ECE 375
Computer Organization and Assembly Language Programming
Fall 2021
Homework #2

Note: Your answers for question 1 will be entered directly into Canvas (details available online). Your solutions for questions 2 - 4 will be written and submitted as a PDF document.

[27 pts]

1- Consider the AVR assembly code provided below. Read the code and use your knowledge of assembly instructions to answer the questions given at the bottom of this page.

```
.EQU var1 = 0b11011010
.EQU var2 = 18
.DEF count = r17

.ORG 0x0000
   RJMP  init

.ORG 0x0046
init:
   LDI      XH, HIGH(resultA)
   LDI      XL, LOW(resultA)
   LDI      YH, HIGH(resultB)
   LDI      YL, LOW(resultB)
   CLR      r10
   LDI      r20, var1
   LDI      count, var2
loop:
   CLC
   ROR      r20
   BRCC     skip
   INC      r1
skip:
   ROL      r10
   DEC      count
   BRNE     loop

   ST       X, r10
   ST       Y+, r1
done:
    RJMP    done

.DSEG
.ORG 0x03FE
resultA: .BYTE 1
resultB: .BYTE 1
```

Answer the following questions (assuming that the code has executed and reached the `done` label):

(a) What is the decimal value of the `count` register?

(b) In total, how many times is the instruction "ROR r20" executed?

(c) What are the hexadecimal values of `r27` and `r26`?

(d) What are the hexadecimal values of `r29` and `r28`?

(e) What hexadecimal address does the assembler assign to the label `resultB`?

(f) What 8-bit binary value is stored in SRAM at the memory location specified by `resultA`?

(g) What decimal value is stored in SRAM at the memory location specified by `resultB`?

Now, suppose that the entire program is executed again, with one small difference. Assume that the first line of the program is replaced with:

```
.EQU var1 = 0b10101010
```

Assuming that the code has executed and reached the `done` label:

(h) What 8-bit binary value is stored in SRAM at the memory location specified by `resultA`?

(i) What decimal value is stored in SRAM at the memory location specified by `resultB`?

[24 pts]

2- Consider the following section of AVR assembly code (from the previous problem) with its equivalent (partially completed) address and binaries on the right.

Determine the values for:

(a) `KKKK dddd KKKK` (@ address $004C)

(b) `d dddd` (@ address $004E)

(c) `kk kkkk k` (@ address $004F)

(d) `d dddd` (@ address $0052)

(e) `r rrrr` (@ address $0055)

```
.EQU var1 = 0b11011010
.EQU var2 = 18
.DEF count = r17

.ORG 0x0046                              Address        Binary
      LDI      XH, HIGH(resultA)         0046:          1110  0000  1011  0011
      LDI      XL, LOW(resultA)          0047:          1110  1111  1010  1110
      LDI      YH, HIGH(resultB)         0048:          1110  0000  1101  0011
      LDI      YL, LOW(resultB)          0049:          1110  1111  1100  1111
      CLR      r10                       004A:          0010  0100  1010  1010
      LDI      r20, var1                 004B:          1110  1101  0100  1010
      LDI      count, var2               004C:          1110  KKKK  dddd  KKKK
loop: CLC                                004D:          1001  0100  1000  1000
      ROR      r20                       004E:          1001  010d  dddd  0111
      BRCC     skip                      004F:          1111  01kk  kkkk  k000
      INC      r1                        0050:          1001  0100  0001  0011
skip: ROL      r10                       0051:          0001  1100  1010  1010
      DEC      count                     0052:          1001  010d  dddd  1010
      BRNE     loop                      0053:          1111  0111  1100  1001
      ST       X, r10                    0054:          1001  0010  1010  1100
      ST       Y+, r1                    0055:          1001  001r  rrrr  1001
done: RJMP     done                      0056:          1100  1111  1111  1111
```

[25 pts]

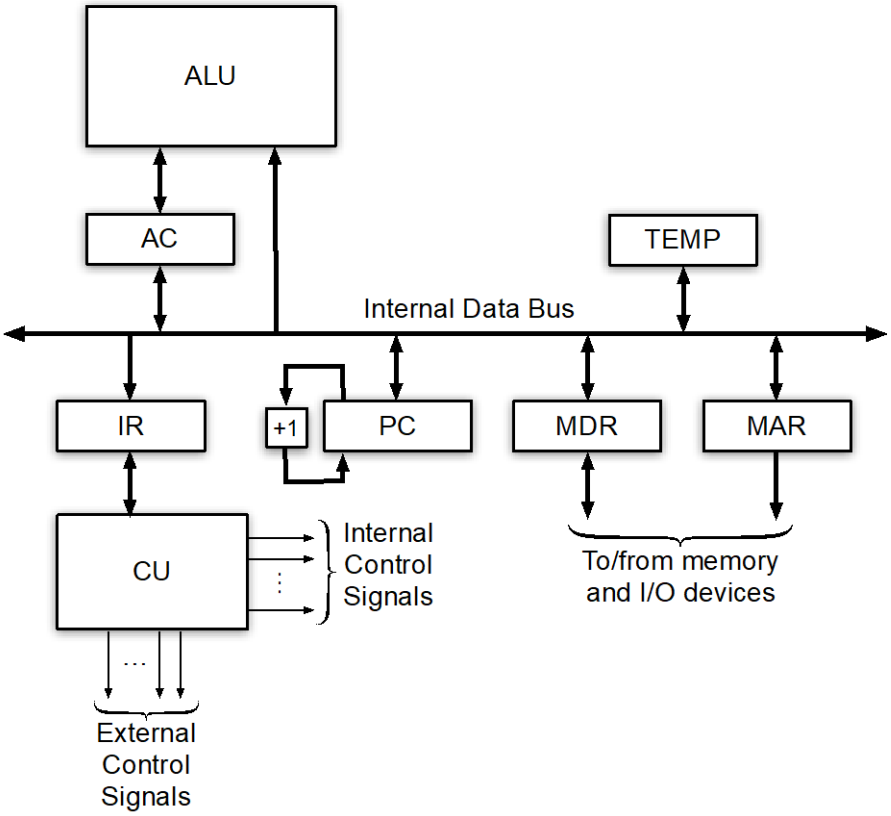
3- Consider the following hypothetical 1-address assembly instruction called "Add Then Store Indirect with Pre-increment" of the form

`ATS +(x)  ; M[x] ← M[x]+1, M[M[x]] ← AC+M[M[x]]`

The instruction will operate using indirect addressing (as illustrated in Figure 2.4a from the textbook).

Suppose we want to implement this instruction on the pseudo-CPU discussed in class augmented with a TEMP register (as shown below). The fetch cycle has been provided below. Give the sequence of *microoperations* required to implement the execute cycle for the above `ATS +(x)` instruction. Be sure to combine multiple register transfer operations into the same microcycle when possible. A correct execute cycle solution should consume no more than 9 microoperations.

Assume an instruction consists of 16 bits: A 4-bit opcode and a 12-bit address. All operands are 16 bits. PC and MAR each contain 12 bits. AC, MDR, and TEMP each contain 16 bits, and IR is 4 bits. Note that the original content of AC should be preserved. Assume PC is currently pointing to the `ATS` instruction and only PC and AC have the capability to increment/decrement themselves.



Fetch Cycle

Cycle 1: MAR ← PC;

Cycle 2: MDR ← M[MAR], PC ← PC+1 ; Read instruction & increment PC

Cycle 3: IR ← MDR_{opcode}, MAR← $MDR_{address}$

[24 pts]

4- In order to make the pseudo-CPU more useful, suppose that we incorporate a *single-port register file* containing 32 8-bit registers (R0-R31). The term *single-port* means that only one register value can be read or written at a time. Suppose the pseudo-CPU is used to implement the AVR instruction `ST -Y, R3`. Give the sequence of microoperations that would be required to Fetch and Execute this instruction (assuming that the AVR architecture uses the little-endian convention). Note that this instruction occupies 16 bits. *Your solution should result in exactly 6 cycles for the fetch cycle and no more than 7 cycles for the execute cycle.* You may assume only the AC and PC registers have the capability to increment/decrement themselves. Assume the MDR register is only 8 bits wide (which implies that memory is organized into consecutive addressable bytes), and AC, PC, IR, and MAR are 16-bits wide. Also, assume the Internal Data Bus is 16 bits wide and thus can handle 8-bit or 16-bit (as well as portions of 8-bit or 16-bit) transfers in one microoperation. Clearly state any other assumptions made.

Hint: Remember that the Y register is actually formed from two individual 8-bit registers. They can be accessed from the Register File.