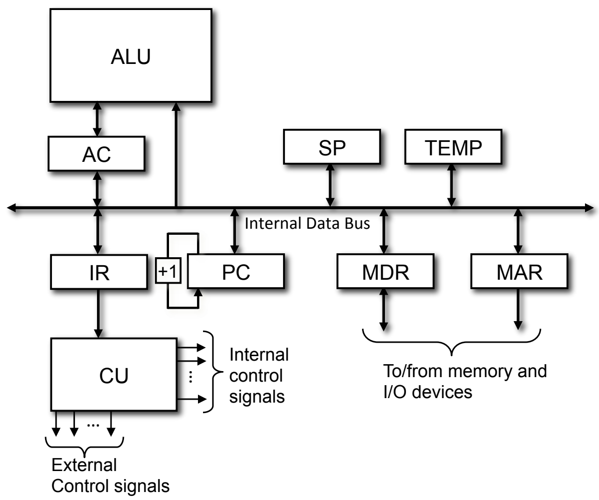
ECE 375
Computer Organization and Assembly Language Programming
Winter 2022
Assignment #2

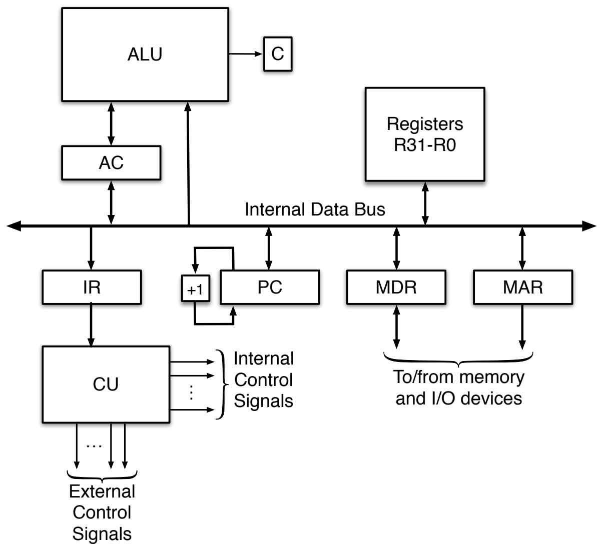
[20 pts]

1- Consider the internal structure of the pseudo-CPU augmented with a Stack Pointer (SP) and a 16-bit Temporary (TEMP) register. Suppose the pseudo-CPU can be used to implement the AVR instruction `CALL` *label* (Long Call to a Subroutine). `CALL` *label* is a four-byte instruction. Give the sequence of microoperations required to Fetch and Execute `CALL` *label*. *Your solution should result in no more than 19 microoperations*. You may assume the SP register has the capability to increment/decrement itself. Assume the PC is currently pointing to the `CALL` instruction and MDR register is 8-bit wide, and SP, PC, IR, and MAR are 16-bit wide. Note that since PC is 16 bits, only the lower 16 bits of the target address (i.e., *label*) are used. In other words, the upper 16 bits represent the opcode and the lower 16 bits represent the target address for the subroutine call. Also, assume Internal Data Bus is 16-bit wide and can handle 8-bit or 16-bit transfers in one microoperation. 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) 32 8-bit registers (R31-R0) and a carry bit (C-bit), which is set/reset after each arithmetic operation. Suppose the pseudo-CPU can be used to implement the AVR instruction `adiw ZH:ZL,32` (*Add immediate to word*). `adiw` is a 16-bit instruction, where the upper byte represents the opcode and the lower byte represents an immediate value, i.e., "`32`" (do not worry about the fact that the actual format is slightly different). Give the sequence of microoperations required to Fetch and Execute the `adiw` instruction. *Your solutions should result in exactly 5 cycles for the fetch cycle and 6 cycles for the execute cycle.* Assume the memory is organized into addressable bytes (i.e., each memory word is a byte), MDR, IR, and AC registers are 8-bit wide, and PC and MAR registers 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 only PC and AC have the capability to increment itself.



[20 pts]

3- Consider the following AVR assembly code that performs 16-bit by 16-bit multiplication (with some information missing). Assume the data memory locations $0100 through $0107 initially have the following values:

Data Memory

Address	content
0100	03
0101	02
0102	0C
0103	01
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:       ____________________    ; Load low byte
4.              ____________________    ; Load high byte
5.              ____________________    ; Load low byte
6.              ____________________    ; Load high byte
7.              ldi    oloop, ___       ; Load counter
8.  MUL16_OLOOP: ____________________   ; Load low byte
9.              ____________________    ; 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
              .org   $0100
    addrA:    .byte  2
    addrB:    .byte  2
    LAddrP:   .byte  4
```

(a) Show the code needed to initialize the `X` (lines 8-9), `Y` (lines 3-4), and `Z` (lines 5-6) pointers to point to labels `addrA`, `addrB`, and `LAddrP`, respectively.
(b) What are the two 16-bit values (in hexadecimal) being multiplied?
(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 first time (i.e., the 1st iteration)?
(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 second time (i.e., the 2nd iteration)?
(e) What is the immediate value needed in the instruction in line 7 for this program to work properly?

[20 pts]

4- For the following AVR code, determine the machine code for each instruction in the program code shown below. Some of the machine codes have already been determined.

```
        .include "m128def.inc"
        .def mpr = r16
        .def count = r17

        .ORG $0000                      Program Address         Binary
START:  RJMP INIT                       0000:             1100  kkkk  kkkk  kkkk
        .ORG $0002                      …
        RCALL ISR                       0002:             1101  kkkk  kkkk  kkkk
        RETI                            0003:             1001  0101  0001  1000
INIT:   {                               0004:
        …                               …
        Interrupt                       …
        Initialization                  …
        Code                            …
        …                               …
        }
        LDI    XH, high(CTR)            000B:             1110  KKKK  dddd  KKKK
        LDI    XL, low(CTR)             000C:             1110  KKKK  dddd  KKKK
        LDI    YH, high(DATA)           000D:             1110  KKKK  dddd  KKKK
        LDI    YL, low(DATA)            000E:             1110  KKKK  dddd  KKKK
WAIT:   RJMP   WAIT                     000F:             1100  kkkk  kkkk  kkkk
        .ORG   $100F                    …
ISR:    IN     mpr, PINA                100F:             1011  0AAd  dddd  AAAA
        ST     Y+, mpr                  1010:             1001  001r  rrrr  1001
        INC    count                    1011:             1001  010d  dddd  0011
        ST     X+, count                1012:             1001  001r  rrrr  1101
        RET                             1013:             1001  0101  0000  1000
        .DSEG
        .ORG   $0100
CTR:    .BYTE  1
DATA:   .BYTE  256
```

[20 pts.]

5- Suppose the following array of numbers are stored in the Data Memory (represented in hexadecimal):

Address	Content
0000:	01
0001:	BE
0002:	35
0003:	EC
0004:	48
0005:	2D
0006:	04
0007:	02

Using AVR assembly language, write a subroutine that (1) determines the smallest number among the 8 numbers stored in memory and (2) stores the smallest number in memory location $0008. Clearly comment and explain your code. Use the skeleton code shown below to implement your subroutine:

```
.ORG  $0046
   …Initialize stack…
   RCALL MIN
   …
   …
.ORG  $0060
MIN:
   …         ; Your code goes here
   …         ;
   RET
```