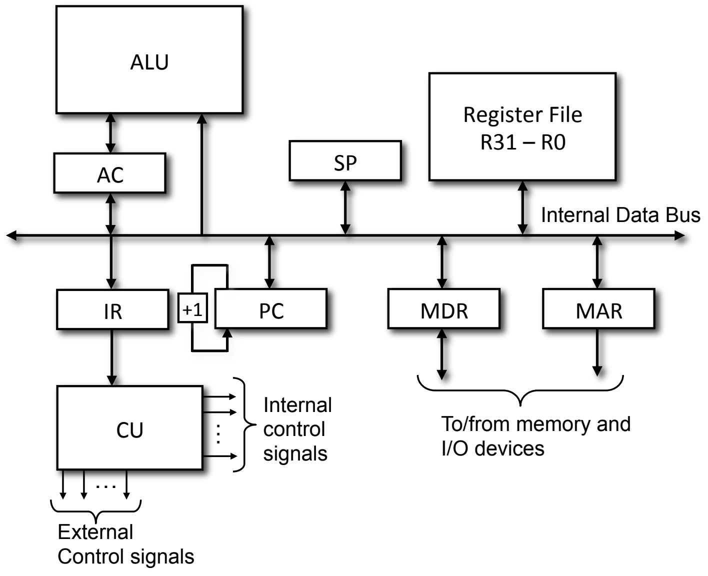
ECE 375
Computer Organization and Assembly Language Programming
Fall 2022
Solutions Set #2

[20 pts]
1- Consider the pseudo-CPU discussed in class augmented with a *single-port register file* (i.e., only one register value can be read at a time) containing 32 8-bit registers (R31-R0) and a Stack Pointer (SP) register. Suppose the pseudo-CPU can be used to implement the AVR instruction `RET` (Return from Subroutine) with the format shown below:

1001	0101	0000	1000

`RET` pops the return address from the stack and jumps to the return address. Give the sequence of microoperations required to Fetch and Execute AVR's `RET` instruction. *Your solutions should result in minimum number of microoperations*. Assume the memory is organized into addressable bytes (i.e., each memory word is a byte), MDR and AC registers are 8-bit wide, and SP, PC, IR, and MAR are 16-bit wide. Also, assume Internal Data Bus is 16-bit wide and thus can handle 8-bit or 16-bit (as well as portion of 8-bit or 16-bit) transfers in one microoperation and SP has the capability to increment itself. *Clearly state any other assumptions made*.



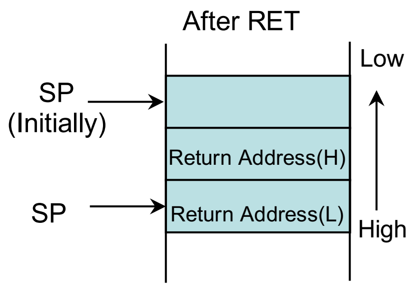
Solution

Step 1: MAR ←PC;
Step 2: MDR ←M(MAR), PC ← PC+1 ; Get the high byte of the instruction and increment PC
Step 3: IR(15…8) ← MDR
Step 4: MAR ←PC;
Step 5: MDR ←M(MAR), PC ← PC+1 ; Get the low byte of the instruction and increment PC
Step 6: IR(7…0) ← MDR ; At this point, CU knows this is RET

Step 7: SP ← SP+1
Step 8: MAR ←SP
Step 9: MDR ← M(MAR), SP ← SP+1 ; Pop the higher byte of return address from the stack
Step 10: MAR ← SP
Step 11: PC(15…8) ← MDR, MDR ← M(MAR) ; Pop the lower byte of return address from the stack

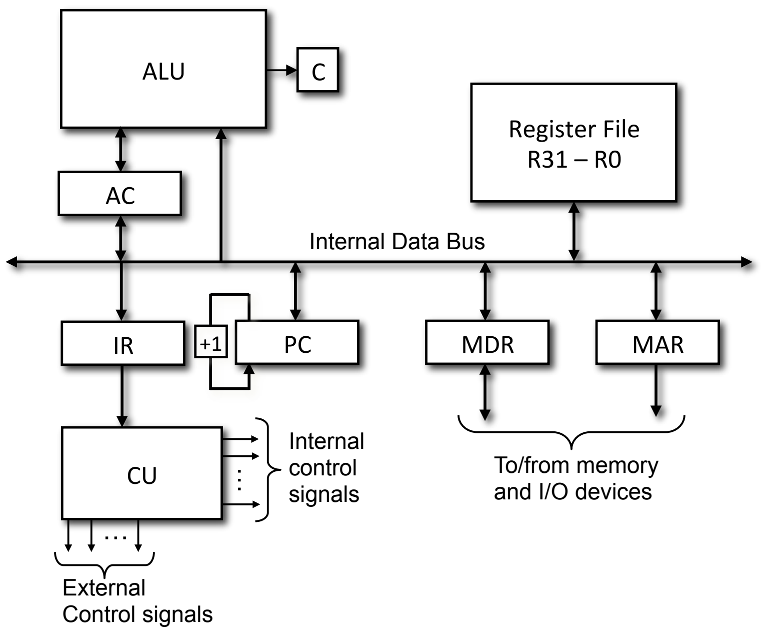
Step 12: PC(7…0) ← MDR

Goto fetch and Execute cycle



[20 pts]

2- Consider the internal structure of the pseudo-CPU discussed in class augmented with a *single-port register file* (i.e., only one register value can be read at a time) containing 32 8-bit registers (R0-R31). Suppose the pseudo-CPU can be used to implement the AVR instruction `ST -X, R3`. Give the sequence of microoperations required to Fetch and Execute AVR's `ST -X, R3` instruction. *Your solutions should result in exactly 6 cycles for the fetch cycle and 7 cycles for the execute cycle.* You may assume only the AC and PC registers have the capability to increment/decrement itself. Assume the MDR register is 8-bit wide (which implies that memory is organized into consecutive addressable bytes), and AC, PC, IR, and MAR are 16-bit wide. Also, assume Internal Data Bus is 16-bit wide and thus can handle 8-bit or 16-bit (as well as portion of 8-bit or 16-bit) transfers in one microoperation. Clearly state any other assumptions made.



Solution

Since `ST -X, R3` is a 16-bit instruction and the memory is organized into consecutive bytes, the instruction occupies two consecutive bytes. Thus, two memory accesses are needed to fetch the instruction into the IR.

Fetch Cycle
Cycle 1: MAR ← PC
Cycle 2: MDR ← M[MAR], PC ← PC+1
Cycle 3: IR(15…8) ← MDR
Cycle 4: MAR ← PC
Cycle 5: MDR ← M[MAR], PC ← PC+1
Cycle 6: IR(7…0) ← MDR

Next, we need to decrement register X, which is in registers R26 (XL) and R27 (XH). Once X is pre-decremented, the address is put into MAR, content of R3 is put into MDR, and stored into the memory. Note that it is also possible to perform Cycles 3 and 6 in parallel.

Execute Cycle
Cycle 1: AC(15…8) ← R27 ; Requires two steps because only one register can be read at a time
Cycle 2: AC(7…0) ← R26
Cycle 3: AC ← AC - 1
Cycle 4: R27 ← AC(15…8), MAR(15…8) ← AC(15…8)
Cycle 5: R26 ← AC(7…0), MAR(7…0) ← AC(7…0)
Cycle 6: MDR ← R3
Cycle 7: M(MAR) ← MDR

[20 pts]
3- Consider the following AVR assembly code for Lab. #4 that performs 16-bit by 16-bit multiplication. Assume the data memory locations $0100 through $0107 initially have the following values:

Data Memory

Address	content
0100	0C
0101	00
0102	0F
0103	02
0104	00
0105	00
0106	00
0107	00

```
    .include "m128def.inc"              ; Include definition file
    .def   rlo = r0                     ; Low byte of MUL result
    .def   rhi = r1                     ; High byte of MUL result
    .def   zero = r2                    ; Zero register
    .def   A = r3                       ; An operand
    .def   B = r4                       ; Another operand
    .def   oloop = r17                  ; Outer Loop Counter
    .def   iloop = r18                  ; Inner Loop Counter
    .org   $0000
1.                rjmp   INIT
                  .org   $0046
2.  INIT:         clr    zero           ; Set zero register to zero
3.  MAIN:         ldi    YL, low(addrB) ; Load low byte
4.                ldi    YH, high(addrB) ; Load high byte
5.                ldi    ZL, low(LAddrP) ; Load low byte
6.                ldi    ZH, high(LAddrP) ; Load high byte
7.                ldi    oloop, 2       ; Load counter
8.  MUL16_OLOOP:  ldi    XL, low(addrA) ; Load low byte
9.                ldi    XH, high(addrA) ; Load high byte
10.               ldi    iloop, 2       ; Load counter
11. MUL16_ILOOP:  ld     A, X+          ; Get byte of A operand
12.               ld     B, Y           ; Get byte of B operand
13.               mul    A,B            ; Multiply A and B
14.               ld     A, Z+          ; Get a result byte from memory
15.               ld     B, Z+          ; Get the next result byte from memory
16.               add    rlo, A         ; rlo <= rlo + A
17.               adc    rhi, B         ; rhi <= rhi + B + carry
18.               ld     A, Z           ; Get a third byte from the result
19.               adc    A, zero        ; Add carry to A
20.               st     Z, A           ; Store third byte to memory
21.               st     -Z, rhi        ; Store second byte to memory
22.               st     -Z, rlo        ; Store first byte to memory
23.               adiw   ZH:ZL, 1       ; Z <= Z + 1
```

```
24.             dec    iloop            ; Decrement counter
25.             brne   MUL16_ILOOP      ; Loop if iLoop != 0
26.             sbiw   ZH:ZL, 1         ; Z <= Z - 1
27.             adiw   YH:YL, 1         ; Y <= Y + 1
28.             dec    oloop            ; Decrement counter
29.             brne   MUL16_OLOOP      ; Loop if oLoop != 0
30. Done:       rjmp   Done
    .dseg
    .org   $0100
    addrA:      .byte  2
    addrB:      .byte  2
    LAddrP:     .byte  4
```

Solution

This code performs the following:

```
          00  0C
     x    02  0F
          00  B4
      00  00
      00  18
  00  00
  00  00  18  B4
```

[4 pts]

(a) What are the two 16-bit values (in hexadecimal) being multiplied?

Solution

The figure below shows where labels `addrA`, `addrB`, and `LAddrP` are pointing to in data memory after executing lines 3-9.

	Data Memory Address	content
addrA:	0100	0C
	0101	00
addrB:	0102	0F
	0103	02
LAddrP:	0104	00
	0105	00
	0106	00
	0107	00

Since X points to $0100 and Y points to $0102, the two values are either $000C and $020F or $0C00 and $0F02. However, since the first multiplication (line 13) performed in the inner loop is $0C x $0F, the two values are $000C and $020F.

[4 pts]

(b) What are the contents of memory locations pointed to by `LAddrP`, `LAddrP+1`, `LAddrP+2`, and `LAddrP+3` after the loop `MUL16_ILOOP` (lines 11-25) completes for the first time?

Solution

Lines 11-13 result in the following:

```
      0C
  x   0F
  00  B4
  (rhi)(rlo)
```

Lines 14-19 result in

$$\begin{array}{lll} & \text{(rhi)} & \text{(rlo)} \\ & 00 & B4 \\ & \text{(B)} & \text{(A)} \\ + & 00 & 00 \\ \hline 00 & 00 & B4 \\ \text{(A)} & \text{(rhi)} & \text{(rlo)} \end{array}$$

Then, lines 20-22 stores these three bytes in

```
0104        B4
0105        00
0106        00
```

[4 pts]
(c) What are the contents of memory locations pointed to by `LAddrP`, `LAddrP+1`, `LAddrP+2`, and `LAddrP+3` after the loop `MUL16_ILOOP` (lines 11-25) completes for the second time?

Solution

$$\begin{array}{ll} & 00 \\ \text{x} & 0F \\ \hline 00 & 00 \\ \text{(rhi)} & \text{(rlo)} \end{array}$$

Lines 14-19 result in

$$\begin{array}{lll} & \text{(rhi)} & \text{(rlo)} \\ & 00 & 00 \\ & \text{(B)} & \text{(A)} \\ + & 00 & 00 \\ \hline 00 & 00 & 00 \\ \text{(A)} & \text{(rhi)} & \text{(rlo)} \end{array}$$

Then, lines 20-22 stores these three bytes in

```
0104        B4
0105        00
0106        00
0107        00
```

[4 pts]
(d) What are the contents of memory locations pointed to by `LAddrP`, `LAddrP+1`, `LAddrP+2`, and `LAddrP+3` after the loop `MUL16_ILOOP` (lines 11-25) completes for the third time?

Solution

$$\begin{array}{ll} & 0C \\ \text{x} & 02 \\ \hline 00 & 18 \\ \text{(rhi)} & \text{(rlo)} \end{array}$$

Lines 14-19 result in

```
      (rhi)(rlo)
      00   18
      (B)  (A)
+     00   00
00    00   18
(A)  (rhi)(rlo)
```

Then, lines 20-22 stores these three bytes in

```
0104       B4
0105       18
0106       00
0107       00
```

[4 pts]

(e) What are the contents of memory locations pointed to by `LAddrP`, `LAddrP+1`, `LAddrP+2`, and `LAddrP+3` after the loop `MUL16_ILOOP` (lines 11-25) completes for the fourth time?

Solution

```
           00
      x    02
      00   00
      (rhi)(rlo)
```

Lines 14-19 result in

```
      (rhi)(rlo)
      00   00
      (B)  (A)
+     00   00
00    00   00
(A)  (rhi)(rlo)
```

Then, lines 20-22 stores these three bytes in

```
0104       B4
0105       18
0106       00
0107       00
```

[20 pts]

4- Consider the following code written in AVR assembly with its equivalent (partially completed) binaries on the right.

```
         .ORG   0x000F           Address        Binary
         LDI    XH, high(CTR)    000F:      1110   KKKK   dddd   KKKK
         LDI    XL, low(CTR)     0010:      1110   KKKK   dddd   KKKK
         LDI    R31, 0x55        0011:      1110   KKKK   dddd   KKKK
         CLR    R5               0012:      0010   01dd   dddd   dddd
LOOP:    SEC                     0013:      1001   0100   0000   1000
         ROL    R31              0014:      0001   11dd   dddd   dddd
         BRCS   SKIP             0015:      1111   00kk   kkkk   k000
         INC    R5               0016:      1001   010d   dddd   0011
SKIP:    CPI    R31, 0xFF        0017:      0011   KKKK   dddd   KKKK
         BRNE   LOOP             0018:      1111   01kk   kkkk   k001
```

```
        ST     X, R5            0019:     1001   001d   dddd   1100
DONE:   JMP    DONE             001A:     1001   010k   kkkk   110k
                                001B:     kkkk   kkkk   kkkk   kkkk
        .DSEG
        .ORG   0x0100
CTR:    .BYTE  1
```

[12 pts]

(a) Explain in words what the program accomplishes when it is executed. That is, explain what it does, how it does it, and how many times it does it. What is the value of location CTR when the execution completes?

Solution

This program loads the value \$55 = 0b01010101 and rotates a bit at a time through the carry to determine the number of 0's. The number of 0's is stored at location CTR. Since 0b01010101 has 4 0's, the value at location CTR will be 4. The will execute 7 times since LSB is already 1. This is done by

- Loading immediate value \$55 into R31
- Rotating R31 left through the carry to see if the shifted bit is not set, and feed in 1's.
- If not set, increment counter (R5).
- BRNE checks if the loop should terminate. And the program terminates when R31 becomes all 1's. The number times the loop executes depends on the position of the least significant zero bit. For the value \$55, the loop executes 7 times.
- BYTE assembler directive allocates 1 byte for storing the count.

[8 pts]

(b) Determine the binary representation for the following:
(i) `KKKK dddd KKKK` (@ address \$0010)
(ii) `dd dddd dddd` (@ address \$0012)
(iii) `kk kkkk k` (@ address \$0018)
(iv) `k kkkk k` (@ address \$001A) and `kkkk kkkk kkkk kkkk` (@ address \$001B)

Solution

(i) Address \$0010: low(CTR) = 00_{16}, thus `KKKK KKKK` = 0000 0000, and XL = R26 = `dddd`= 11010 (with implied 1 in the MSB).
(ii) Address \$0012: `d dddd`=0 0101 and `d dddd`=0 0101 (note that first one indicates red locations and second one indicates blue locations). Actually, the opcode is identical to EOR.
(iii) Address \$0018: target address is \$0013 thus 0013_{16}-0019_{16}=-0006_{16}=1111010_2 => `kk kkkk k` = 11 1101 0
(iv) Address \$001A: target address is \$001A = 0000 0000 0001 1010
=> k kkkk k kkkk kkkk kkkk kkkk = 0 0000 0 0000 0000 0001 1010

[20 pts]

5- Using AVR assembly language, write a subroutine `XOR` that evaluates the logical exclusive-OR of two operands *A* and *B*. The subroutine will be implemented using a skeleton code shown below:

```
.ORG   0x000F
       RCALL XOR
       …
       …
.ORG   0x010F
XOR:   …   ; Your code goes here
       …   ;
       RET
```

[16 pts]

(a) Assume the value *A* is in R1 and value *B* in R2 when the subroutine is called. However, there is a catch! You can use any AVR assembly instructions **except** `EOR` and `AND` instructions. In addition, you may not destroy (overwrite) the original contents of registers R1 and R2.

Solution

Exclusive-OR of A and B can be performed by the function AB'+A'B. However, since you are not allowed to use EOR or AND, you can only do this using OR and complement operations. To generate the proper equation, we use DeMorgan's theorem, which leads to the following steps:
AB'+A'B
(AB')''+(A'B)''
(A'+B)'+(A+B')'

Now, we have all the operations in terms of OR and NOT. Assuming the operands A and B are in R1 and R2, respectively, when the subroutine is called, one possible code for implementing (A'+B)'+(A+B')' is given by

```
XOR:
   MOV    R3, R1       ;
   COM    R3           ; A'
   OR     R3, R2       ; R3=A'+B
   COM    R3           ; R3 =(A'+B)'
   MOV    R4, R2       ;
   COM    R4           ; B'
   OR     R4,R1        ; R4=A+B'
   COM    R4           ; R4=(A+B')'
   OR     R3, R4       ; R3=(A'+B)' + (A+B')'
   RET
```

[4 pts]
(b) Show the contents of the stack right after `RCALL` is made.

Solution

Since the return address of the subroutine call is right after `RCALL`, i.e., 0010_{16}, the content of the stack is as shown below.