Introduction to Computer Systems
2013 Fall Midterm Examination

Name_____________ Student No._______________ Score__________

Problem 1: (7 points)

1.

2.

Problem 2: (10 points)

1.

2.

3.

4.

Problem 3: (21 points)

1

2

3.

4

5

6

Problem 4: (8 points)

1.

2.

3.

Problem 5: (6 points)

1.

2.

Problem 6: (48 points)

1.

rjne rB	rjne %ebx
[1]	[7]
[2]	[8]
[3]	[9]
[4]	[10]
[5]	[11]
[6]	[12]

2.

3

4.

5.

Problem 1: (7 points)

```
1  #include "csapp.h"
2
3  int main(void)
4  {
5     int status;
6     if (Fork() == 0) {
7        printf("a");
8        exit(0);
9     } else {
10       printf("b");
11       waitpid(-1, &status, 0);
12       if (WIFEXITED(status))
13          printf("%d", WEXITSTATUS(status));
14    }
15    printf("c");
16    exit(1);
17 }
```

1. Please list ALL possible output sequences of the above program. (4’).
2. If the statement **`exit(0)`** in **`line-8`** was REMOVED, please list ALL possible outputs. (3’)

Problem 2: (10points)

Consider the following code.

```
1  #include "csapp.h"
2
3  int main()
4  {
5      pid_t pid;
6      char* args[] = {"/bin/ls", NULL};
7      if ((pid = Fork()) == 0) {
8          if ((pid = Fork()) > 0)
9              exit(0);
10
11         printf("A\n");
12         if(execve(args[0], args, NULL) < 0) {
13             printf("%s: execve command error!\n", args[0]);
14             exit(0);
15         }
16         printf("C\n");
17         exit(0);
18     }
19     while (waitpid(-1, NULL, 0) > 0) { // empty }
20     printf("B\n");
21     exit(0);
22 }
```

Suppose the origin process is **process#1**, the child process forked in **line-7** is **process#2**, the child process forked in **line 8** is **process#3**

1. Point out which process exits after **line-9** is executed. (2′)
2. Please answer whether the character **"C"** will be printed out. Why? (3′)
3. Which of the characters will be printed out FIRST, **A** or **B**? Is it DETERMINISTIC? Why? (3′)
4. When **process#3** terminates, which process will reap it. (2′)

Problem 3: (21points)

```
1 .Loop
2   mulss (%rax, %rdx, 4), %xmm0
3   mulq $2, %rdx
4   cmpq %rdx, %rbp
5   jp .Loop
```

1. Please rewrite the code of the **n**th cycle of the loop using register renaming (3'). (Note that you should separate the **mulss** into **load** and **mul**, and provide the registers updated by each instruction.)
2. Please provide the data-flow graphic of above code like figure 5.13 in ICS book (3').
3. Please classify all the registers used in the code into the four categories, **read-only**, **write-only**, **local** and **loop** (4').
4. Assume that the latency of integer multiple is 2 and the latency of single FP multiply is 3, please provide the data-flow graphic like figure 5.15 in ICS book and **bold** the critical path in your graphic. (5'). (Note that you should make difference between the two **muls** in the graph.)
5. Please calculate the **CPE** with the assumption in question. 4. (2')
6. If the instruct at **line 3** is replaced by "**mulss %xmm0 %rdx**", and there is only one multiplier (issue time 1), what's the **CPE** now? (4')

Problem 4: (8points)

```
int fac_u1(int n, int *data)
{
  int i, result = 1;
  for(i = 0; i < n; i++)
    result = result * data[i];
  return result;
}
```

```
int fac_u2(int n, int *data) {
  int i, result = 1;
  for(i = 0; i < n-1; i+=2)
    result = (result*data[i])
             * data[i+1];
  return result;
}
```

1. Is the result of function **fac_u2** equivalent to that of function **fac_u1**? If NOT, what value of argument '**n**' will return different results? (2')
2. Please modify function **fac_u2** such that its return value always equals to **fac_u1** (3')
3. If we replace inner loop code of function **fac_u2** to:

   result = result * (data[i]*data[i+1]);

   Please explain why the new version has better performance. (3')

Problem 5: (6points)

// sum	// sum_b4

`int a[8][8], b[8][8];`
`int i, j, k, sum=0;`
`for(i=0; i<8; i++)`
` for(j=0; j<8; j++)`
`  for(k=0; k<8; k++)`
`    sum += a[i][k] * b[k][j];`
	`int a[8][8], b[8][8];`
	`int i, j, k, jj, kk, sum=0;`
	`for(i=0; i<8; i++)`
	` for(jj=0; jj<8; jj+=4)`
	`  for(kk=0; kk<8; kk+=4)`
	`   for(j=jj; j<jj+4; j++)`
	`    for(k=kk; k<kk+4; k++)`
	`      sum += a[i][k] * b[k][j];`

Suppose above code block of function **sum** and **sum_b4** is executed on a machine with following conditions:

1) **sizeof(int) == 4.**
2) The machine has a **fully Associate** cache with **8 blocks**, each block is **16 bytes.** Its replacement policy is **LRU**
3) Code uses memory access only for the array data, other variables are held in registers.
4) The start address of array **A** is **0x00000000** and **B** is **0xF0000000.** Both of the arrays are stored in **row major.**
5) cache is always **cold** before running above code.

1. How many accesses to arrays **A** and **B** will result in cache misses after running above code of function **sum**, please provide some explanation to your answer. (3')
2. How many accesses to arrays **A** and **B** will result in cache misses after running above code of function **sum_b4**, please provide some explanation to your answer.(3')

Problem 6: (48points)

Suppose we add a new instruction **rjXX rB** to Y86 instruction sets, it uses the following encoding.

Byte	0		1	
rjXX rB	**E**	**fn**	**F**	**rB**

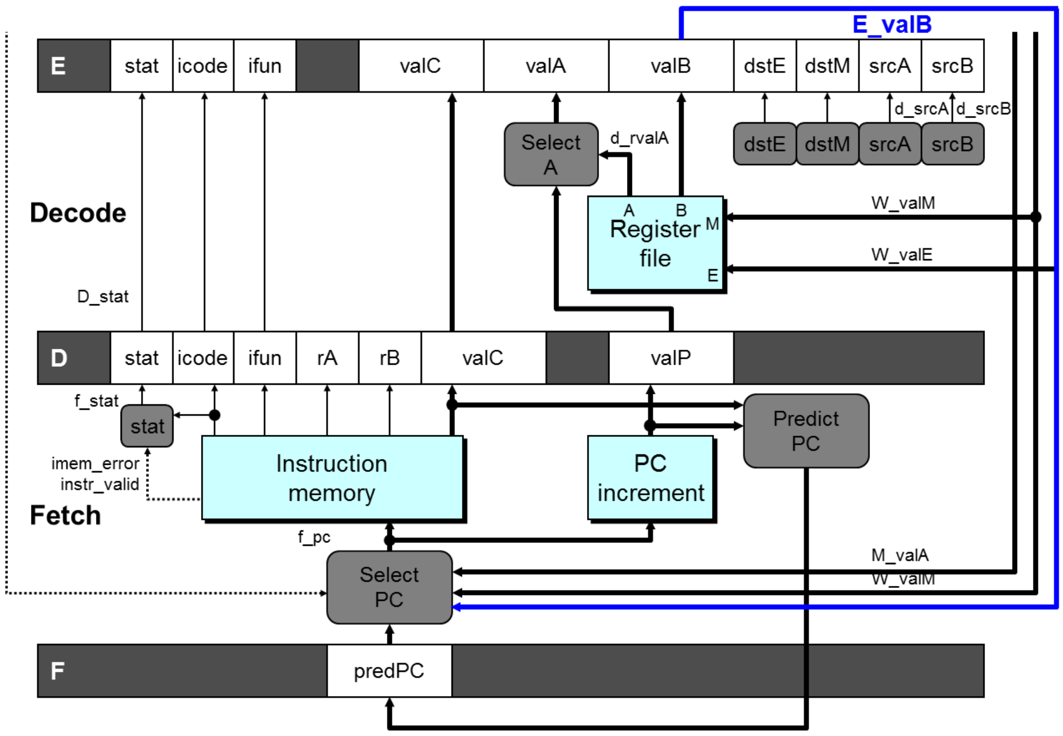
rjXX rB is a new INDIRECT jump instruction (e.g., **rjmp**, **rjle**, **rjl**, **rje**, **rjne**, **rjge**, and **rjg**). Our design uses the always **TAKEN** branch prediction strategy for conditional jump. The jump condition is the same with **jXX**. (NOTE that the original direct jump instructions **jXX** are unchanged)

1. Fill in the function of each stage for **rjne rB** instruction in Y86 sequential implementation like Figure 4.21. (NOTE: fill ALL functions for each stage, and fill in '–' for empty stage) (12')

Instruction:

```
0x009:   xorl   %eax, %eax
0x00f:   irmovl $0x023, %ebx
0x015:   irmovl $0x9,   %eax
0x01b:   rjne %ebx
0x01d:   halt
```

Stage	Generic	Specific
	rjne rB	rjne %ebx
Fetch	[1]	[7]
Decode	[2]	[8]
Execute	[3]	[9]
Memory	[4]	[10]
Write Back	[5]	[11]
PC Update	[6]	[12]



2. As shown in above new PIPE logic figure, we add a forwarding logic from **E_valB** to **f_pc** to support **rjXX** instruction, since the target address require read from register file. Please describe all possible hazards due to new instruct **rjXX**. You need provide detail explanation and list detection conditions like Figure 4.64 and control action like Figure 4.66. (10')

(NOTE: there are two different hazards.)

3. The original PIPE implementation of Y86 should be modified to support the **`rjXX`** instruction. Please describe the modification and provide **`HCL`** of **`f_pc`**, **`D_bubble`** and **`E_bubble`** logic. (NOTE: Only need to write the code about **`rjXX`** instruction) (12′)

   For example:
   ```
   bool instr_valid = f_icode in { IRJXX };
   ```

4. Please list all hazard combinations (arise simultaneously) including **`rjXX`** instruction. You need draw pipeline states figures like Figure 4.67 and list pipeline control action like table in Practice Problem 4.35 for each combination (9')

5. Calculate the CPI of new pipeline system with following penalties (3')

Cause	**Instruction Frequency**	**Condition Frequency**
Load/Use	0.15	0.20
Return	0.01	1.00
jXX	0.15	0.40 (NOT TAKEN)
rjXX	0.05	0.40 (NOT TAKEN)

   If we change design to always use **NOT TAKEN** branch prediction strategy for all conditional jump (both **`jXX`** and **`rjXX`**), then please calculate new CPI with above penalties. (2′)