Winnie_Woo_HW2

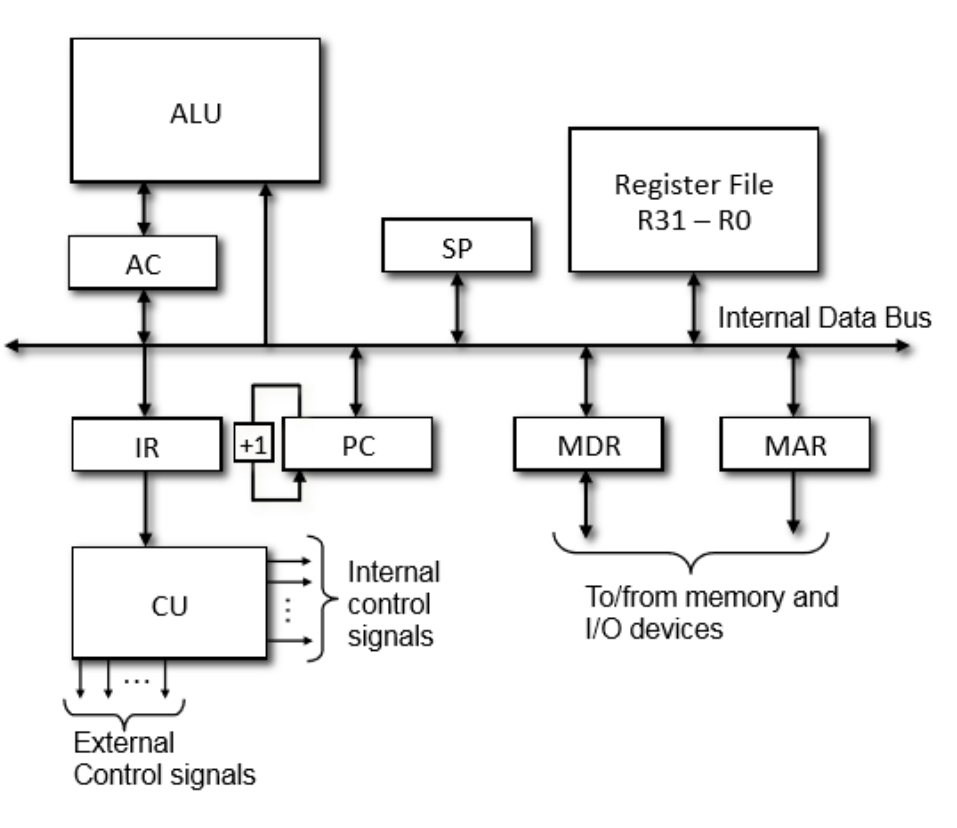
Sunday, October 23, 2022 9:03 PM

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

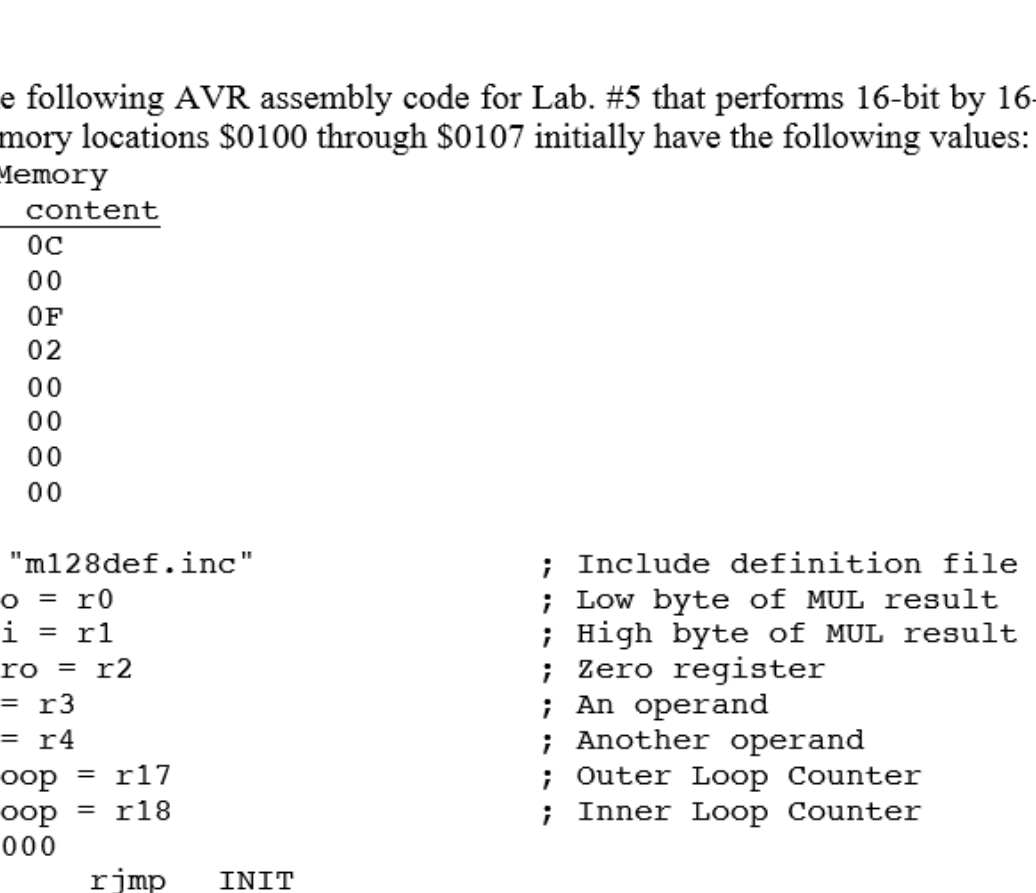
[20 pts]
1- Consider the pseudo-CPU discussed in class augmented with a *single-port register file* (i.e., only one register value can be read at a time) containing 32 8-bit registers (R31-R0) and a Stack Pointer (SP) register. Suppose the pseudo-CPU can be used to implement the AVR instruction RET (Return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



[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.



[20 pts]
3- Consider the following AVR assembly code for Lab. #5 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
```

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

[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?
[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)

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

1a.

Fetch cycle

Cycle 1: MAR ← PC, PC ← PC + 1
Cycle 2: MDR ← M(MAR), MAR ← PC
Cycle 3: HIGH (IR) ← MDR, MDR ← M(MAR), PC ← PC + 1
Cycle 4: LOW (IR) ← MDR

b. Execute Cycle

Cycle 1: SP ← SP + 1
Cycle 2: MAR ← SP, SP ← SP + 1
Cycle 3: MDR ← M(MAR), MAR ← SP
Cycle 4: HIGH (PC) ← MDR, MDR ← M(MAR)
Cycle 5: LOW (PC) ← MDR

2. ST -X, R3

Fetch cycle

Cycle 1: MAR ← PC
Cycle 2: MDR ← M(MAR), PC ← PC + 1
Cycle 3: $IR_{(7...0)}$ ← MDR ; little endian convention, first bytes will be bits 7..0
Cycle 4: MAR ← PC
Cycle 5: MDR ← M(MAR), PC ← PC + 1
Cycle 6: $IR_{(15...8)}$ ← MDR

Execute Cycle R26 (XL) R27 (XH)

Cycle 1: $AC_{(15..8)}$ ← R27
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

3a. 0x00C, 0x20F
b. 0xB4, 0x00, 0x00, 0x00
c. 0xB4, 0x00, 0x00, 0x00
d. 0xB4, 0x18, 0x00, 0x00
e. 0xB4, 0x18, 0x00, 0x00

4a. loads CTR address into X pointer & loads 0x55 into R31 and resets R5 register to 0x00. Within the loop, carry flag is set to 1, Then check to see if 1 was carried, if so skip otherwise increment register by 1. Then check if value is equal to 0XFF if not the case, loop again. If at the max, store value to X pointer address. loop is executed 5 times.

b. KKKK dddd KKKK @$0010
0000 r26
0000 1010 0000

ii dd dddd dddd @ $0012
10 1010 1010

iii KK KKKK KK @ $0018

iv. K KKKK K
0 0000 0

KKKK KKKK KKKK KKKK
0000 0000 0001 1010

5a. XOR:

clr result		; clear output register
cp A, B		; compare A & B
in mpr, SREG		; Read SREG
SBRS mpr,1		; set bit if Z flag is 0
ldi result, 1		
RET		

b. location is 0x10FD