Introduction to Computer Systems 2011
Second Midterm Examination

Name_____ Student No._____ Score_____

Problem 1: (18 points)

Read the following program and answer the questions.

```
 1 #include <stdio.h>
 2 #include <unistd.h>
 3 #include <sys/types.h>
 4 #include <sys/wait.h>
 5 #include <stdlib.h>
 6
 7 int main(int argc, char **argv)
 8 {
 9     int i = 0, n = 3, status;
10
11     while (i < n) {
12         printf("i=%d\n", i);
13         if (fork()) {
14             i++;
15         } else {
16             i += 2;
17             printf("i=%d\n", i)
18             exit(i);
19         }
20     }
21
22     while (waitpid(-1, &status, 0) > 0) {
23         if (WIFEXITED(status) != 0) {
24             printf("exit status=%d\n", WEXITSTATUS(status));
25         }
26     }
27
28     exit(0);
29 }
```

1) What is the maximum value of the variable "**i**" during the whole execution of the program? (2')

2) How many "i=0", "i=1", "i=2" will be printed on the screen? (2' * 3)

   "i=0": __[1]___; "i=1": __[2]___; "i=2": __[3]___;

3) Suppose all the processes terminated normally, how many "**`exit status ...`**" will be printed on the screen by code at line **24** "**`printf("exit status=%d\n", WEXITSTATUS(status))`**"?(2') Please write down all possible output of the code from line **21** to line **27**. (3')

4) Suppose all the processes terminated normally, if we change the keyword "**`while`**" to "**`if`**" at line **22**, how many "**`exit status ...`**" will be printed on the screen by the code at line **24** "**`printf("exit status=%d\n", WEXITSTATUS(status))`**"?(2') Please write down all possible output of the code from line **21** to line **27**. (3')

Problem 2: (29points)

Suppose a computer with following features:

- ✧ Memory accesses are to **2-byte words**.
- ✧ It's a **little endian** type machine.
- ✧ Physical addresses are **12-bits** wide.
- ✧ The cache uses **LRU replacement policy**.

The contents of the cache are as follows. (Hexadecimal):

2-way Set Associative												
	Line 0						Line1					
Set Index	Tag	Valid	B0	B1	B2	B3	Tag	Valid	B0	B1	B2	B3
0	C7	1	86	30	3F	10	05	1	40	67	C2	3B
1	45	1	06	78	07	C5	38	0	00	BC	0B	37
2	91	1	60	4F	E0	23	F0	0	C0	88	30	12
3	06	0	B7	26	2D	47	32	1	12	08	7B	AD

Given the following fields of physical address:

- ✧ CO: Byte offset within the cache line
- ✧ CI: The cache (set) index
- ✧ CT: The cache tag

1. Indicate the length of every field of above cache. (1' * 3 = 3')

	Fields	**Length (bit)**
Cache	**CT**	**[1]**

	CI	[2]
	CO	[3]

2. List all of the hex physical address range that will hit in **Set 1**.(4')

3. Suppose a program running on this machine and access the memory (load) as the following trace, fill the blanks in the table. If there is a cache miss, enter "-" in the "Value Returned" row, otherwise fill in the true value **in hex** the cache returns. (8')

Binary Address	Hit or Miss?	Value Returned
1100 0111 0000	[1]	[2]
1100 1000 0010	[3]	[4]
0011 1000 0110	[5]	[6]
1001 0001 1000	[7]	[8]

4. Consider the program execution on above machine. You are given the following definitions:

```
struct node {
    short id;
    short v;
};
struct node tree[4][4];
int i, j;
```

- ✧ *sizeof*(short) is equal to **2**.
- ✧ The array **tree** is allocated at physical address **0**.
- ✧ The cache above is now cleaned up (empty now).
- ✧ The only memory accesses are to the entries of the array **tree**. (**i** and **j** are stored in registers).

Please determine the cache performance of the following codes

```
for(i=0; i<4; i++){
    for(j=0; j<4; j++){
        tree[i][j].id = i * j;
        tree[i][j].v = i + j;
    }
}
```

1) What is the total number of writes that ***miss*** in the cache? _____. (1')

2) What is the ***miss rate***? _____. (2')

If we change the code (the cache is still empty)

```
for(i=0; i<4; i++){
    for(j=0; j<4; j++){
```

```
        tree[i][j].id = i * j;
    }
}
for(i=0;i<4;i++){
    for(j=0;j<4;j++){
        tree[i][j].v = i + j;
    }
}
```

3) What is the total number of writes that ***miss*** in the cache? _____. (1’)

4) What is the ***miss rate***? _____. (2’)

5) After finish running above code, fill all the content of **Set 1** in hex (8’)

2-way Set Associative							
	Line 0			Line1			
Set Index	**Tag**	**Valid**	**B0 B1 B2 B3**	**Tag**	**Valid**	**B0 B1 B2 B3**	
1	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	

Problem 3: (25points)

Read the following code of finding the maximum value in a linked list.

```
   typedef struct list_t {
      int data;
      struct list_t *next;
   } list_t;
1. int max(list_t *p)
2. {
3.     int max = 0;
4.     while(p) {
5.         if(p->data > max)
6.             max = p->data;
7.         p = p->next;
8.     }
9.     return max;
10.}
```

Assembly code for the while loop:

```
    .L1
1.    movl    (%eax),%ecx
2.    cmpl    %ecx,%edx
3.    jge     .L2
4.    movl    %ecx,%edx
    .L2
5.    movl    4(%eax),%eax
6.    testl   %eax,%eax
7.    jne     .L1
```

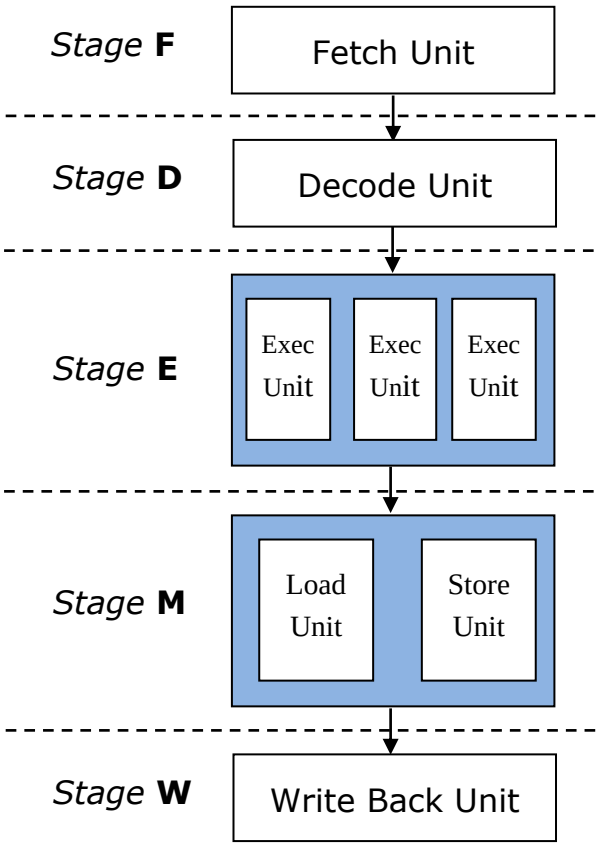
Assume that there is only **one load unit** for the processor, and the **issue time** of this load unit is **1** and the latency time is **3** for a load operation. We don't take cache into consideration, thus all the memory operations have the identical latency. Besides the load unit, all other processor resources such as integer/branch units are **unlimited** and both issue time and latency time for these units are **1**. The "if" condition branch (line 3 in assembly code) is NOT taken and the "while" loop branch (line 7 in assembly code) is taken. (**Hint**: The register-to-register move instruction does not take load unit but an integer unit, while move from memory to register will take one load unit.)

1. Translate the assembly code into **executing unit operations**. Different versions of registers should be renamed. (7')
2. Draw computation graph according to the scheduling of operations for the **first** iteration of the loop, and give the CPE of the while loop.(6' + 2')

3. Suppose the number of **load unit** is unlimited, and the issue time and the latency remain unchanged. Draw computation graph according to the scheduling of operations for the **first** iteration of the loop, and give the CPE of the while loop again.(6’ + 2’)
4. What is the bottleneck of this piece of code when all the processor resources are unlimited? (2’)

Problem 4: (28points)

Consider the following figure; it illustrates a five stage pipeline processor similar to that in your text book (Figure 4.53, Page 334 in English Book). Notice, there are several differences different from the architecture from that in your book.



- **<u>First difference</u>** is stage **E** has three **multi-cycle** execution units, which performs equally in function. The addition and subtraction take **2 cycles** to completion. Every time it sets or reads ***CC*** also takes **1 cycle**. (**NOTE**: Each of them can only handle one instruction at a time**,** that is, other instructions must wait in its stage **D** until one of the execution units is free.)
- **<u>Second difference</u>** is that **Memory Unit** is separated into two parts: the **Load Unit** and the **Store Unit**. The Load Unit can only ***read*** memory while the Store Unit can only ***write*** memory. Every memory access takes **3 cycles,** for a non-memory instruction, it takes **1 cycle** to

pass Stage **M**.
(**NOTE**: if two instructions access the same memory address, the latter one has to wait until the former one finished. If they come at the same time, the read operation owns a higher priority.)

- ✧ **Third difference** is that this architecture uses an instruction set called "W86" based on the "Y86" instruction set in your text book. Different with Y86, W86 supports at most **10** bytes instruction encodings.
(**NOTE**: You can regard W86 as an extension of the Y86 instruction set.)
- ✧ **Fourth difference** is that this architecture is **2-issue** in-order pipeline processor. It means the stage **F** can fetch at most **two** instructions and all the state registers between stages can also store at most two instructions' states. There are also *forwarding* data path from stage **W**, stage **M** and stage **E** to stage **D**.
(**NOTE**: If it encounters the hazards that can't be solved, it will stall until the operands are available.)

1. Please answer the questions below (2' * 3 = 6')
   1) How many cycles are needed to complete the instruction "**iaddl %eax, %ebx**"? (Update PC is not taken into account) (2')
   2) How many cycles are needed to complete the instruction "**pushl %ecx**"? (Update PC is not taken into account) (2')
   3) Suppose the following instruction to be fetched is "**jmp $41**", can we fetch the next instruction predicted simultaneously (in the same cycle)? Why? (2')
2. **XCHG** is a new instruction in W86 instruction set. It uses the following encodings:

Byte	**0**	**1**	**2**		**6**	**10**
xchg rA, D(rB)	**C**	**0**	**rA**	**rB**	**D**	

XCHG exchanges the values in rA and D(rB). Describe the computations performed to implement these two instructions. Please fill the blank of certain stage with "Nothing to do" if the stage has no work to do to accomplish this instruction. (2' * 5 = 10')

Stage	**xchg**
Fetch	[1]
Decode	[2]
Execute	[3]
Memory	[4]
Write Back	[5]
PC Update	**PC←valP**

3. This problem is based on the following code, which will be executed on the processor described above. Assume in cycle **0**, no instruction is executed, and in cycle **1**, the first two instructions are fetched. Fill in the blanks. (Stage: **F**, **D**, **E**, **M**, **W**, **done** or **non-fetched**, uncertain value marked with "**--**") (8').

```
0x000:      irmovl $11, %eax
0x006:      irmovl $3, %ebx
0x00c:      iaddl %ebx, %ebx
0x00e:      irmovl $0x44F, %edx
0x014:      mrmovl 0(%edx), %ecx
0x01a:      rrmovl %ecx, %edx
0x020:      subl  %eax,  %ebx
0x022:     irmovl %0xDEADBEEF, %ecx
0x028:      rmmovl %edx, 0(%eax)
```

Cycle 5:

Instruction	Stage
0x006	[1]
0x00c	[2]
0x00e	[3]
Register	Value
%eax	[4]

Cycle 7:

Instruction	Stage
0x006	[5]
0x00c	[6]
0x00e	[7]
0x014	[8]
Register	Value
%eax	[9]
%ebx	[10]
%edx	[11]

Cycle 9:

Instruction	Stage
0x00c	[12]
0x001a	[13]
0x020	[14]
Register	Value
%ebx	[15]
%edx	[16]

4. Finally, the last instruction in this instruction flow will exit its stage **W** in Cycle _____?(4')

Solution

Problem 1: (18points)

1

2 [1] [2] [3]

3

4

Problem 2: (29 points)

1 [1] [2] [3]

2

3 [1] [2]
[3] [4]
[5] [6]
[7] [8]

4 1) 2)

3) 4)

5)

[1] [2] [3]

[4] [5] [6]

Problem 3: (25 points)

1

2 **CPE =**

3 **CPE =**

4

Problem 4: (28 points)

1 1)

2)

3)

2

3 [1] [2] [3] [4]
[5] [6] [7] [8]
[9] [10] [11]
[12] [13] [14]
[15] [16]

4