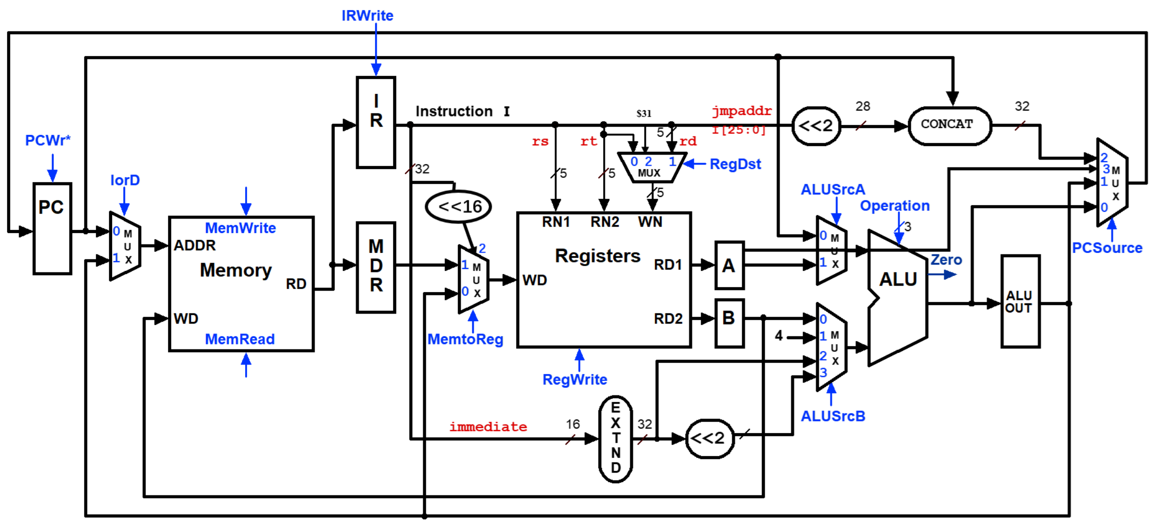
计算机组成原理作业3-2

2021141450109

4.x.1

A datapath which contains jal, jr, lui, ori.



1.jal(J-type)

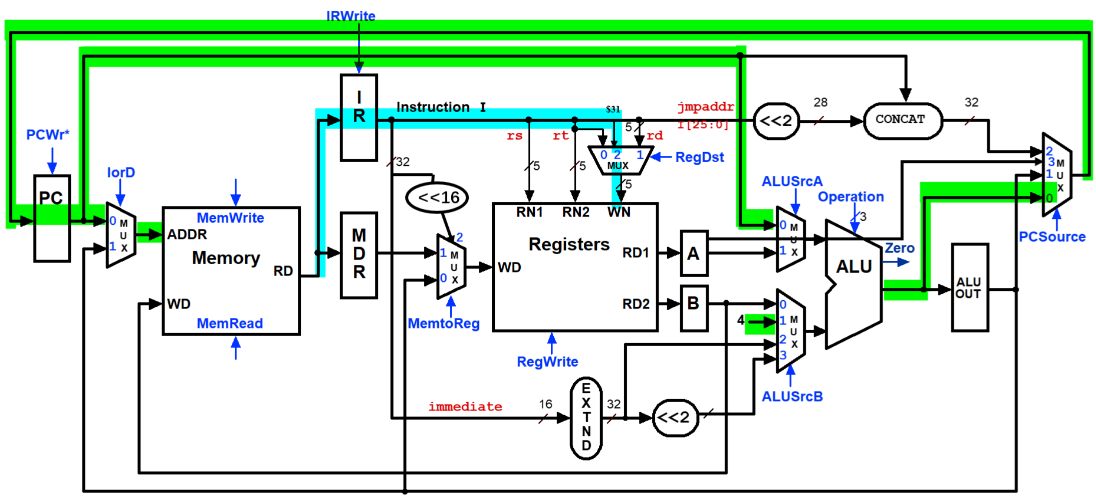
jal address

1. IF:

   IR=Memory[PC]

   PC=PC+4

   $31=PC

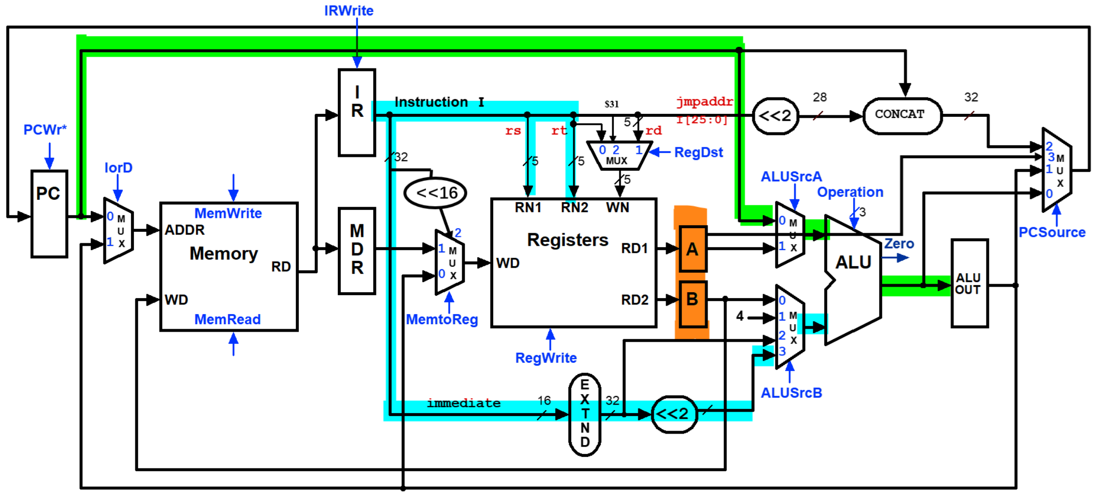


2. ID

   A=Reg[IR[25-21]]

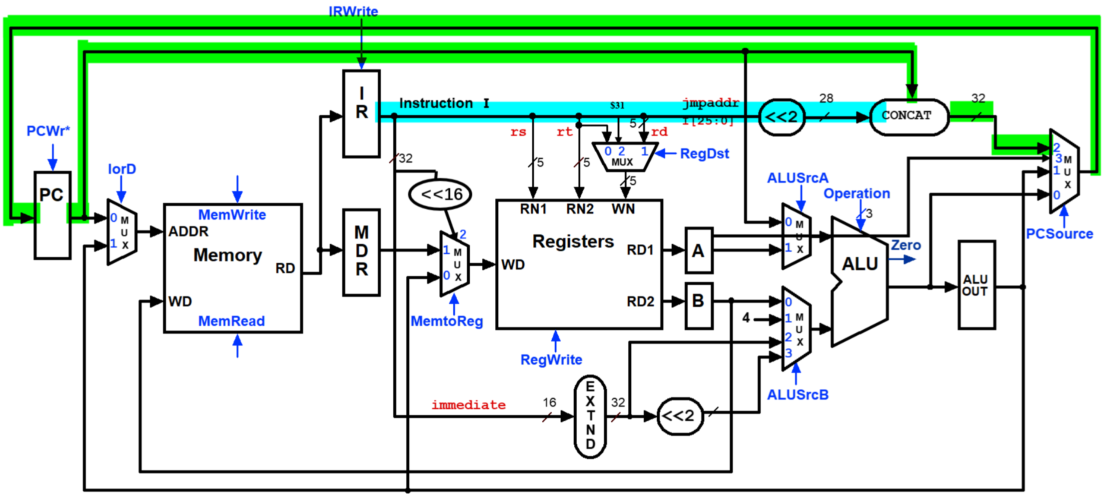
   B=Reg[IR[20-16]]

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



3. EX

PC=PC[31-28]||(IR[25-0]<<2)



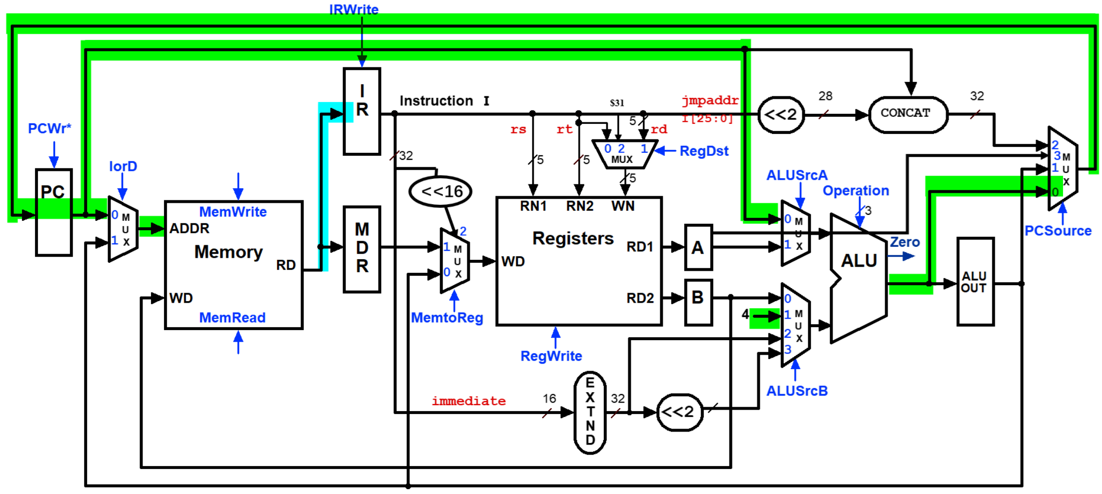
2.jr(R-type)

jr rs

1. IF:

IR=Memory[PC]

PC=PC+4

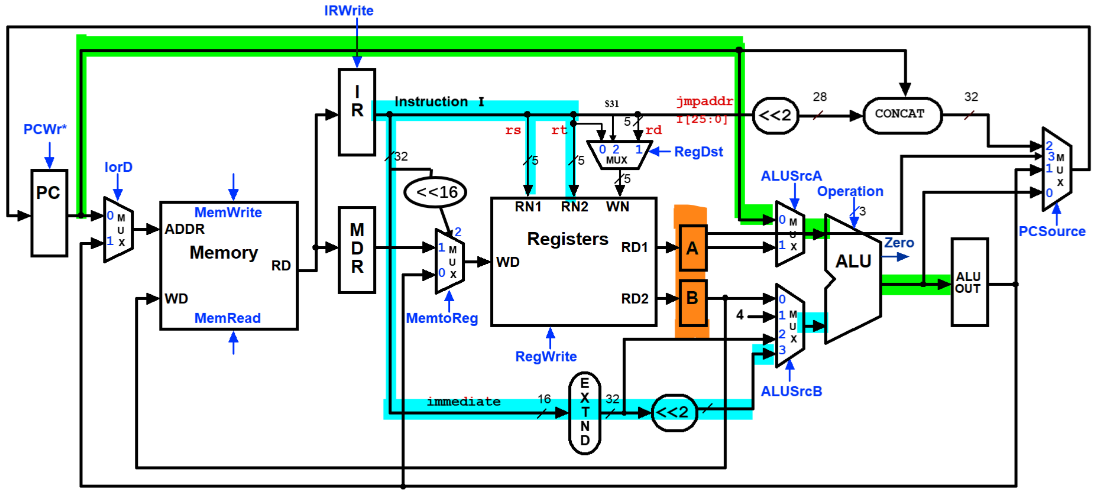


2. ID

A=Reg[IR[25-21]]

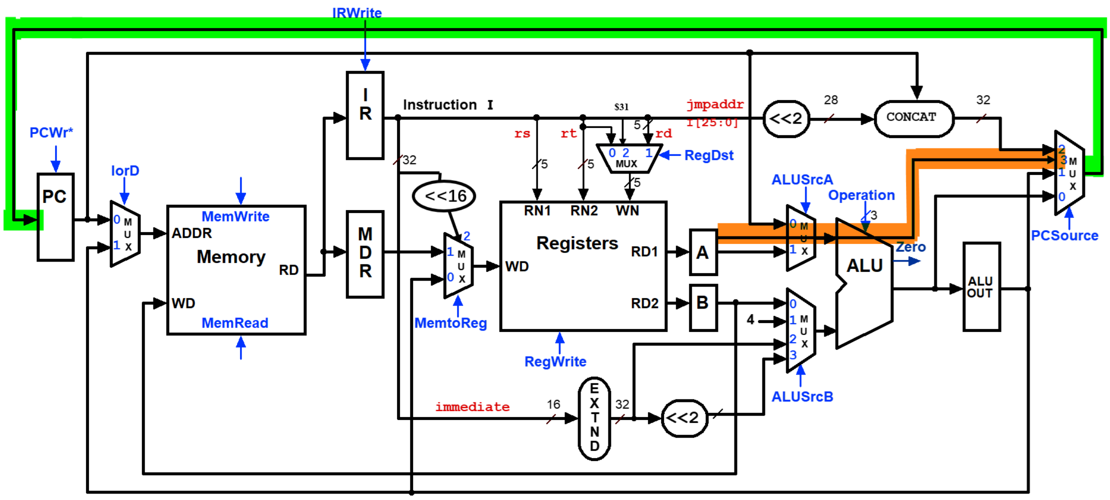
B=Reg[IR[20-16]]

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



3. EX

PC=A



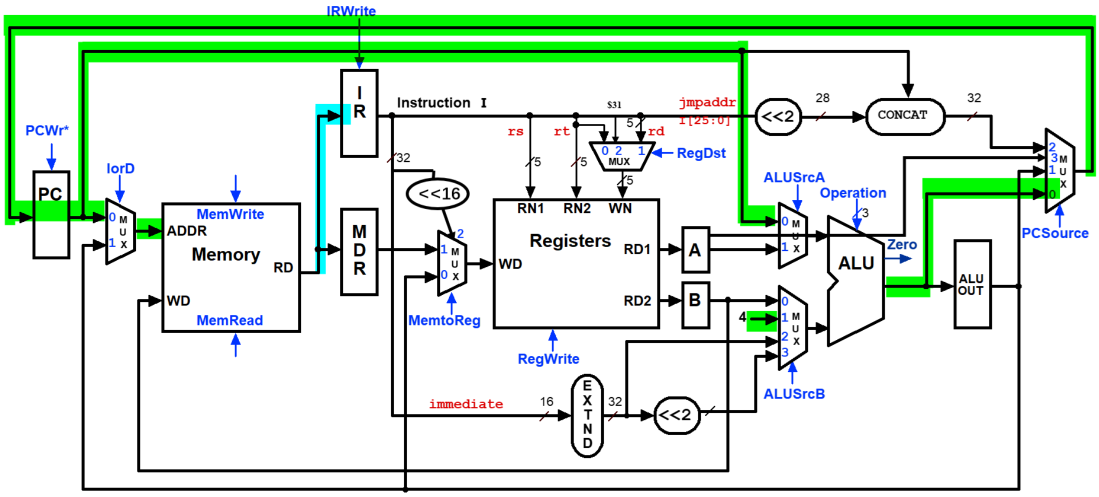
3.lui(I-type)

lui rt immediate

1. IF:

   IR=Memory[PC]

   PC=PC+4

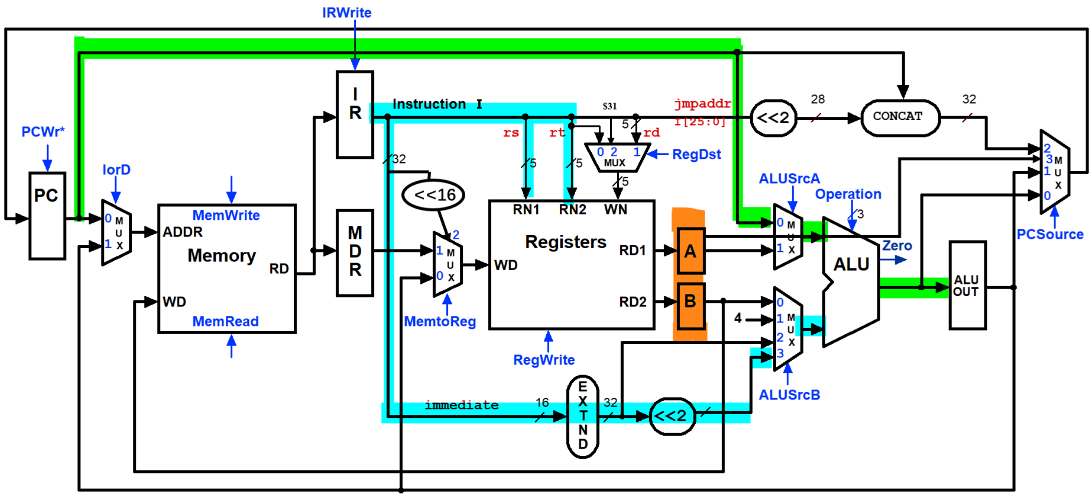


2. ID:

   A=Reg[IR[25-21]]

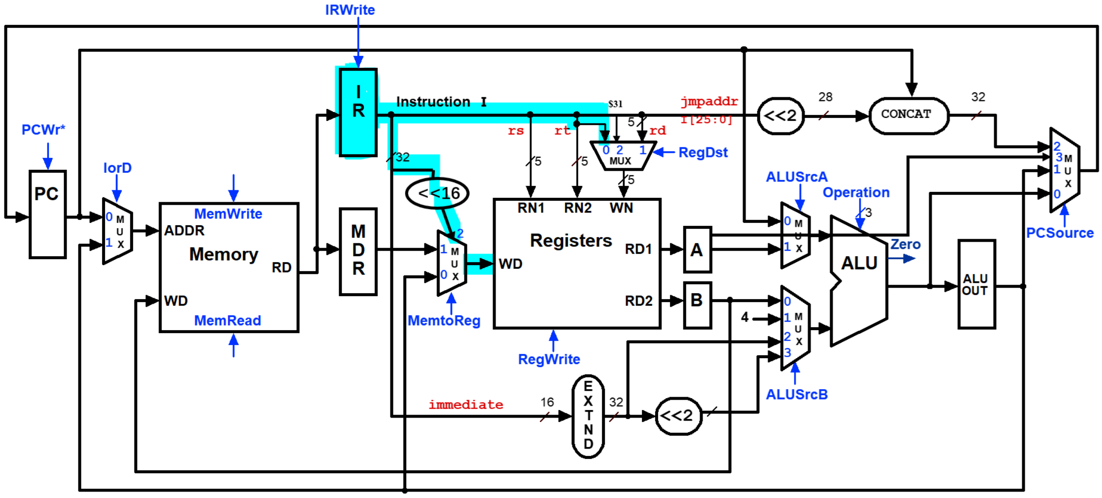
   B=Reg[IR[20-16]]

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



3. EX:

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



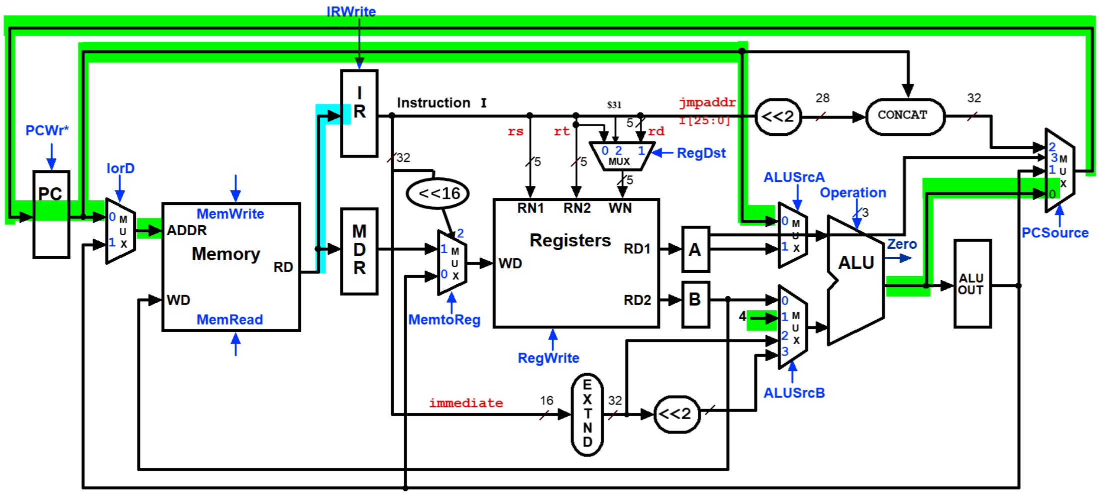
4. ori(I-type)

ori rs rt immediate

1. IF:

   IR=Memory[PC]

   PC=PC+4

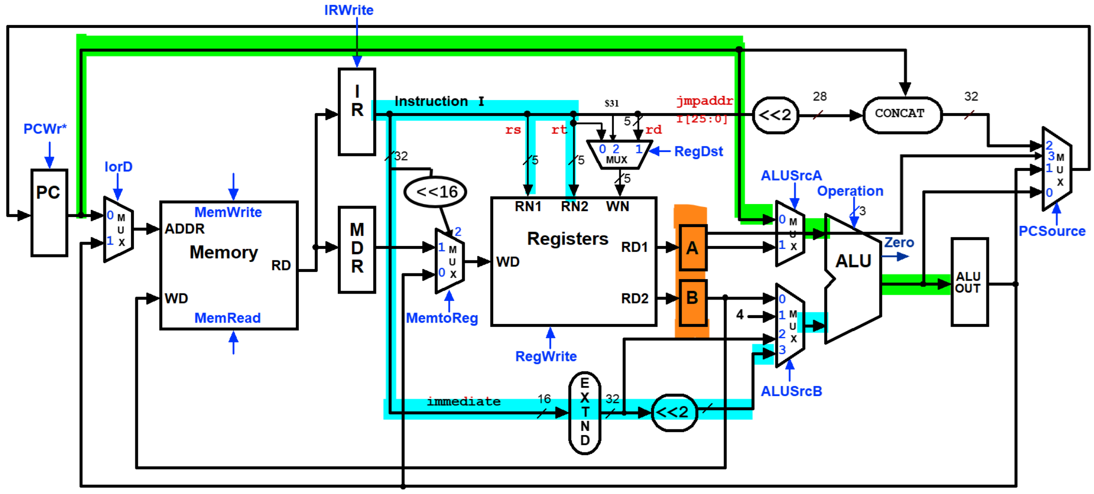


2. ID:

   A=Reg[IR[25-21]]

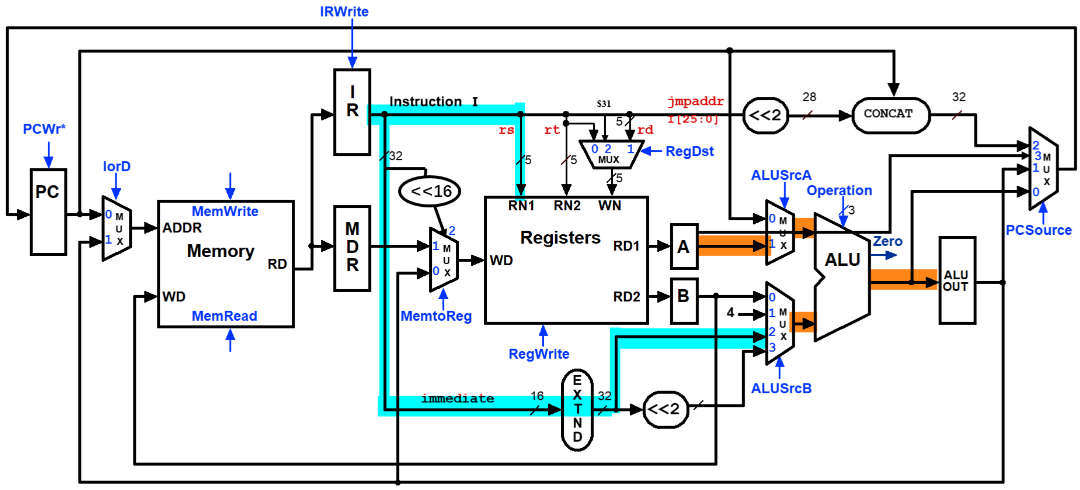
   B=Reg[IR[20-16]]

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



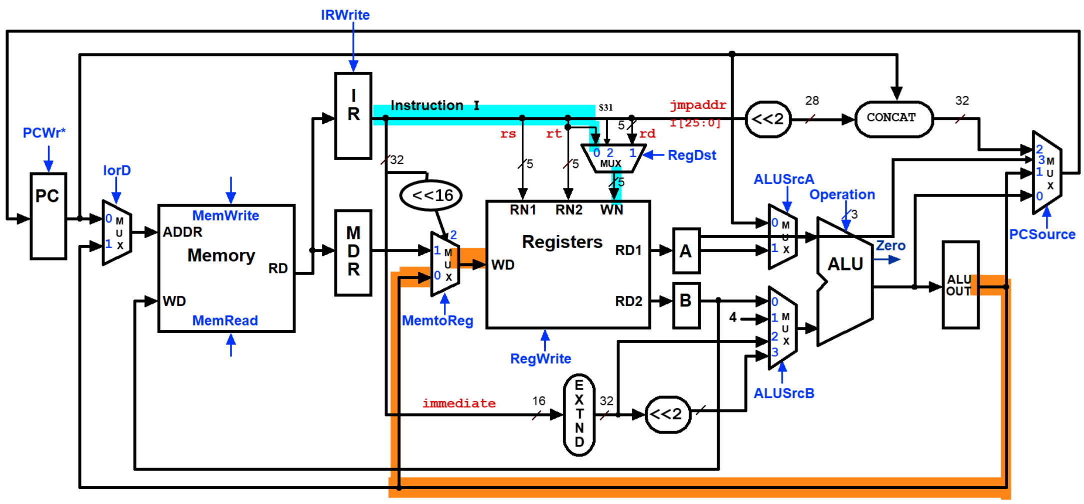
3. EX:

ALUOut=A or IR[15-0]



4. MEM:

Reg[IR[20-16]]=ALUOut



4.x.2

(1)

```
int MaxValue(int a[], int count) {
    int mx = a[0];
    for (int i = 1; i < count; i++) {
        if (a[i] > mx)mx = a[i];
    }
    return mx;
}
```

(2)

```
.data
a:          .word 7 -7 7 9 11 # an arbitray array and its length is 5

.text
main:
            la $a0,a   # la is a pseudo-instruction which can be mixed by lui and ori
            addi $a1,$zero,5 # a1 represents the length of the array
            addi $sp,$sp,-4 # push ra into the stack
            sw $ra,0($sp)
            jal subB # jump to subB
            lw $ra,0($sp) # pop the ra from the stack
            addi $sp,$sp,4
            jr $ra # returns control to the caller

subB:
            addi $sp,$sp,-4 # push ra into the stack
            sw $ra,0($sp)
            lw $s1,0($a0) # store $a0 in the $s1
            addi $a1,$a1,-1 # $a1-=1
            addi $a0,$a0,4 # $a0 represents the current address of the array
            slti $t0,$a1,0 # set $t0=1 when $a1<0
            beq $t0,$zero,Loop # $t0=0 means the loop hasn't finished
            addi $sp,$sp,4
            jr $ra # returns control to the caller

Loop:
            slt $t1,$s0,$s1 # set $t1=1 when $s0<$s1
            beq $t1,$zero,subB # $t1=0 means $s0>$s1, so we don't need to change the maxvalue
            add $s0,$zero,$s1  #s0 represents the maxvalue
            jal subB # jump to subB
```

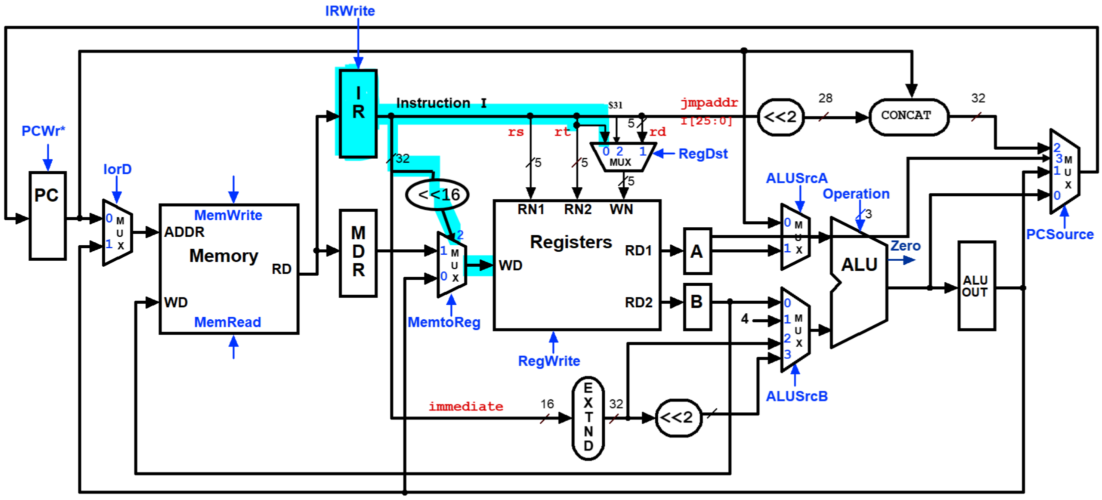
R16 (s0) = 0000000b

We can see the maxvalue of the arbitray array is b, which determines the accuracy of this program.

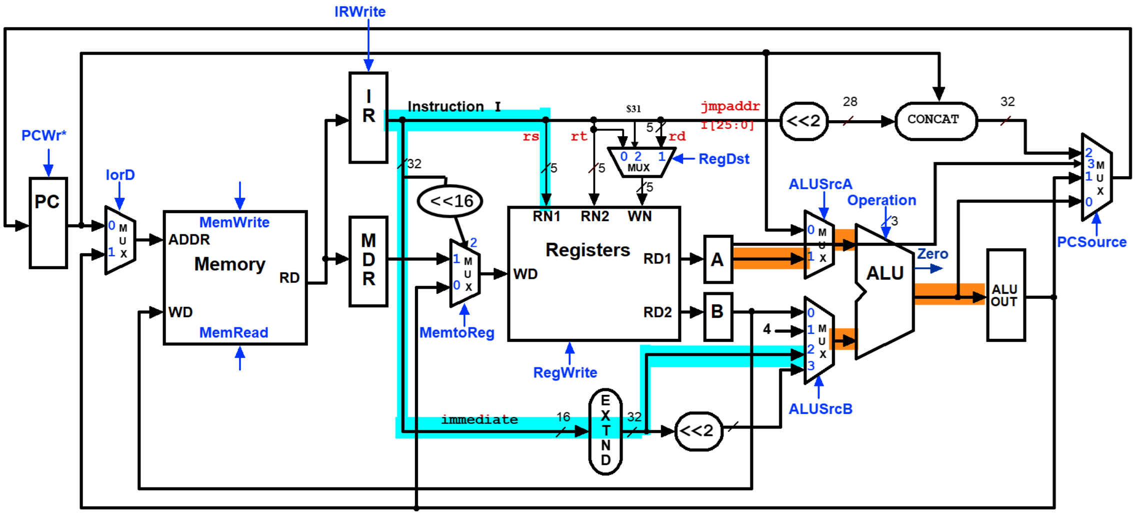
(3)

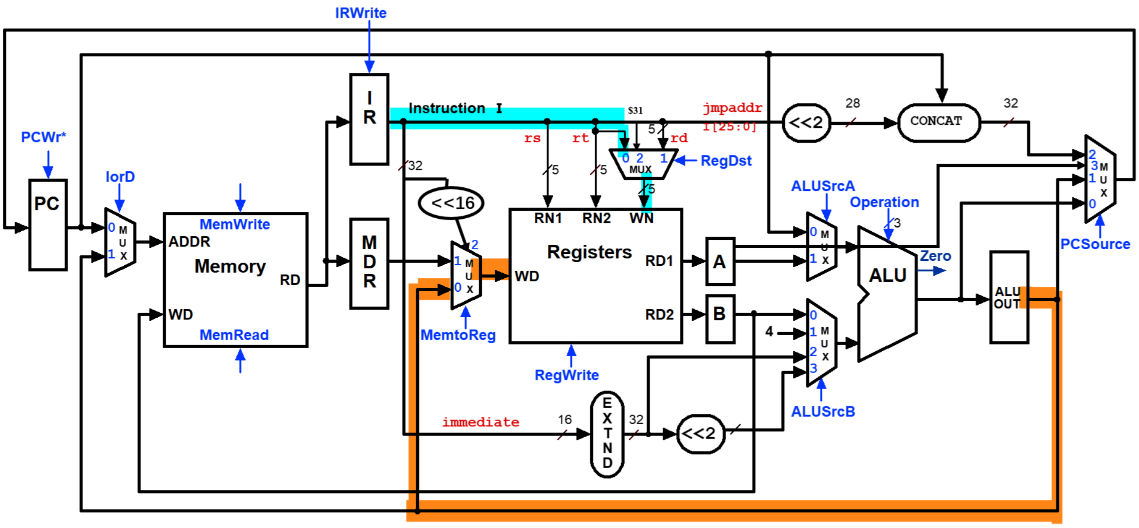
1. la is a pseudo-instruction which can be mixed by lui and ori. They can be implemented just like 4.x.1. This instruction can get the address of a and store it to the destination register $a0.

   lui:



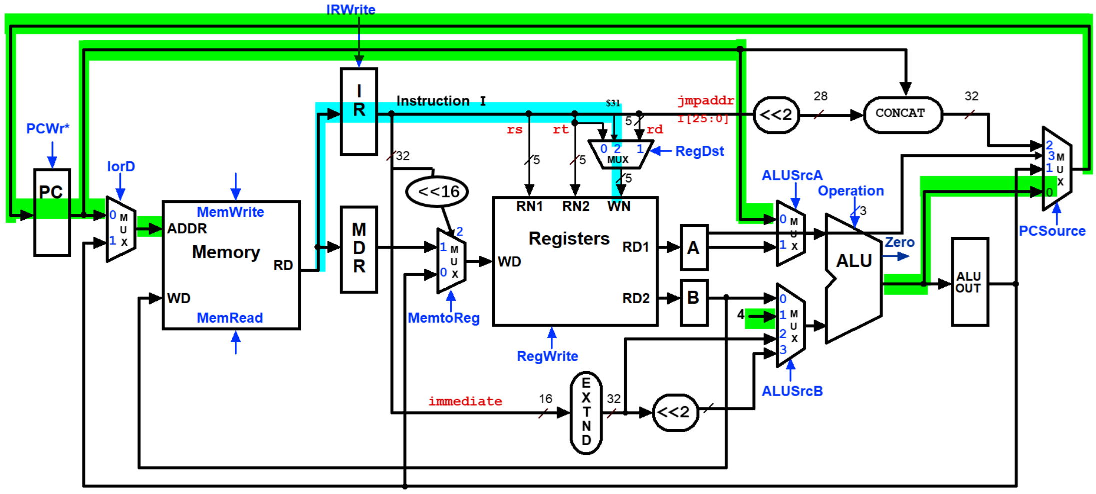
   ori:





2. jal SubB

The register that is used for linkage is register `$31`, which is called `$ra` by the extended assembler. It holds the return address for a subroutine. The instruction that puts the return address into `$ra` is (usually) the `jal` instruction. And in this program, we jump to the function SubB by this instruction.



3. jr $ra

The `jr` instruction returns control to the caller. It copies the contents of `$ra` into the PC:

```
jr  $ra       # PC <— $ra
              # A branch delay
              # slot follows this instruction.
```

Usually you think of this as "jumping to the address in `$ra`."

To make the instruction more general, it can be used with any register rs, not just `$ra`. Like all jump and branch instructions, the `jr` instruction is followed by a branch delay.