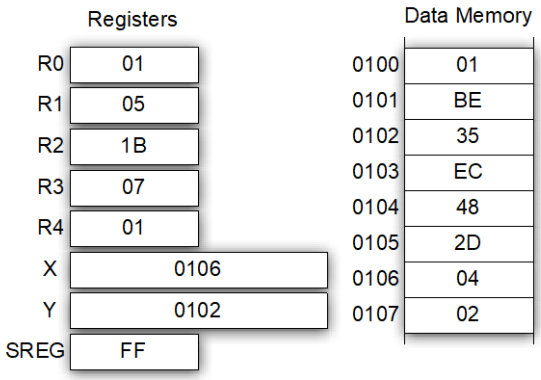
ECE 375
Computer Organization and Assembly Language Programming
Fall 2021

[22 pts]

1- Based on the initial register and data memory contents shown below (represented in hexadecimal), show how these contents are modified (in hexadecimal) after executing each of the following AVR assembly instructions. Do not be concerned about what happens to the Status Register (SREG) *after* the operation.
Each instruction is independent from the others.

(a) `add  R3, R2`
(b) `ld  R0, X+`
(c) `andi  R3, $42`
(d) `muls  R1, R28`
(e) `sbr  R26, 15`
(f) `ser  R2`



[15 pts]

2- Consider a CPU with 3-address format that has a memory unit with a capacity of 3000 words and supports 112 unique instructions. The instruction word is divided into five parts: *opcode* (Opcode) field, *indirect addressing mode* (I) bit, and the three *address* (Address) fields. For your information, given an address, indirect addressing is where

- If I-bit = 0, the operand is located in M[address].
- If I-bit = 1, the operand is located in M[M[address]].

Indicate the number of bits required for each part of the instruction. Also calculate the total number of bits needed to represent an instruction.

[15 pts]

3- Explain the meaning of each term in the list below. Be sure that you describe how each item is used within the pseudo-computer discussed in class.

(a) Operation code (opcode)
(b) ALU
(c) Effective Address
(d) Program Counter (PC)
(e) Internal Data Bus

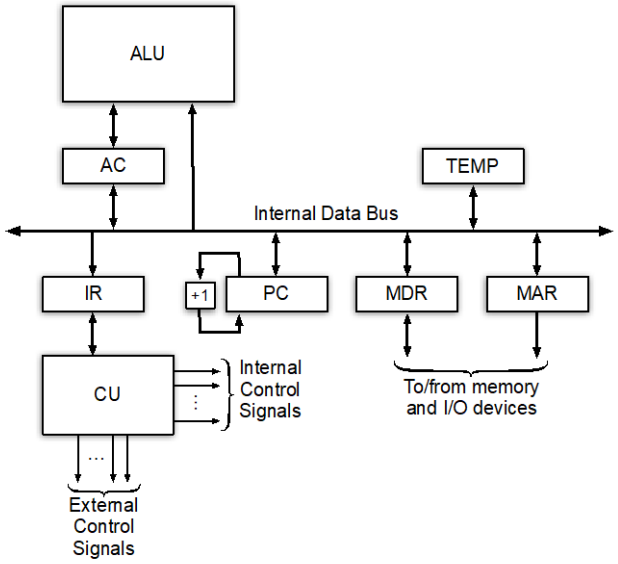
[24 pts]

4- Imagine that you want to introduce a new assembly code instruction: "Increment and Save" of the form

```
INCS   Y       ; M(Y) ← M(Y) + 1
```

Within the execute cycle this instruction increments M(Y) (placing the new value back into memory). Be sure that the original value of the AC register is restored prior to the end of your execute cycle.

We want to implement this instruction on the pseudo-CPU shown below. Note that this implementation is nearly identical to our example from lecture, except that this version has been augmented with an additional register named "TEMP" which can be used to hold temporary values as desired. 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. Give the sequence of *microoperations* required to implement the Execute cycle for the above INCS instruction (the Fetch cycle is given below). Your solution for the execute cycle should result in no more than 5 microcycles. **Hint: On these types of problems, it is always acceptable to combine multiple register transfer operations into the same microcycle when possible.** Assume PC is currently pointing to the INCS instruction and that only PC and AC have the capability to increment/decrement themselves.



Fetch Cycle

Step 1: MAR ←PC;

Step 2: MDR ←M(MAR), PC ← PC+1 ; Read inst. & increment PC

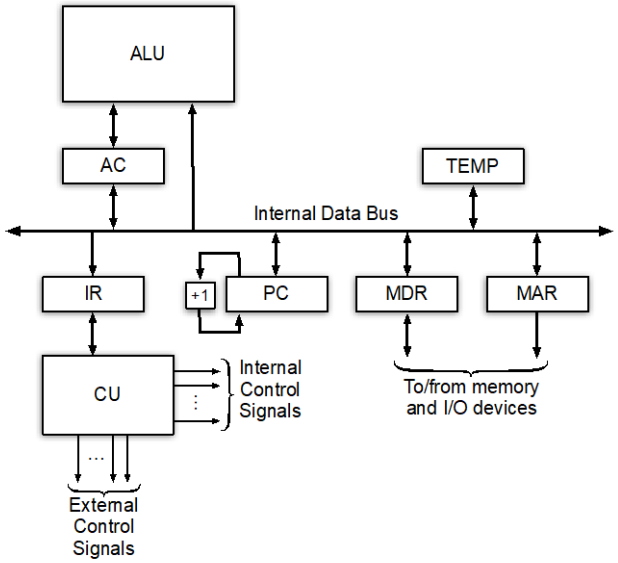
Step 3: IR ← MDR_{opcode}, MAR← $MDR_{address}$

[24 pts]

5- Consider the following hypothetical 1-address assembly instruction called "Store Accumulator Indirect with Post-decrement" of the form

STA (x)- ; M(M(x)) ← AC, M(x) ← M(x) - 1

Suppose we want to implement this instruction on the pseudo-CPU discussed in class (with an additional register TEMP as before). 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. Give the sequence of *microoperations* required to implement the Execute cycle (the Fetch cycle is given below) for the above STA (x)- instruction. Your solution should result in no more than 8 microcycles (again, combine multiple register transfer operations into the same microcycle when possible). Assume that the PC is currently pointing to the STA (x)- instruction and only PC and AC have the capability to increment/decrement themselves. Be sure that the original value of the AC register is restored prior to the end of your execute cycle.



Fetch Cycle

Step 1: MAR ←PC;

Step 2: MDR ←M(MAR), PC ← PC+1 ; Read inst. & increment PC

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