Winnie_Woo_HW1

Thursday, October 6, 2022   2:38 PM

ECE 375
Computer Organization and Assembly Language Programming
Fall 2022
Assignment #1

[20 pts]
1- (a) Suppose a processor or CPU supports 64 different instructions and has a memory of 1K (K = 1024) words. Determine the size of each memory word for the following instruction formats:
(i) 3-address instruction format
(ii) 2-address instruction format
(iii) 1-address instruction format

(b) Explain the advantages and disadvantages of the three instruction formats.

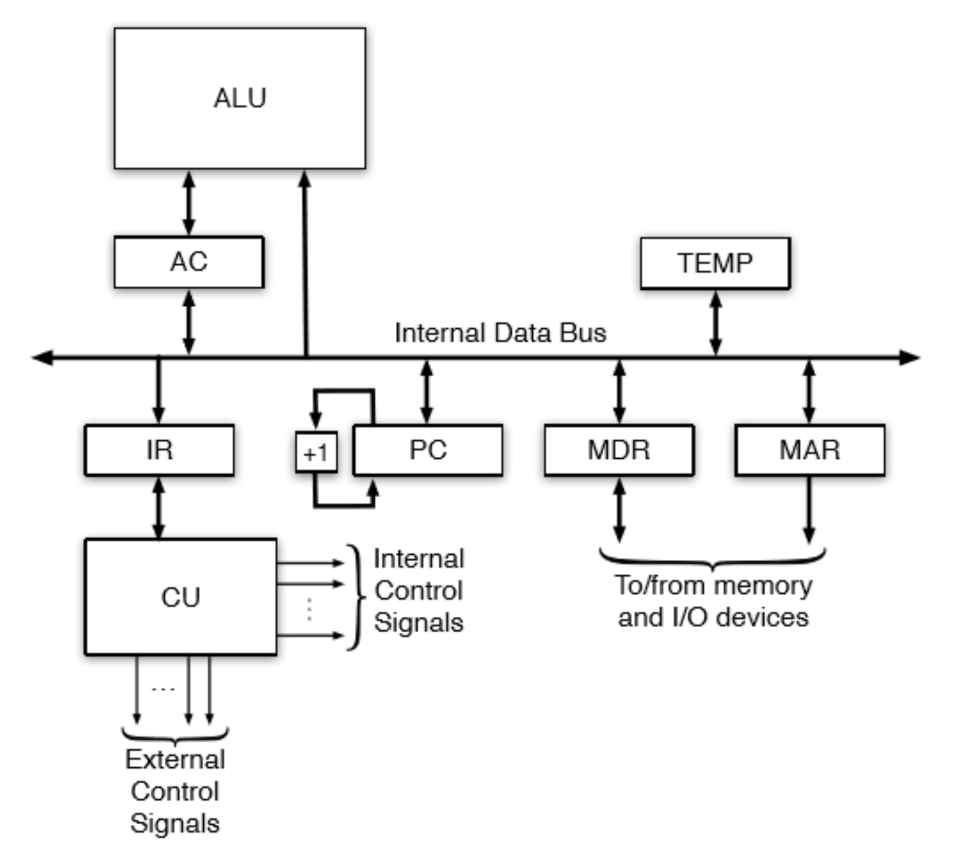
[20 pts]
2- Show a block diagram of the hardware necessary (similar to Figures 3.12 - 3.13 in the textbook) to connect AC to the Internal Data Bus and the ALU in the pseudo-CPU using a tri-state buffer and a multiplexer (MUX).

[20 pts]
3- For the pseudo-CPU shown in Figure 3.9 in the textbook, explain whether or not each of the following microoperations can be performed in a single clock cycle. Assume PC and MAR each contain 12 bits, AC and MDR each contain 16 bits, and IR is 4 bits.
(a) MAR ← AC, MDR ← MAR
(b) IR ← MDR, MAR ← MDR
(c) MAR ← MDR, MDR ← M(MAR)
(d) MDR ← AC + 1
(e) AC ← MDR, PC ← PC + 1
(f) PC ← PC + AC

[20 pts]
4- Consider the following hypothetical 1-address assembly instruction called "Add Then Store Indirect with Pre-decrement" of the form

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

Suppose we want to implement this instruction on the pseudo-CPU discussed in class augmented with TEMP register as shown below. Give the sequence of *microoperations* required to implement the Execute cycle (Fetch cycle is given below) for the above ADDTHENSTORE -(x) instruction. *Your solution should result in exactly 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 ADDTHENSTORE instruction and only PC and AC have the capability to increment/decrement itself.



Fetch Cycle
Cycle 1: MAR ← PC;
Cycle 2: MDR ← M[MAR], PC ← PC+1   ; Read inst. & increment PC
Cycle 3: IR ← MDR_{opcode}, MAR ← $MDR_{address}$

[20 pts]
5- 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. *Instructions are unrelated.*

```
(i)   ldi   r27, 85
(ii)  ror   r2
(iii) adc   r2, r1
(iv)  sts   $0007, r28
(v)   sbiw  XH:XL, 2
```

Registers

Register	Value
R0	01
R1	05
R2	1B
R3	07
R4	01
X	0106
Y	0102
SREG	FF

Data Memory

Address	Value
0100	01
0101	BE
0102	35
0103	EC
0104	48
0105	2D
0106	04
0107	02

Memory size = 1K $2^{10} = 1024$
$\log_2 64 = 6$ bits

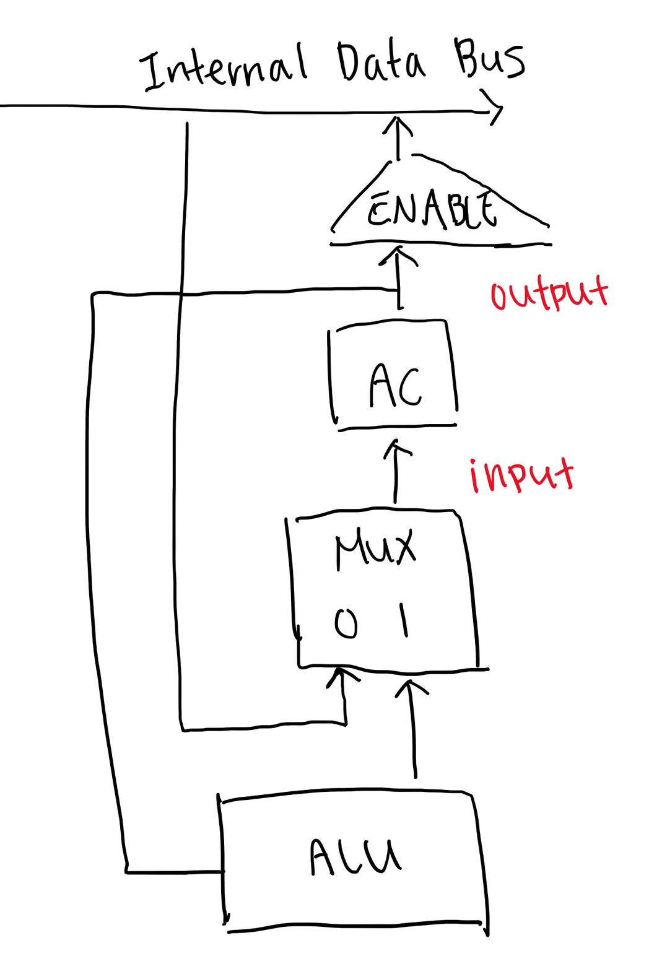
1a.
i. | opcode (6 bits) | Address (10 bits) | Address (10 bits) | Address (10 bits) | → 36 bits

ii. | opcode (6 bits) | Address (10 bits) | Address (10 bits) | → 26 bits

iii | opcode (6 bits) | Address (10 bits) | → 16 bits

b. Advantages are that it requires shorter programs. However the disadvantage is that it requires more number of bits.

2.



3a. MAR ← AC, MDR ← MAR
Can't be performed in the same cycle because we're using the data stored in MAR

b. IR ← MDR, MAR ← MDR
Can be performed in the same cycle because its reading data from the MDR to 2 different locations

c. MAR ← MDR, MDR ← M(MAR)
Can't be performed in the same cycle because MDR is being read & writing to the same source

d. MDR ← AC+1
Can't be performed in the same cycle because 1 is being added to AC at the same time its being read

e. AC ← MDR, PC ← PC+1
Can be performed in the same cycle because PC is not used in the first call

f. PC ← PC + AC
Can't be performed in the same cycle because you're modifying & writing to it

4.

1. temp ← AC
2. AC ← MDR, MDR ← M[MAR]
3. AC ← AC+1
4. MDR ← AC
5. M[MAR] ← MDR, MAR ← MDR
6. MDR ← M[MAR]
7. AC ← AC+MDR
8. M[MAR] ← MDR, MDR ← AC
9. AC ← temp

5 i. ldi r27, 85   * load upper byte
r27 = 0x55

ii. ror r2   * rotate right value
0x1B, 00011011
= 0x8D

iii adc r2, r1   * add with carry
1B + 05 = 0x20

iv. sts $0007, r28   * stores 1 Byte from R28 to 0007
r28 = 0x02
$0007 = 0x02

v. sbiw XH:XL, 2   * subtract r27:r26 - 2
r27:r26 - 2 = 0x104