COMP1411 (Spring 2021) Introduction to Computer Systems

Take-home exam Due Date: 23:59pm, 8th May, 2021

Answer Book

Name	
Student number	

Instructions:

- Please fill in your name and student number in the above table.
- Please type your answers into this answer book, and then submit this answer book. Scanned images of hand-writing will not be accepted.
- You must answer questions by yourself only. You are not allowed t

Question 1. [6 marks]

Note that 83 is (1010011) in binary,

which equals 2^0+2^1+2^4+2^6. So x * 83 = x * (2^0+2^1+2^4+2^6) = x * 2^0 + x * 2^1 + x * 2^4 + x * 2^6. Because 2^i = (1<<i), we can rewrite the original statement using only + and << operators on x.

Rewrite:

y = x + (x<<1) + (x<<4) + (x<<6)

Question 2. [14 marks]

My SID = 0x200******.

Binary form is: 0010 0000 0000 1000 0100 0100 0101 1001

****Assembly code Process:****

1. After "sarl $21, %eax",
    1. %eax = 0000 0000 0000 0000 0000 0001 0000 0000
2. After "sall $21, %eax",
    1. %eax = 0010 0000 0000 0000 0000 0000 0000 0000
3. After "movl $0xC1E00000, %ebx",
    1. %ebx = 1100 0001 1110 0000 0000 0000 0000 0000
4. After "andl %ebx, %eax",
    1. %eax = 0000 0000 0000 0000 0000 0000 0000 0000
5. After "movl $0x40000000, %ecx",
    1. %ecx = 0100 0000 0000 0000 0000 0000 0000 0000
6. After "orl %ecx, %eax",
    1. %eax = 0100 0000 0000 0000 0000 0000 0000 0000

****Convertion Process:****

- Sign bit = 0 so it's positive.
    - s=0.

- Exp bits = 1000 0000.
    - E = Exp-Bias = 128 - (2^7-1) = 128 - 127 = 1

- M bits = 000 0000 0000 0000 0000 0000.
    - Frac = 0.
    - M = 1.Frac = 1.0

So in the end the decimal is (-1)^0 * 1.0 * 2^1 = 2.0

Answer is 2.0

Question 3. [8 marks]

5.5 in binary form is 101.1 = 1.011 * 2^2

- Since -5.5 is negative, the sign bit is 1.
- E = Exp - Bias = Exp - (2^5-1) = 2.
  - Exp = 33.
  - The binary form is 100001
- Frac = 011

So the final result in 10-bit binary format is: 1 100001 011

Question 4. [22 marks]

4(a)

****Convention:****

1. t1, t2, ... means time 1, time 2.... .
2. I1, I2, ... means instruction 1, instruction 2, ...
3. Starting time is inclusive but ending time is exclusive, e.g. starting time = t0 and ending time = t10 means the instruction starts at the beginning of time t0 and ends immediately before time t10 (not using time t10).

****Execution of each instruction:****

1. irmovq $-1, %rax:
    1. Starting time: t0
    2. F: t0+10 = t10
    3. D: t10 + 1 = t11
    4. E: t11 + 2 = t13
    5. M: t13 + 1 = t14
    6. W: t14 + 2 = t16
    7. P: t16 + 1 = t17
    8. Ending time: t17
2. xorq %rax, %rax
    1. Starting time: t10, after the previous instruction finished its F stage.
    2. F: t10 + 10 = t20
    3. D: t20 + 2 = t22
    4. E: t22 + 2 = t24
    5. M: t24 + 1 = t25
    6. W: t25 + 2 = t27

7. P: t27 + 1 = t28
8. Ending time: t28

3. rmmovq %rbx, 10(%rcx)
    1. Starting time: t20, after the previous 2 instructions finished their F stages.
    2. F: t20 + 10 = t30
    3. D: t30 + 2 = t32
    4. E: t32 + 2 = t34
    5. M: t34 + 12 = t46
    6. W: t46 + 1 = t47
    7. P: t47 + 1 = t48
    8. Ending time: t48
4. rmmovq %rax, 20(%rcx)
    1. Starting time: t30, after the previous 3 instructions finished their F stages.
    2. F: t30 + 10 = t40
    3. D: t40 + 2 = t42
    4. E: t42 + 2 = t44
    5. M: t46 + 12 = t58 (expects to start at t44 but has to wait until I3 completes its M stage at t46)
    6. W: t58 + 1 = t59
    7. P: t59 + 1 = t60
    8. Ending time: t60
5. jg .L1: (xorq has set the ZF=1 because xoring with itself always resulting in zero so no branch taken)
    1. Starting time: t40, after the previous 4 instructions finished their F stages.
    2. F: t40 + 10 = t50
    3. D: t50 + 1 = t51
    4. E: t51 + 2 = t53
    5. M: t58+1 = t59 (waits until I4's completion on M)
    6. W: t59 + 1 = t60
    7. P: t60 + 1 = t61

8. Ending time: t61

6. rrmovq %rbx, %rax

    1. Starting time: t50, after the previous 5 instructions finished their F stages.

    2. F: t50 + 10 = t60

    3. D: t60 + 2 = t62

    4. E: t62 + 2 = t64

    5. M: t64 + 1 = t65

    6. W: t65 + 2 = t67

    7. P: t67 + 1 = t68

    8. Ending time: t68

7. ret

    1. Starting time: t68, after the previous 6 instructions finished their F stages.

    2. Ending time: t68. ret instruction takes 0 time to execute by the given assumption.

4(b)

The main bottleneck comes from the fact that accessing data memory (the M stage) takes too much time such that it not only increases the execution time for a single instruction but also delays following instructions from entering their M stages. This phenomenon can be best illustrated by I3 and I4:

1. The M stage takes 12 time, which is approximately the same amount of time to execute another instruction without operation in M stage.

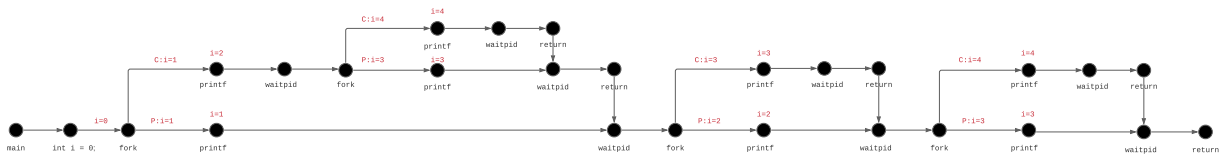
2. Due to the prolonged access time of I3's M stage, I4 and I5 have to wait for extra 2 and 5 time units to enter their M stages respectively.

To mitigate this performance loss, we can introduce L1 data cache to the system. Note that the two memory addresses accessed at I3 and I4 are off by 10 only, which implies that the first cache miss at I3 must bring in cache the content at I4's access address too thus securing a cache hit at I4.If the pipeline is not evenly allocated, both F and M times will be reduced by the cache

Besides, we even can put more transistors on the CPU to improve the efficiency.

Question 5. [25 marks]

5(a)



5(b)

1. After the first fork at main after i=0, there are 2 feasible outputs : 12 or 21, depending on whether the parent prints first or the children prints first. We call the parent and children as P1 and C1 respectively.

2. After the second fork at C1, there are 2 branches C1 and C2. The prints at C1 and C2 have 2 feasible outputs: 234 or 243, depending on whether C1 prints first or C2 prints first after C1's second fork. Combining this with P1's first print, there are 8 feasible outputs depending on where P1's print gets inserted: taking 234 as an example, P1's print can be inserted as 1234, 2134, 2314, 2341. So 8 feasible outputs in total now.

3. Let C3 be the child branch born after P1's second fork. There are two feasible outputs: 23 or 32, depending on whether P1 prints first or C3 prints first. Combining this with previous results, there are 8 * 2 = 16 feasible outputs in total.

4. Let C4 be the child branch born after P1's third fork. There are two feasible outputs: 34 or 43, depending on whether P1 prints first or C4 prints first. Combining this with previous results, there are 16 * 2 = 32 feasible outputs in total.

Answer: 32 feasible outputs.

List the possible outputs :

12343234; 12343243; 12342334;12342343;

21343234; 21343243; 21342334; 21342343;

23143234; 23143243; 23142334; 23142343;

23413234; 23413243; 23412334; 23412343;

12433234; 12433243; 12432334; 12432343;

21433234; 21433243; 21432334; 21432343;

24133234; 24133243; 24132334; 24132343;

24313234; 24313243; 24312334; 24312343;

Question 6. [25 marks]

6(a)

The blocks:

```assembly
; Bibj means Block i, byte j
irmovq $1, %rsi ;B1b0 ~ B1b9, Block 1
irmovq $4, %rdi ;B1b10 ~ B2b3, Block 1 ~ 2
irmovq $0, %rax ;B2b4 ~ B2b13, Block 2
addq %rsi, %rax ;B2b14 ~ B2b15, Block 2
L0:
rrmovq %rax, %rbp ;B3b0 ~ B3b1, Block 3
andq %rsi, %rbp ;B3b2 ~ B3b3, Block 3
nop ;B3b4 ~ B3b4, Block 3
nop ;B3b5 ~ B3b5, Block 3
nop ;B3b6 ~ B3b6, Block 3
je .L1 ;B3b7 ~ B3b15, Block 3
addq %rbx, %rcx ;B4b0 ~ B4b1, Block 4
addq %rsi, %rcx ;B4b2 ~ B4b3, Block 4
addq %rdi, %rcx ;B4b4 ~ B4b5, Block 4
nop ;B4b6 ~ B4b6, Block 4
jmp .L2 ;B4b7 ~ B4b15, Block 4
L1:
irmovq $5, %rcx ;B5b0 ~ B5b9, Block 5
andq %rdi, %rcx ;B5b10 ~ B5b11, Block 5
addq %rsi, %rbx ;B5b12 ~ B5b13, Block 5
subq %rbx, %rcx ;B5b14 ~ B5b15, Block 5
```

L2:

addq %rsi, %rax ;B6b0 ~ B6b1, Block 6

rrmovq %rax, %rbp ;B6b2 ~ B3b3, Block 6

subq %rdi, %rbp ;B6b4 ~ B6b5, Block 6

jle .L0 ;B6b6 ~ B6b14, Block 6

ret ;B6b15 ~ B6b15, Block 6

```

The functional-equivalent C code is as the following.

```c
int i = 0;

int c;

int b;

i = i + 1;


do {

b = i & 1;

if (b == 0) {

c = 3;

}

else {

c = c + 6;

}

i = i + 1;

} while (i <= 4);
```
```

return;

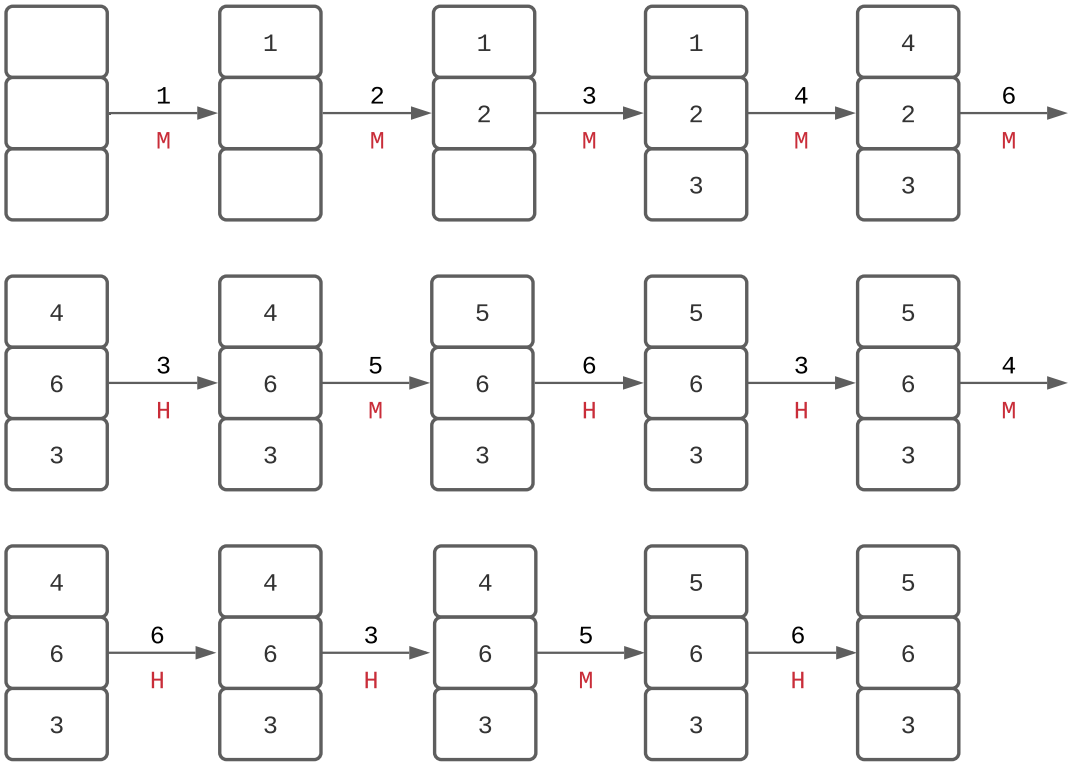
```

Basically, the program initialize i to 1 and use it as loop variable with increment 1. The loop condition is i <= 4 so there are 4 iterations in total. In every iteration, the program branches based on if i is odd or even. If i is even then c=3 otherwise c = c + 6. Note the program is only functional-equivalent to the original assembly program, not completely equivalent.

So the sequence of block numbers accessed during execution is :

1, 2, 3, 4, 6, 3, 5, 6, 3, 4, 6, 3, 5, 6

**6(b)**
```


1
M
2
M
3
M
4
M
6
M
3
H
5
M
6
H
3
H
4
M
6
H
3
H
5
M
6
H