COMP1411 (Spring 2022) Introduction to Computer Systems

Individual Assignment 2 Duration: 00:00, 19-Mar-2022 ~ 23:59, 20-Mar-2022

Name	**ZHOU, Siyu**
Student number	

Question 1. [3 marks]

In this question, we use the Y86-64 instruction set (please refer to Lecture 4-6).

1(a) [1 mark]

Write the machine code encoding of the assembly instruction:

"`mrmovq 0x15F(%rbx), %rax`".

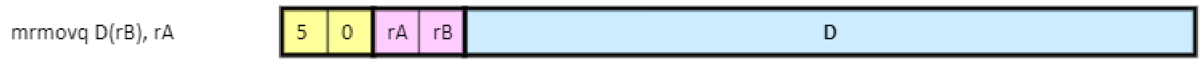
Please write the bytes of the machine code in hex-decimal form, i.e., using two hex-decimal digits to represent one byte. You are allowed to leave spaces between adjacent bytes for better readability. The machine has a little-endian byte ordering.

Show your steps. Only giving the final result will NOT get a full mark of this question.

Answer:

`mrmovq 0x15F(%rbx), %rax`

follows this instruction:



icode: ifun = 50

rA = 0

rB = 3

D = 15F

50 03 5F 01 00 00 00 00 00 00

1(b) [2 marks]

Consider the execution of the instruction "`mrmovq 0x15F(%rbx), %rax`". Assume that for now, the data in register `%rbx` is 0x200, just before executing this instruction, the value of PC is 0x420. We use "**vm**" to represent the data read from the main memory.

Describe the steps done in the following stages: Fetch, Decode, Execute, Memory, Write Back, PC update, by filling in the blanks in the table below.

Note that you are required to fill in the generic form of each step in the second column; and in the third column, fill in the steps for the instruction "`mrmovq 0x15F(%rbx), %rax`" with the above given values. If you think there should not be a step in some stage, just leave the blanks unfilled.

The symbol "←" means reading something from the right side and assign the value to the left side. X:Y means assign the highest 4 bits of a byte to X, and assign the lowest 4 bits of the byte to Y.

Answer:

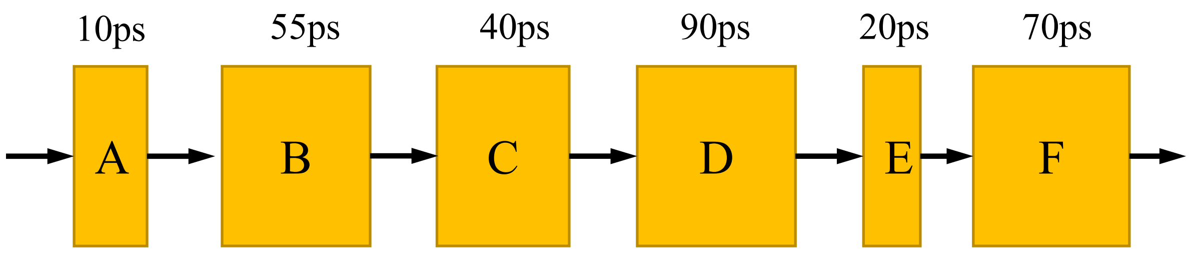
Stages	**mrmovq D(rB), rA**	**mrmovq 0x15F(%rbx), %rax**

Fetch	icode: ifun ← **$\mathbf{M_1}$[PC]**
	rA:rB ← **$\mathbf{M_1}$[PC+1]**
	valC ← **$\mathbf{M_8}$[PC+2]**
	valP ← **PC+10**
		icode: ifun ← **$\mathbf{M_1}$[0x420]=5:0**
		rA:rB ← **$\mathbf{M_1}$[0x421] = 0:3**
		valC ← **$\mathbf{M_8}$[0x422] = 0x15F**
		valP ← **0x420+10 = 0x42A**

Decode	valB ← **R[rB]**	valB ← **R[%rbx] = 0x200**
Execute	valE ← **valB+valC**	valE ← **0x200 + 0x15F = 0x35F**
Memory	valM ← M_8[**valE**]	valM ← M_8[**0x35F**]
Write back	R[**rA**] ← **valM**	R[**%rax**] ← **valM = $\mathbf{M_8}$[0x35F]**
PC update	PC ← **valP**	PC ← **0x42A**

Question 2. [3 marks]

Suppose a combinational logic is implemented by 6 serially connected components named from A to F. The whole computation logic can be viewed as an instruction. The number on each component is the time delay spent on this component, in time unit ps, where 1ps = 10^{-12} second. Operating each register will take 20ps.



Throughput is defined as how many instructions can be executed on average in one second for a pipeline, and the unit of throughput is IPS, instructions per second.

Latency refers to the time duration starting from the very first component and ending with the last register operation finished, the time unit for latency is ps.

For throughput, please write the result in the form X.XX * 10^Y IPS, where X.XX means one digit before the dot and two fractional digits after the dot, and Y is the exponent.

2(a) Make the computation logic a 3-stage pipeline design that has the maximal throughput. Note that a register shall be inserted after each stage to separate their combinational logics. [1.5 marks]

- Please answer how to partition the stages.
- Please compute the throughput and latency for your pipeline design, with steps.

How to partition:
Add cut between C, D and D, E.
10 + 55 + 40 +20 = 135ps = 1.35 * 10^{-10} second
1.35 * 10^{-10} = 3.75
Throughput = 1/(1.35 * 10^{-10}) = 7.41*10^9 IPS

2(b) Make the computation logic a 4-stage pipeline design that has the maximal throughput. Note that a register shall be inserted after each stage to separate their combinational logics. [1.5 marks]

- Please answer how to partition the stages.
- Please compute the throughput and latency for your pipeline design, with steps.

Add cut between B&C, C&D and D&E
90+20 = 225ps = 1.10 * 10^{10} s
Throughput = 1/(2.25 * 10^{-10}) = 9.09*109 IPS

Question 3. [4 marks]

The following byte sequence is the machine code of a function compiled with the Y86-64 instruction set (refer to Lecture 6). The memory address of the first byte is 0x200. Note that the byte sequence is written in hex-decimal form, i.e., each number/letter is one hex-decimal number representing 4 binary bits, and two numbers/letters represent one byte. Assume the machine is a big-endian byte order machine. Assume that by default the value in register %rax will be returned. The machine has a little-endian byte ordering.

30 F0 0000000000000028

30 F3 0000000000000000

30 F1 0000000000000002

70 00 000000000022B

60 03

61 10

62 00

76 0000000000000227

20 30

90

Please write out the assembly instructions (in Y86-64 instruction set) corresponding to the machine codes given by the above bytes sequence, and explain what this function is computing.

Memory address	**Bytes**	**Machine codes**	**Meaning**
0x200	**30 F0 80 00 00 00 00 00 00**	**irmovq 0x28, %rax**	**Set %rax as 40**
0x20A	**30 F3 00 00 00 00 00 00 00 00**	**irmovq $0, %rbx**	**Set %rbx as 0**
0x214	**30 F1 02 00 00 00 00 00 00 00**	**irmovq $2, %rcx**	**Set register %rcx as 2**
0x21E	**70 2B 01 00 00 00 00 00 00**	**jmp 0x22B**	**Jump to the address 0x022B**
0x227	**60 03**	**addq %rax, %rbx**	**Add %rax and %rbx and put answer in %rbx**
0x229	**61 10**	**subq %rcx, %rax**	**Subtract %rax by %rcx and put it in %rax**
0x22B	**62 00**	**andq %rcx, %rax**	**Execute the & operation to %rax and %rax, and put answer in %rax**

0x22D	**76 17 02 00 00 00 00 00 00**	**Jg 0x227**	**Jump the address of 0x227 when b > a (a, b are register in previous instruction)**
0x236	**20 30**	**rrmovq %rbx, %rax**	**Move data from %rbx to %rax**
0x238	**90**	**ret**	**Stop execution**

This function is computing sum of even number in 40 to 0.