COMP1411 (Spring 2024) Introduction to Computer Systems

Individual Assignment 2 Duration: 00:00, 16-Mar-2024 ~ 23:59, 18-Mar-2024

Name	**SUN Hansong**
Student number	**23097775D**

Question 1. [1 mark]

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

1(a) [1 mark]

Write the machine code encoding of the assembly instruction:

```
rmmovq %rax, 0x1124(%rsp)
```

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*:**

The icode: ifun of the operation rmmovq is 4:0.

And rA:rB is 0:4 according to the table.

And D is 2411000000000000 in little-endian.

So the encoding result is 40042411000000000000.

1(b) [3 marks]

Consider the execution of the instruction "`rmmovq %rax, 0x1124(%rsp)`". Assume that the data in register `%rsp` is 0x456, the value of PC is 0x360. Use "**`vm`**" to represent the data that will be written to 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 "`rmmovq %rax, 0x1124(%rsp)`" with the above given values.

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	**rmmovq rA, D(rB)**	**rmmovq %rax, 0x1124(%rsp)**

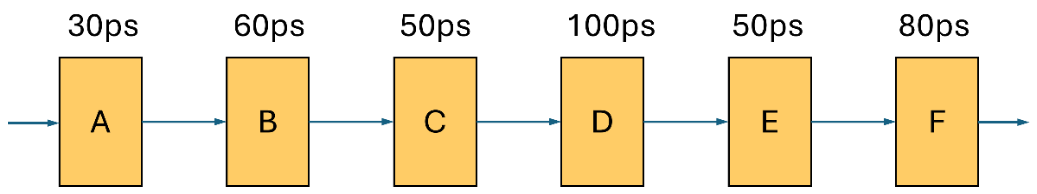
Fetch	icode: ifun ← M_1[PC]
	rA:rB ← M_1[PC+1]
	valC ← M_8[PC+2]
	valP ← PC+10
		icode: ifun ← M_1[0x360] = 4:0
		rA:rB ← M_1[0x360+1] = 0:4
		valC ← M_1[0x360+2] = 0x1124
		valP ← 0x360+10 = 0x36A

Decode	valA ← R[rA]
	valB ← R[rB]
		valA ← R[%rax] = vm
		valB ← R[%rsp] = 0x456

Execute	valE ← valB+valC	valE ← 0x456 + 0x1124 = 0x157A
Memory	M_8[valE] ← valA/vm	M_8[0x157A] ← valA/vm
Write back		
PC update	PC ← valP	PC ← 0x36A

Question 2. [2 marks]

Suppose a combinational logic is implemented by 6 serially connected components 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 in the long run, 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.

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. By default, a register will be inserted after the last stage, i.e., after step F.

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

To make the throughput maximum, we need to make the time of three parts approximate.

Case 1: AB | CD | EF, the latency is (50+100+20)*3 = 510ps

Case 2: ABC | D | EF, the latency is (30+60+50+20)*3 = 480ps

It can be shown that Case 2 has the least latency.

So, the best way to partition the stages is ABC | D | EF.

In this case, the throughput = 3*1/((30+60+50+20)*10^{-12}) = 1.875*10^{10} IPS

And the latency is (30+60+50+20)*3 = 480ps

Question 3.   [4 marks]

The following byte sequence is the machine codes of a C 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 format, 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 little-endian byte order machine.** Assume that by default the value in register %rax will be returned.

30F314000000000000000030F10200000000000000063006233732E02000000000000006030611370160200000000000000090

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.

According to the Y86-64, the sequence can be divided into:

0x200: 30|F3|1400000000000000

0x20A: 30|F1|0200000000000000

0x214: 63|00

0x216: 62|33

0x218: 73|2E02000000000000

0x221: 60|30

0x223: 61|13

0x225: 70|1602000000000000

0x22E: 90

And the instructions are:

irmovq $0x14 %rbx   R[%rbx]=0x14

irmovq $0x02 %rcx   R[%rcx]=0x02

xorq %rax %rax   R[%rax]=0

andq %rbx %rbx   R[%rbx]=0x14 ZF=0

je 0x22E

addq %rbx %rax

subq %rcx %rbx

jmp 0x216

ret

We can find that the instructions between je and jmp form a loop. The exit signal of the loop is R[%rbx] = 0

And the results in each time are the following:

1. R[%rax] = 0 + 0x14 = 0x14, R[%rbx] = 0x14 – 0x02 = 0x12, R[%rbx] != 0
2. R[%rax] = 0x14 + 0x12 = 0x26, R[%rbx] = 0x12 – 0x02 = 0x10, R[%rbx] != 0
3. …
4. R[%rax] = 0x14 + 0x12 + 0x10 + 0x0E + 0x0C + 0x0A + 0x08 + 0x06 + 0x04 + 0x02 = 0x6E, R[%rbx] = 0x02 – 0x02 =0, R[%rbx] = 0

So, the final return value R[%rax] = 0x6E