COMP1411 (Spring 2023)   Introduction to Computer Systems

(UPDATED on 22PM, 10-Mar-2023)

Individual Assignment 2   Duration: 12:00, 10-Mar-2023 ~ 23:59, 12-Mar-2023

Name	**Zhu Jin Shun**
Student number	**22101071d**

Question 1.   [3 marks]

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:

```
mrmovq 0x1356(%rbp), %rdi
```

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 D(rB), rA (8 bytes)

mrmovq 0x1356 (%rbp), %rdi

mrmovq- 50

rB=%rbp=5

rA=%rdi=7

rA:rB=75

0x1356=56 13 00 00 00 00 00 00

Hex-decimal form:

50 75 56 13 00 00 00 00 00 00

1(b) [2 marks]

Consider the execution of the instruction "`mrmovq 0x1356(%rbp), %rdi`". Assume that for now, the data in register `%rbp` is 0x334 just before executing this instruction, the value of PC is 0x540. 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 0x1356(%rbp), %rdi`" 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 0x1356(%rbp), %rdi**

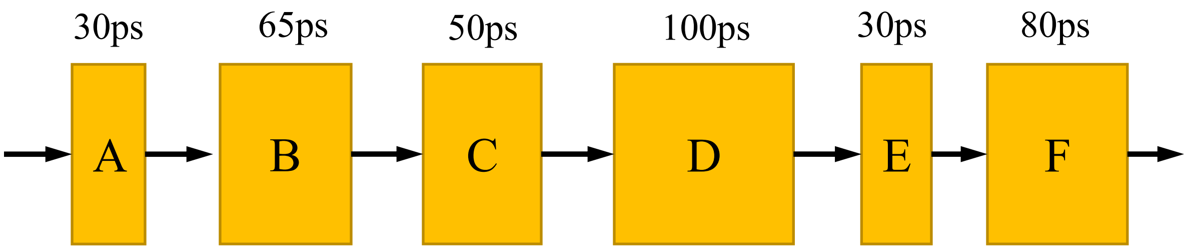
Fetch	icode: ifun ← M_1[PC]
	rA:rB ← M_1[PC+1]
	valC ← M_8[PC+2]
	valP ← PC+10
		icode: ifun ← M_1[0x540]=5:0
		rA:rB ← M_1[0x540+1]=7:5
		valC ← M_8[0x540+2]=0x1356
		valP ← 0x540+0x00A=0x54A

Decode	valB ← R[rB]	valB ← R[%rbp]=0x334
Execute	valE ← valB+valC	valE ← _0x1356+0x334=0x168A
Memory	valM ← M_8[valE]	valM ← M_8[0x168A]

Write back	R[rA] ← valM	R[%rdi] ← vm(valM)
PC update	PC ← valP	PC ←0x54A

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 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.

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. By default, a register will be inserted after the last stage, i.e., after step F. [1.5 marks]

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

Three stages

One way that results in the best throughout and latency out of total 10 possibilities.

ABC| D | EF

145ps| 100ps | 110ps

Throughput=1/((145+20)*10^-12)=6.06*10^9 IPS

Latency= (145+20) *3=495ps

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. By default, a register will be inserted after the last stage, i.e., after step F. [1.5 marks]

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

Four Stages:

One way that results in the best throughout and latency out of total 10 possibilities.

AB | C |D | EF

95ps | 50ps |100ps | 110ps

Throughput=1/((110+20)*10^-12)=7.69*10^9 IPS

Latency= (110+20)*4=520 ps

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 0x1500. 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 little-endian byte order machine.** Assume that by default the value in register %rax will be returned.

30F33200000000000000030F10000000000000000030F00100000000000
000702B150000000000000603161036233762715000000000000000201090

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.

Overall diagram:

Address	Machine code	Assembly code
0x1500	**30 F3 32 00 00 00 00 00 00 00**	irmovq $50,%rbx
0x150A	**30 F1 00 00 00 00 00 00 00 00**	irmovq $0,%rcx
0x1514	**30 F0 00 00 00 00 00 00 00 01**	irmovq $72057594037927926,%rax
0x151E	**70 2B 15 00 00 00 00 00 00**	jmp 0x152B
0x1527	**6031**	addq %rbx, %rcx
0x1529	**6103**	subq %rax,%rbx
0x152B	**6233**	andq %rbx,%rbx
0x152D	**76 27 15 00 00 00 00 00 00**	jg 0x1527
0x1536	**20 10**	rrmovq %rcx, %rax
0x1538	**90**	ret

Process and use of each function:

30 F3 32 00 00 00 00 00 00 00

30-irmovq;

F3 -operands field;

F:no register 3:%rbx

32 00 00 00 00 00 00 00= 0x32=3*16^1+2^16^0=50

0x1500: irmovq $50,%rbx

This function is used to move value $50 into register %rbx. %rbx=50

30 F1 00 00 00 00 00 00 00 00

30-irmovq;F1 -operands field;

F:no register 1:%rcx

00 00 00 00 00 00 00 00 =0

0x1500+0x00010=0x150A

0x150A: irmovq $0,%rcx

This function is used to move value 0 into register %rcx. %rcx=0

30 F0 00 00 00 00 00 00 00 01

30-irmovq

F0 -operands field; F:no register 0:%rax

00 00 00 00 00 00 00 01 =0*16^0+0*16^1+0*16^2+0*16^3+0*16^4+0*16^5+0*16^6+0*16^7+1*16^8=72057594037927926

0x150A+0x00010=0x1514

0x1514: irmovq $72057594037927926,%rax

This function is used to move value $72057594037927926 into register %rax. %rax=$72057594037927926

70 2B 15 00 00 00 00 00 00

70-jmp value:0x152B

0x1514+0x00010=0x151E

0x151E :jmp 0x152B

This function is to jump to address 0x152B

60 31

60:addq

3:%rbx

1:%rcx

0x151E+0x0009=0x1527

0x1527:addq %rbx, %rcx

This function is to add the value in %rbx into %rcx. %rcx= rcx%+%rbx

61 03

61:subq

0:%rax

3:%rb

0x1527+0x0002=0x1529

0x1529: subq %rax,%rbx

This function is to subtract %rbx with %rax. %rbx=%rbx -%rax

62 33

62:andq

3:%rbx

3:rbx

0x1529+0x0002=0x152B

0x152B:andq %rbx,%rbx

This function is to calculate the same value of %rbx calculated before with the %rbx after. %rbx= %rbx &%rbx.

76 27 15 00 00 00 00 00 00

76:jg

2715000000000000=0x1527

0x152B+0x0002=0x152D

0x152D: jg 0x1527

This function is to jump to address 0x1527 under condition ~(SF^OF)& ~ZF and will be true if it is greater.

20 10

20:rrmovq

1:%rcx

0:%rax

0x152D+0x0009=0x1536

0x1536:rrmovq %rcx, %rax

This function is to move the value in register %rcx into register %rax

%rax=%rcx

90

90:ret

0x1536+0x0002=0x1538

0x1538:ret

This function is to return value

The whole function is a while loop and it adds values to %rbx, %rcx in a sequence and then calculates using addq, andq, subq then returns the final %rax value by moving the %rcx value calculated.

During the process, because of operator jg, the identify is jg=~(SF^OF)& ~ZF, it will have two situations.

If and%rbx,%rbx>0 it returns to 1

If and%rbx,%rbx <=0 it returns to 0