（装订线内不要答题）

复旦大学软件学院

2017～2018 学年第一学期期中考试试卷

√A 卷　　□B 卷

课程名称： 计算机系统基础　　**课程代码：** SOFT130057.01

开课院系： 软件学院　　**考试形式：**

姓名：　　**学号：**　　**专业：**

题 号	1	2	3	4	5	6	7	总 分
得 分								

（共 7 道大题，时间 120 分钟）

1. Suppose that the target code of 400000 instructions run on a 50MHZ processor, and the program is composed of four instructions. According to the program tracking experiment results, the instruction-mixing ratio and the number of each instruction required are as follows:

Instruction type	CPI	Instruction mixing ratio
Arithmetic and logic	1	50%
Cache hits load/store	2	20%
Branch	8	20%
Cache miss memory access	20	10%

a. Calculates the average CPI of the program running on the single processor with the data above. (2’)

b. According to the obtained CPI of question a, calculates the corresponding MIPS and the time required for the program to run. (2’)

c. Explain the advantages and disadvantages of single-cycle and multi-cycle CPU execution. (2’)

d. Suppose we consider two different design methods for conditional branch instructions:

> CPUA: set the conditional code by compare instruction;
> CPUB: a comparison procedure is included in branch instructions.

In two CPU, conditional branch instructions take 2 clock cycles and all other instruction takes 1 clock cycles. For CPUA, branch instruction accounted for 25%; because before each branch instruction it needs to have compare instruction, compare instruction also accounted for 25%. Since CPUA does not need to compare in branching, it is assumed that its clock cycle time is 1.5 times faster than that of CPUB. Which CPU is faster? If the clock cycle time of CPUA is only 1.2 times faster than that of CPUB, which CPU is faster. (4’)

Solution:

2. Given the following instruction sequence:

```
I1:   LD    r3, 34, r8      // r3 = Mem(34(r8))
I2:   LD    r5, 45, r7
I3:   MULT  r1, r5, r4      // r1 = r5 * r4
I4:   MULT  r11, r1, r5
I5:   DIV   r10, r1, r3     // r10 = r1/r3
I6:   ADD   r3, r11, r5     // r3 = r11 + r5
```

Suppose that the super-pipeline has the following function units:

One 1-cycle load unit,

One 1-cycle add unit,

Two 5-cycle multiply unit,

And one 20-cycle divide unit.

	Issue	Read Op	Exec	Write Result
I1: LD r3, 34, r8	1	2	3	4
I2: LD r5, 45, r7	5	6		
I3: MULT r1, r5, r4	6			
I4: MULT r11, r1, r5				

The following figure shows the execution status at 6th cycle.

Please answer the following questions.

a. Can the instruction I3 read its ops at the 7th cycle? Explain why.(3’)

b. Can the instruction I4 issue at the 7th cycle? Explain why.(3’)

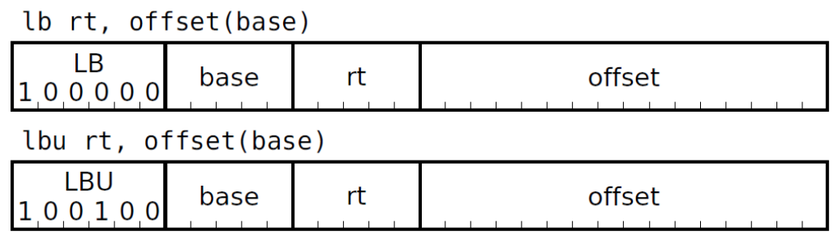
c. In a typical pipeline like Y86, what influence will the dependence cause?(2’)

d.To eliminate the influence, what techniques can be used? (hint: You can discuss from both hardware and software techniques)(2’)

Solution:

3. Modify the single-cycle MIPS processor shown on the next:

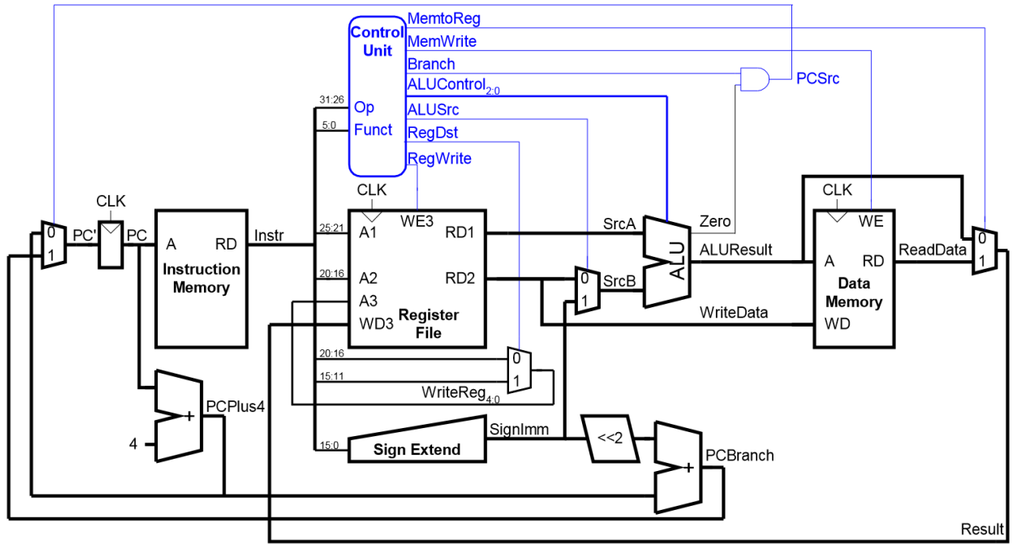
a. to implement the lb (“Load byte”) and lbu (“Load Byte Unsigned”) instructions. They are encoded as shown below



These instructions add the contents of the base register to a sign-extended *offset* immediate to form an address, fetch a byte from that address, and finally write the byte into the *rt* register.

The two instructions differ in how they handle the top 24 bits. lb treats the byte as a signed value and sign-extends the most significant bit of the byte across the top 24 bits; lbu treats the byte as unsigned and simply fills the top 24 bits of *rt* with 0’s.

Start from the base single-cycle processor shown on the next page and add components to the data path, name any new control signals, and show how to add to or modify the main decoder to accommodate these new instructions. (8’)



Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp
R-type	000000	1	1	0	0	0	0	1-
lw	100011	1	0	1	0	0	1	00

b. to implement the bgezal (“Branch on Greater than or Equal to Zero and Link”) instruction. This is encoded as shown below

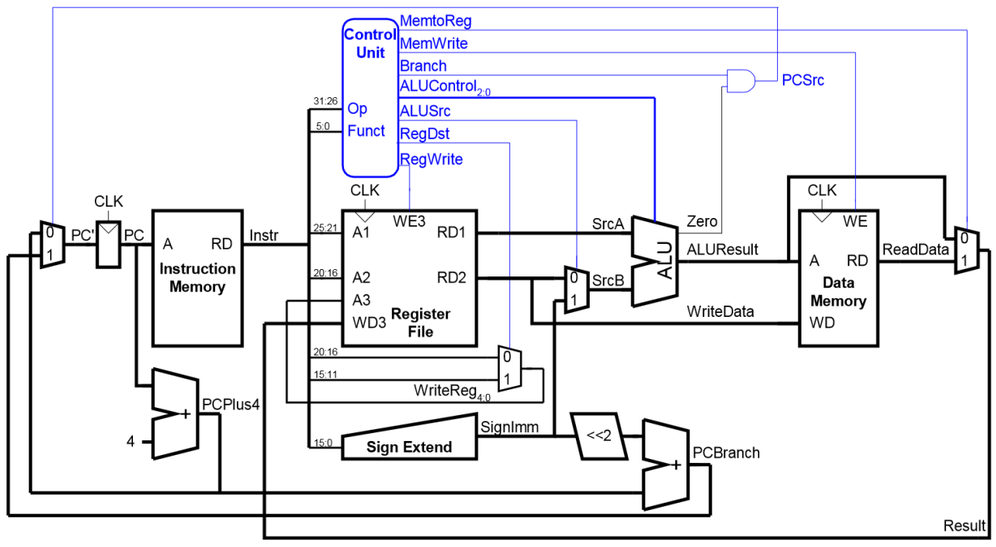
```
bgezal rs, offset
```

REGIMM	rs	BGEZAL	offset
0 0 0 0 0 1		1 0 0 0 0	

This instruction reads the register *rs* and, if it is greater than or equal to zero, passes control to the instruction *offset* words away from the PC (i.e., as the bne and other branch instructions do), otherwise it passes control to the next instruction in series.

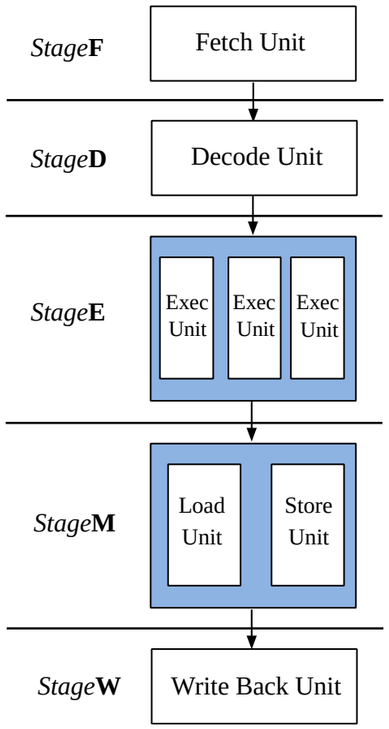
Regardless of the contents of *rs*, this instruction also writes the value of PC + 4 into register 31 ($ra), providing the “and link” function.

As in the previous problem, start from the base single-cycle processor shown on the next page and add components and rules. (8’)



Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp
R-type	000000	1	1	0	0	0	0	1-
lw	100011	1	0	1	0	0	1	00

4. 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.



- ✧ **First difference** 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.)
- ✧ **Second difference** 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.)

a. Answer the questions below:

◇How many cycles are needed to complete the instruction

"**iaddl %eax, %ebx**"? (Update PC is not taken into account) (2')

◇How many cycles are needed to complete the instruction

"**pushl %ecx**"? (Update PC is not taken into account) (2')

◇Suppose the following instruction to be fetched is "**jmp $41**", can we fetch the next instruction predicted simultaneously (in the same cycle)? Why? (2')

Solution:

b. **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

c. 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]

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

5. 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 C0
2	91	1	60	4F	E0	23	F0	0	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

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

	Fields	**Length (bit)**
Cache	**CT**	**[1]**
	CI	**[2]**
	CO	**[3]**

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

Solution:

c. 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]**

d. 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;
```

I) *sizeof*(short) is equal to **2**.
II) The array **tree** is allocated at physical address **0**.
III) The cache above is now cleaned up (empty now).
IV) 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]**

6. Nowadays, developers are facing the application that owns immense data throughput and intensive calculation intrinsically, aka big-data application. Back in the 1970s, the Americans first used computers in common families to compose a cluster for the decomposition of the computation task for launching spacecraft, which is badly intensive. Coming to the next millennium, a group of Yahoo proposed a framework named hadoop for applications with increasing data throughput. Nowadays, industry tycoons are continuously involved in the competition of big-data products. For example, the new generation of Microsoft OS Azure is designed with the distributive attribute. Since most of us will feed on the industry or the relative jobs, how can we keep pace with the trend of technology revolution which occurs beyond our scope with limited resources? Please talk about your ideas with no nonsense.(7’)