Homework 9

Problem 1

Suppose the order of the third and fourth cases (the two forwarding sources from the memory stage) in the HCL code for d_valA were reversed, as follows in table 1. Describe the resulting behavior of the rrmovl instruction (line5) for the following program:

```
int d_valA=[
      D_icode in { ICALL, IJXX } : D_valP;
      d_srcA == e_dstE : e_valE;
      d_srcA == M_dstM : m_valM;
      d_srcA == M_dstE : M_valE;
      d_srcA == W_dstM : W_valM;
      d_srcA == W_dstE :W_valE;
      1 : d_rvalA;
];
```

Table 1

```
1    irmovl $5, %edx
2    irmovl $0x100, %esp
3    rmmovl %edx, 0(%esp)
4    popl %esp
5    rrmovl %esp, %eax
```

Answer：If the two cases were reversed, then the write back from M_valE would take priority, causing the incremented stack pointer to be passed as the argument to the rrmovl instruction. This would not be consistent with the convention for handling popl %esp determined in section 4.1.6 in ICS book.
得分点：popl %esp 的行为会发生改变，或者算出来%eax 的值会变成 0x104

Problem 2

Considering the codes below and answer the following questions:

Prog1	**Prog2**
irmovl $1, %ebx irmovl $2, %ecx addl %ebx, %ecx halt	rrmovl %eax, %ebx rrmovl %edx, %eax

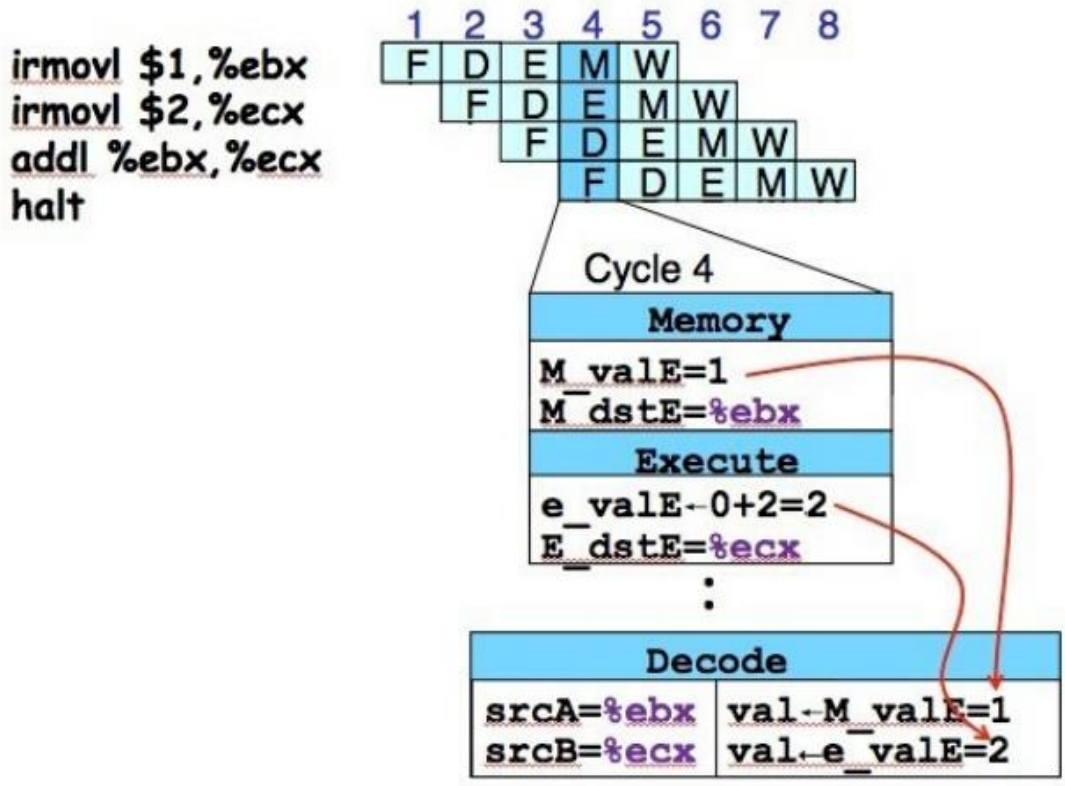
1. How to solve above data hazard in **Prog1** by Forwarding ? You can draw

a picture to illustrate yourself.

2. In **Prog2**, Is there a Data Hazard? If yes, how to solve it. If no, explain why.

Answer：

1. In cycle 4, the decode stage logic detects a pending write M_valE to register %ebx in the memory stage. It also detects that a new value e_valE is being computed for register %ecx in the execute stage. **It uses these as the values for valA and valB rather than the values read from the register file.**
   得分点：指明在 addl 指令在 Decode 阶段时，是从 Memory 阶段得到的%ebx 和 Execute 阶段得到的%ecx。



2. No, the read of %eax of the first instruction is in **Stage 2 Decode**, while the write of %eax of the next instruction is in **Stage 5 WriteBack**, there will be **no data hazard.**

Problem 3

```
#demo-retB.ys
0x000:      irmovl Stack,%esp      #Intialize stack pointer
0x006:      call p         #Procedure call
0x00b:      irmovl $5,%esi      #Return point
0x011:      halt
0x020: .pos 0x20
0x020: p:   irmovl $-1,%edi     #procedure
```

```
0x026:      ret
0x027:      irmovl $1,%eax     #Should not be executed
0x02d:      irmovl $2,%ecx     #Should not be executed
0x033:      irmovl $3,%edx     #Should not be executed
0x039:      irmovl $4,%ebx     #Should not be executed
0x100: .pos 0x100
0x100: Stack:       #Stack: Stackpointer
```

1. During the above example's executing, how many hazards will happen? Please pointed them out.
2. How will the data hazard in the above code be handled? (detail required)
3. What's the difference between **stall** and **bubble**?

Answer:

1. There are **3 hazard** in the above code:
   - **Data hazard** between *"irmovl Stack,%esp"* and *"call p"*. For the previous instruction write to %esp and the next instruction read %esp.
   - **Data hazard** between *"call p"* and *"ret"*. For they both read and write to %esp.
   - **Control hazard** due to the *"ret"* instruction.

2. The data hazard will be handled by Forwarding.
   a) The first data hazard between *"irmovl Stack,%esp"* and *"call p"* will be handled by forwarding. The call instruction at stage **Decode** will get the value of %esp from the **E_valE** from the stage **Execute** of irmovl instruction.
   b) The second Data hazard between "call p" and "ret" will be handled by forwarding. The ret instruction at the **Decode** stage will get the value from **M_valE** from the stage **Memory** of the call instruction.

3. A pipeline stall is when a pipeline stage's input memory remains the same from one cycle to the next. A pipeline bubble is when a pipeline stage's input memory changes to that of a nop instruction, i.e., an instruction that does nothing. On any cycle when an instruction stalls in one stage, a bubble is injected into the subsequent stage.