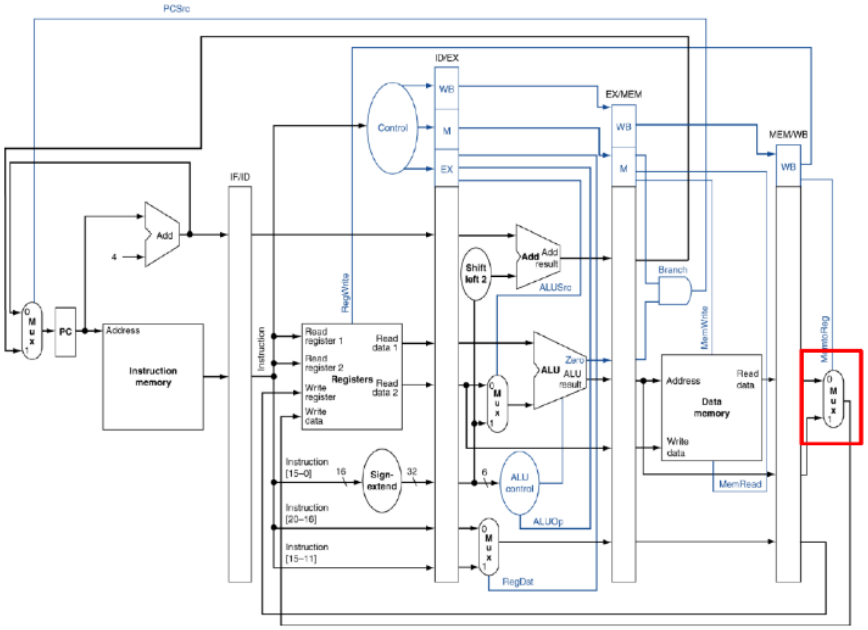
Computer Organization Lab4

Architecture diagrams:



Hardware module analysis:

- The following modules are the same as in Lab3:
  Adder, MUX_2to1, Shift_Left_Two_32, Sign_Extend

- ALU

  The idea is the same as in Lab3. The difference is that two functions, xor and multiplication, are added.

- ALU_Ctrl

	Input									Output			
	fc5	fc4	fc3	fc2	fc1	fc0	op2	op1	op0	ctr3	ctr2	ctr1	ctr0
add	1	0	0	0	0	0	1	1	0	0	0	1	0
sub	1	0	0	0	1	0	1	1	0	0	1	1	0
and	1	0	0	1	0	0	1	1	0	0	0	0	0

or	1	0	0	1	0	1	1	1	0	0	0	0	1
slt	1	0	1	0	1	0	1	1	0	0	1	1	1
xor	1	0	0	1	1	0	1	1	0	0	0	1	1
mult	0	1	1	0	0	0	1	1	0	1	0	0	0
addi	x	x	x	x	x	x	0	0	0	0	0	1	0
slti	x	x	x	x	x	x	1	0	1	0	1	1	1
beq	x	x	x	x	x	x	0	0	1	0	1	1	0
lw	x	x	x	x	x	x	0	0	0	0	0	1	0
sw	x	x	x	x	x	x	0	0	0	0	0	1	0

given function field　　ALU opcode　　ALU control

➔ ctr3 = fc4 & op1
ctr2 = (~fc2 & fc1 & op1) | op0
ctr1 = (~fc3 & ~fc2) | fc1 | (~op1)
ctr0 = (fc0 & op1) | (fc2 & fc1 & op1) | (fc3 & fc1 & op1) | (op2 & op0)

The difference is that jr, jp, and jal are removed while xor and mult are added, so the logical formulas for "ctr" are different.

➢ Decoder

	Input						Output						
	op5	op4	op3	op2	op1	op0	RD	alu op2	alu op1	alu op0	Src	Brh	RW
R-f	0	0	0	0	0	0	1	1	1	0	0	0	1
addi	0	0	1	0	0	0	0	0	0	0	1	0	1
slti	0	0	1	0	1	0	0	1	0	1	1	0	1
beq	0	0	0	1	0	0	x	0	0	1	0	1	0
lw	1	0	0	0	1	1	0	0	0	0	1	0	1
sw	1	0	1	0	1	1	x	0	0	0	1	0	0

given op field　　RegDst　　ALU opcode　　ALUSrc　　Branch　　RegWrite

➔ RD = ~op3 & ~op2 & ~op1
alu_op2 = (~op3 & ~op2 & ~op1) | (~op2 & op1 & ~op0)

alu_op1 = ~op3 & ~op2 & ~op1
alu_op0 = ~op5 & (op2 | op1)
Src = op5 | op3
Brh = op2
RW = (~op2 & ~op0) | (op5 & ~op3)

	Input						Output		
	op5	op4	op3	op2	op1	op0	memread	memwrite	memtoreg
R-f	0	0	0	0	0	0	0	0	1
addi	0	0	1	0	0	0	0	0	1
slti	0	0	1	0	1	0	0	0	1
beq	0	0	0	1	0	0	0	0	x
lw	1	0	0	0	1	1	1	0	0
sw	1	0	1	0	1	1	0	1	x

➔ memread = op5 & ~op3
memwrite = op5 & op3
memtoreg = ~op0

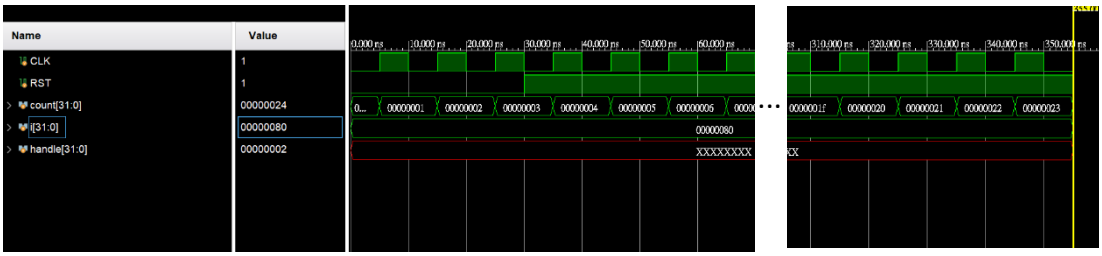
The difference are that jp, jal, and jump are removed, and the value for MemtoReg exchanges, so the logical formulas are a little different. Besides, RegDst and MemtoReg change from 2-bit to 1-bit.

➢ Pipelined_CPU

By reference to the architecture diagram, include all the required modules, and carefully connect the wires between them, done!

This is the part that differs the most from Lab3. In a pipelined CPU, pipeline registers are important components that hold information produced in previous cycle. Therefore, the operation is separated into several stages and the corresponding control signal lines are needed.

Simulation results:



RST is triggered at the half of the third cycle, so PC doesn't proceed to the next instruction until the fourth cycle. From that time on, when CLK is triggered, PC reads in next address and count is incremented by one as well. It ends at the 36th cycle since the value of PC (132) is larger than 128, then print out the results.

✓ test_1

This test data does the basic instructions. It does a few arithmetic and bitwise logical operation on some registers. The value of r8 is one since r1(3) is smaller than r2(4). Among them, the value of r1 is stored to m1, and the value is loaded back to r10. Moreover, r2 doesn't equal to r1, so then instructions won't branch back to the beginning, but execute to the end sequentially.

```
============== Final Result ==============
- Register File -
 r0 =    0  r1 =    3  r2 =    4  r3 =    1
 r4 =    6  r5 =    2  r6 =    7  r7 =    1
 r8 =    1  r9 =    0 r10 =    3 r11 =    0
r12 =    0 r13 =    0 r14 =    0 r15 =    0
r16 =    0 r17 =    0 r18 =    0 r19 =    0
r20 =    0 r21 =    0 r22 =    0 r23 =    0
r24 =    0 r25 =    0 r26 =    0 r27 =    0
r28 =    0 r29 =    0 r30 =    0 r31 =    0

- Memory Data -
 m0 =    0  m1 =    3  m2 =    0  m3 =    0
 m4 =    0  m5 =    0  m6 =    0  m7 =    0
 m8 =    0  m9 =    0 m10 =    0 m11 =    0
m12 =    0 m13 =    0 m14 =    0 m15 =    0
m16 =    0 m17 =    0 m18 =    0 m19 =    0
m20 =    0 m21 =    0 m22 =    0 m23 =    0
m24 =    0 m25 =    0 m26 =    0 m27 =    0
m28 =    0 m29 =    0 m30 =    0 m31 =    0
```

✓ test_2

This test data includes the new instructions, xor and mult. For example, the value of r4 derives from r11*r11, which is the square of 7, equals 49. And r6 is from the bitwise xor operation on r11 and r2. The value of r8 is one since r11(7) is smaller than constant ten. Among them, the value of r11 is stored to m1, and the value is loaded back to r10.

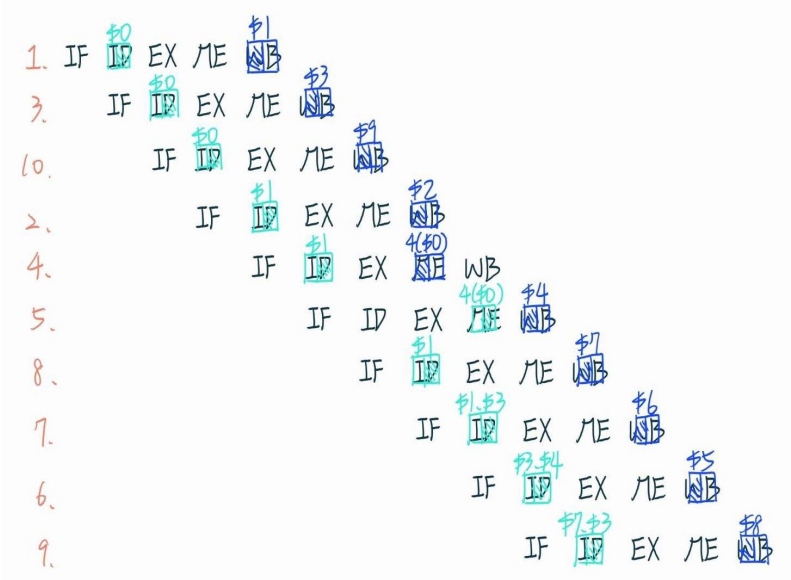
```
========== Final Result ==========
- Register File -
 r0 =    0  r1 =    0  r2 =    4  r3 =    5
 r4 =   49  r5 =    0  r6 =    3  r7 =    5
 r8 =    1  r9 =    0 r10 =    7 r11 =    7
r12 =    0 r13 =    0 r14 =    0 r15 =    0
r16 =    0 r17 =    0 r18 =    0 r19 =    0
r20 =    0 r21 =    0 r22 =    0 r23 =    0
r24 =    0 r25 =    0 r26 =    0 r27 =    0
r28 =    0 r29 =    0 r30 =    0 r31 =    0

- Memory Data -
 m0 =    0  m1 =    7  m2 =    0  m3 =    0
 m4 =    0  m5 =    0  m6 =    0  m7 =    0
 m8 =    0  m9 =    0 m10 =    0 m11 =    0
m12 =    0 m13 =    0 m14 =    0 m15 =    0
m16 =    0 m17 =    0 m18 =    0 m19 =    0
m20 =    0 m21 =    0 m22 =    0 m23 =    0
m24 =    0 m25 =    0 m26 =    0 m27 =    0
m28 =    0 m29 =    0 m30 =    0 m31 =    0
```

✓ test_3_hazard

StepI: color the stage where a read/write happens

StepII: reorder the instructions such that no hazard exists

➔ rearrangement:



➔ machine code:

00100000000000010000000000010000 ...(I1)
00100000000000110000000000001000 ...(I3)
00100000000010010000000001100100 ...(I10)
00100000010001000000000000000100 ...(I2)
10101100000000010000000000000100 ...(I4)
10001100000001000000000000000100 ...(I5)
00100000001001110000000000001010 ...(I8)
00000000011000010011000000100000 ...(I7)
00000000100000110010100000100010 ...(I6)
00000000111000110100000000100100 ...(I9)

➔ result:

```
============ Final Result ============
- Register File -
 r0 =   0  r1 =  16  r2 =  20  r3 =   8
 r4 =  16  r5 =   8  r6 =  24  r7 =  26
 r8 =   8  r9 = 100 r10 =   0 r11 =   0
r12 =   0 r13 =   0 r14 =   0 r15 =   0
r16 =   0 r17 =   0 r18 =   0 r19 =   0
r20 =   0 r21 =   0 r22 =   0 r23 =   0
r24 =   0 r25 =   0 r26 =   0 r27 =   0
r28 =   0 r29 =   0 r30 =   0 r31 =   0

- Memory Data -
 m0 =   0  m1 =  16  m2 =   0  m3 =   0
 m4 =   0  m5 =   0  m6 =   0  m7 =   0
 m8 =   0  m9 =   0 m10 =   0 m11 =   0
m12 =   0 m13 =   0 m14 =   0 m15 =   0
m16 =   0 m17 =   0 m18 =   0 m19 =   0
m20 =   0 m21 =   0 m22 =   0 m23 =   0
m24 =   0 m25 =   0 m26 =   0 m27 =   0
m28 =   0 m29 =   0 m30 =   0 m31 =   0
```

Problems you met and solutions:

Although TAs have highlighted the multiplexer with a red box, I still forgot to exchange the values in the truth table, so I cannot get the correct answer at first.

The solution was to double-check the architecture diagram then I found it was my fault. After revising the truth table, I could get the correct answer.

Summary:

This is the fourth time doing the lab, and I'm quite familiar with the way of implementation now. In this lab, constructing pipeline registers is the most impressive part. The idea is different from the previous ones, I have to pass some data and control signals stage by stage. In fact, I was dazzled by connecting those multiple wires. Fortunately, I did right at the first time!