1.

Assumption:

PC is currently pointing the address of higher byte of *CALL* instruction. After incrementing the PC, PC is pointing the address of lower byte of CALL instruction. After incrementing the PC, PC is pointing *target address*.

|  |  |
| --- | --- |
| PC -> | high(Call instruction) |
| PC+1 -> | Low(Call instruction) |
| PC+2 -> | *target address*. |

CALL *label* ; STACK <- PC, PC <- k

In 8 bit memories, The address pointed by PC in the memory has higher byte of opcode, and then the next address in the memory has lower byte of opcode.

|  |  |
| --- | --- |
| Address (pointed by PC) | HIGH(Opcode) |
| Address (pointed by PC) + 1 | LOW(Opcode) |

Fetch Cycle)

1. MAR <-PC, TEMP <- PC
2. MDR <- M[MAR], PC,<- PC+1
3. HIGH(IR) <- MDR, MAR <- PC,
4. MDR<-M[MAR], PC <- PC+1 ; PC is currently pointing target address
5. LOW(IR) <- MDR

Execute Cycle)

1. MDR <- LOW(TEMP), MAR <- SP
2. M[MAR] <- MDR, SP <- SP-1
3. MDR <- HIGH(TEMP), MAR <- SP
4. M[MAR] <- MDR, SP <- SP-1

2.

Assumptions:

PC is currently pointing the instruction of *ADIW* instruction. After incrementing PC, PC is pointing the immediate value.

|  |  |
| --- | --- |
| PC -> | ADIW |
| PC+1 -> | Immediate Value (ex. 0x20) |

ADIW ZH:ZL, 32 ; ZH : ZL <- ZH: ZL + 32

Fetch Cycle)

1. MAR <- PC
2. MDR <-M[MAR], PC <-PC+1
3. IR <-MDR, MAR <-PC
4. MDR <- M[MAR], PC <- PC+1
5. AC <- MDR

Execute Cycle)

1. AC <- AC+ZL
2. ZL <- AC
3. AC <- ZH
4. IF(C = 1) THEN AC <- AC+1
5. AC <- AC
6. ZH <- AC

3.

(a)

;Y register points the label addrB

3. ldi YL, $02

4. ldi YH, $01

;Z register points the label LAddrP

5. ldi ZL, $04

6. ldi ZH, $01

;X register points the label addrA

8. ldi XL, $00

9. ldi XH, $01

(b) 0x0203, 0x010C

(c)

LAddrP: 24

LAddrP +1: 00

LAddrP +2: 00

LAddrP +3: 00

(Note: LAddrP = $0104)

(d)

LAddrP: 24

LAddrP +1: 18

LAddrP +2: 00

LAddrP +3: 00

(Note: LAddrP = $0104)

(e) Immediate value for line 7: 2 (Decimal) -> ldi oloop, 2

4.

(Note: The letter colored blue is machine code of each instruction)

RJMP INIT : 1100 kkkk kkkk kkkk

RCALL ISR : 1101 kkkk kkkk kkkk

RETI : 1001 0101 0001 1000

LDI XH, high(CTR): 1110 kkkk dddd kkkk

LDI XL, low(CTR): 1110 kkkk dddd kkkk

LDI YH, high(DATA) 1110 kkkk dddd kkkk

LDI YL, low(DATA) 1110 kkkk dddd kkkk

RJMP WAIT 1100 kkkk kkkk kkkk

IN mpr, PINA 1011 0AAd dddd AAAA

ST Y+, mpr 1001 001r rrrr 1001

INC count 1001 010d dddd 0011

ST X+, count 1001 001r rrrr 1101

RET 1001 0101 0000 1000

5.

.ORG $0046

…Initialize stack…

RCALL MIN

…

…

.ORG $0060

MIN:

ldi XL, $00 ;X-reg points 0x0000

ldi XH, $00

ld r16, X+ ; load 0x01 to r16 with post increment

LOOP:

ld r15, X ; loads the content X is pointing -starts from 0x0001

cp r15, r16 ; compare contents of two registers

brlt Change ; if the content of r15 is less than r16, go to Change

inc XL ; increment lower-byte of address

cpi XL, $08 ; compare lower-byte of address is $08

brne LOOP ; if not keep looping

Change:

mov r16, r15 ; moving much less value to r16.

cp r16, r15 ; making condition to go back to loop

breq LOOP ; because of ‘mov’, condition satisfied. Go back to loop

st X, r16 ; after existing the loop, store the minimum value to the address 0x0008

RET

(Note: The code above is a pseudo code. Thus, it will make errors when you actually run the code.)

(For real world assembly coding, if I can set the variables, I would set r15 as mpr(.def mpr = r15), and r16 as mini(.def mini = r16))

(Furthermore, I assume that the code has definition file – inc.”m128def.inc”, .cseg, .dseg, and etc. for properly running code.)