HYCON TECHNOLOGY

CPU Core Application Description

Comparisons and Supplements of Instruction Set H08A and H08B

Table of Contents

1. Comparison of Instruction Set H08A and H08B

Instruction				Description	Cycles	Status Affected
H08A		H08B				
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDC	f,d,a	ADDC	f,d,a	Add W, F and C, and place the result to W or F.	1	C,DC,N,OV,Z
ADDF	f,d,a	ADDF	f,d,a	Add W and F, and place the result to W or F.	1	C,DC,N,OV,Z
ADDL	k	ADDL	k	Add constant k and W, and place the result to W.	1	C,DC,N,OV,Z
ANDF	f,d,a	ANDF	f,d,a	AND W and F, and place the result to W or F.	1	N,Z
ANDL	k	ANDL	k	AND constant k and W, and place the result to W.	1	N,Z
ARLC	f,d,a	-	-	Rotate left F value and C, and place the result to W or F.	1	C,N,OV,Z
ARRC	f,d,a	-	-	Rotate right F value, MSB remains unchanged, LSB moves to C	1	C,N,Z
CLRF	f,a	CLRF	f,a	Clear zero of F value.	1	None
COMF	f,d,a	COMF	f,d,a	Complement F value, and place the result to W or F.	1	N,Z
CPSE	f,a	CPSE	f,a	If F value = W value, skip the next instruction.	1(2)(3)	None
CPSG	f,a	CPSG	f,a	If F value > W value, skip the next instruction.	1(2)(3)	None
CPSL	f,a	CPSL	f,a	If F value < W value, skip the next instruction.	1(2)(3)	None
DCF	f,d,a	DCF	f,d,a	Subtract 1 of F value and place the result to W or F.	1	C,DC,N,OV,Z
DCSUZ	f,d,a	DCSUZ	f,d,a	Subtract 1 of F value. If the value≠0, skip the next instruction and place the result to W or F.	1(2)(3)	None
DCSZ	f,d,a	DCSZ	f,d,a	Subtract 1 of F value. If the value = 0, skip the next instruction, and place the result to W or F.	1(2)(3)	None
INF	f,d,a	INF	f,d,a	Add 1 to F value and place the result to W or F.	1	C,DC,N,OV,Z
INSUZ	f,d,a	INSUZ	f,d,a	Add 1 to F value. If the value≠0, skip the next instruction and place the result to W or F.	1(2)(3)	None
INSZ	f,d,a	INSZ	f,d,a	Add 1 to F value. If the value=0, skip the next instruction, and place the result to	1(2)(3)	None

				W or F.		
IORF	f,d,a	IORF	f,d,a	Inclusive OR W and F, and place the result to W or F.	1	N,Z
IORL	k	IORL	k	OR constant k and W, and place the result to W.	1	N,Z
LBSR	k	-	-	Move constant k to register, BSRCN.	1	None
LDPR	k,f	-	-	Move constant k (9-bit) to the register, FSR (f = 0 ~ 1).	2	None
MULF	f,a	-	-	Multiply W and F.	2	None
MULL	k	-	-	Do multiplication of constant k and W.	2	None
MVF	f,d,a	MVF	f,d,a	Move W value to F(d=1) or move F value to W(d=0).	1	None
MVFF	fs,fd	-	-	Move Fs data to Fd.	2	None
MVL	k	MVL	k	Move constant k to W.	1	None
RETL	k	RETL	k	Place the top-of-stack value to PC, and configure W as k. Main program will be executed from current PC value.	2	None
RLF	f,d,a	RLF	f,d,a	Rotate left F value and place the result to W or F.	1	N,Z
RLFC	f,d,a	RLFC	f,d,a	Rotate left F value and C, and place the result to W or F.	1	C,N,Z
RRF	f,d,a	RRF	f,d,a	Rotate right F value and place the result to W or F.	1	N,Z
RRFC	f,d,a	RRFC	f,d,a	Rotate right F value and C, and place the result to W or F.	1	C,N,Z
SETF	f,a	SETF	f,a	Configure F value as 0xFF.	1	None
SUBC	f,d,a	SUBC	f,d,a	Subtract W of F value and reverse C, and place the result to W or F.	1	C,DC,N,OV,Z
SUBF	f,d,a	SUBF	f,d,a	Subtract W of F value and place the result to W or F.	1	C,DC,N,OV,Z
SUBL	k	SUBL	k	Subtract constant k and W and place the result to W.	1	C,DC,N,OV,Z
SWPF	f,d,a	SWPF	f,d,a	Switch the high and low b bit of F value and place the result to W or F.	1	None
TFSZ	f,a	TFSZ	f,a	Test if F value = 0. Skip the next instruction if the value =0.	1(2)(3)	None
XORF	f,d,a	XORF	f,d,a	Exclusive OR W and F, and place the result to W or F.	1	N,Z
XORL	k	XORL	k	Exclusive OR constant k and W, and place the result to W.	1	N,Z

CONTROL OPERATIONS						
CALL	n,s	CALL	n	Store the PC value of the next instruction to the top-of-stack and jump to address, n.	2	None
CWDT	-	CWDT	-	Clear zero of watch dog timer.	1	TO
IDLE	-	IDLE	-	Access into idle mode.	1	IdleB
JC	n	-	-	If C = 1, jump to address n.	1(2)	None
JMP	n	JMP	n	Unconditionally jump to address n.	2	None
JN	n	-	-	If N = 1, jump to address n.	1(2)	None
JNC	n	-	-	If C = 0, jump to address n.	1(2)	None
JNN	n	-	-	If N = 0, jump to address n.	1(2)	None
JNO	n	-	-	If OV = 0, jump to address n.	1(2)	None
JNZ	n	-	-	If Z = 0, jump to address n.	1(2)	None
JO	n	-	-	If OV = 1, jump to address n.	1(2)	None
JZ	n	-	-	If Z = 1, jump to address n.	1(2)	None
NOP	-	NOP	-	Blank instruction.	1	None
POP	-	-	-	Subtract 1 of stack pointer register, read out the pointed stack value to register, TOS.	1	None
PUSH	-	-	-	Add 1 to stack pointer, store the address to register, TOS.	1	None
RCALL	n	-	-	Store the PC value of the next instruction to top-of-stack, and jump to address n, $-1024 \leqq n \leqq 1023$.	2	None
RET	s	RET	-	Return to main program from vice program and place the top-of-stack value to PC. Main program will be executed from current PC value.	2	None
RETI	s	RETI	-	Return to main program from interrupt, and place the top-of-stack value to PC. Main program will be executed from current PC value.	2	GIE
RJ	n	-	-	Unconditionally jump to address n, $-1024 \leqq n \leqq 1023$.	2	None

SLP	f,a	SLP	f,a	Access into sleep mode.		PD
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f,b,a	BCF	f,b,a	Configure a specific bit of f as 0.	1	None
BSF	f,b,a	BSF	f,b,a	Configure a specific bit of f as 1.	1	None
BTGF	f,b,a	BTGF	f,b,a	NOT a specific bit of f.	1	None
BTSS	f,b,a	BTSS	f,b,a	Test if a specific bit of F is 1. Skip the next instruction if the value is 1.	1(2)(3)	None
BTSZ	f,b,a	BTSZ	f,b,a	Test if a specific bit of F is 0. Skip the next instruction if the value is 0.	1(2)(3)	None
PROGRAM MEMORY OPERATIONS						
MVLP	k	-	-	Move constant k(0≦k≦16384d) to TABLE pointer (TBLPTRU/TBLPTRH/TBLPTRL)	2	None
TBLR	*	-	-	Make the contents of TBLPTR as address pointer, read program memory contents to register, TBLDH/TBLDL.	2	None
TBLR	*+	-	-	Make the contents of TBLPTR as address pointer, read program memory contents to register, TBLDH/TBLDL. The address pointer will add 1 automatically.	2	None

Remark

f	Register	b	Register b bit
n	Memory address	k	8 bit constant
d	Data stored place; d = 0 means it is saved in accumulator W; d = 1 means it is saved in register f.		
a	Memory address where data is stored, a=0 means it is saved in current memory address; a=1 means it is saved in the appointed memory address of register BSR.		
s	s = 1: backup/return wreg, status and bsr register value; s = 0: no backup action.		

Notice: There is only one set of Shadow register that stored only the latest inputted value.

2. Instruction Supplement Description

2.1 PCLATL Description

General programs can be written in sequence. Use JMP or RJ to jump to appointed program and execute. If users would like to jump the program to different address, operate PCLATL can achieve the objective.

General PCLATL operation can be implemented through instructions such as MVF, ADDF, AND….etc. PCLATH and PCLATU value must be confirmed first before operating PCLATL.

Note: It is suggested to use RJ. When it is necessary to use JMP, users must notice whether smart complier function was activated and the jump range fell in the range of -1024 ~ +1023. It must followed by a short instruction, otherwise it may jump to different line when PCLATL value is operated through ADDF and SUBF.

Some operation notices are listed hereunder：

To operate PCLATL by ADDF and SUBF, PCLATH must be dealt with caution if the sum of WREG and PCLATL exceeds 0x100. PCLATU must be changed in advance or it is likely to jump the wrong program.

For example：the program needs WREG value to judge jump address.

```
    MVL     High JPB                ; PCLATH stores the initial jump address (bit 15~8)
    MVF     PCLATH,F,ACCE
    MVF     MainCount+1,W,ACCE      ; Read jump value
    ADDF    PCLATL,F,ACCE           ; Current PCLATL + jump value
JPB:
    RJ      MainDaPro0
    RJ      MainDaPro1
    RJ      MainDaPro2
    RJ      MainDaPro3
    RJ      MainDaPro4
    RJ      MainDaPro5
    RJ      MainDaPro6
    RJ      MainDaPro7
```

The above program seems fine, however, if JPB address =0x00FF, 0x01FF…, jump address may be wrong.
Therefore, it is recommended to modify the program as in below to ensure JPB will not jump to error address:

```
        MVL     High JPB                ; PCLATH stores the initial jump address (bit 15~8)
        MVF     PCLATH,F,ACCE
        MVF     MainCount+1,W,ACCE
        ADDL    Low JPB                 ; test if the sum of jump value and JPB address exceeds 0x100
                                        ; (Low JPB)+( MainCount+1) ->W
        BTSZ    STAUTS,C,ACCE
        INF     PCLATH,F,ACCE           ; if the value outstripped 0x100, [PCLATU,PCLATH] must + 1
        MVF     PCLATL,F,ACCE           ; current PCLATL + jump value
    JPB:
        RJ      MainDaPro0
        RJ      MainDaPro1
```

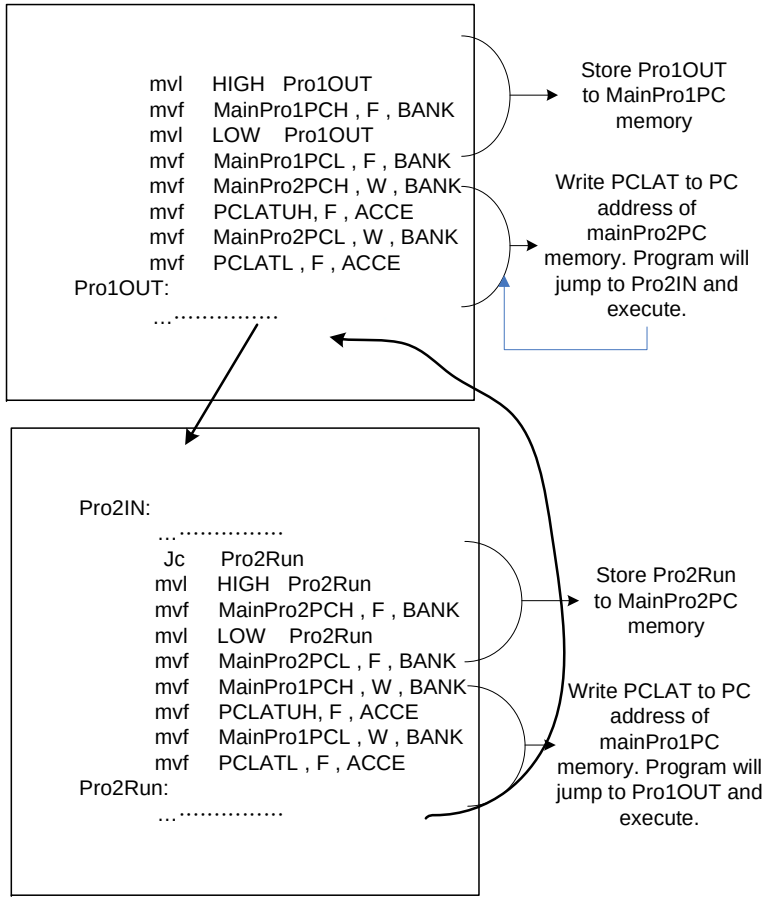
Also can specify the address to predict that PCLATH will not be changed and shorter instruction can be used to complete PC jump, for example:

```
            MVF     MainCount+1,W,ACCE
            JMP     JPB
            ……
            MVF     Table_Index,W,ACCE
            CALL    Table
ORG 0700H   ;; Move table or branch to the last 256 instructions. (In 2K word Rom size, the last 256 instructions are 0700H~07FFH)
    JPB:
            ADDF    PCLATL,F,ACCE
            RJ      MainDaPro0
            RJ      MainDaPro1
    Table:
```

```
ADDF    PCLATL,F,ACCE
RETL    055H
RETL    0AAH
```

When a program is under execution, users would like to jump to another program section, he must wait for the completion of another program to a certain stage, then jump back to the original one to continue.

For example:



Notice: Please do not use MVFF to move PCLATL value

2.2 POP Description

POP is used to deal with program memory stacks. POP discards the value of TOSU, TOSH and TOSL, and subtracts 1 of STKPTR. If the stack layer is not enough, stores TOSU, TOSH and TOSL to memory and uses POP to release one stack layer. Use PCLATL stored stack value to return.

POP and PCLATL can be implemented when CALL is the latest layer and one layer must be saved for interrupt. Below provides an example to illustrate this function.

```
MainLoop:
        call    Tst1
        ……
        jmp     MainLoop

Tst1:
        ……
        call    Tst2
        ……
        ret
Tst2:
        ……
        call    Tst3
        ……
        ret
Tst3:
        ……
        call    Tst4
        ……
        ret
Tst4:
        ……
        call    Tst5
        ……
        ret
Tst5:
        ……
        mvff    TOSU,STKBufU
        mvff    TOSH,STKBufH
        mvff    TOSL,STKBufL
```

```
        POP
        call    Tst6
        mvff    STKBufU,PCLATU
        mvff    STKBufH,PCLATH
        mvf     STKBufL,W,BANK
        mvf     PCLATL,F,ACCE       → return to Tst4
Tst6:
        …….
        ret
```

2.3 XOR Application

Example 1: XOR can be used for equivalent values judgment:

```
        mvl     3
        xorf    Temp , W , BANK
        jz      ValEQU
        ……..
        ValEQU:
        ……..
```

Example 2: XOR can be used for 2 memory value switching function:

```
        Switching memory TEMP0 and TEMP1 value
        mvf     TEMP0 , W , ACCE
        xorf    TEMP1 , W , ACCE
        xorf    TEMP0 , F , ACCE
        xorf    TEMP1 , F , ACCE
```

2.4 Object File

Object file is saved in binary format and can offer reference arguments through Global for external program. The arguments can be the definition of Labe and SRAM.

For example: Subroutine source code:

```
        Global  TEMP, MA, INDF0, POINC0, FSR0H, FSR0L
        Global  Clear RAM
        Global  F, W, ACCE, BANK
        TEMP    EQU     080h
        MA      EQU     090h
        INDF0   EQU     000h
        POINC0  EQU     001h
```

```
        FSR0H       EQU         00Fh
        FSR0L       EQU         010h
        F           EQU         1
        W           EQU         0
        ACCE        EQU         0
        BANK        EQU         1
ClearRAM:
        MVL         80h
        MVF         FSR0L, F, ACCE
        CLRF        FSR0H, ACCE
 ClearLoop:
        CLRF        POINC0, ACCE
        TFSZ        FSR0L, ACCE
        JMP         Clear Loop
        RET
```

After compile, Subroutine.obj may be generated and may release some arguments:

SRAM arguments: TEMP, MA, INDF0, POINC0, FSR0H, FSR0L

Constant defined arguments: F, W, ACCE, BANK

Label argument: Clear RAM

Main program can quote Subroutine.obj declared arguments

```
    ORG         0000h
    JMP         Begin
Begin:
      CALL          ClearRAM
Mainloop:
    MVL         0A0h
    MVF         TEMP,F,ACCE
    ….
    JMP         Mainloop
    INCLUDE     Subroutine.obj
```

Object File can quote external arguments, through EXTERN call

For example: External arguments MXX is defined as SRAM address, 0A0h

```
    EXTERN      MXX
    CLRF        MXX, ACCE
```

3. Revision Record

Major differences are stated thereinafter:

Version	Page	Revision Summary
V01	ALL	First edition.
V02	ALL	Format change, Add instruction supplement description.
V04	-	Delete DAW instruction
V05	P7~9	Update PCLAT demo code and description
V06	P7~9	Update PCLAT demo code and description