Exercise 3

4.1

4.1.1

RegWrite = 1 ALUMux = 0 MemRead = 0 MemWrite = 0 ALUop = AND Branch = 0 RegMux = 0

For RegWrite, we need to write information into the registers, so it is 1.

For ALUMux, we need to use the value of registers to calculate, so we put it into 0 to use them.

For MemRead and MemWrite, we have no need to do any operations on Memory, so we put 0 to both of them.

For Branch, we don't use it.

For RegMux, we want to write the value of ALUOut to the regester, so we put it into 0.

4.1.2

Instruction memory, PC, Instruction Add, Registers, ALU.

4.1.3

Not used: the Add of Branch.

No outputs: Data memory.

4.2

4.2.1

PC, Instruction memory, Registers, Data memory, Instuction Add, ALU.

4.2.2

We need to add a Mux. We can see the Reg[Rd] and Reg[Rs] need to do a ALU operation. But currently we can only do the operations between Reg[Rs] and Reg[Rt]. So we need to use a Mux to store the Reg[Rd] as Read register 2.

4.2.3

As we add a Mux, there should be a new signal to control it. When it is the LWI instruction, we need to use the Reg[Rs] and Reg[Rd] as the Read registers.

4.3

4.3.1

We will use the longest way to calculate this, which means the LW instruction. First we will go through the I-Mem, then the Regs part, the Mux part, ALU part, D-Mem part, Mux part. So the total time would be 400 + 2 * 30 + 120 + 200 + 350 = 1130ps. And the time after improvement is 1130 + 300 = 1430ps.

4.3.2

The original time is 1430 – 300 = 1130ps. As we decline 5% instructions now, so the average time would be 1430 * 0.95 = 1358.5ps. Actually we slow down.

4.3.3

We originally use 1 I-Mem, 2 Add, 3 Mux, 1 ALU, 1 Regs, 1 D-Mem, 1 Control block. The total cost is 1000 + 2 * 30 + 3 * 10 + 100 + 200 + 2000 + 500 = 3890. While the current cost is 3890 + 100 = 3990. So we cost more but the performance becomes worse. The cost/performance becomes larger after adding the improvement.

4.4

4.4.1

In this situation, we only need to calculate the cost of latency of Instruction Memory as it is the longest way to go.

So the latency is 200ps.

4.4.2

In this situation, the longest way would be go from Instruction Memory, then the Sign-extend, Shift left 2, Add, Mux.

So the latency is 200 + 15 + 10 + 70 + 20 = 315ps

4.4.3

In this situation, the longest way would be go from Instrution Memory,

then the Registers, Mux, ALU, then another Mux.

So the latency is 200 + 90 + 20 + 90 + 20 = 420ps

4.4.4

PC-relative branches

4.4.5

The unconditional PC-relative branch

4.4.6

If the latency of shift is less than 105ps, the cycle time would not change. But if it is larger than 105ps, assuming it is xps the cycle time would add (x – 105) ps.

4.x

4.x.1

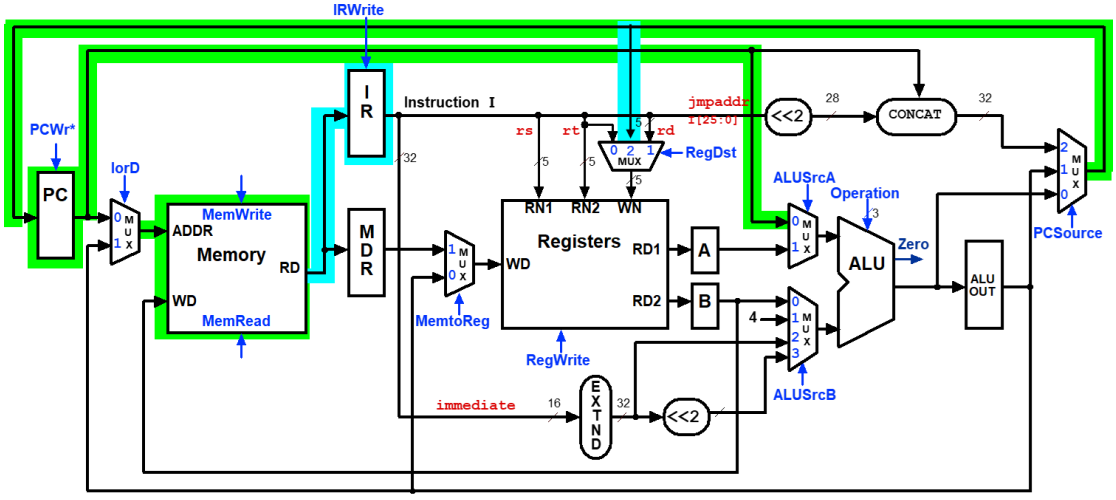
For jal:

(1)Instruction Fetch

<1>Fetch the instuction and store it in the IR

<2>Add 4 to the PC

<3>Store the current PC to $rs

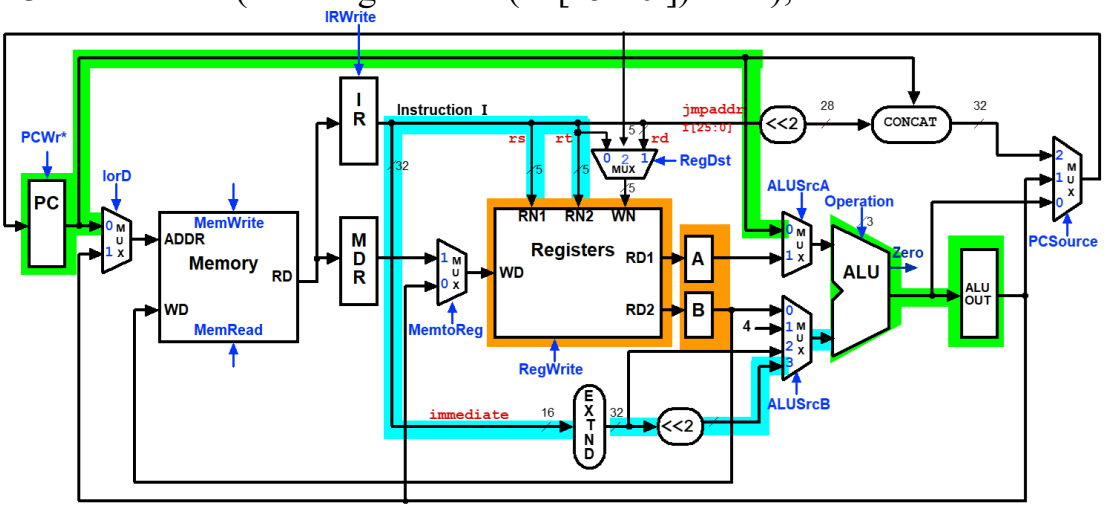


(2)Instruction Decode & Register Fetch

<1>A = Reg[IR[25 - 21]];

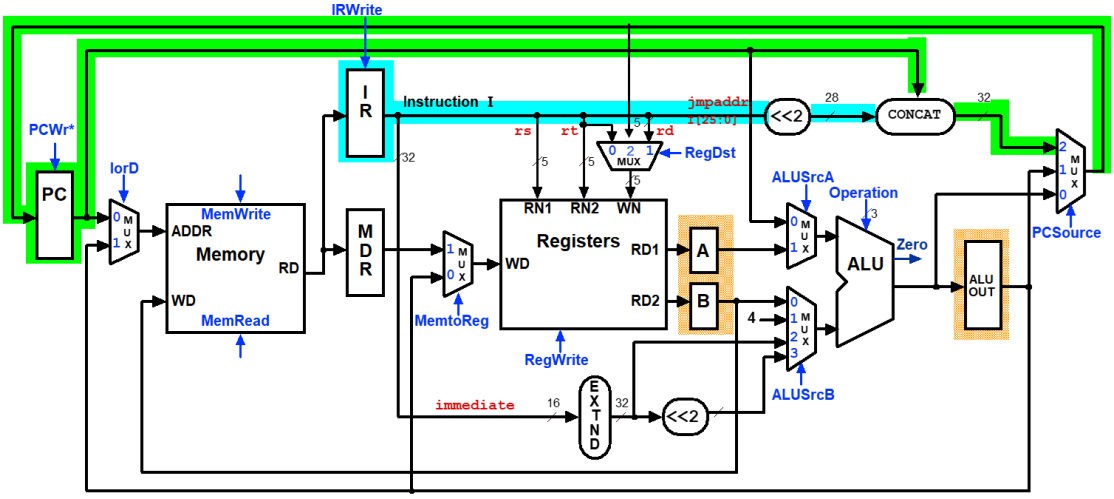
<2>B = Reg[IR[20 - 15]];

<3>ALUOut = (PC + sign-extend(IR[15 - 0]) << 2);



(3)Jump Instructions

<1>PC = PC[21-28] concat (IR[25-0] << 2);

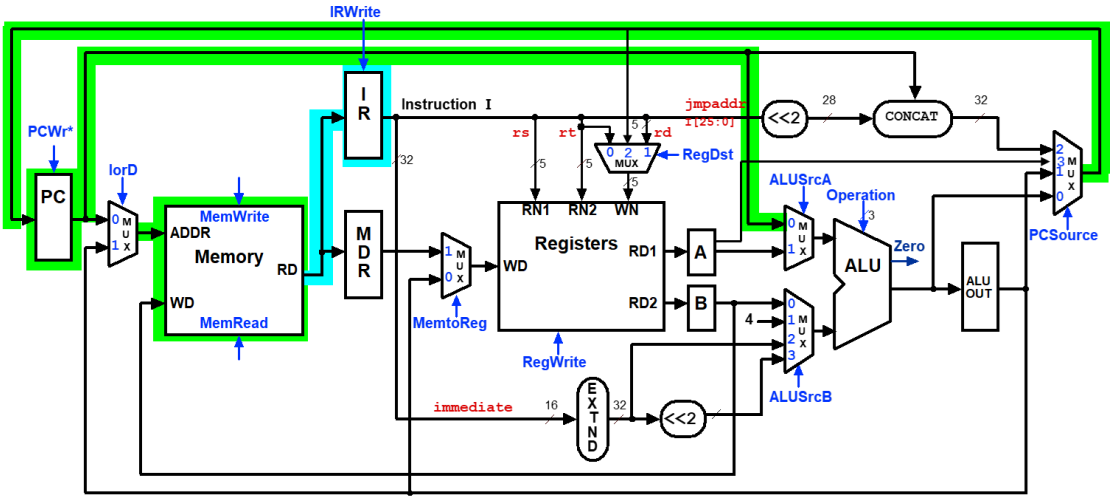


For jr:

(1)Instruction Fetch

<1>Fetch the instuction and store it in the IR

<2>Add 4 to the PC

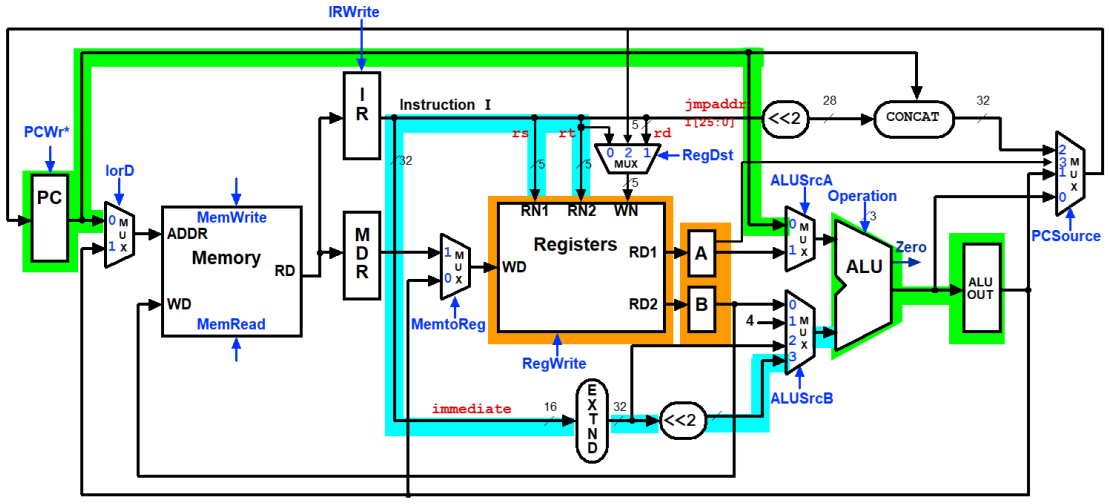


(2)Instruction Decode & Register Fetch

<1>A = Reg[IR[25 - 21]];

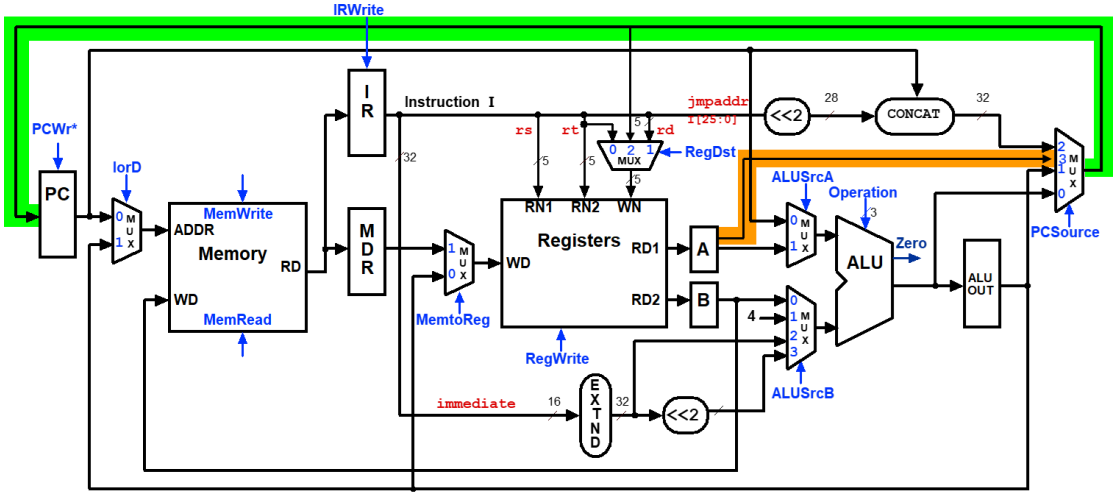
<2>B = Reg[IR[20 - 15]];

<3>ALUOut = (PC + sign-extend(IR[15 - 0]) << 2);



(3)Jump Instructions

<1>PC = PC[21-28] concat (IR[25-0] << 2);

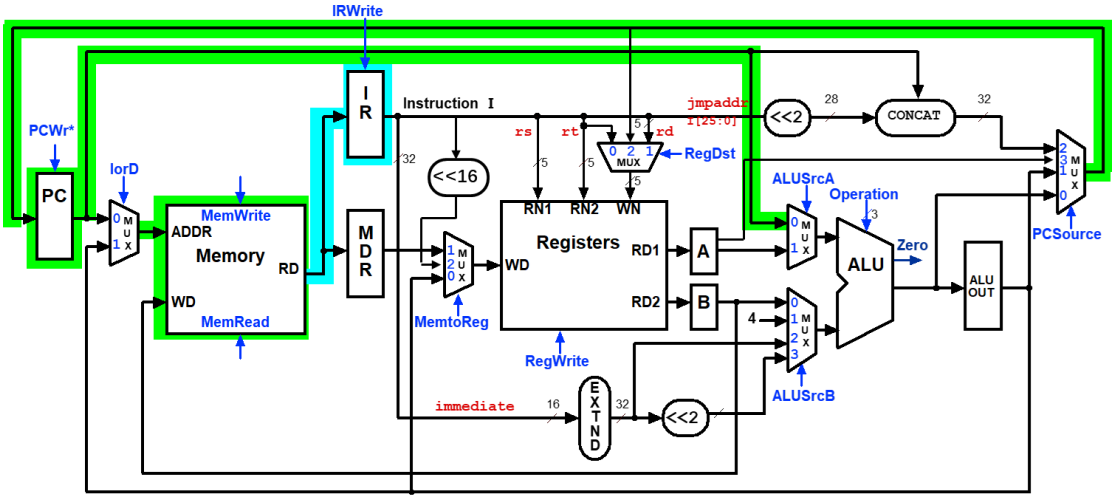


For lui:

(1)Instruction Fetch

<1>Fetch the instuction and store it in the IR

<2>Add 4 to the PC

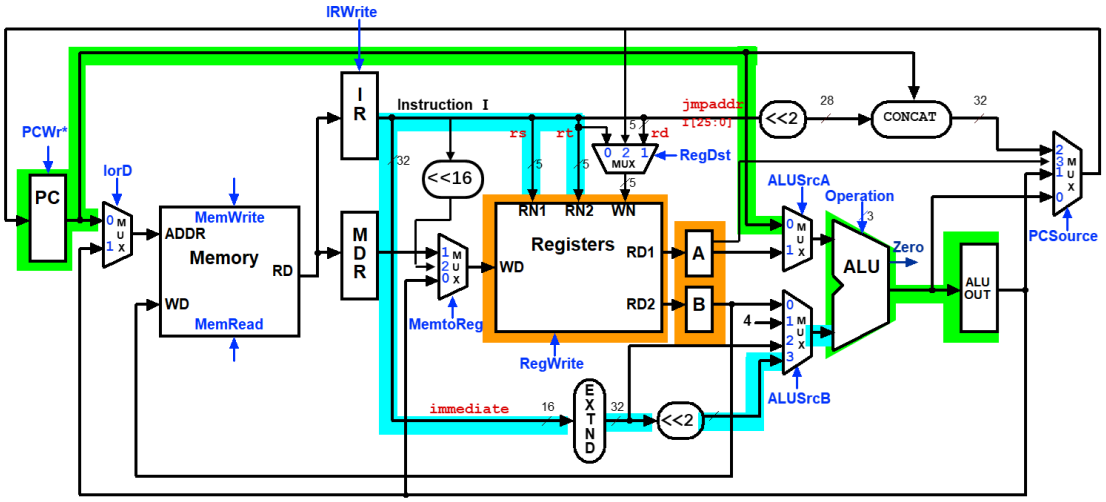


(2)Instruction Decode & Register Fetch

<1>A = Reg[IR[25 - 21]];

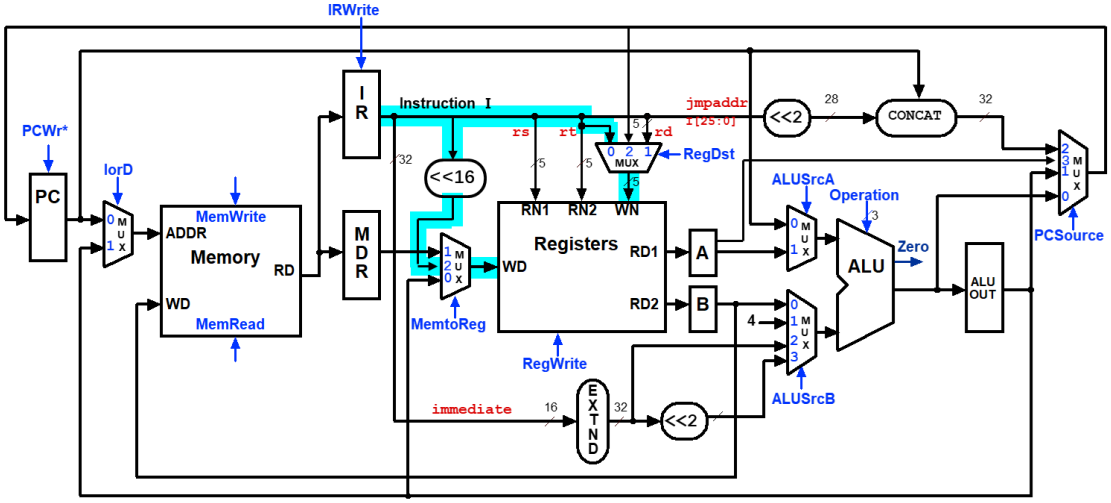
<2>B = Reg[IR[20 - 15]];

<3>ALUOut = (PC + sign-extend(IR[15 - 0]) << 2);



(3)Store the immediate

<1>Reg[IR[20 - 16]] = (IR[15 - 0] << 16)

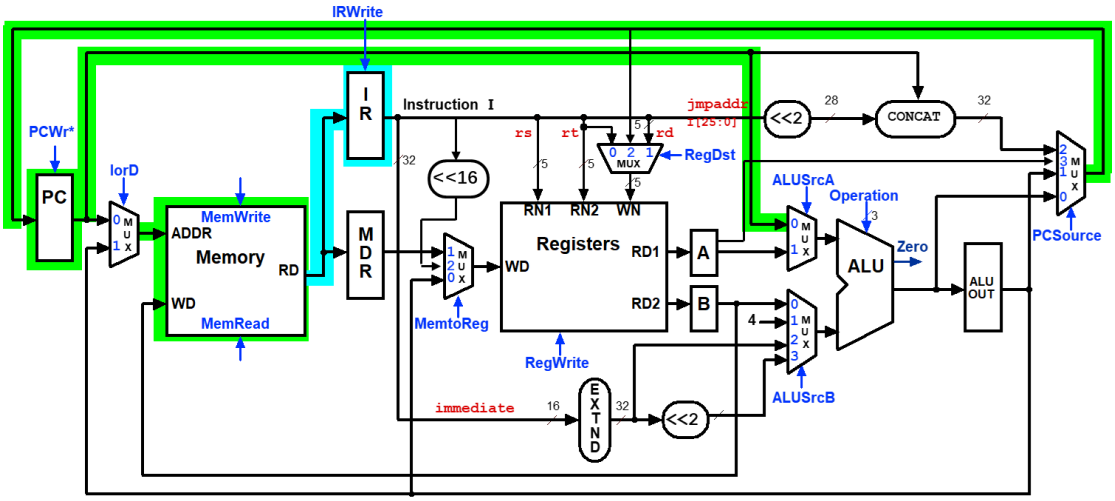


For ori:

(1)Instruction Fetch

<1>Fetch the instuction and store it in the IR

<2>Add 4 to the PC

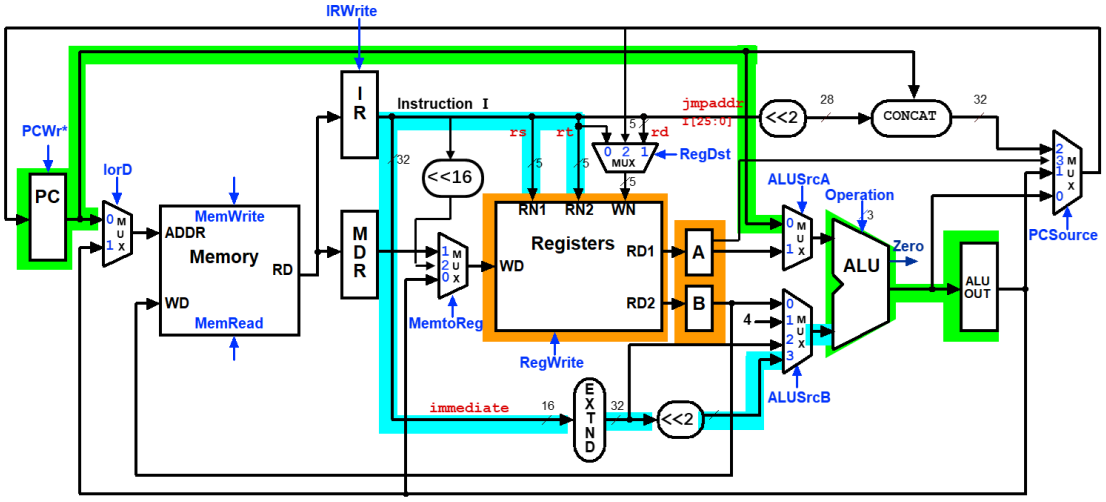


(2)Instruction Decode & Register Fetch

<1>A = Reg[IR[25 - 21]];

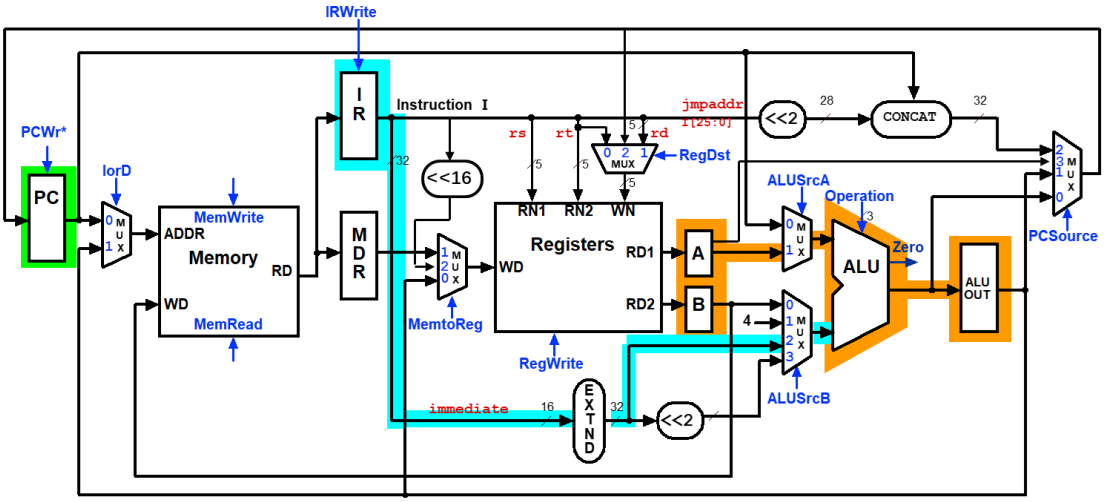
<2>B = Reg[IR[20 - 15]];

<3>ALUOut = (PC + sign-extend(IR[15 - 0]) << 2);



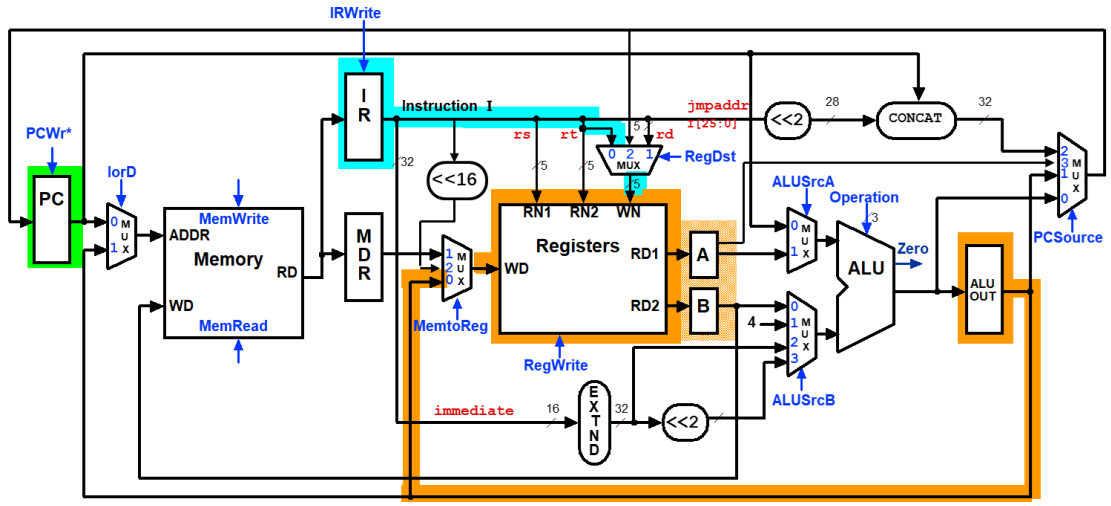
(3)Execution

<1>ALUOut <= A OR sign-extend(IR[15:0]);



(4)Compeletion

<1>Reg[IR[20-16]] = ALUOut;



4.x.2

(1)

The codes are as followings.

```
int MaxValue(int a[], int len)
{
	for(int i = 0; i < len; i++)
	{
		if(maxn > a[i])
		maxn = a[i];
	}
	return maxn;
}
```

(2)

The MIPS codes are as followings.

(We assume that the address of the array is $a0, the length of the array is $a1)

SubA:

```
addi $sp, $sp, -4    #clear a space for $ra
sw $ra, 0($sp)    #store the $ra
lw $v0, 0($a0)    #initialize the value of $v0
jal SubB    #jump to the SuB
lw $ra, 0($sp)    #get the $ra
addi $sp, $sp, 4    #recover the $sp
jr $ra
```

SubB:

```
addi $t0, $a0, 0    #get the address of array a
addi $t1, $a1, 0    #get the value of array a
beq $t1, $zero, loop    #go into the Loop
jr $ra
```

Loop:

```
beq $t1, $zero, SubB
addi $t1, $t1, -1    #len--
addi $t0, $t0, 4    #the current element's place of array a
lw $t2, 0($t0)    #get the length of a[i]
slt $t3, $t2, $v0    #compare
beq $t3, $zero, Swap    #if a[i] > maxn, maxn = a[i]
```

Swap:

```
addi $v0, $t2, 0    #maxn = a[i]
j Loop
```

(3)

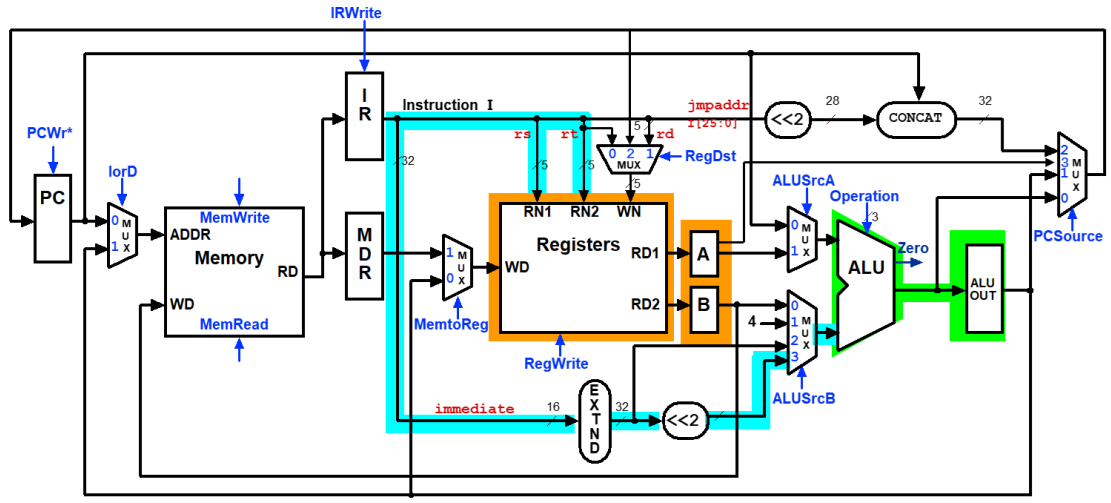
The codes of SubB is quite simple, the main job is done in the Loop.

For SubB:

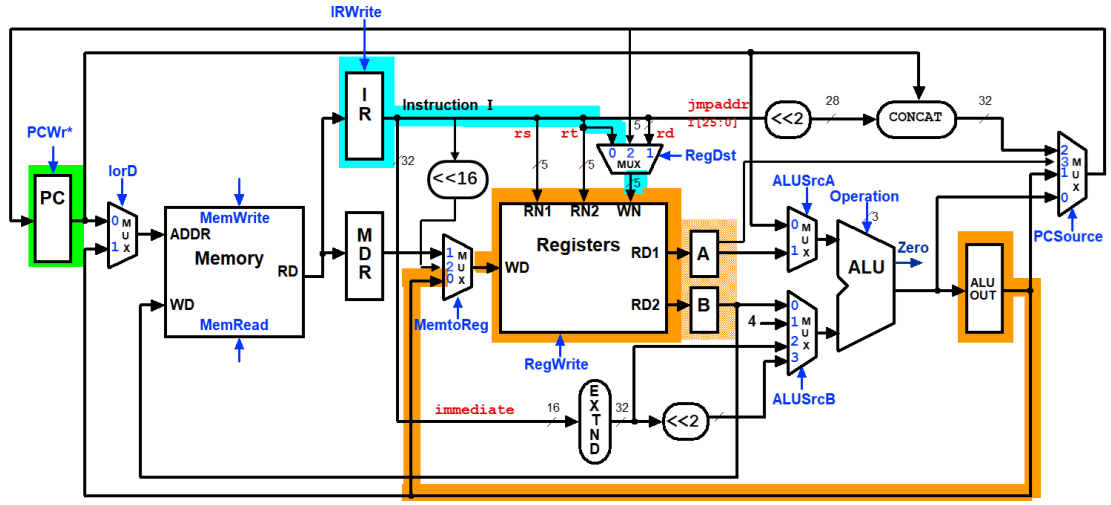
<1>addi $t0, $a0, 0   #get the address of array a

This is an I-type instruction.

The first two steps are all the same:Instruction Fetch, Instruction Decode & Register Fetch.



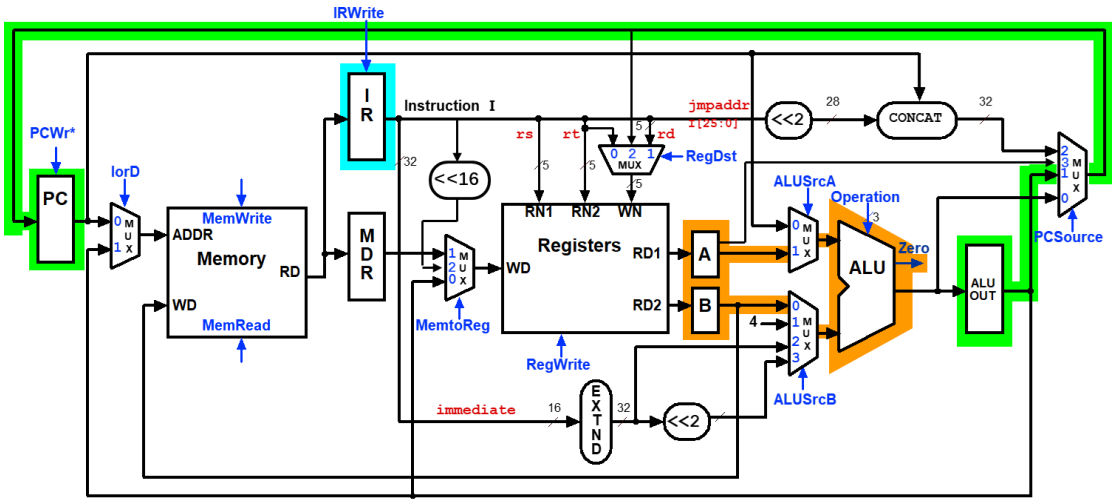
In the third step, it will calculate the value of Reg[a0] + 0 to ALUOut.
So it will use the immediate and Reg[a0], ALU to do this job.



At last it will store the value to Reg[t0].
So it will use the ALUOut and MemtoReg Mux would be 0 to store the value to Reg[t0].

<2> addi $t1, $a1, 0 #get the length of array a
It is the same as <1>.

<3> beq $t1, $zero, loop #go into the Loop
This is a Branch instruction.
The first two steps are all the same:Instruction Fetch, Instruction Decode & Register Fetch.



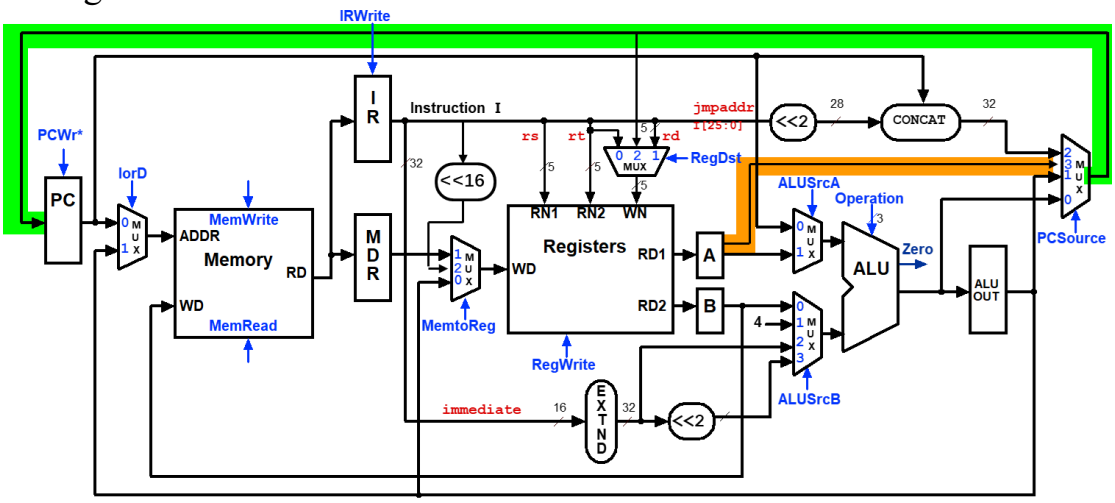
In the third step, it would compare if Reg[t1] = Reg[zero], and if it is, it

will pass the value of ALUOut to PC. So at this time, it would use the A, B and ALU to compare if Reg[t1] = Reg[zero], also the ALUOut would pass the value to PC. If Reg[t1] = Reg[zero], the value of ALUOut would sucessfully go into PC.

<4> jr $ra

This is a Jump insturction.

The first two steps are all the same:Instruction Fetch, Instruction Decode & Register Fetch.



In the third step, the value of Reg[ra] would directly go into the PC just like this.