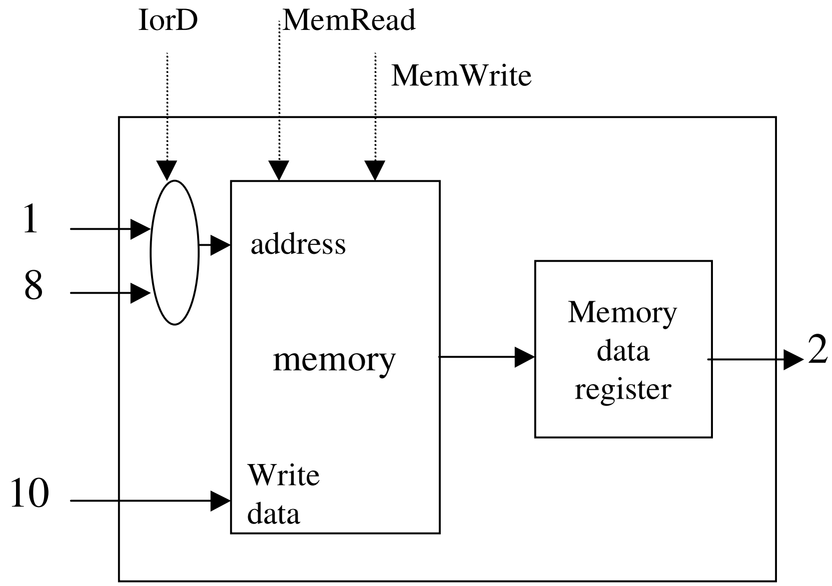
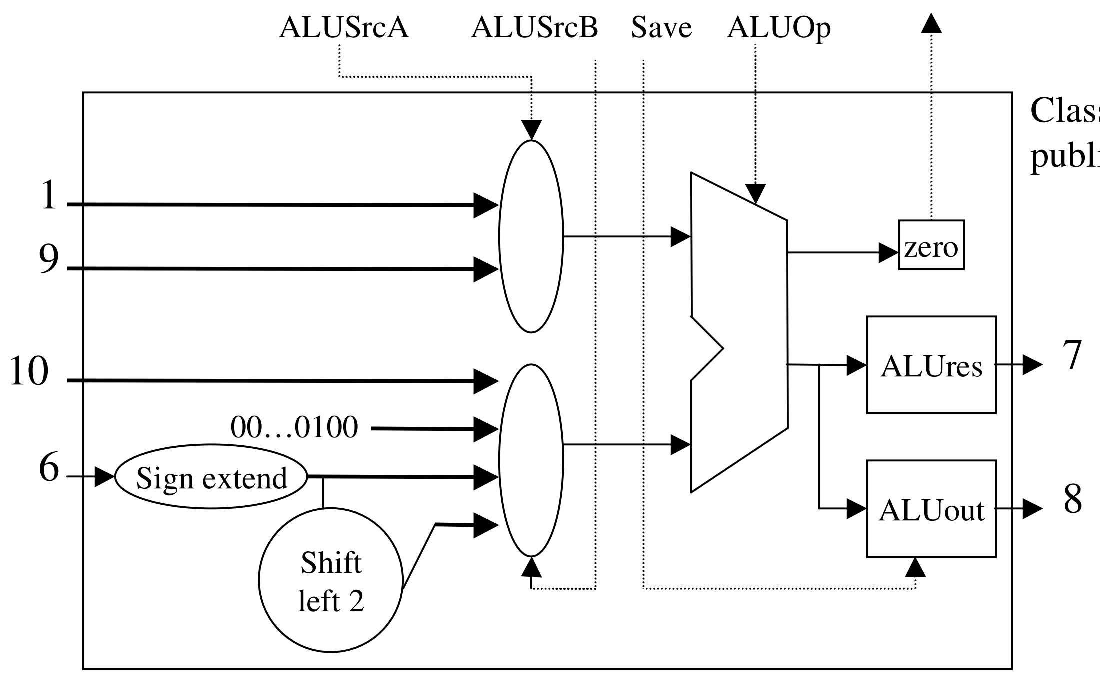
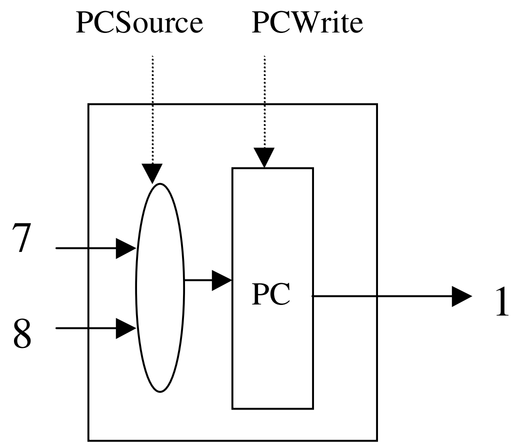


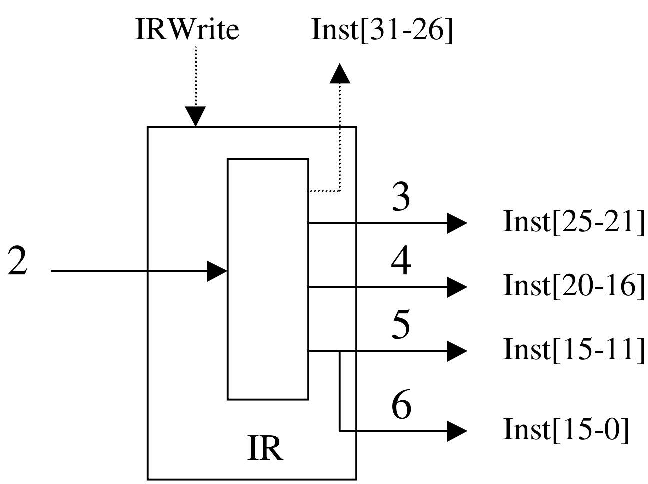

```
Class mem_unit{
   public: unsigned long MemData ;
           void  MemRead(bit IorD) ;
           void MemWrite(bit IorD) ;
}
```



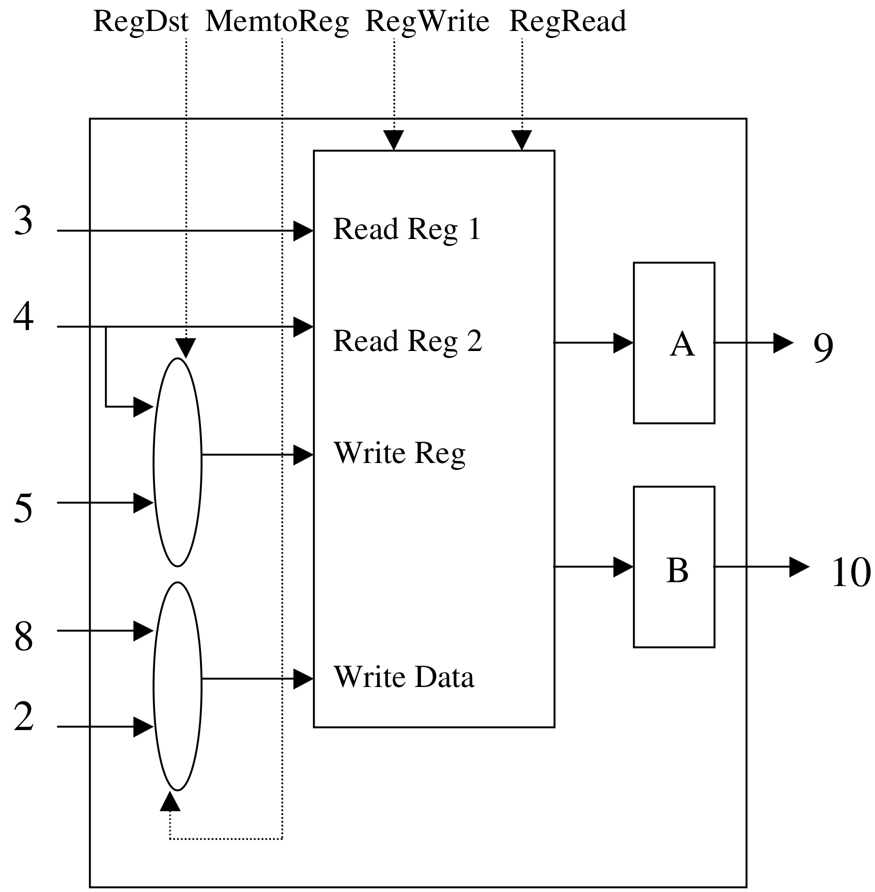

```
Class alu_unit{
public: void ALUadd(bit ALUSrcA, bit[2] ALUSrcB);
        void ALUsub(bit ALUSrcA, bit[2] ALUSrcB);
        void ALUor(bit ALUSrcA, bit[2] ALUSrcB);
        void Save( );
        unsigned long ALUOut ;
        unsigned long ALUres;
        bit Zero ;
          }
```



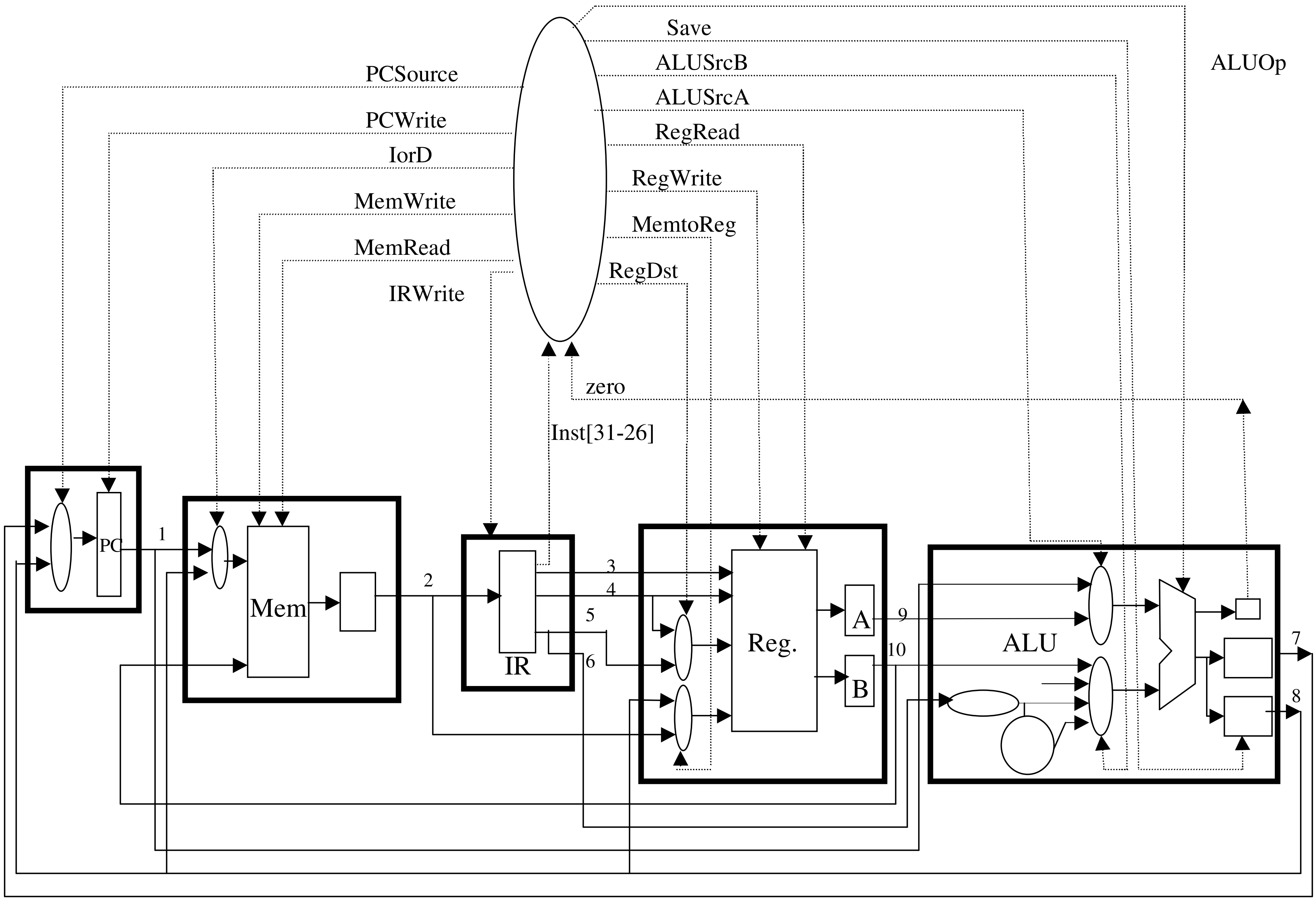

```
Class PC_unit{
  public: unsigned long pc;
            void PCWrite(bit PCSource);
}
```

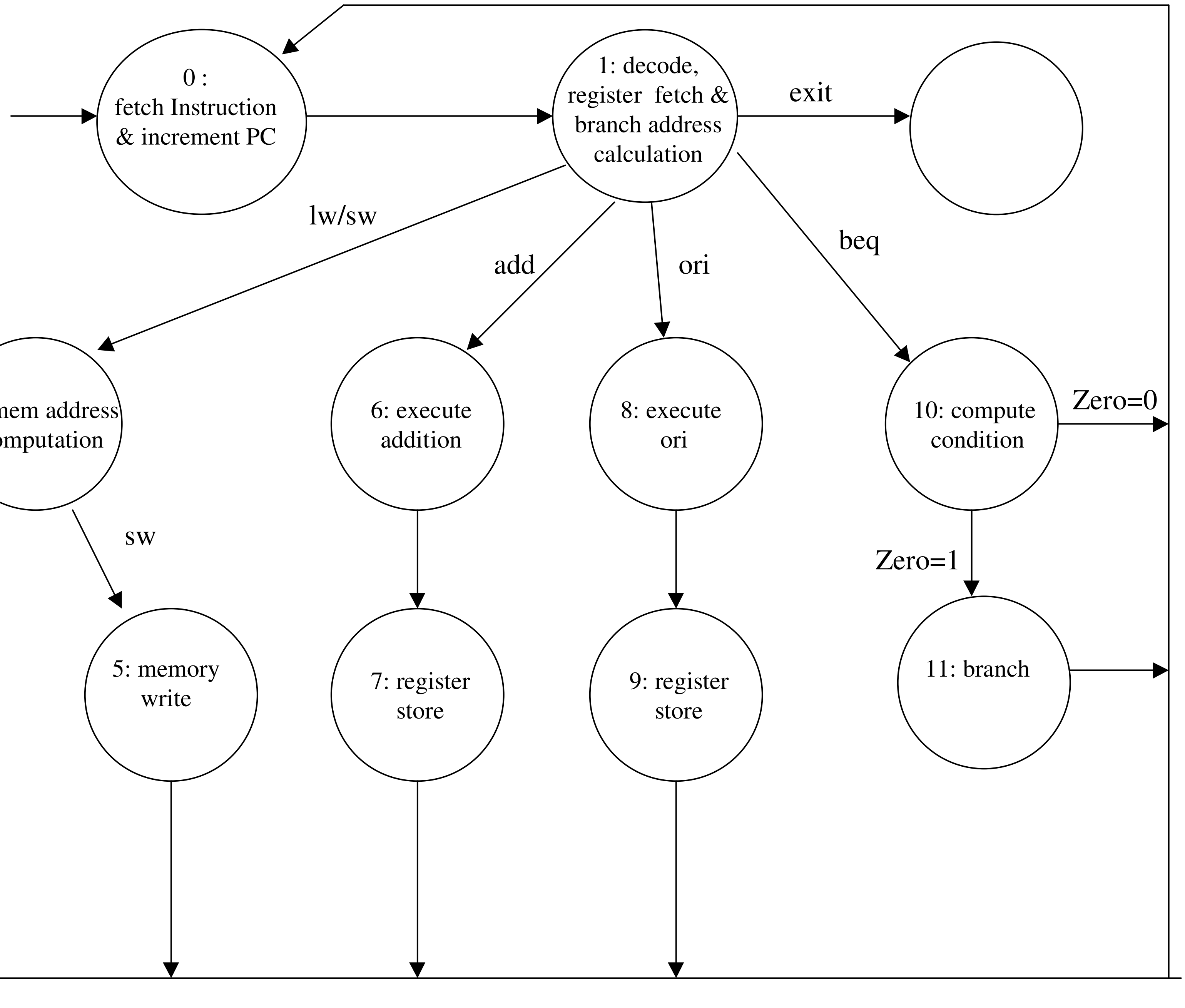



```
Class IR_unit{
  public : bit[6]  opcode ;       /* Inst[31-26] */
           bit[5]  srca_reg ;     /* Inst[25-21] */
           bit[5]  srcb_reg;      /* Inst[20-16] */
           bit[5] dest_reg ;      /* Inst[15-11] */
           bit[16] immediate;     /* Inst[15-0]   */
           void IRWrite();
}
```




```
Class Registers_unit{
  public: unsigned long A;
          unsigned long B;
          void RegWrite(bit RegDest, bit MemtoReg);
          void RegRead() ;
}
```


Save
ALUSrcB
ALUSrcA
RegRead
RegWrite
MemtoReg
RegDst
PCSource
PCWrite
IorD
MemWrite
MemRead
IRWrite
ALUOp
zero
Inst[31-26]
PC
1
Mem
2
IR
3
4
5
6
Reg.
A
B
9
10
ALU
7
8



0 :
fetch Instruction
& increment PC
1: decode,
register fetch &
branch address
calculation
exit
lw/sw
add
ori
beq
2: mem address
computation
6: execute
addition
8: execute
ori
10: compute
condition
Zero=0
lw
sw
Zero=1
3: memory
read
5: memory
write
7: register
store
9: register
store
11: branch
4: register
store