and stores the result in <op1>.

Flags: **Z:** set if result is zero. Reset if not.
N: set if the MSB of result is 1. Reset if not.

Example: Given: IDH:IDL0 = 01FFh, eid = 1

```
AND     R0, 7Ah             // R0 ← R0 & DM[017Ah]

AND     R1, #40h            // R1 ← R1 & 40h

AND     R0, R1              // R0 ← R0 & R1

AND     R1, @ID0 + 3        // R1 ← R1 & DM[01FFh], IDH:IDL0 ← 0202h
AND     R1, @[ID0 – 5]      // R1 ← R1 & DM[01FAh], IDH:IDL0 ← 01FAh
AND     R1, @[ID0 + 7]!     // R1 ← R1 & DM[0206h], IDH:IDL0 ← 01FFh
AND     R1, @[ID0 – 2]!     // R1 ← R1 & DM[01FDh], IDH:IDL0 ← 01FFh
```

In the first instruction, if eid bit in SR0 is zero, register R0 has garbage value because data memory DM[0051h-007Fh] are not mapped in S3CB205/FB205. In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

AND SR0 — Bit-wise AND with SR0

Format: AND SR0, #imm:8

Operation: SR0 ← SR0 & imm:8

AND SR0 performs the bit-wise AND operation on the value of SR0 and imm:8 and stores the result in SR0.

Flags: –

Example: Given: SR0 = 11000010b

```
nIE     EQU     ~02h
nIE0    EQU     ~40h
nIE1    EQU     ~80h

        AND     SR0, #nIE | nIE0 | nIE1

        AND     SR0, #11111101b
```

In the first example, the statement “AND SR0, #nIE|nIE0|nIE1” clear all of bits of the global interrupt, interrupt 0 and interrupt 1. On the contrary, cleared bits can be set to ‘1’ by instruction “OR SR0, #imm:8”. Refer to instruction OR SR0 for more detailed explanation about enabling bit.

In the second example, the statement “AND SR0, #11111101b” is equal to instruction DI, which is disabling interrupt globally.

BANK — GPR Bank selection

Format: BANK #imm:2

Operation: SR0[4:3] ← imm:2

Flags: –

NOTE: For explanation of the CalmRISC banked register file and its usage, please refer to chapter 3.

Example:

```
BANK    #1          // Select register bank 1
LD      R0, #11h    // Bank1's R0 ← 11h

BANK    #2          // Select register bank 2
LD      R1, #22h    // Bank2's R1 ← 22h
```

BITC — Bit Complement

Format: BITC adr:8.bs

bs: 3-digit bit specifier

Operation: R3 ← ((adr:8) ^ (2**bs)) if (TF == 0)

(adr:8) ← ((adr:8) ^ (2**bs)) if (TF == 1)

BITC complements the specified bit of a value read from memory and stores the result in R3 or back into memory, depending on the value of TF. TF is set or clear by BMS/BMC instruction.

Flags: **Z:** set if result is zero. Reset if not.

NOTE: Since the destination register R3 is fixed, it is not specified explicitly.

Example: Given: IDH = 01, DM[0180h] = FFh, eid = 1

```
BMC                 // TF ← 0
BITC    80h.0       // R3 ← FEh, DM[0180h] = FFh

BMS                 // TF ← 1
BITC    80h.1       // DM[0180h] ← FDh
```

BITR — Bit Reset

Format: BITR adr:8.bs

bs: 3-digit bit specifier

Operation: R3 ← ((adr:8) & ($(11111111)_2$ - (2**bs))) if (TF == 0)

(adr:8) ← ((adr:8) & ($(11111111)_2$ - (2**bs))) if (TF == 1)

BITR resets the specified bit of a value read from memory and stores the result in R3 or back into memory, depending on the value of TF. TF is set or clear by BMS/BMC instruction.

Flags: **Z:** set if result is zero. Reset if not.

NOTE: Since the destination register R3 is fixed, it is not specified explicitly.

Example: Given: IDH = 01, DM[0180h] = FFh, eid = 1

```
BMC                 // TF ← 0
BITR    80h.1       // R3 ← FDh, DM[0180h] = FFh

BMS                 // TF ← 1
BITR    80h.2       // DM[0180h] ← FBh
```

BITS — Bit Set

Format: BITS adr:8.bs

bs: 3-digit bit specifier.

Operation: R3 ← ((adr:8) | (2**bs)) if (TF == 0)

(adr:8) ← ((adr:8) | (2**bs)) if (TF == 1)

BITS sets the specified bit of a value read from memory and stores the result in R3 or back into memory, depending on the value of TF. TF is set or clear by BMS/BMC instruction.

Flags: **Z:** set if result is zero. Reset if not.

NOTE: Since the destination register R3 is fixed, it is not specified explicitly.

Example: Given: IDH = 01, DM[0180h] = F0h, eid = 1

```
BMC                 // TF ← 0
BITS    80h.1       // R3 ← 0F2h, DM[0180h] = F0h

BMS                 // TF ← 1
BITS    80h.2       // DM[0180h] ← F4h
```

BITT — Bit Test

Format: BITT adr:8.bs

bs: 3-digit bit specifier.

Operation: Z ← ~((adr:8) & (2**bs))

BITT tests the specified bit of a value read from memory.

Flags: **Z:** set if result is zero. Reset if not.

Example: Given: DM[0080h] = F7h, eid = 0

```
        BITT    80h.3           // Z flag is set to '1'
        JR      Z, %1           // Jump to label %1 because condition is true.
        •
        •
        •
%1      BITS    80h.3
        NOP
        •
        •
        •
```

BMC/BMS – TF bit clear/set

Format: BMS/BMC

Operation: BMC/BMS clears (sets) the TF bit.

TF ← 0 if BMC

TF ← 1 if BMS

TF is a single bit flag which determines the destination of bit operations, such as BITC, BITR, and BITS.

Flags: –

NOTE: BMC/BMS are the only instructions that modify the content of the TF bit.

Example:

```
BMS                     // TF ← 1
BITS    81h.1

BMC                     // TF ← 0
BITR    81h.2
LD      R0, R3
```

CALL — Conditional Subroutine Call (Pseudo Instruction)

Format: CALL cc:4, imm:20
CALL imm:12

Operation: If CALLS can access the target address and there is no conditional code (cc:4), CALL command is assembled to CALLS (1-word instruction) in linking time, else the CALL is assembled to LCALL (2-word instruction).

Example:

```
        CALL    C, Wait         // HS[sptr][15:0] ← current PC + 2, sptr ← sptr + 2
        •                       // 2-word instruction
        •
        •
        CALL    0088h           // HS[sptr][15:0] ← current PC + 1, sptr ← sptr + 2
        •                       // 1-word instruction
        •
        •
Wait:   NOP                     // Address at 0088h
        NOP
        NOP
        NOP
        NOP
        RET
```

CALLS — Call Subroutine

Format: CALLS imm:12

Operation: HS[sptr][15:0] ← current PC + 1, sptr ← sptr + 2 if the program size is less than 64K word.

HS[sptr][19:0] ← current PC + 1, sptr ← sptr + 2 if the program size is equal to or over 64K word.

PC[11:0] ← imm:12
CALLS unconditionally calls a subroutine residing at the address specified by imm:12.

Flags: –

Example:

```
        CALLS   Wait
        •
        •
        •
Wait:   NOP
        NOP
        NOP
        RET
```

Because this is a 1-word instruction, the saved returning address on stack is (PC + 1).

CLD — Load into Coprocessor

Format: CLD imm:8, <op>

<op>: GPR

Operation: (imm:8) ← <op>

CLD loads the value of <op> into (imm:8), where imm:8 is used to access the external coprocessor's address space.

Flags: –

Example:

```
AH      EQU   00h
AL      EQU   01h
BH      EQU   02h
BL      EQU   03h
        •
        •
        •
        CLD   AH, R0        // A[15:8] ← R0
        CLD   AL, R1        // A[7:0] ← R1

        CLD   BH, R2        // B[15:8] ← R2
        CLD   BL, R3        // B[7:0] ← R3
```

The registers A[15:0] and B[15:0] are Arithmetic Unit (AU) registers of MAC816.
Above instructions generate SYSCP[7:0], nCLDID and CLDWR signals to access MAC816.

CLD — Load from Coprocessor

Format: CLD <op>, imm:8

<op>: GPR

Operation: <op> ← (imm:8)

CLD loads a value from the coprocessor, whose address is specified by imm:8.

Flags: **Z:** set if the loaded value in <op1> is zero. Reset if not.
N: set if the MSB of the loaded value in <op1> is 1. Reset if not.

Example:

```
AH      EQU   00h
AL      EQU   01h
BH      EQU   02h
BL      EQU   03h
        •
        •
        •
        CLD   R0, AH        // R0 ← A[15:8]
        CLD   R1, AL        // R1 ← A[7:0]

        CLD   R2, BH        // R2 ← B[15:8]
        CLD   R3, BL        // R3 ← B[7:0]
```

The registers A[15:0] and B[15:0] are Arithmetic Unit (AU) registers of MAC816.
Above instructions generate SYSCP[7:0], nCLDID and CLDWR signals to access MAC816.

COM — 1's or Bit-wise Complement

Format: COM <op>

<op>: GPR

Operation: <op> ← ~<op>

COM takes the bit-wise complement operation on <op> and stores the result in <op>.

Flags: **Z:** set if result is zero. Reset if not.
N: set if the MSB of result is 1. Reset if not.

Example: Given: R1 = 5Ah

```
COM     R1                  // R1 ← A5h, N flag is set to '1'
```

COM2 — 2's Complement

Format: COM2 <op>

<op>: GPR

Operation: <op> ← ~<op> + 1

COM2 computes the 2's complement of <op> and stores the result in <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative.

Example: Given: R0 = 00h, R1 = 5Ah

```
COM2    R0          // R0 ← 00h, Z and C flags are set to '1'.

COM2    R1          // R1 ← A6h, N flag is set to '1'.
```

COMC — Bit-wise Complement with Carry

Format: COMC <op>

<op>: GPR

Operation: <op> ← ~<op> + C

COMC takes the bit-wise complement of <op>, adds carry and stores the result in <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: If register pair R1:R0 is a 16-bit number, then the 2's complement of R1:R0 can be obtained by COM2 and COMC as following.

```
COM2    R0
COMC    R1
```

Note that Z flag do not exactly reflect result of 16-bit operation. For example, if 16-bit register pair R1: R0 has value of FF01h, then 2's complement of R1: R0 is made of 00FFh by COM2 and COMC. At this time, by instruction COMC, zero (Z) flag is set to '1' as if the result of 2's complement for 16-bit number is zero. Therefore when programming 16-bit comparison, take care of the change of Z flag.

COP — Coprocessor

Format: COP #imm:12

Operation: COP passes imm:12 to the coprocessor by generating SYSCP[11:0] and nCOPID signals.

Flags: –

Example:

```
COP     #0D01h          // generate 1 word instruction code(FD01h)
COP     #0234h          // generate 1 word instruction code(F234h)
```

The above two instructions are equal to statement “ELD A, #1234h” for MAC816 operation. The microcode of MAC instruction “ELD A, #1234h” is “FD01F234”, 2-word instruction. In this, code ‘F’ indicates ‘COP’ instruction.

CP — Compare

Format: CP <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> + ~<op2> + 1

CP compares the values of <op1> and <op2> by subtracting <op2> from <op1>. Contents of <op1> and <op2> are not changed.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero (i.e., <op1> and <op2> are same). Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: R0 = 73h, R1 = A5h, IDH:IDL0 = 0123h, DM[0123h] = A5, eid = 1

```
CP      R0, 80h          // C flag is set to '1'

CP      R0, #73h         // Z and C flags are set to '1'

CP      R0, R1           // V flag is set to '1'

CP      R1, @ID0         // Z and C flags are set to '1'
CP      R1, @[ID0 – 5]
CP      R2, @[ID0 + 7]!
CP      R2, @[ID0 – 2]!
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

CPC — Compare with Carry

Format: CPC <op1>, <op2>

<op1>: GPR
<op2>: adr:8, GPR

Operation: <op1> ← <op1> + ~<op2> + C

CPC compares <op1> and <op2> by subtracting <op2> from <op1>. Unlike CP, however, CPC adds (C - 1) to the result. Contents of <op1> and <op2> are not changed.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: If register pair R1:R0 and R3:R2 are 16-bit signed or unsigned numbers, then use CP and CPC to compare two 16-bit numbers as follows.

```
CP      R0, R1
CPC     R2, R3
```

Because CPC considers C when comparing <op1> and <op2>, CP and CPC can be used in pair to compare 16-bit operands. But note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit comparison, take care of the change of Z flag.

DEC — Decrement

Format: DEC <op>

<op>: GPR

Operation: <op> ← <op> + 0FFh

DEC decrease the value in <op> by adding 0FFh to <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: R0 = 80h, R1 = 00h

```
DEC     R0          // R0 ← 7Fh, C, V and N flags are set to '1'

DEC     R1          // R1 ← FFh, N flags is set to '1'
```

DECC — Decrement with Carry

Format: DECC <op>

<op>: GPR

Operation: <op> ← <op> + 0FFh + C

DECC decrease the value in <op> when carry is not set. When there is a carry, there is no change in the value of <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: If register pair R1:R0 is 16-bit signed or unsigned number, then use DEC and DECC to decrement 16-bit number as follows.

```
DEC     R0
DECC    R1
```

Note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit decrement, take care of the change of Z flag.

DI — Disable Interrupt (Pseudo Instruction)

Format: DI

Operation: Disables interrupt globally. It is same as “AND SR0, #0FDh” .
DI instruction sets bit1 (ie: global interrupt enable) of SR0 register to “0”

Flags: –

Example: Given: SR0 = 03h

```
DI                                  // SR0 ← SR0 & 11111101b
```

DI instruction clears SR0[1] to ‘0’, disabling interrupt processing.

EI — Enable Interrupt (Pseudo Instruction)

Format: EI

Operation: Enables interrupt globally. It is same as “OR SR0, #02h” .
EI instruction sets the bit1 (ie: global interrupt enable) of SR0 register to “1”

Flags: –

Example: Given: SR0 = 01h

```
EI                                  // SR0 ← SR0 | 00000010b
```

The statement “EI” sets the SR0[1] to ‘1’, enabling all interrupts.

IDLE — Idle Operation (Pseudo Instruction)

Format: IDLE

Operation: The IDLE instruction stops the CPU clock while allowing system clock oscillation to continue. Idle mode can be released by an interrupt or reset operation.
The IDLE instruction is a pseudo instruction. It is assembled as “SYS #05H”, and this generates the SYSCP[7-0] signals. Then these signals are decoded and the decoded signals execute the idle operation.

Flags: –

NOTE: The next instruction of IDLE instruction is executed, so please use the NOP instruction after the IDLE instruction.

Example:

```
IDLE
NOP
NOP
NOP
•
•
•
```

The IDLE instruction stops the CPU clock but not the system clock.

INC — Increment

Format: INC <op>

<op>: GPR

Operation: <op> ← <op> + 1

INC increase the value in <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: R0 = 7Fh, R1 = FFh

```
INC     R0          // R0 ← 80h, V flag is set to '1'
INC     R1          // R1 ← 00h, Z and C flags are set to '1'
```

INCC — Increment with Carry

Format: INCC <op>

<op>: GPR

Operation: <op> ← <op> + C

INCC increase the value of <op> only if there is carry. When there is no carry, the value of <op> is not changed.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: exclusive OR of V and MSB of result.

Example: If register pair R1:R0 is 16-bit signed or unsigned number, then use INC and INCC to increment 16-bit number as following.

```
INC     R0
INCC    R1
```

Assume R1:R0 is 0010h, statement “INC R0” increase R0 by one without carry and statement “INCC R1” set zero (Z) flag to ‘1’ as if the result of 16-bit increment is zero. Note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit increment, take care of the change of Z flag.

IRET — Return from Interrupt Handling

Format: IRET

Operation: PC ← HS[sptr - 2], sptr ← sptr - 2

IRET pops the return address (after interrupt handling) from the hardware stack and assigns it to PC. The ie (i.e., SR0[1]) bit is set to allow further interrupt generation.

Flags: –

NOTE: The program size (indicated by the nP64KW signal) determines which portion of PC is updated. When the program size is less than 64K word, only the lower 16 bits of PC are updated (i.e., PC[15:0] ← HS[sptr – 2]**)**.
When the program size is 64K word or more, the action taken is PC[19:0] ← HS[sptr - 2].

Example:

```
SF_EXCEP:   NOP                 // Stack full exception service routine
            •
            •
            •
            IRET
```

JNZD — Jump Not Zero with Delay slot

Format: JNZD <op>, imm:8

<op>: GPR (bank 3's GPR only)

imm:8 is an signed number

Operation: PC ← PC[delay slot] - 2's complement of imm:8

<op> ← <op> - 1

JNZD performs a backward PC-relative jump if <op> evaluates to be non-zero. Furthermore, JNZD decrease the value of <op>. The instruction immediately following JNZD (i.e., in delay slot) is always executed, and this instruction must be 1 cycle instruction.

Flags: –

NOTE: Typically, the delay slot will be filled with an instruction from the loop body. It is noted, however, that the chosen instruction should be "dead" outside the loop for it executes even when the loop is exited (i.e., JNZD is not taken).

Example: Given: IDH = 03h, eid = 1

```
        BANK    #3
        LD      R0, #0FFh           // R0 is used to loop counter
        LD      R1, #0
%1      LD      IDL0, R0
        JNZD    R0, %B1             // If R0 of bank3 is not zero, jump to %1.
        LD      @ID0, R1            // Clear register pointed by ID0
        •
        •
        •
```

This example can be used for RAM clear routine. The last instruction is executed even if the loop is exited.

JP — Conditional Jump (Pseudo Instruction)

Format: JP cc:4 imm:20
JP cc:4 imm:9

Operation: If JR can access the target address, JP command is assembled to JR (1 word instruction) in linking time, else the JP is assembled to LJP (2 word instruction) instruction.
There are 16 different conditions that can be used, as described in table 7-6.

Example:

```
%1      LD      R0, #10h        // Assume address of label %1 is 020Dh
        •
        •
        •
        JP      Z, %B1          // Address at 0264h

        JP      C, %F2          // Address at 0265h
        •
        •
        •
%2      LD      R1, #20h        // Assume address of label %2 is 089Ch
        •
        •
        •
```

In the above example, the statement “JP Z, %B1” is assembled to JR instruction. Assuming that current PC is 0264h and condition is true, next PC is made by PC[11:0] ← PC[11:0] + offset, offset value is “64h + A9h” without carry. ‘A9’ means 2’s complement of offset value to jump backward. Therefore next PC is 020Dh. On the other hand, statement “JP C, %F2” is assembled to LJP instruction because offset address exceeds the range of imm:9.

JR — Conditional Jump Relative

Format: JR cc:4 imm:9

cc:4: 4-bit condition code

Operation: PC[11:0] ← PC[11:0] + imm:9 if condition is true. imm:9 is a signed number, which is sign-extended to 12 bits when added to PC.
There are 16 different conditions that can be used, as described in table 7-6.

Flags: –

NOTE: Unlike LJP, the target address of JR is PC-relative. In the case of JR, imm:9 is added to PC to compute the actual jump address, while LJP directly jumps to imm:20, the target.

Example:

```
        JR      Z, %1           // Assume current PC = 1000h
        •
        •
        •
%1      LD      R0, R1          // Address at 10A5h
        •
        •
        •
```

After the first instruction is executed, next PC has become 10A5h if Z flag bit is set to '1'. The range of the relative address is from +255 to –256 because imm:9 is signed number.

LCALL — Conditional Subroutine Call

Format: LCALL cc:4, imm:20

Operation: HS[sptr][15:0] ← current PC + 2, sptr ← sptr + 2, PC[15:0] ← imm[15:0] if the condition holds and the program size is less than 64K word.

HS[sptr][19:0] ← current PC + 2, sptr ← sptr + 2, PC[19:0] ← imm:20 if the condition holds and the program size is equal to or over 64K word.

PC[11:0] ← PC[11:0] + 2 otherwise.
LCALL instruction is used to call a subroutine whose starting address is specified by imm:20.

Flags: –

Example:

```
LCALL    L1

LCALL    C, L2
```

Label L1 and L2 can be allocated to the same or other section. Because this is a 2-word instruction, the saved returning address on stack is (PC + 2).

LD adr:8 — Load into Memory

Format: LD adr:8, <op>

<op>: GPR

Operation: DM[00h:adr:8] ← <op> if eid = 0
DM[IDH:adr:8] ← <op> if eid = 1

LD adr:8 loads the value of <op> into a memory location. The memory location is determined by the eid bit and adr:8.

Flags: –

Example: Given: IDH = 01h

```
LD      80h, R0
```

If eid bit of SR0 is zero, the statement “LD 80h, R0” load value of R0 into DM[0080h], else eid bit was set to ‘1’, the statement “LD 80h, R0” load value of R0 into DM[0180h]

LD @idm — Load into Memory Indexed

Format: LD @idm, <op>

<op>: GPR

Operation: (@idm) ← <op>

LD @idm loads the value of <op> into the memory location determined by @idm. Details of the @idm format and how the actual address is calculated can be found in chapter 2.

Flags: –

Example: Given R0 = 5Ah, IDH:IDL0 = 8023h, eid = 1

```
LD      @ID0, R0          // DM[8023h] ← 5Ah
LD      @ID0 + 3, R0      // DM[8023h] ← 5Ah, IDL0 ← 26h
LD      @[ID0-5], R0      // DM[801Eh] ← 5Ah, IDL0 ← 1Eh
LD      @[ID0+4]!, R0     // DM[8027h] ← 5Ah, IDL0 ← 23h
LD      @[ID0-2]!, R0     // DM[8021h] ← 5Ah, IDL0 ← 23h
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

LD — Load Register

Format: LD <op1>, <op2>

<op1>: GPR
<op2>: GPR, SPR, adr:8, @idm, #imm:8

Operation: <op1> ← <op2>

LD loads a value specified by <op2> into the register designated by <op1>.

Flags: **Z:** set if result is zero. Reset if not.
N: exclusive OR of V and MSB of result.

Example: Given: R0 = 5Ah, R1 = AAh, IDH:IDL0 = 8023h, eid = 1

```
LD      R0, R1              // R0 ← AAh

LD      R1, IDH             // R1 ← 80h

LD      R2, 80h             // R2 ← DM[8080h]

LD      R0, #11h            // R0 ← 11h

LD      R0, @ID0+1          // R0 ← DM[8023h], IDL0 ← 24h
LD      R1, @[ID0-2]        // R1 ← DM[8021h], IDL0 ← 21h
LD      R2, @[ID0+3]!       // R2 ← DM[8026h], IDL0 ← 23h
LD      R3, @[ID0-5]!       // R3 ← DM[801Eh], IDL0 ← 23h
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

LD — Load GPR:bankd, GPR:banks

Format: LD <op1>, <op2>

<op1>: GPR: bankd
<op2>: GPR: banks

Operation: <op1> ← <op2>

LD loads a value of a register in a specified bank (banks) into another register in a specified bank (bankd).

Flags: **Z:** set if result is zero. Reset if not.
N: exclusive OR of V and MSB of result.

Example:

```
LD      R2:1, R0:3          // Bank1's R2 ← bank3's R0

LD      R0:0, R0:2          // Bank0's R0 ← bank2's R0
```

LD — Load GPR, TBH/TBL

Format: LD <op1>, <op2>

<op1>: GPR
<op2>: TBH/TBL

Operation: <op1> ← <op2>

LD loads a value specified by <op2> into the register designated by <op1>.

Flags: **Z:** set if result is zero. Reset if not.
N: exclusive OR of V and MSB of result.

Example: Given: register pair R1:R0 is 16-bit unsigned data.

```
LDC     @IL             // TBH:TBL ← PM[ILX:ILH:ILL]
LD      R1, TBH         // R1 ← TBH
LD      R0, TBL         // R0 ← TBL
```

LD — Load TBH/TBL, GPR

Format: LD <op1>, <op2>

<op1>: TBH/TBL
<op2>: GPR

Operation: <op1> ← <op2>

LD loads a value specified by <op2> into the register designated by <op1>.

Flags: –

Example: Given: register pair R1:R0 is 16-bit unsigned data.

```
LD      TBH, R1         // TBH ← R1
LD      TBL, R0         // TBL ← R0
```

LD SPR — Load SPR

Format: LD <op1>, <op2>

<op1>: SPR
<op2>: GPR

Operation: <op1> ← <op2>

LD SPR loads the value of a GPR into an SPR.
Refer to Table 3-1 for more detailed explanation about kind of SPR.

Flags: –

Example: Given: register pair R1:R0 = 1020h

```
LD      ILH, R1          // ILH ← 10h
LD      ILL, R0          // ILL ← 20h
```

LD SPR0 — Load SPR0 Immediate

Format: LD SPR0, #imm:8

Operation: SPR0 ← imm:8

LD SPR0 loads an 8-bit immediate value into SPR0.

Flags: –

Example: Given: eid = 1, idb = 0 (index register bank 0 selection)

```
LD      IDH, #80h          // IDH point to page 80h
LD      IDL1, #44h
LD      IDL0, #55h
LD      SR0, #02h
```

The last instruction set ie (global interrupt enable) bit to '1'.
Special register group 1 (SPR1) registers are not supported in this addressing mode.

LDC — Load Code

Format: LDC <op1>

<op1>: @IL, @IL+

Operation: TBH:TBL ← PM[ILX:ILH:ILL]

ILL ← ILL + 1 (@IL+ only)

LDC loads a data item from program memory and stores it in the TBH:TBL register pair.

@IL+ increase the value of ILL, efficiently implementing table lookup operations.

Flags: –

Example:

```
LD      ILX, R1
LD      ILH, R2
LD      ILL, R3
LDC     @IL             // Loads value of PM[ILX:ILH:ILL] into TBH:TBL

LD      R1, TBH         // Move data in TBH:TBL to GPRs for further processing
LD      R0, TBL
```

The statement “LDC @IL” do not increase, but if you use statement “LDC @IL+”, ILL register is increased by one after instruction execution.

LJP — Conditional Jump

Format: LJP cc:4, imm:20

cc:4: 4-bit condition code

Operation: PC[15:0] ← imm[15:0] if condition is true and the program size is less than 64K word. If the program is equal to or larger than 64K word, PC[19:0] ← imm[19:0] as long as the condition is true. There are 16 different conditions that can be used, as described in table 7-6.

Flags: –

NOTE: LJP cc:4 imm:20 is a 2-word instruction whose immediate field directly specifies the target address of the jump.

Example:

```
        LJP     C, %1           // Assume current PC = 0812h
        •
        •
        •
%1      LD      R0, R1          // Address at 10A5h
        •
        •
        •
```

After the first instruction is executed, LJP directly jumps to address 10A5h if condition is true.

LLNK — Linked Subroutine Call Conditional

Format: LLNK cc:4, imm:20

cc:4: 4-bit condition code

Operation: If condition is true, IL[19:0] ← {PC[19:12], PC[11:0] + 2}.

Further, when the program is equal to or larger than 64K word, PC[19:0] ← imm[19:0] as long as the condition is true. If the program is smaller than 64K word, PC[15:0] ← imm[15:0].
There are 16 different conditions that can be used, as described in table 7-6.

Flags: –

NOTE: LLNK is used to conditionally to call a subroutine with the return address saved in the link register (IL) without stack operation. This is a 2-word instruction.

Example:

```
        LLNK    Z, %1           // Address at 005Ch, ILX:ILH:ILL ← 00:00:5Eh
        NOP                     // Address at 005Eh
        •
        •
        •
%1      LD      R0, R1
        •
        •
        •
        LRET
```

LNK — Linked Subroutine Call (Pseudo Instruction)

Format: LNK cc:4, imm:20
LNK imm:12

Operation: If LNKS can access the target address and there is no conditional code (cc:4), LNK command is assembled to LNKS (1 word instruction) in linking time, else the LNK is assembled to LLNK (2 word instruction).

Example:

```
        LNK     Z, Link1        // Equal to “LLNK Z, Link1”
        LNK     Link2           // Equal to “LNKS Link2”
        NOP
        •
        •
        •
Link2:  NOP
        •
        •
        •
        LRET

Subroutines     section CODE, ABS 0A00h
        Subroutines
Link1:  NOP
        •
        •
        •
        LRET
```

LNKS — Linked Subroutine Call

Format: LNKS imm:12

Operation: IL[19:0] ← {PC[19:12], PC[11:0] + 1} and PC[11:0] ← imm:12
LNKS saves the current PC in the link register and jumps to the address specified by imm:12.

Flags: –

NOTE: LNKS is used to call a subroutine with the return address saved in the link register (IL) without stack operation.

Example:

```
        LNKS    Link1           // Address at 005Ch, ILX:ILH:ILL ← 00:00:5Dh
        NOP                     // Address at 005Dh
        •
        •
        •
Link1:  NOP
        •
        •
        •
        LRET
```

LRET — Return from Linked Subroutine Call

Format: LRET

Operation: PC ← IL[19:0]
LRET returns from a subroutine by assigning the saved return address in IL to PC.

Flags: –

Example:

```
        LNK     Link1
Link1:  NOP
        •
        •
        •
        LRET                        ;  PC[19:0] ← ILX:ILH:ILL
```

NOP — No Operation

Format: NOP

Operation: No operation.

When the instruction NOP is executed in a program, no operation occurs. Instead, the instruction time is delayed by approximately one machine cycle per each NOP instruction encountered.

Flags: –

Example:

```
NOP
```

OR — Bit-wise OR

Format: OR <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> | <op2>
OR performs the bit-wise OR operation on <op1> and <op2> and stores the result in <op1>.

Flags: **Z:** set if result is zero. Reset if not.
N: exclusive OR of V and MSB of result.

Example: Given: IDH:IDL0 = 031Eh, eid = 1

```
OR      R0, 80h          // R0 ← R0 | DM[0380h]

OR      R1, #40h         // Mask bit6 of R1

OR      R1, R0           // R1 ← R1 | R0

OR      R0, @ID0         // R0 ← R0 | DM[031Eh], IDL0 ← 1Eh
OR      R1, @[ID0-1]     // R1 ← R1 | DM[031Dh], IDL0 ← 1Dh
OR      R2, @[ID0+1]!    // R2 ← R2 | DM[031Fh], IDL0 ← 1Eh
OR      R3, @[ID0-1]!    // R3 ← R3 | DM[031Dh], IDL0 ← 1Eh
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

OR SR0 — Bit-wise OR with SR0

Format: OR SR0, #imm:8

Operation: SR0 ← SR0 | imm:8

OR SR0 performs the bit-wise OR operation on SR0 and imm:8 and stores the result in SR0.

Flags: –

Example: Given: SR0 = 00000000b

```
EID     EQU     01h
IE      EQU     02h
IDB1    EQU     04h
IE0     EQU     40h
IE1     EQU     80h

        OR      SR0, #IE | IE0 | IE1

        OR      SR0, #00000010b
```

In the first example, the statement “OR SR0, #EID|IE|IE0” set global interrupt(ie), interrupt 0(ie0) and interrupt 1(ie1) to ‘1’ in SR0. On the contrary, enabled bits can be cleared with instruction “AND SR0, #imm:8”. Refer to instruction AND SR0 for more detailed explanation about disabling bit.

In the second example, the statement “OR SR0, #00000010b” is equal to instruction EI, which is enabling interrupt globally.

POP — POP

Format: POP

Operation: sptr ← sptr – 2

POP decrease sptr by 2. The top two bytes of the hardware stack are therefore invalidated.

Flags: –

Example: Given: sptr[5:0] = 001010b

POP

This POP instruction decrease sptr[5:0] by 2. Therefore sptr[5:0] is 001000b.

POP — POP to Register

Format: POP <op>

<op>: GPR, SPR

Operation: <op> ← HS[sptr - 1], sptr ← sptr - 1

POP copies the value on top of the stack to <op> and decrease sptr by 1.

Flags: **Z:** set if the value copied to <op> is zero. Reset if not.
N: set if the value copied to <op> is negative. Reset if not.
When <op> is SPR, no flags are affected, including Z and N.

Example:

```
POP     R0      // R0 ← HS[sptr-1], sptr ← sptr-1

POP     IDH     // IDH ← HS[sptr-1], sptr ← sptr-1
```

In the first instruction, value of HS[sptr-1] is loaded to R0 and the second instruction “POP IDH” load value of HS[sptr-1] to register IDH. Refer to chapter 5 for more detailed explanation about POP operations for hardware stack.

PUSH — Push Register

Format: PUSH <op>

<op>: GPR, SPR

Operation: HS[sptr] ← <op>, sptr ← sptr + 1

PUSH stores the value of <op> on top of the stack and increase sptr by 1.

Flags: –

Example:

```
PUSH    R0          // HS[sptr] ← R0, sptr ← sptr + 1

PUSH    IDH         // HS[sptr] ← IDH, sptr ← sptr + 1
```

In the first instruction, value of register R0 is loaded to HS[sptr-1] and the second instruction “PUSH IDH” load value of register IDH to HS[sptr-1]. Current HS pointed by stack point sptr[5:0] be emptied. Refer to chapter 5 for more detailed explanation about PUSH operations for hardware stack.

RET — Return from Subroutine

Format: RET

Operation: PC ← HS[sptr - 2], sptr ← sptr – 2

RET pops an address on the hardware stack into PC so that control returns to the subroutine call site.

Flags: –

Example: Given: sptr[5:0] = 001010b

```
        CALLS   Wait                    // Address at 00120h
        •
        •
        •
Wait:   NOP                             // Address at 01000h
        NOP
        NOP
        NOP
        NOP
        RET
```

After the first instruction CALLS execution, “PC+1”, 0121h is loaded to HS[5] and hardware stack pointer sptr[5:0] have 001100b and next PC became 01000h. The instruction RET pops value 0121h on the hardware stack HS[sptr-2] and load to PC then stack pointer sptr[[5:0] became 001010b.

RL — Rotate Left

Format: RL <op>

<op>: GPR

Operation: C ← <op>[7], <op> ← {<op>[6:0], <op>[7]}

RL rotates the value of <op> to the left and stores the result back into <op>. The original MSB of <op> is copied into carry (C).

Flags:
C: set if the MSB of <op> (before rotating) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after rotating) is 1. Reset if not.

Example: Given: R0 = 01001010b, R1 = 10100101b

```
RL      R0          // N flag is set to '1', R0 ← 10010100b
RL      R1          // C flag is set to '1', R1 ← 01001011b
```

RLC — Rotate Left with Carry

Format: RLC <op>

<op>: GPR

Operation: C ← <op>[7], <op> ← {<op>[6:0], C}

RLC rotates the value of <op> to the left and stores the result back into <op>.
The original MSB of <op> is copied into carry (C), and the original C bit is copied into <op>[0].

Flags: **C:** set if the MSB of <op> (before rotating) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after rotating) is 1. Reset if not.

Example: Given: R2 = A5h, if C = 0

```
RLC     R2                  // R2 ← 4Ah, C flag is set to '1'

RL      R0
RLC     R1
```

In the second example, assuming that register pair R1:R0 is 16-bit number, then RL and RLC are used for 16-bit rotate left operation. But note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit decrement, take care of the change of Z flag.

RR — Rotate Right

Format: RR <op>

<op>: GPR

Operation: C ← <op>[0], <op> ← {<op>[0], <op>[7:1]}

RR rotates the value of <op> to the right and stores the result back into <op>. The original LSB of <op> is copied into carry (C).

Flags:
C: set if the LSB of <op> (before rotating) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after rotating) is 1. Reset if not.

Example: Given: R0 = 01011010b, R1 = 10100101b

```
RR      R0          // No change of flag, R0 ← 00101101b
RR      R1          // C and N flags are set to '1', R1 ← 11010010b
```

RRC — Rotate Right with Carry

Format: RRC <op>

<op>: GPR

Operation: C ← <op>[0], <op> ← {C, <op>[7:1]}

RRC rotates the value of <op> to the right and stores the result back into <op>. The original LSB of <op> is copied into carry (C), and C is copied to the MSB.

Flags:
C: set if the LSB of <op> (before rotating) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after rotating) is 1. Reset if not.

Example: Given: R2 = A5h, if C = 0

```
RRC     R2                  // R2 ← 52h, C flag is set to '1'

RR      R0
RRC     R1
```

In the second example, assuming that register pair R1:R0 is 16-bit number, then RR and RRC are used for 16-bit rotate right operation. But note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit decrement, take care of the change of Z flag.

SBC — Subtract with Carry

Format: SBC <op1>, <op2>

<op1>: GPR
<op2>: adr:8, GPR

Operation: <op1> ← <op1> + ~<op2> + C

SBC computes (<op1> - <op2>) when there is carry and (<op1> - <op2> - 1) when there is no carry.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated.
N: set if result is negative. Reset if not.

Example:

```
SBC     R0, 80h         //  If eid = 0, R0 ← R0 + ~DM[0080h] + C
                        //  If eid = 1, R0 ← R0 + ~DM[IDH:80h] + C

SBC     R0, R1          //  R0 ← R0 + ~R1 + C

SUB     R0, R2
SBC     R1, R3
```

In the last two instructions, assuming that register pair R1:R0 and R3:R2 are 16-bit signed or unsigned numbers. Even if the result of “ADD R0, R2” is not zero, zero (Z) flag can be set to ‘1’ if the result of “SBC R1,R3” is zero. Note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit addition, take care of the change of Z flag.

SL — Shift Left

Format: SL <op>

<op>: GPR

Operation: C ← <op>[7], <op> ← {<op>[6:0], 0}

SL shifts <op> to the left by 1 bit. The MSB of the original <op> is copied into carry (C).

Flags:
C: set if the MSB of <op> (before shifting) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after shifting) is 1. Reset if not.

Example: Given: R0 = 01001010b, R1 = 10100101b

```
SL      R0          // N flag is set to '1', R0 ← 10010100b
SL      R1          // C flag is set to '1', R1 ← 01001010b
```

SLA — Shift Left Arithmetic

Format: SLA <op>

<op>: GPR

Operation: C ← <op>[7], <op> ← {<op>[6:0], 0}

SLA shifts <op> to the left by 1 bit. The MSB of the original <op> is copied into carry (C).

Flags:
C: set if the MSB of <op> (before shifting) is 1. Reset if not.
Z: set if result is zero. Reset if not.
V: set if the MSB of the result is different from C. Reset if not.
N: set if the MSB of <op> (after shifting) is 1. Reset if not.

Example: Given: R0 = AAh

```
SLA     R0                  // C, V, N flags are set to '1', R0 ← 54h
```

SR — Shift Right

Format: SR <op>

<op>: GPR

Operation: C ← <op>[0], <op> ← {0, <op>[7:1]}

SR shifts <op> to the right by 1 bit. The LSB of the original <op> (i.e., <op>[0]) is copied into carry (C).

Flags: **C:** set if the LSB of <op> (before shifting) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after shifting) is 1. Reset if not.

Example: Given: R0 = 01011010b, R1 = 10100101b

```
SR      R0          // No change of flags, R0 ← 00101101b
SR      R1          // C flag is set to '1', R1 ← 01010010b
```

SRA — Shift Right Arithmetic

Format: SRA <op>

<op>: GPR

Operation: C ← <op>[0], <op> ← {<op>[7], <op>[7:1]}

SRA shifts <op> to the right by 1 bit while keeping the sign of <op>. The LSB of the original <op> (i.e., <op>[0]) is copied into carry (C).

Flags:
C: set if the LSB of <op> (before shifting) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after shifting) is 1. Reset if not.

NOTE: SRA keeps the sign bit or the MSB (<op>[7]) in its original position. If SRA is executed 'N' times, N significant bits will be set, followed by the shifted bits.

Example: Given: R0 = 10100101b

```
SRA     R0      // C, N flags are set to '1', R0 ← 11010010b
SRA     R0      // N flag is set to '1', R0 ← 11101001b
SRA     R0      // C, N flags are set to '1', R0 ← 11110100b
SRA     R0      // N flags are set to '1', R0 ← 11111010b
```

STOP — Stop Operation (pseudo instruction)

Format: STOP

Operation: The STOP instruction stops the both the CPU clock and system clock and causes the microcontroller to enter the STOP mode. In the STOP mode, the contents of the on-chip CPU registers, peripheral registers, and I/O port control and data register are retained. A reset operation or external or internal interrupts can release stop mode. The STOP instruction is a pseudo instruction. It is assembled as “SYS #0Ah”, which generates the SYSCP[7-0] signals. These signals are decoded and stop the operation.

NOTE: The next instruction of STOP instruction is executed, so please use the NOP instruction after the STOP instruction.

Example:

```
STOP
NOP
NOP
NOP
•
•
•
```

In this example, the NOP instructions provide the necessary timing delay for oscillation stabilization before the next instruction in the program sequence is executed. Refer to the timing diagrams of oscillation stabilization, as described in Figure 15-4, 15-5

SUB — Subtract

Format: SUB <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> + ~<op2> + 1

SUB adds the value of <op1> with the 2's complement of <op2> to perform subtraction on <op1> and <op2>

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: IDH:IDL0 = 0150h, DM[0143h] = 26h, R0 = 52h, R1 = 14h, eid = 1

```
SUB     R0, 43h          // R0 ← R0 + ~DM[0143h] + 1 = 2Ch

SUB     R1, #16h         // R1 ← FEh, N flag is set to '1'

SUB     R0, R1           // R0 ← R0 + ~R1 + 1 = 3Eh

SUB     R0, @ID0+1       // R0 ← R0 + ~DM[0150h] + 1, IDL0 ← 51h
SUB     R0, @[ID0-2]     // R0 ← R0 + ~DM[014Eh] + 1, IDL0 ← 4Eh
SUB     R0, @[ID0+3]!    // R0 ← R0 + ~DM[0153h] + 1, IDL0 ← 50h
SUB     R0, @[ID0-2]!    // R0 ← R0 + ~DM[014Eh] + 1, IDL0 ← 50h
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode. The example in the SBC description shows how SUB and SBC can be used in pair to subtract a 16-bit number from another.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

SWAP — Swap

Format: SWAP <op1>, <op2>

<op1>: GPR
<op2>: SPR

Operation: <op1> ← <op2>, <op2> ← <op1>

SWAP swaps the values of the two operands.

Flags: –

NOTE: Among the SPRs, SR0 and SR1 can not be used as <op2>.

Example: Given: IDH:IDL0 = 8023h, R0 = 56h, R1 = 01h

```
SWAP    R1, IDH          // R1 ← 80h, IDH ← 01h
SWAP    R0, IDL0         // R0 ← 23h, IDL0 ← 56h
```

After execution of instructions, index registers IDH:IDL0 (ID0) have address 0156h.

SYS — System

Format: SYS #imm:8

Operation: SYS generates SYSCP[7:0] and nSYSID signals.

Flags: –

NOTE: Mainly used for system peripheral interfacing.

Example:

```
SYS     #0Ah

SYS     #05h
```

In the first example, statement “SYS #0Ah” is equal to STOP instruction and second example “SYS #05h” is equal to IDLE instruction. This instruction does nothing but increase PC by one and generates SYSCP[7:0] and nSYSID signals.

TM — Test Multiple Bits

Format: TM <op>, #imm:8

<op>: GPR

Operation: TM performs the bit-wise AND operation on <op> and imm:8 and sets the flags. The content of <op> is not changed.

Flags: **Z:** set if result is zero. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: R0 = 01001101b

```
TM      R0, #00100010b      // Z flag is set to '1'
```

XOR — Exclusive OR

Format: XOR <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> ^ <op2>

XOR performs the bit-wise exclusive-OR operation on <op1> and <op2> and stores the result in <op1>.

Flags: **Z:** set if result is zero. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: IDH:IDL0 = 8080h, DM[8043h] = 26h, R0 = 52h, R1 = 14h, eid = 1

```
XOR     R0, 43h             // R0 ← 74h
XOR     R1, #00101100b      // R1 ← 38h
XOR     R0, R1              // R0 ← 46h
XOR     R0, @ID0            // R0 ← R0 ^ DM[8080h], IDL0 ← 81h
XOR     R0, @[ID0-2]        // R0 ← R0 ^ DM[807Eh], IDL0 ← 7Eh
XOR     R0, @[ID0+3]!       // R0 ← R0 ^ DM[8083h], IDL0 ← 80h
XOR     R0, @[ID0-5]!       // R0 ← R0 ^ DM[807Bh], IDL0 ← 80h
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

NOTES