# ECE243

## CPU

Prof. Enright Jerger

## IMPLEMENTING A SIMPLE CPU

- How are machine instructions implemented?
- What components are there?
- How are they connected and controlled?

# MINI ISA:

- every instruction is 1-byte wide
  - data and address values are also 1-byte wide
- address space
  - byte addressable (every byte has an address)
  - 8 addr bits => 256 byte locations
- 4 registers:
  - k0..k3
- PC (resets to $80)
- Condition codes:
  - Z (zero), N (negative)
  - these are used by branches

# Some Definitions:

- IMM3: a 3-bit signed immediate, 2 parts:
  - 1 sign bit: sign(IMM3)
  - 2 bit value: value(IMM3)
- IMM4: a 4-bit signed immediate
- IMM5: a 5-bit unsigned immediate
- R1, R2: registers variables
  - represent one of k0..k3
- SE8(X):

- means sign-extend value X to 8 bits

- NOTE: ALL INSTS DO THIS LAST:
  - PC = PC + 1
    = PC + sizeof(Instr)

# Mini ISA Instructions

## load R1 (R2):

R1 = mem[R2]
PC = PC+1

## store R1 (R2):

mem[R2] = R1
PC = PC+1

## add R1 R2

R1 = R1 + R2
if (R1 == 0) Z=1, else Z=0
if (R1 < 0) N=1, else N=0
PC = PC+1

* Set cond. codes

## sub R1 R2

R1 = R1 - R2
Set cond. codes
PC = PC+1

## nand R1 R2

R1 = R1 bit-wise NAND R2

Set cond. codes

PC = PC + 1

## ori IMM5

R1 = R1 bitwise-OR Imm5

Set cond. Codes

PC = PC + 1

## shift R1 IMM3

if (sign(Imm3)) R1 = R1 << val(Imm3)

else R1 = R1 >> val(Imm3)

Set cond. Codes

PC = PC + 1

## bz IMM4

if (Z == 1) PC = PC + SE8(Imm4)

PC = PC + 1

## bnz IMM4

if (Z == 0) PC = PC + SE8(Imm4)

PC = PC + 1

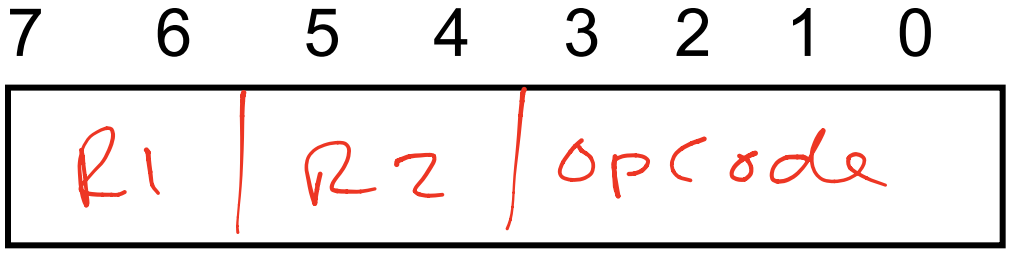
bpz IMM4

if (N==0) PC = PC + SE8(Imm4)

PC = PC+1

# ENCODINGS: Inst(opcode)

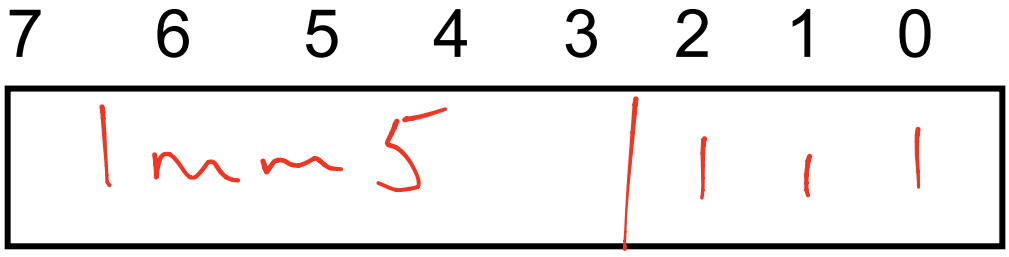
- Load(0000), store(0010), add(0100), sub(0110), nand(1000):




Ex: add R2 R1
encoding: 1001 0100

Ex: load R3 (R0)
encoding: 1100 0000

- Ori:

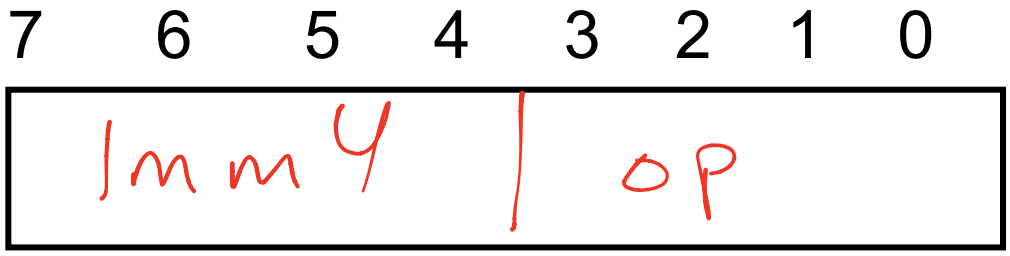


Ex: ori 5
encoding 0010 1111

- Shift:


Ex: Shift k3 2
encoding 1101 0011

- BZ(0101), BNZ(1001), BPZ(1101):



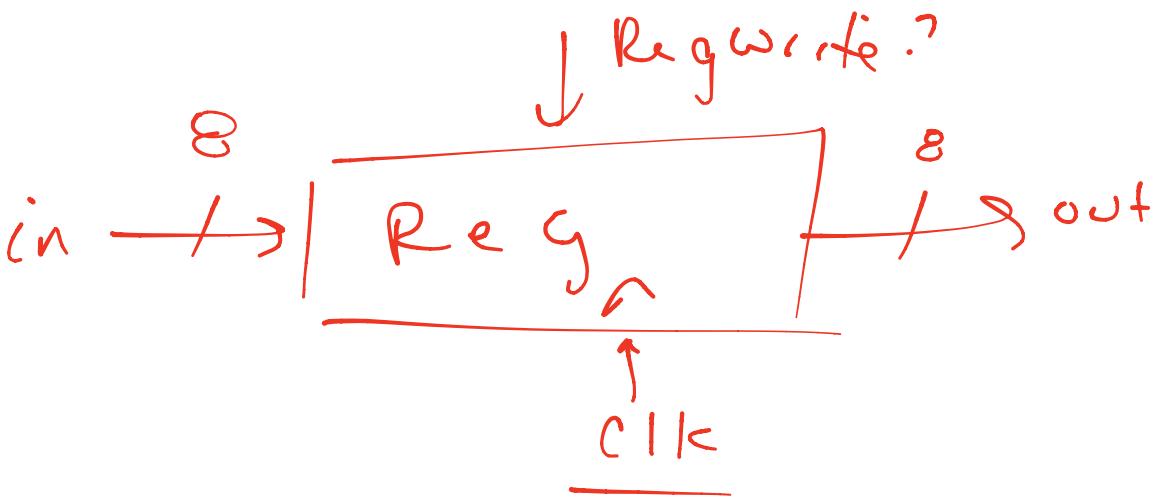

Ex: bnz -2
encoding: 1110 1001

# DESIGNING A CPU

- Two main components:
  - *datapath* and *control*
- datapath:
  - registers, functional units, muxes, wires
  - must be able to perform all steps of every inst
- control:
  - a finite state machine (FSM)
  - commands the datapath
  - performs: fetch, decode, read, execute, write, get next inst

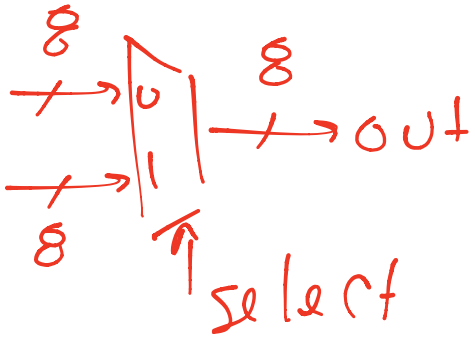
# CPU: basic components

# REGISTERS



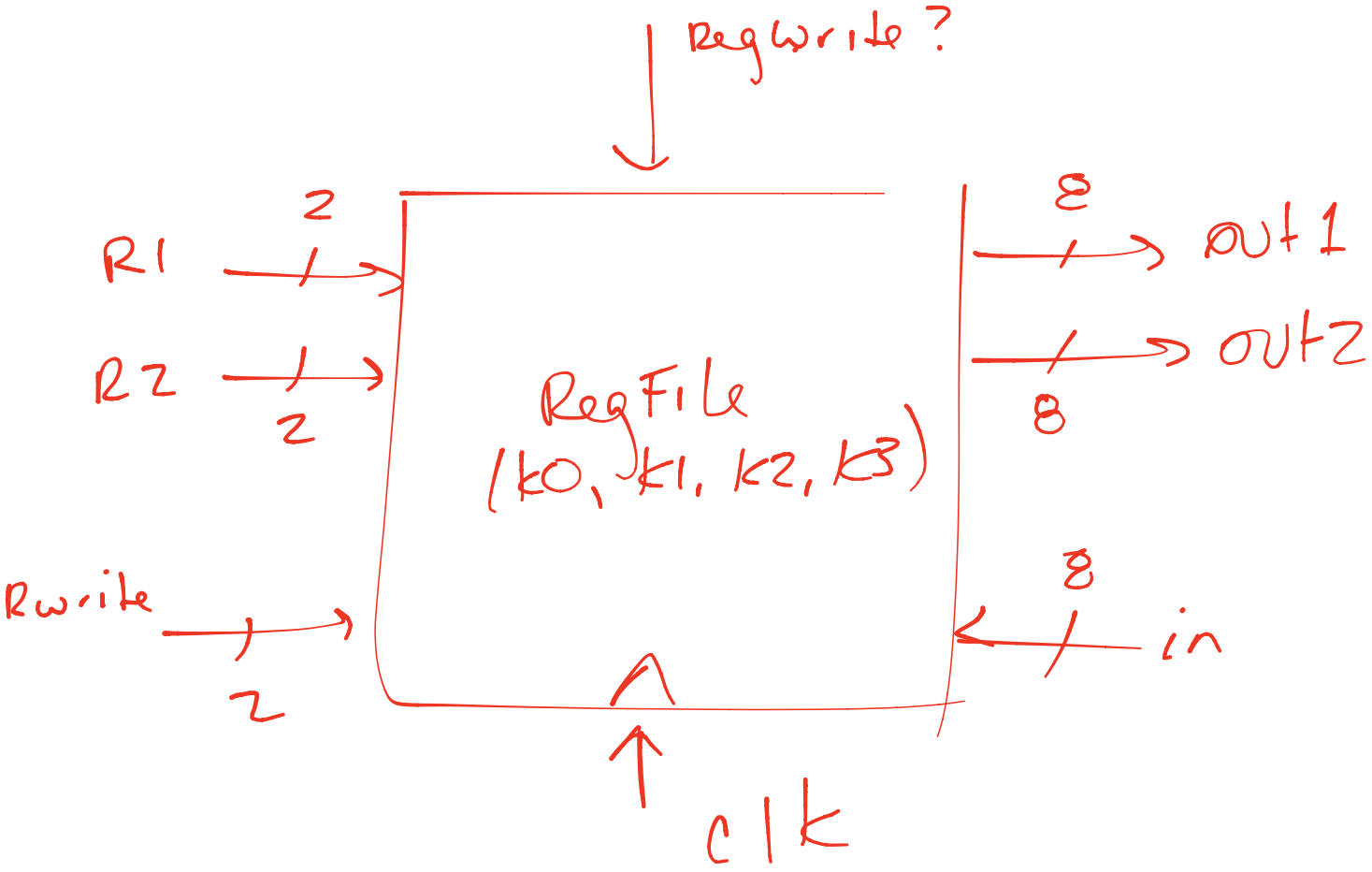

- **REGISTERS**
  - can always read
  - we assume falling-edge-triggered
  - in is stored if REGWrite=1 on falling clock edge
  - we won't normally draw the clock input

# MUXES




- 'select' signal chooses which input to route to output

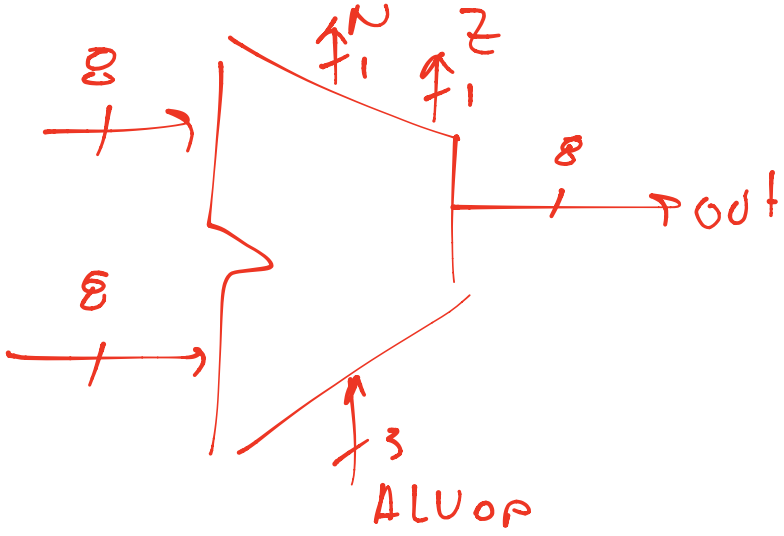
# REGISTER FILE




- Out1 is the value of reg indexed by R1
- Out2 is the value of reg indexed by R2
- if REGWrite is 1 when clock goes low
  - then the value on 'in' is written to reg indexed by Rwrite

# ALU (arithmetic logic unit)

- ALUop:
  - add = 000
  - sub = 001
  - or = 010
  - nand = 011
  - shift = 100

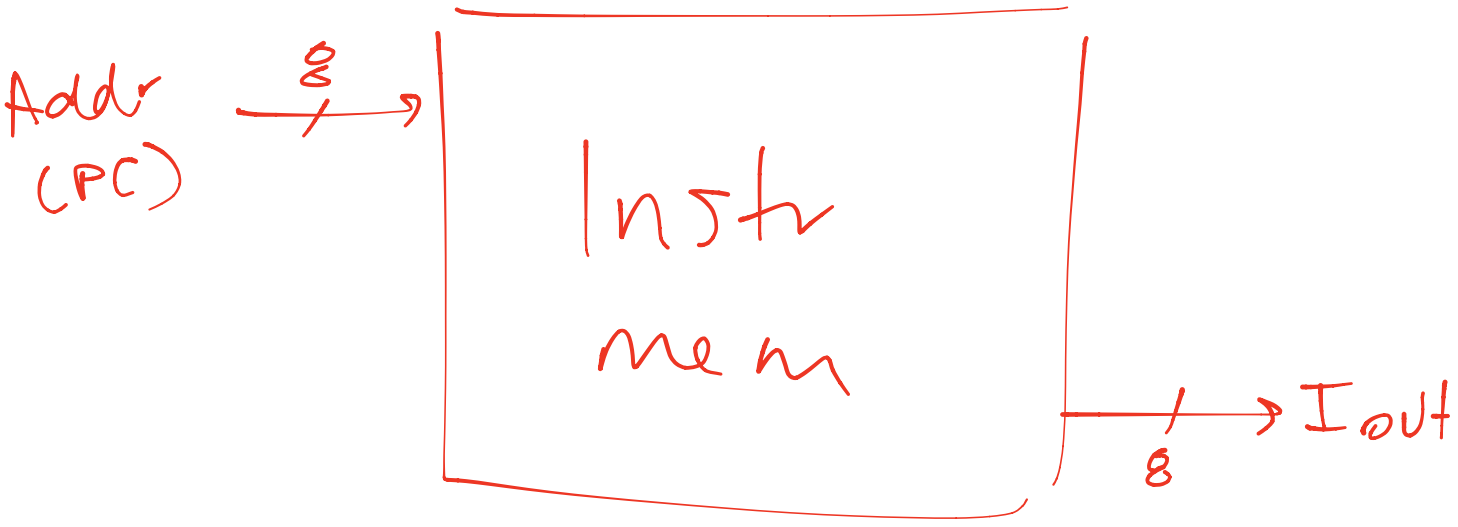


- $\overline{Z}$ = nor(out7,out6,out5…out0)
- N = out bit 7  (implies negative---sign bit)

# MEMORY

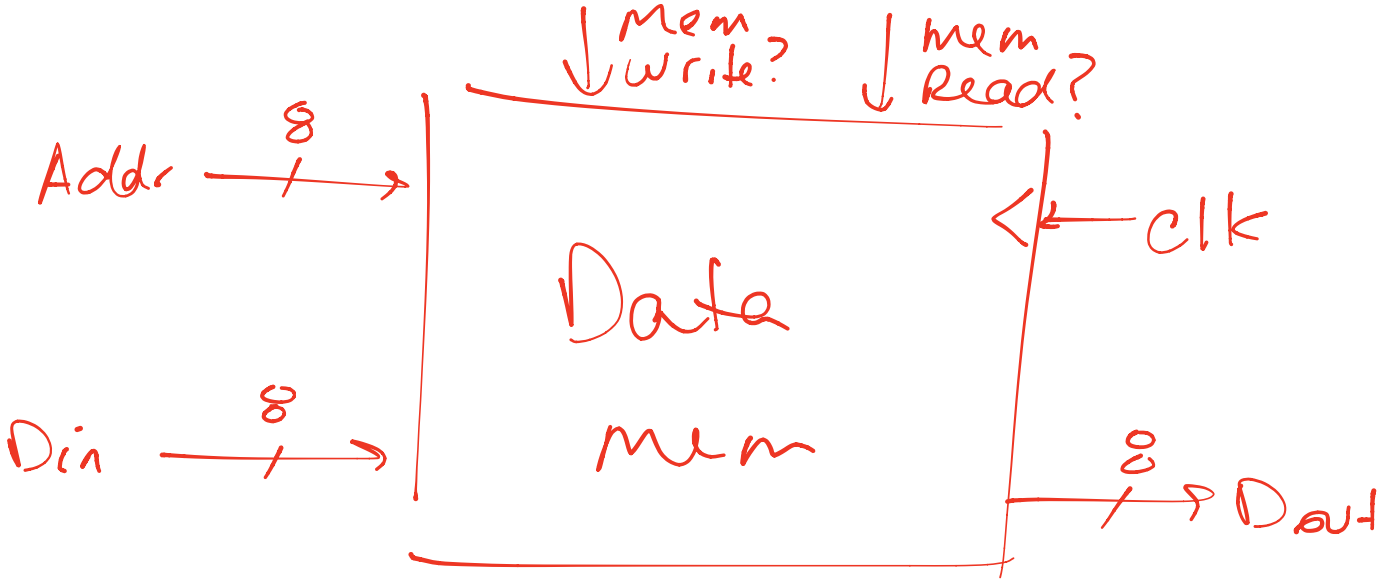
- our CPU has two memories for simplicity:
  - instruction memory and data memory
  - known as a “Harvard architecture”

# INSTRUCTION MEM



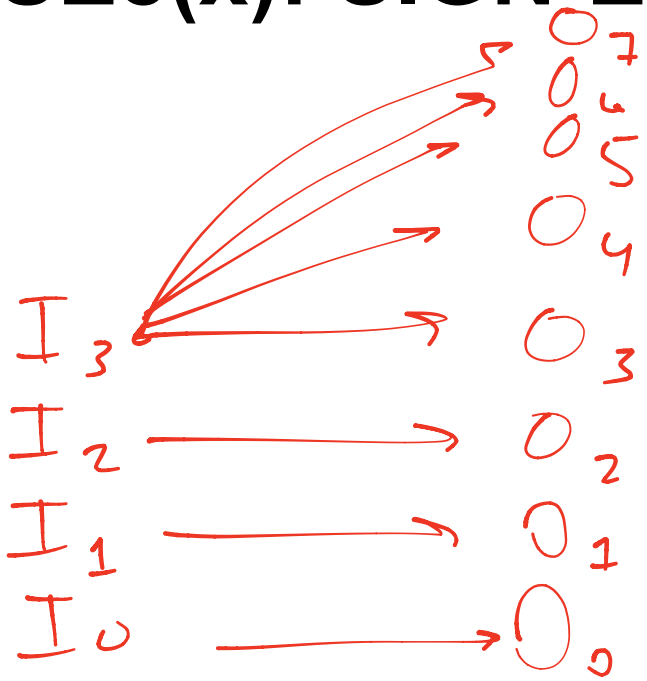

- is read only
- Iout is set to the value indexed by the address

# DATA MEMORY



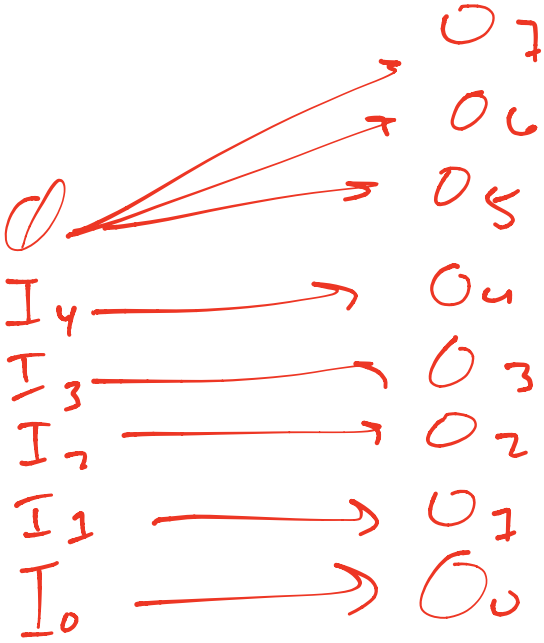

- can read or write
  - but only one in a given clock cycle
- on falling clock edge:
  - if MEMWrite==1: value on Din is stored at addr
  - if MEMRead==1: value at addr is output on Dout

# SE8(x): SIGN-EXTEND TO 8 BITS



- assuming 4-bit input
- Recall: want:
  - SE8(0100) -> 00000100
  - SE8(1100) -> 11111100
- In bits i3,i2,i1,i0;  out bits o7…o0

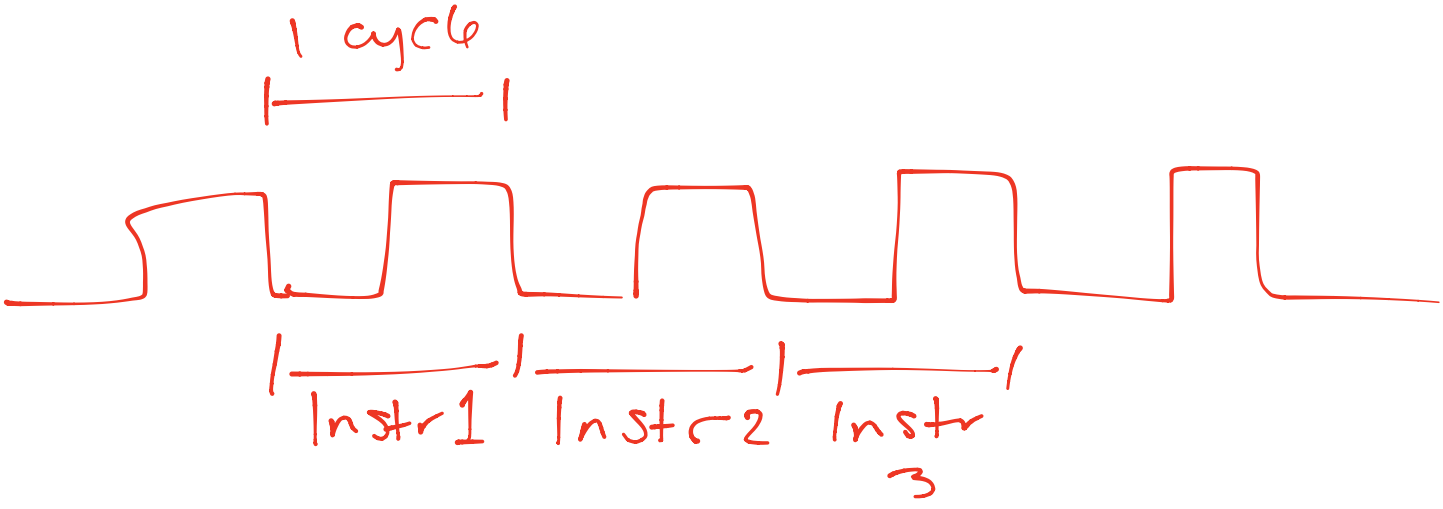
# ZE8(x): ZERO EXTEND TO 8 bits




- assuming 5-bit input
- Recall: want
  - ZE8(00100) -> 00000100
  - ZE8(11100) -> 00011100
- In bits i4,i3,i2,i1,i0;  out bits o7…o0

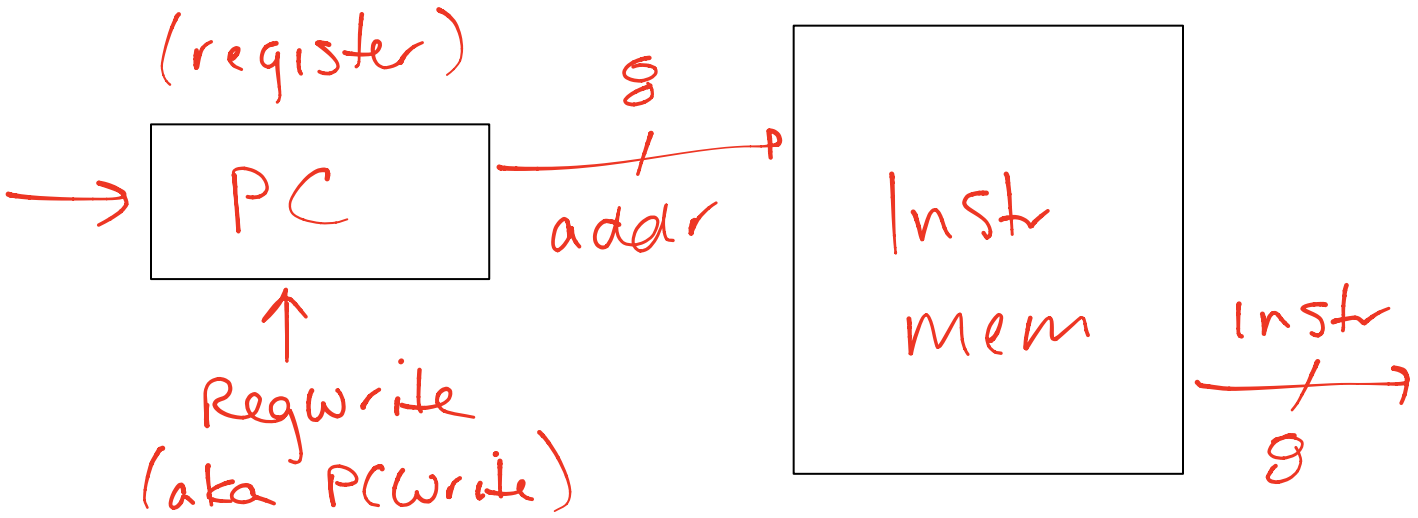
# CPU: Single Cycle Implementation

## SINGLE CYCLE DATAPATH



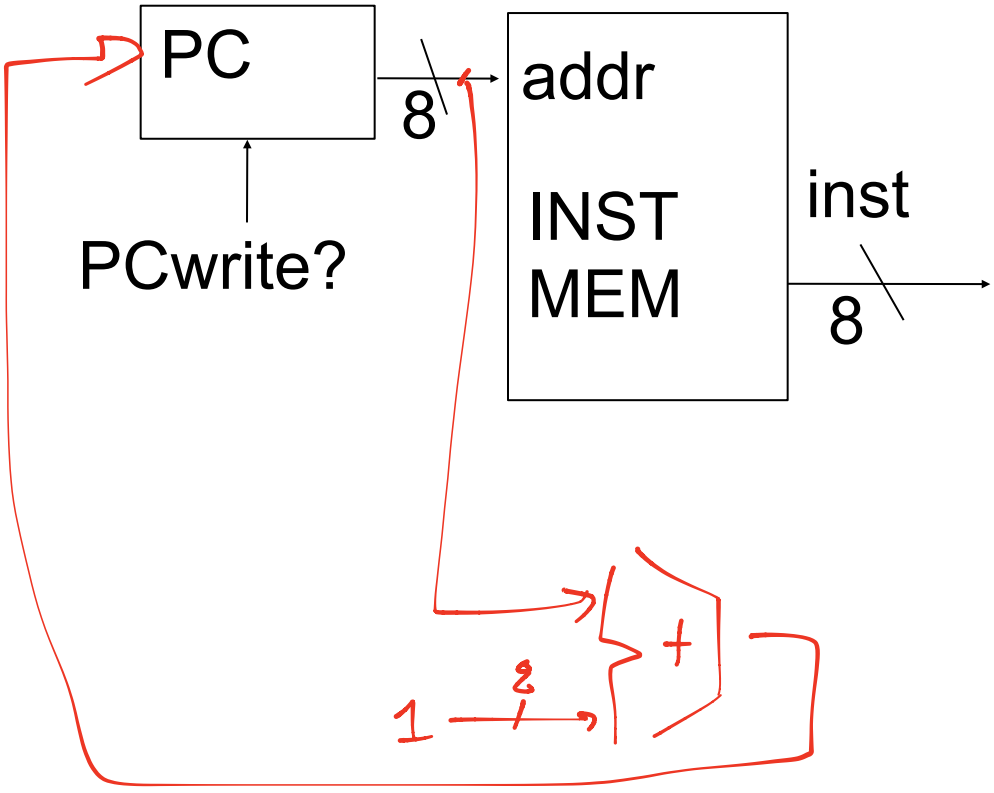

- each instruction executes entirely
  - in one cycle of the cpu clock
- registers are triggered by the falling edge
  - new values begin propagating through datapath
  - some values may be temporarily incorrect
- the clock period is large enough to ensure:
  - that all values correct before next falling edge

# FETCH

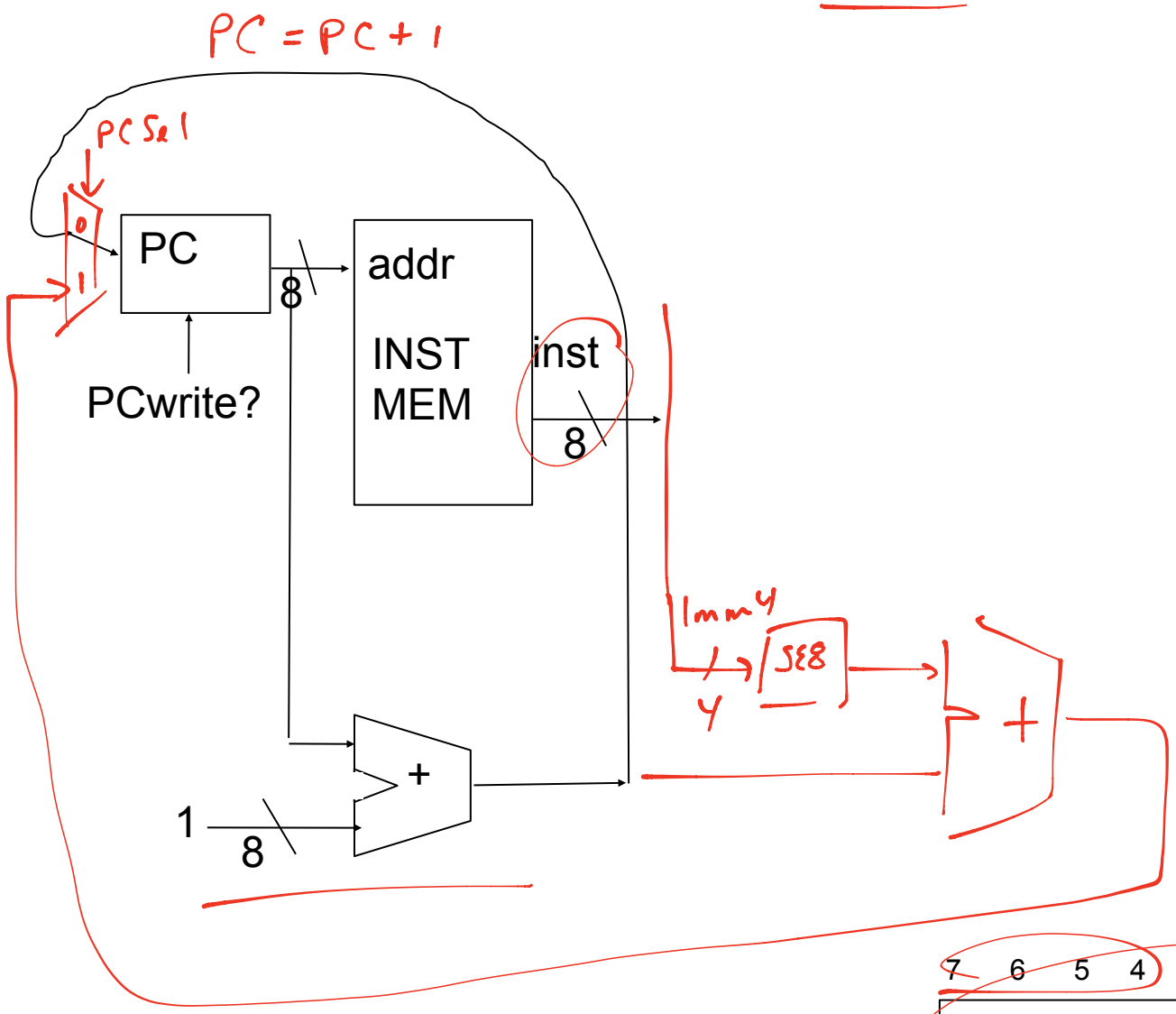


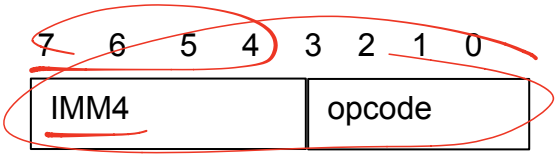

- needed by every instruction
  - i.e., every instruction must be fetched

**PC = PC + 1**



# BRANCHES: BZ IMM4

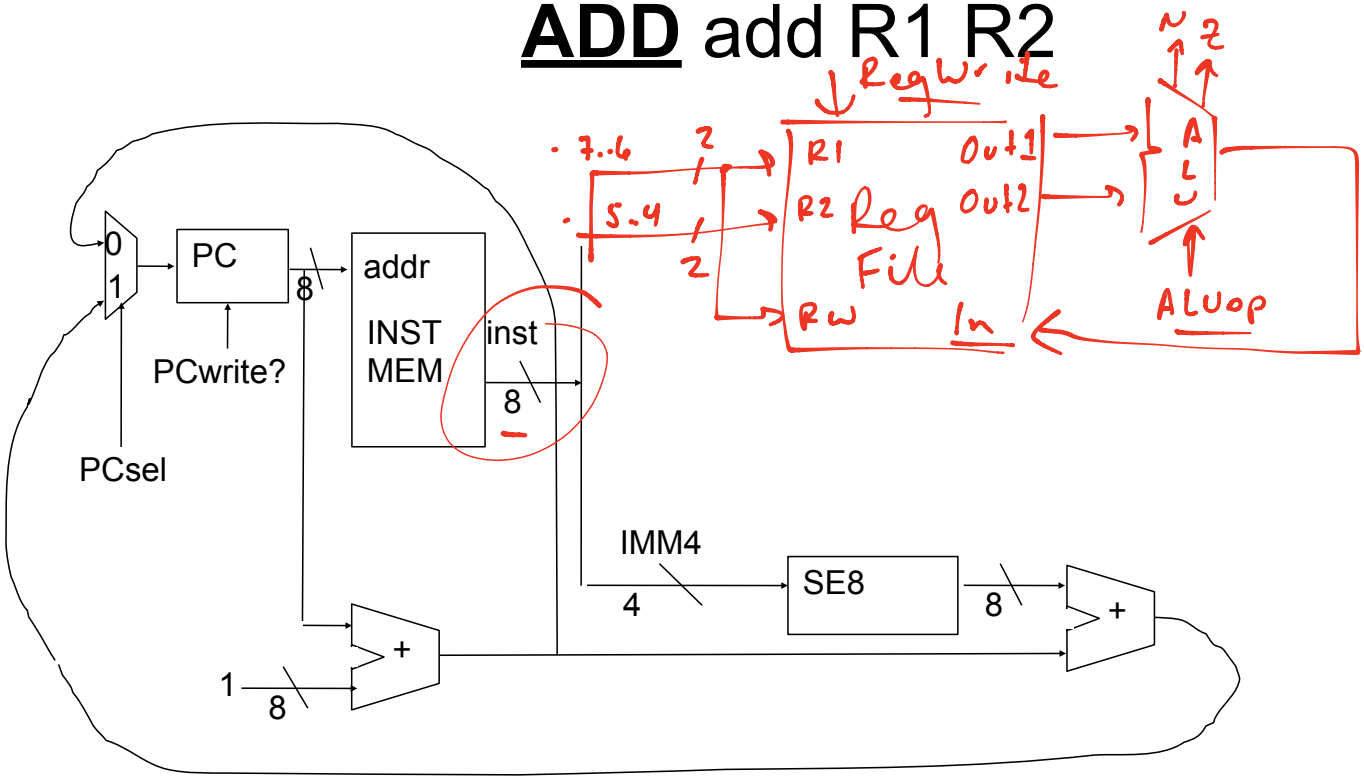


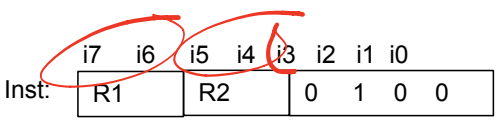

- (if branch is taken does: PC = PC + IMM4 + 1)

same for all branches

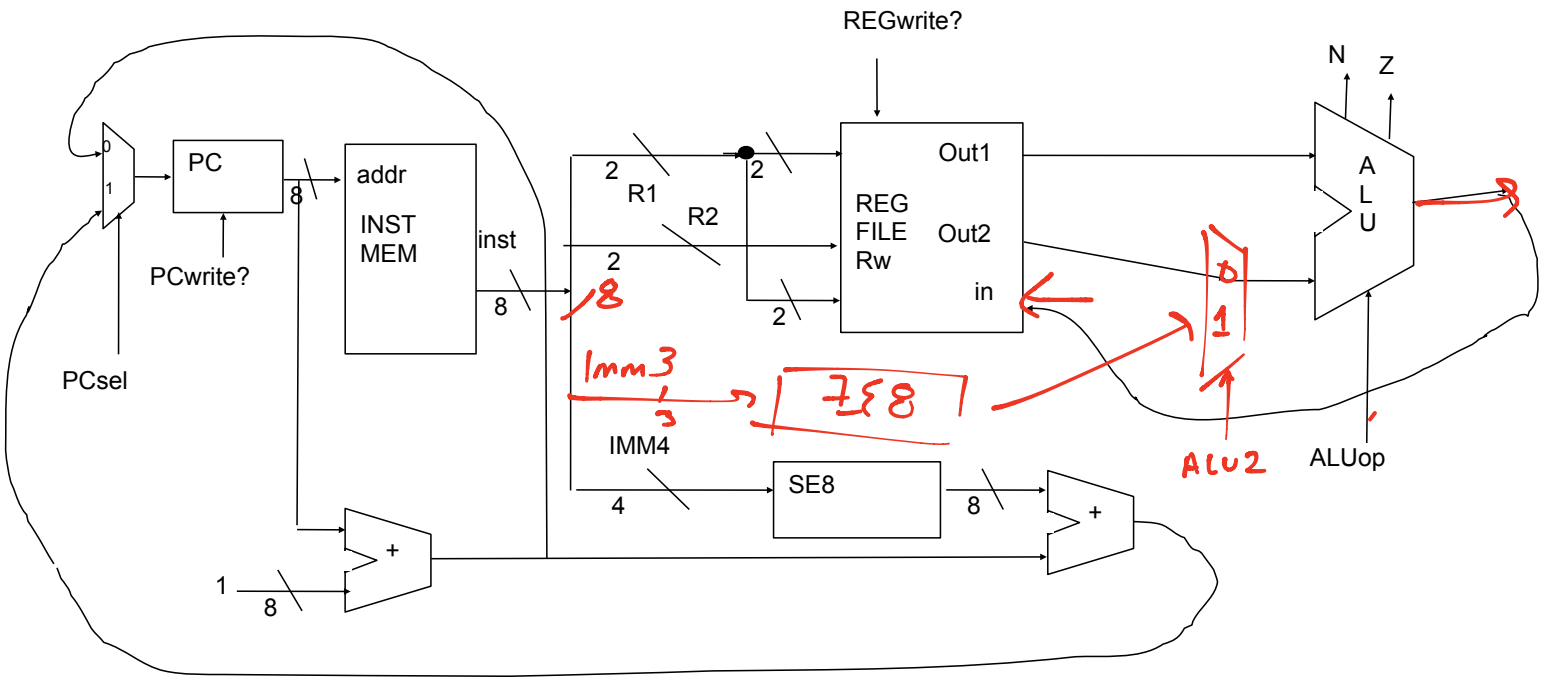
# **ADD** add R1 R2




- does r1 = r1 + r2
- same datapath for sub and nand

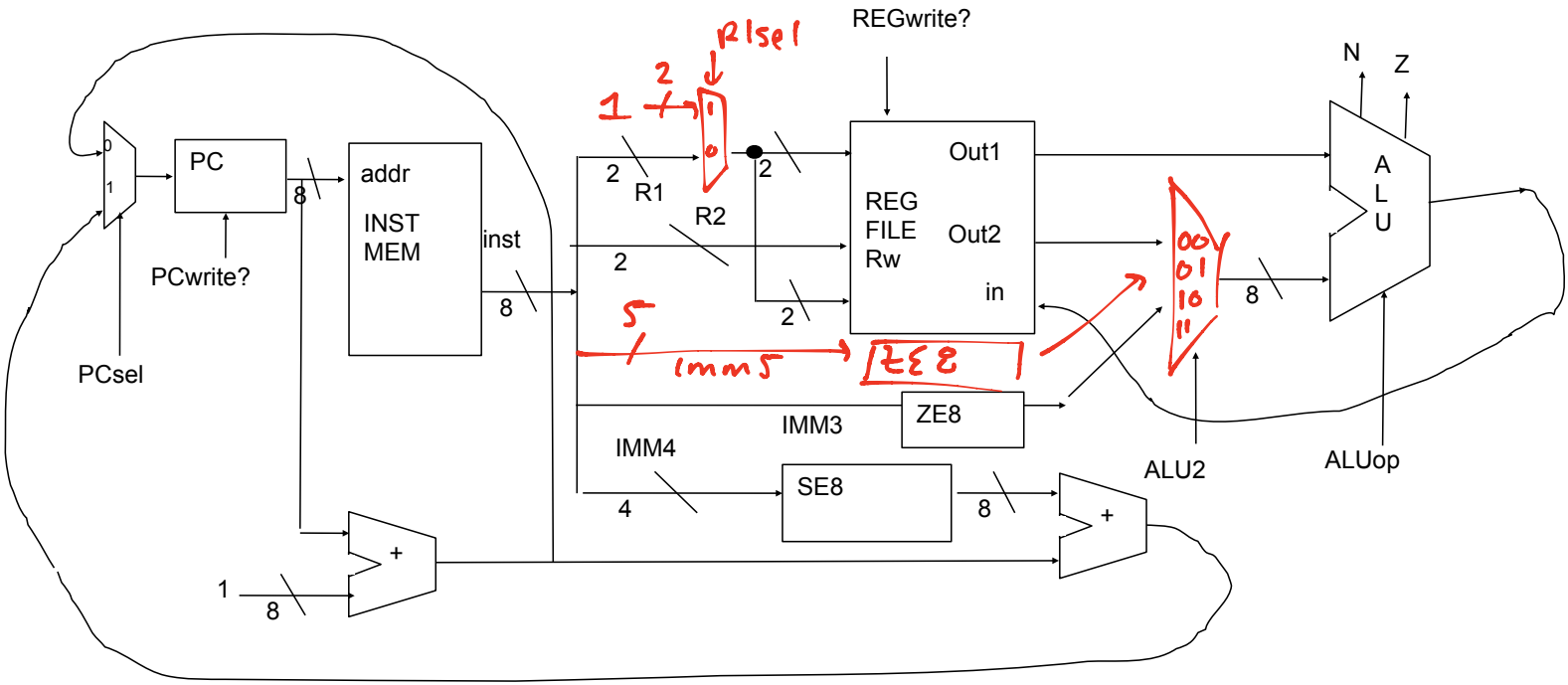


# **SHIFT**: SHIFT R1 IMM3

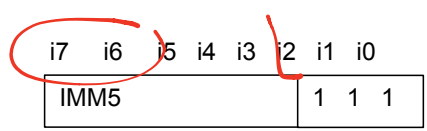



| i7 | i6 | i5 | i4 | i3 | i2 | i1 | i0 |
|---|---|---|---|---|---|---|---|
| R1 | | IMM3 | | | | 0 | 1 1 |

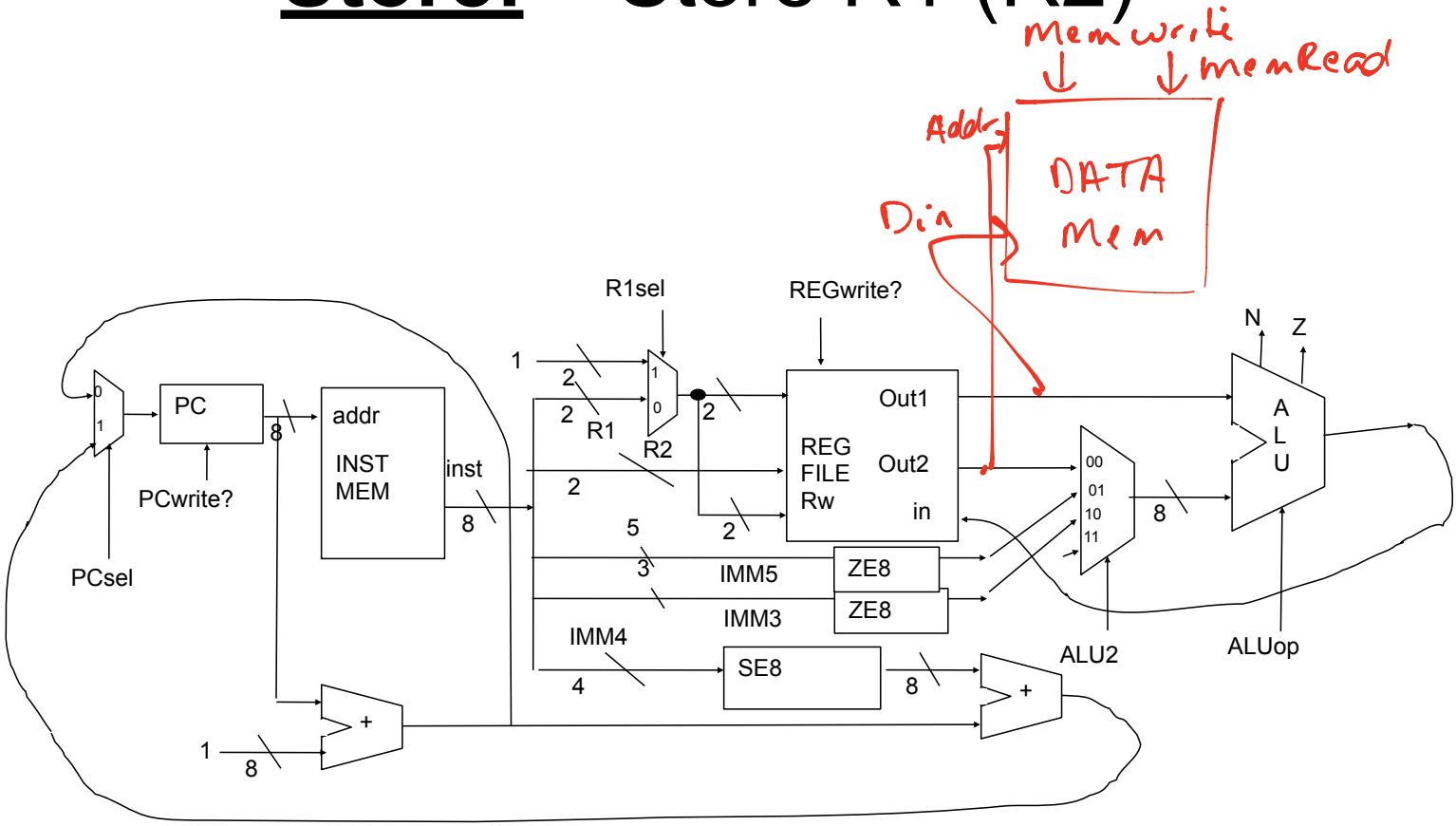
# **ORI**: ORI IMM5



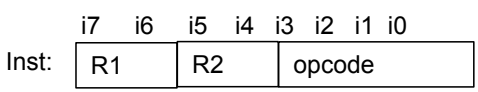

- does: k1 <- k1 bitwise-or IMM5



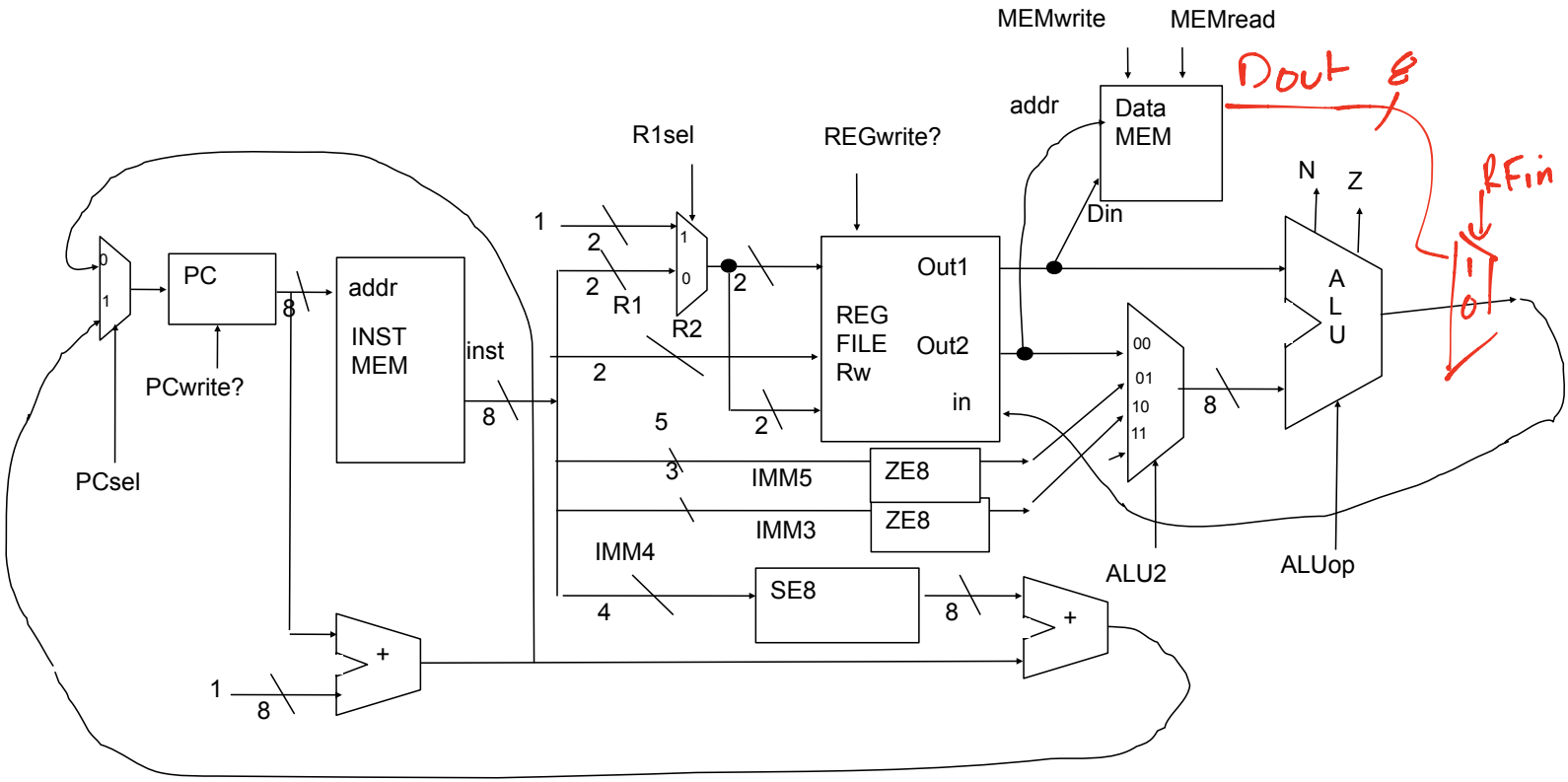
# **Store:** Store R1 (R2)



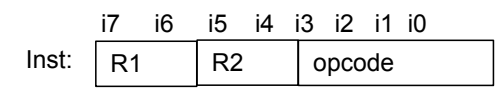

- does: mem[r2] = r1



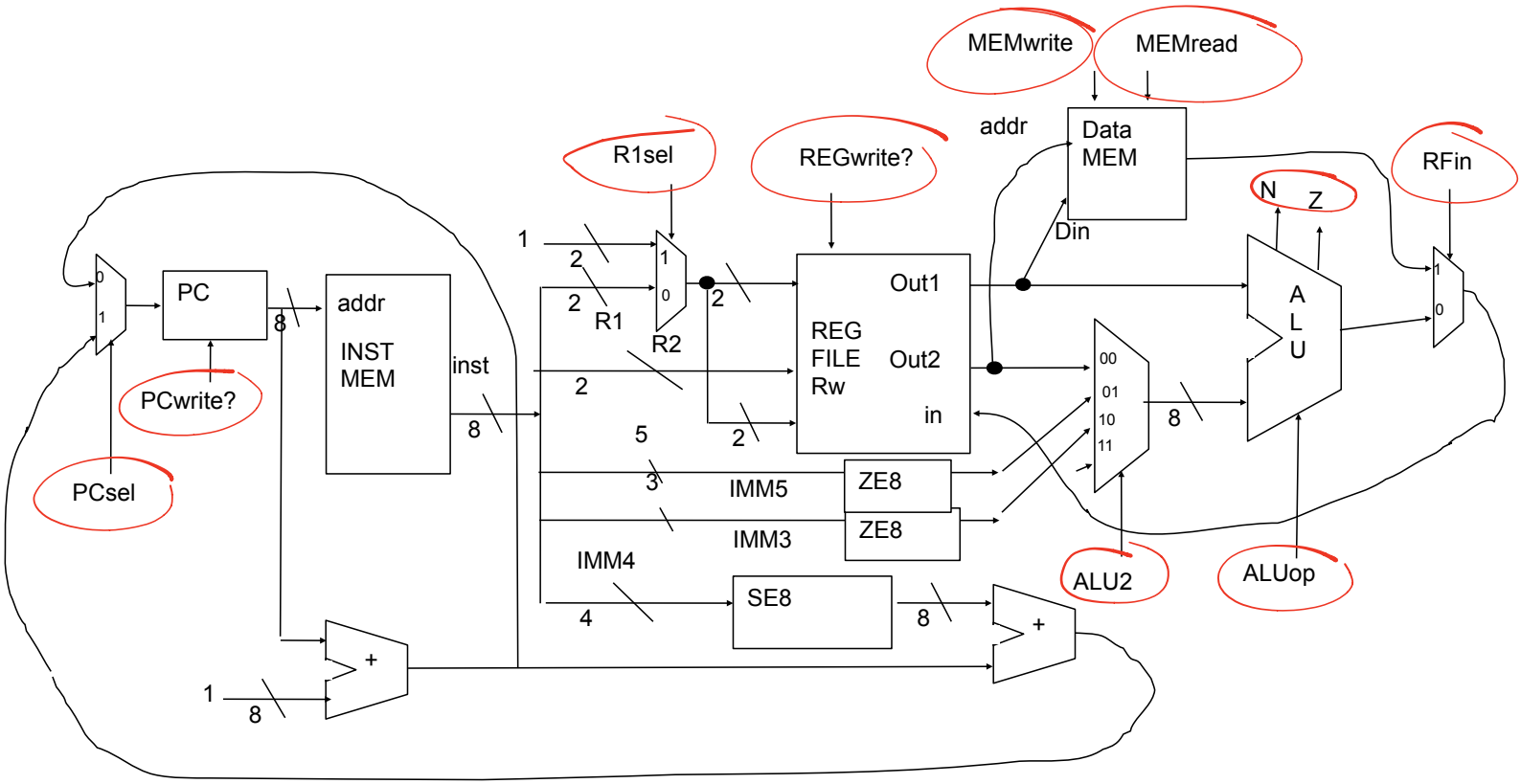
# **Load**: Load R1 (R2)




- does: r1 = mem[r2]



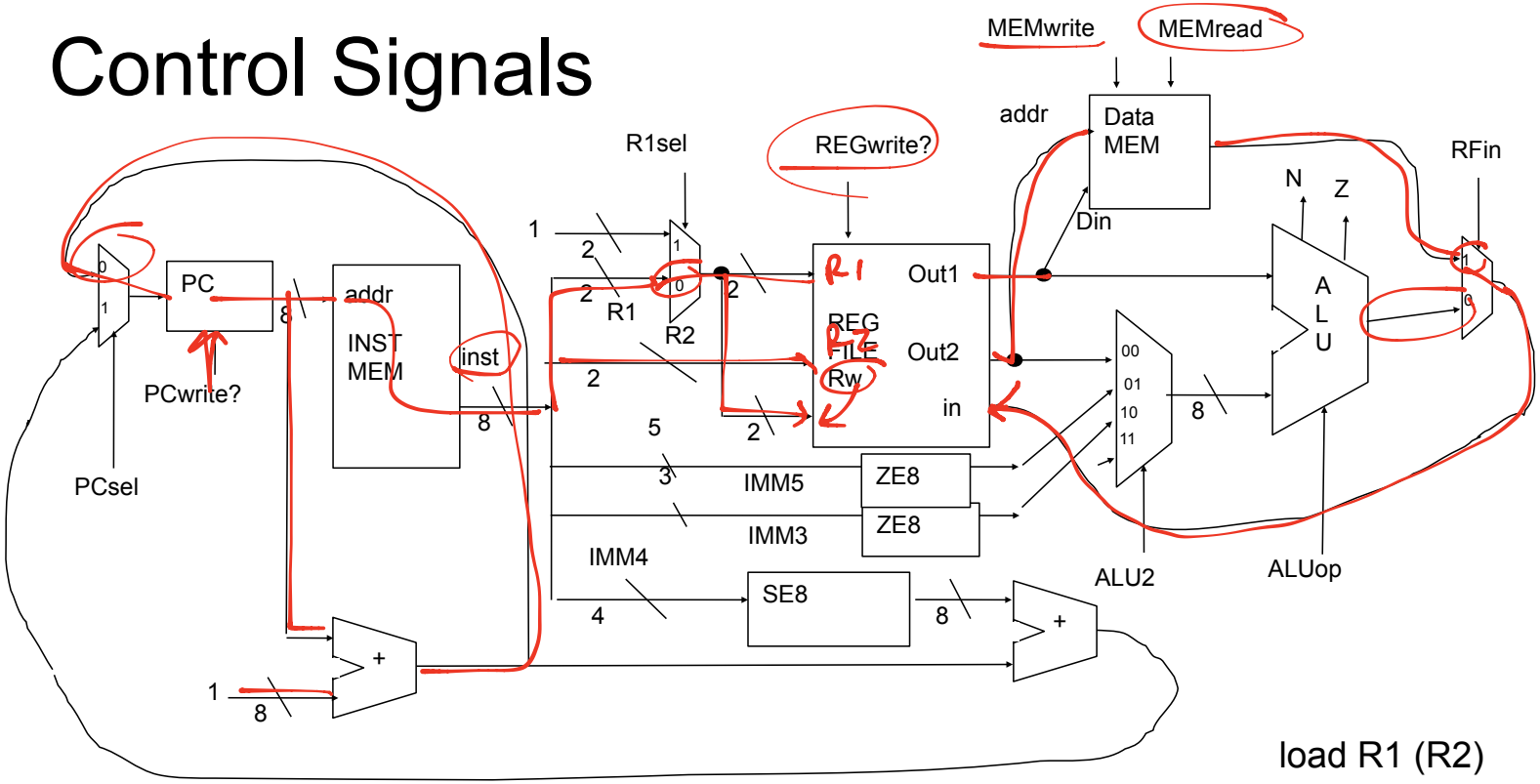
# Final Datapath!



# DESIGNING THE CONTROL UNIT

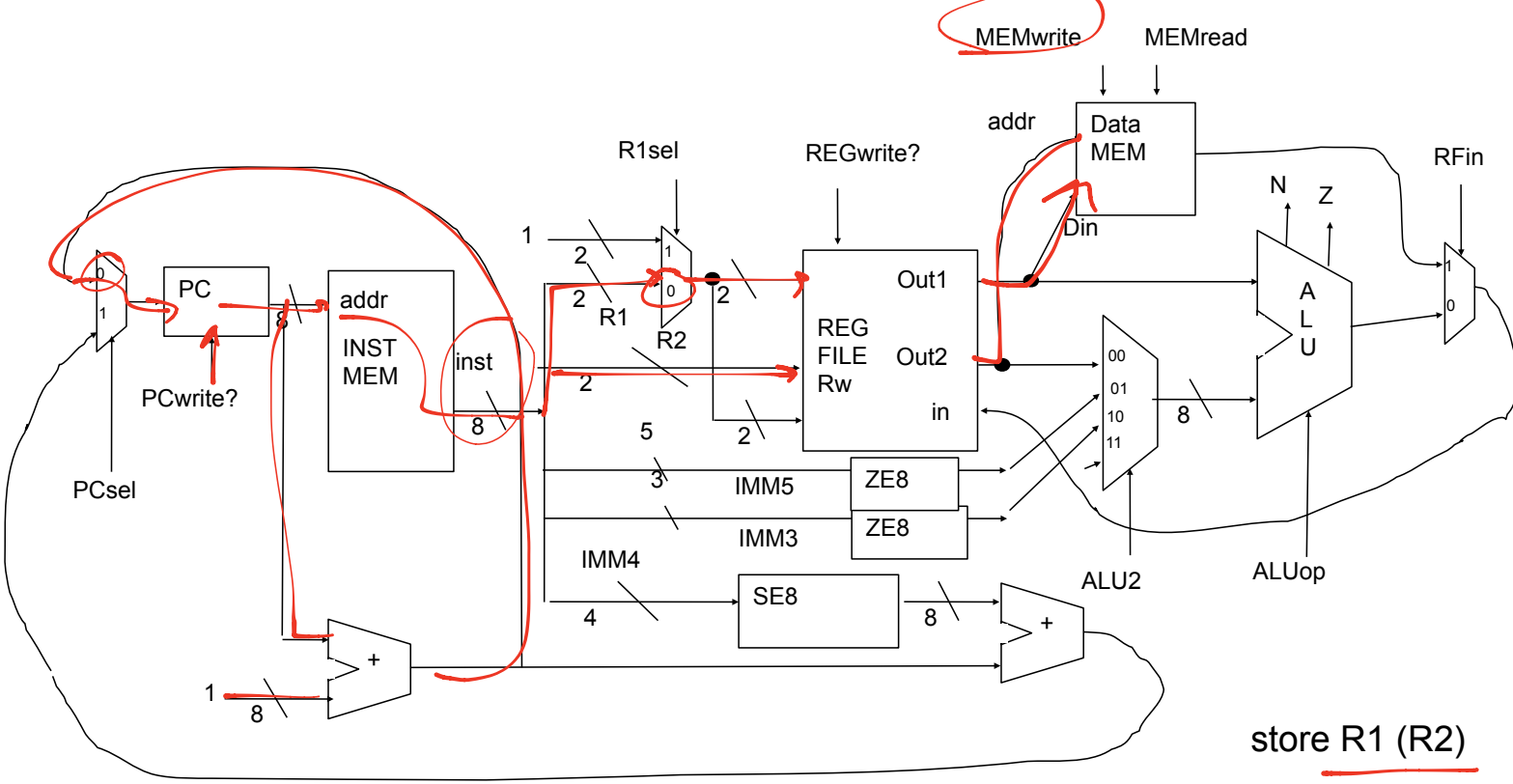

- CONTROL SIGNALS TO GENERATE:
  - PCsel, PCwrite, REGwrite, MEMread, MEMwrite, R1sel, ALUop, ALU2, RFin

# Control Signals

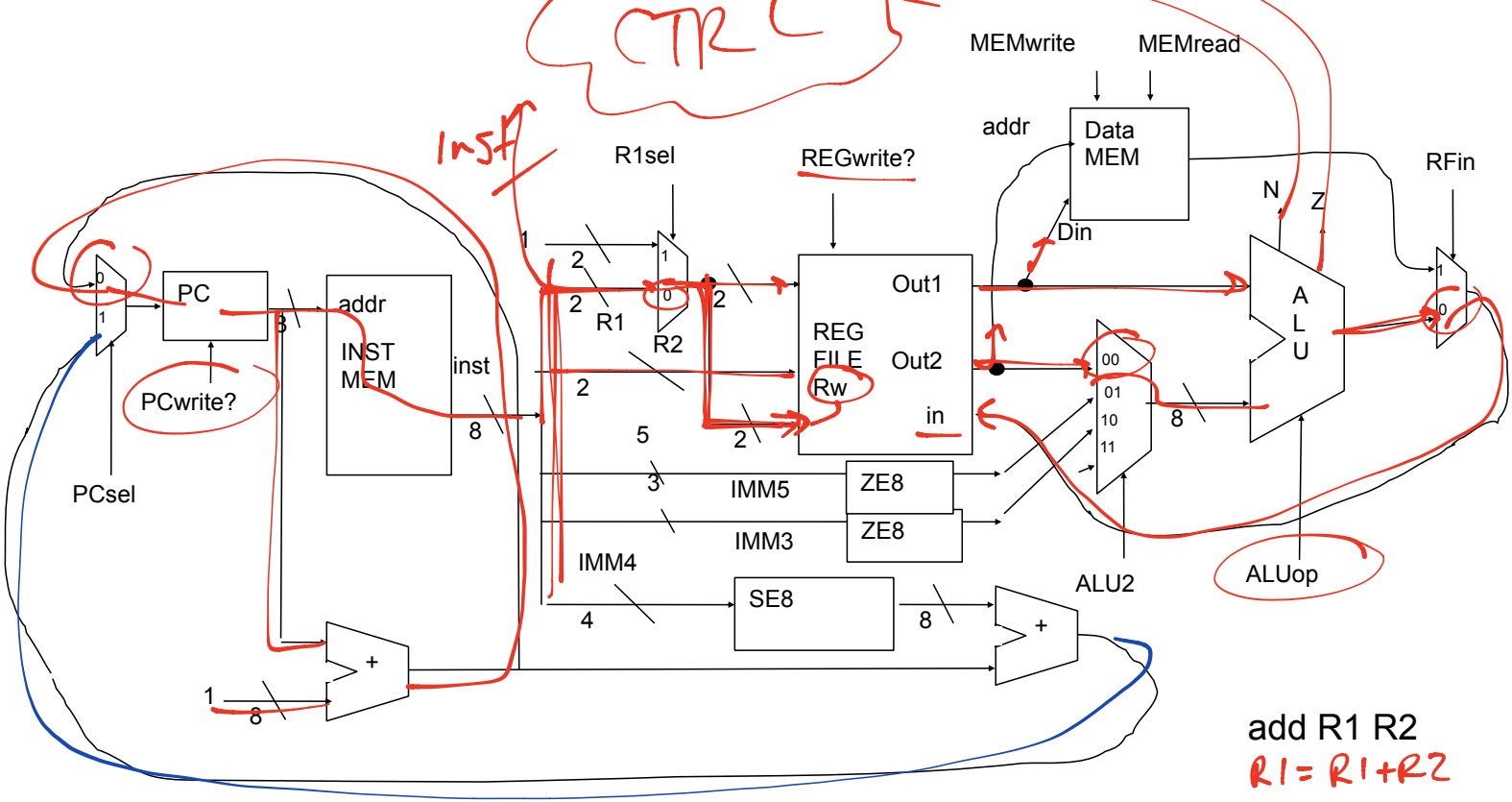



load R1 (R2)

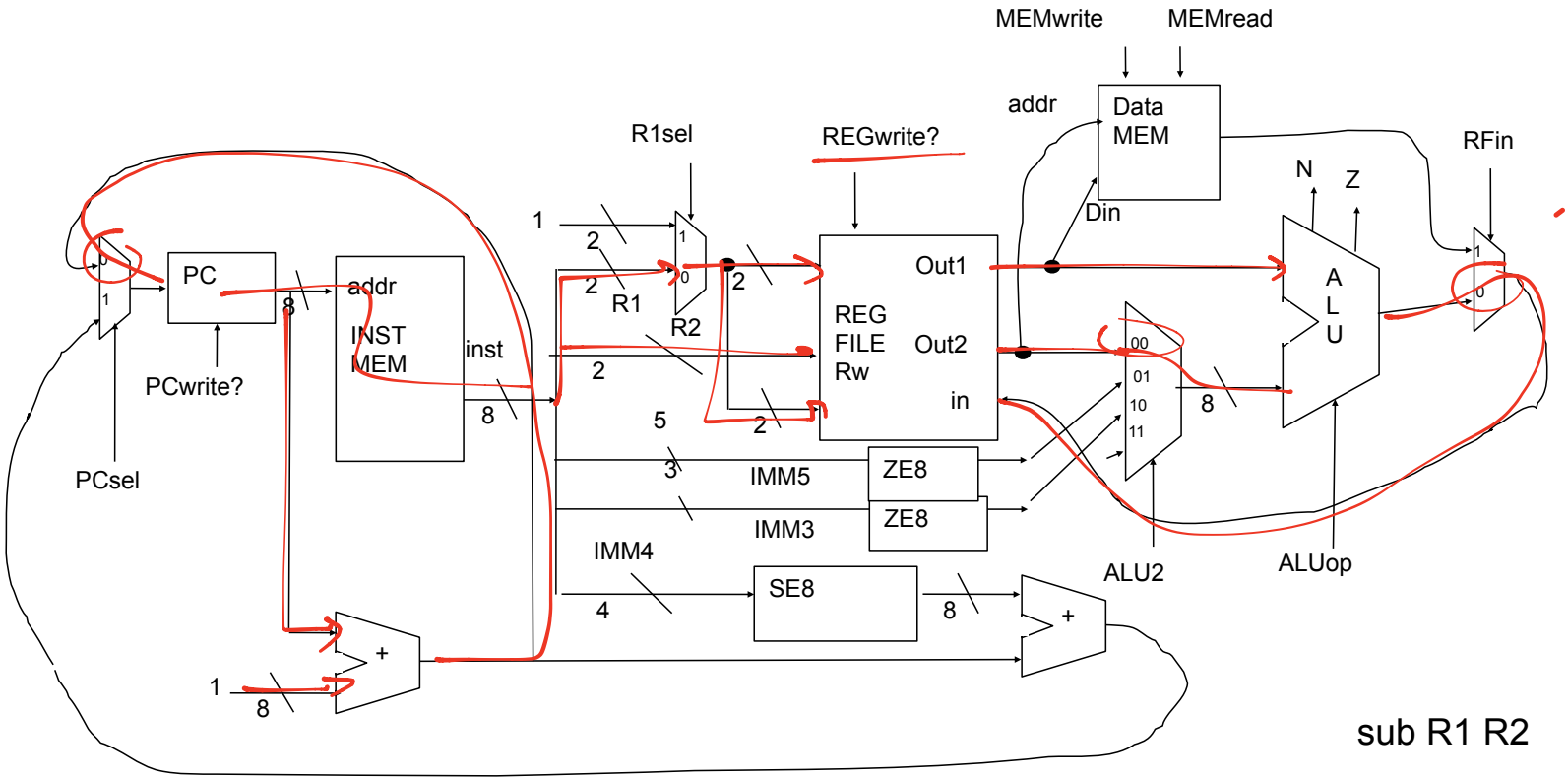
| | INPUTS | | | OUTPUTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **INST** | **Inst bits 3-0** | **N** | **Z** | **PC Sel** | **PC Write** | **Reg Write** | **Mem Read** | **R1Sel** | **Mem Write** | **ALU2** | **RFin** | **ALUop** |
| **LOAD** | **0000** | **X** | **X** | 0 | 1 | 1 | 1 | 0 | 0 | XX | 1 | XXX |



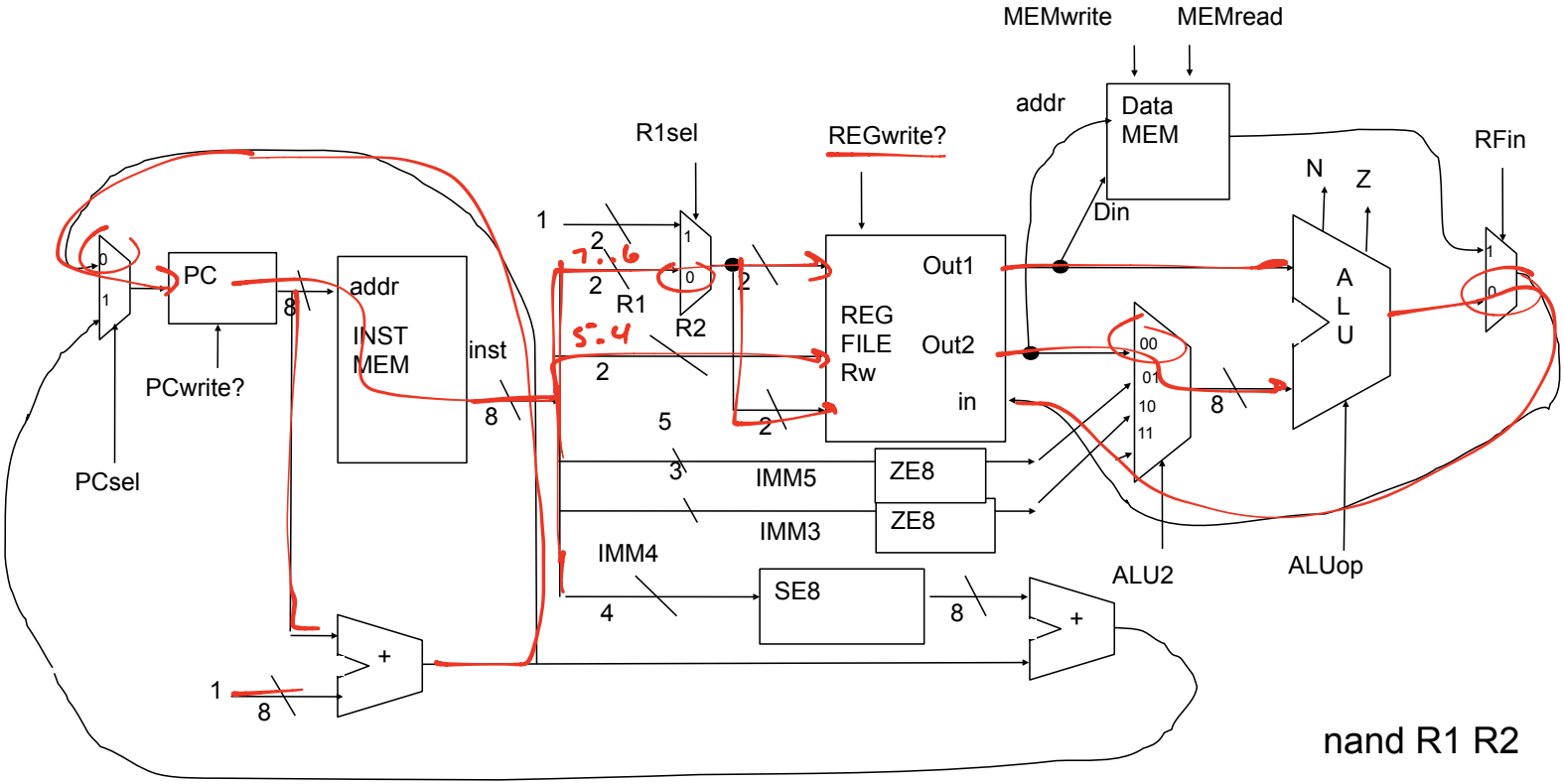

| | INPUTS | | | OUTPUTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **INST** | **Inst bits 3-0** | **N** | **Z** | **PC Sel** | **PC Write** | **Reg Write** | **Mem Read** | **R1Sel** | **Mem Write** | **ALU2** | **RFin** | **ALUop** |
| **STORE** | **0010** | **X** | **X** | 0 | 1 | 0 | 0 | 0 | 1 | X X | ✓ | XXX |



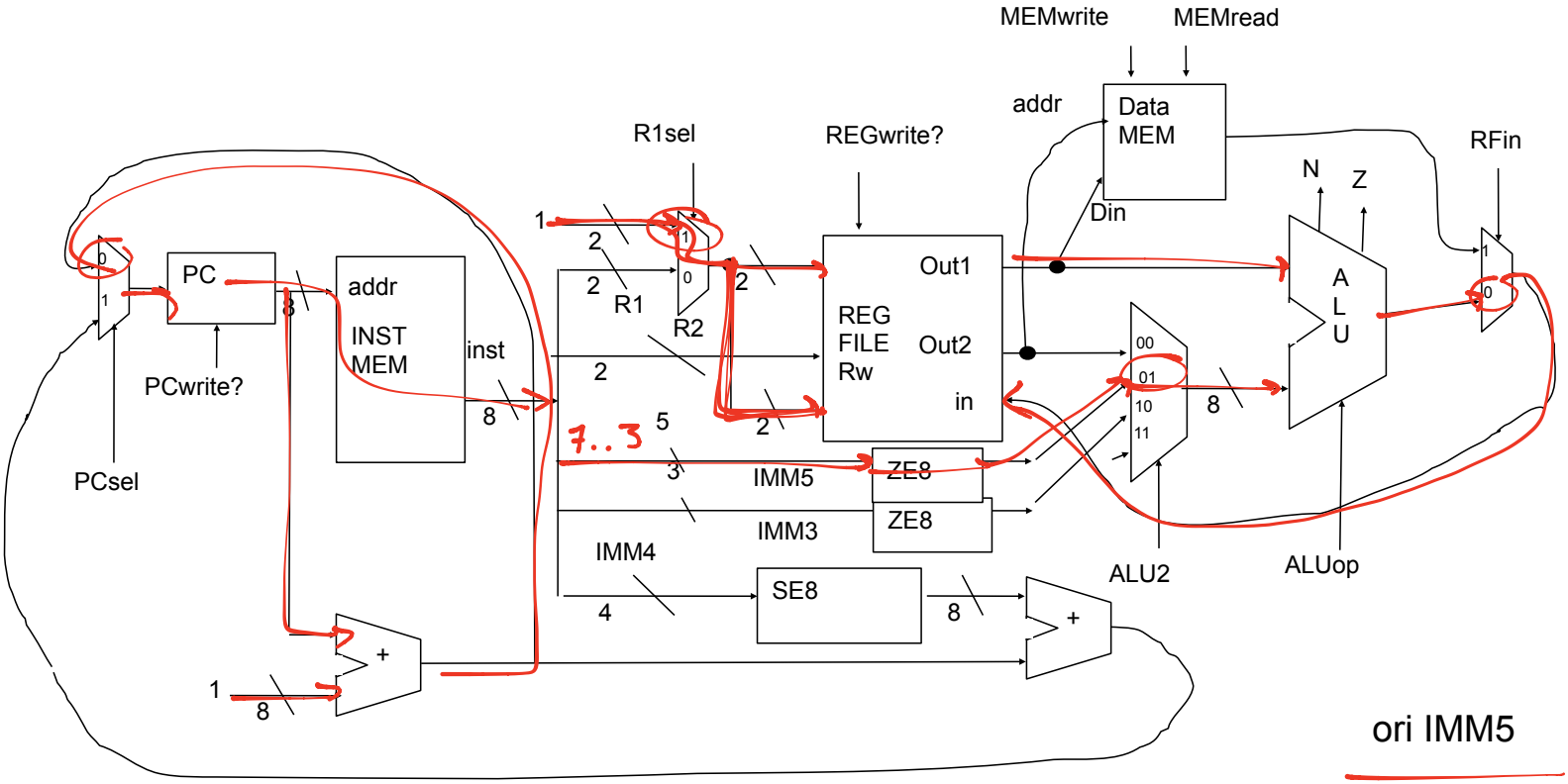

| | INPUTS | | | OUTPUTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **INST** | **Inst bits 3-0** | **N** | **Z** | **PC Sel** | **PC Write** | **Reg Write** | **Mem Read** | **R1Sel** | **Mem Write** | **ALU2** | **RFin** | **ALUop** |
| **ADD** | **0100** | **X** | **X** | 0 | 1 | 1 | 0 | 0 | 0 | 00 | 0 | 000 |



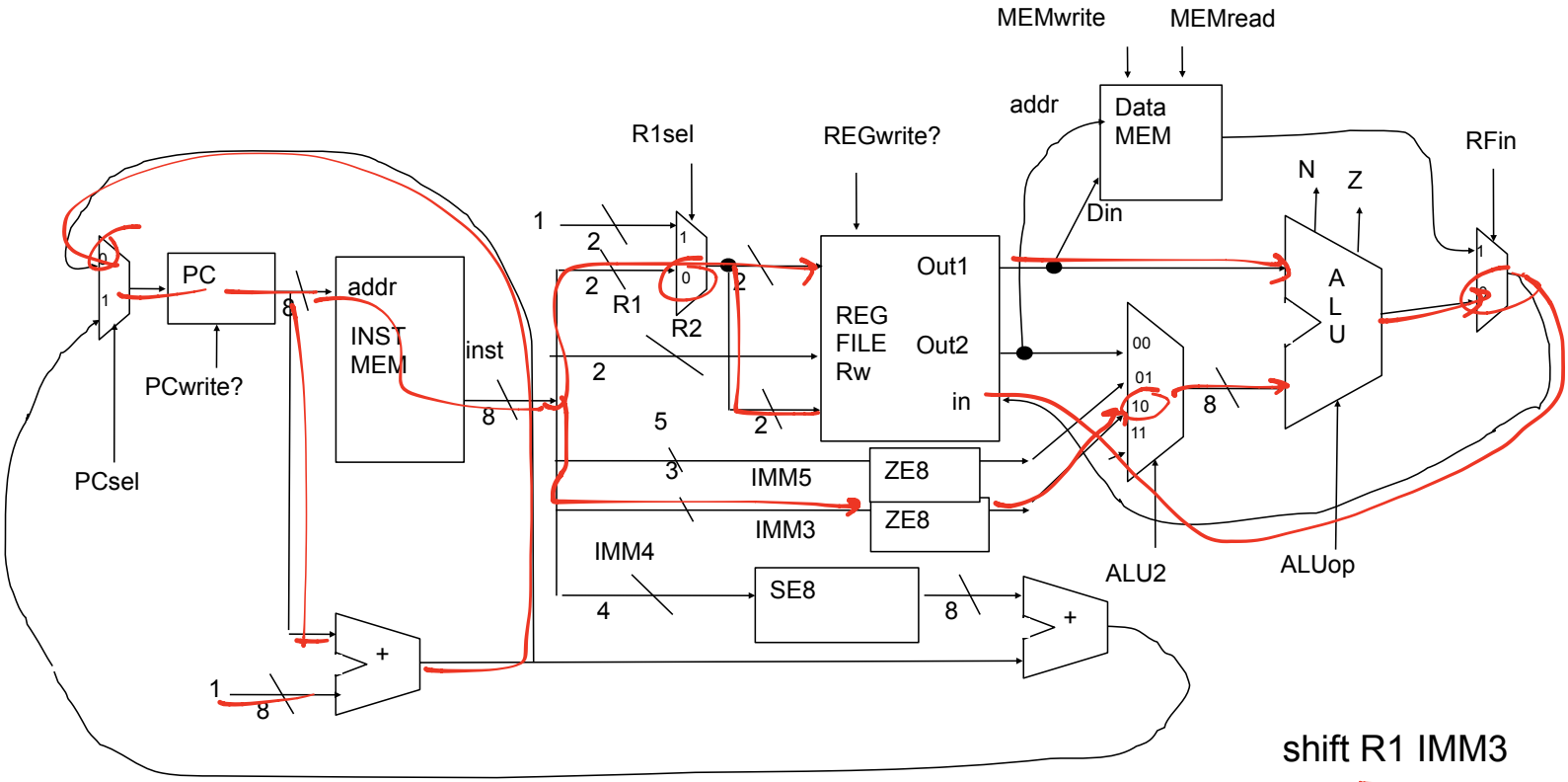

| | INPUTS | | | OUTPUTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **INST** | **Inst bits 3-0** | **N** | **Z** | **PC Sel** | **PC Write** | **Reg Write** | **Mem Read** | **R1Sel** | **Mem Write** | **ALU2** | **RFin** | **ALUop** |
| **SUB** | **0110** | **X** | **X** | 0 | 1 | 1 | 0 | 0 | 0 | 00 | 0 | 001 |



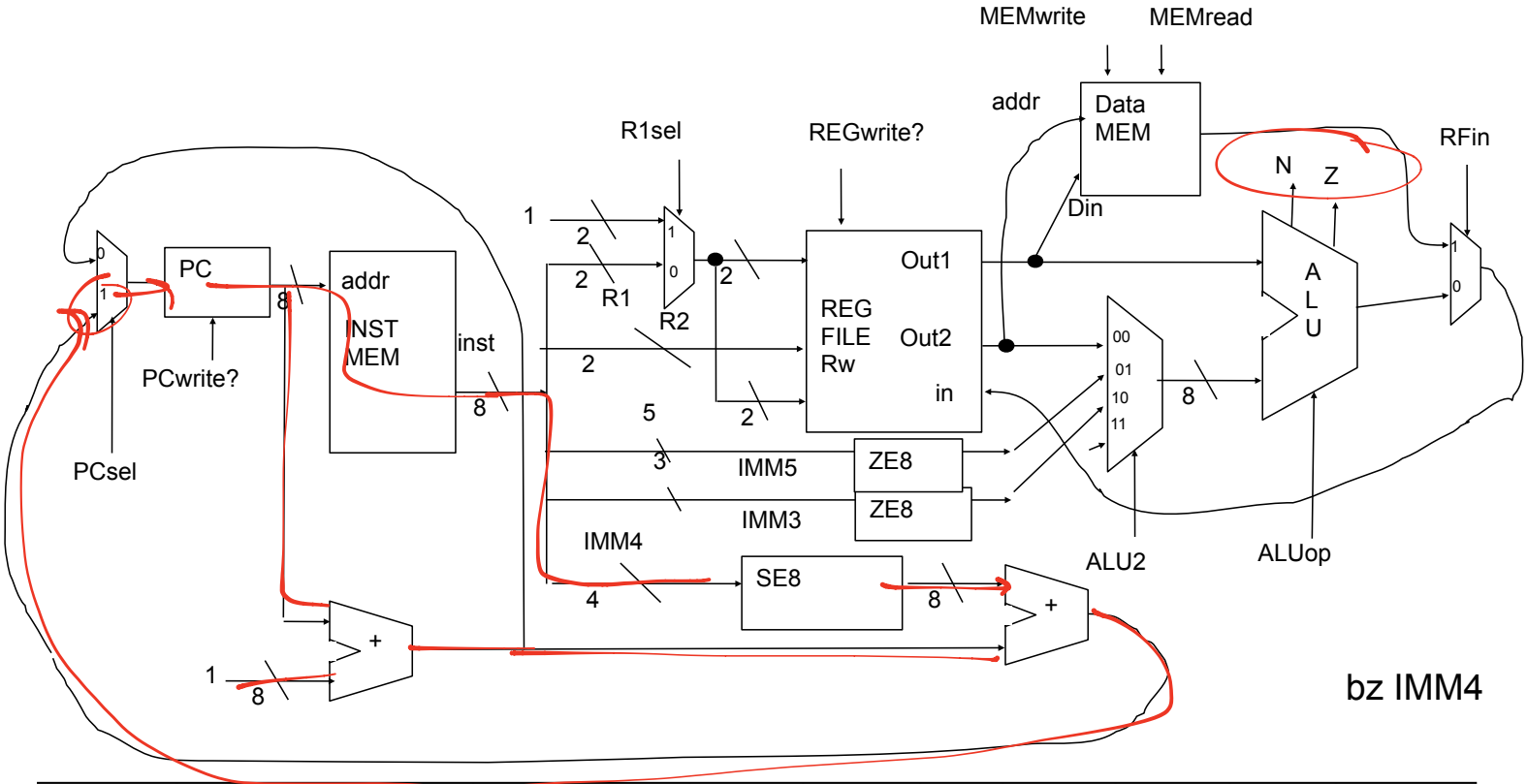

| | INPUTS | | | OUTPUTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **INST** | **Inst bits 3-0** | **N** | **Z** | **PC Sel** | **PC Write** | **Reg Write** | **Mem Read** | **R1Sel** | **Mem Write** | **ALU2** | **RFin** | **ALUop** |
| **NAND** | **1000** | **X** | **X** | 0 | 1 | 1 | 0 | 0 | 0 | 00 | 0 | 011 |



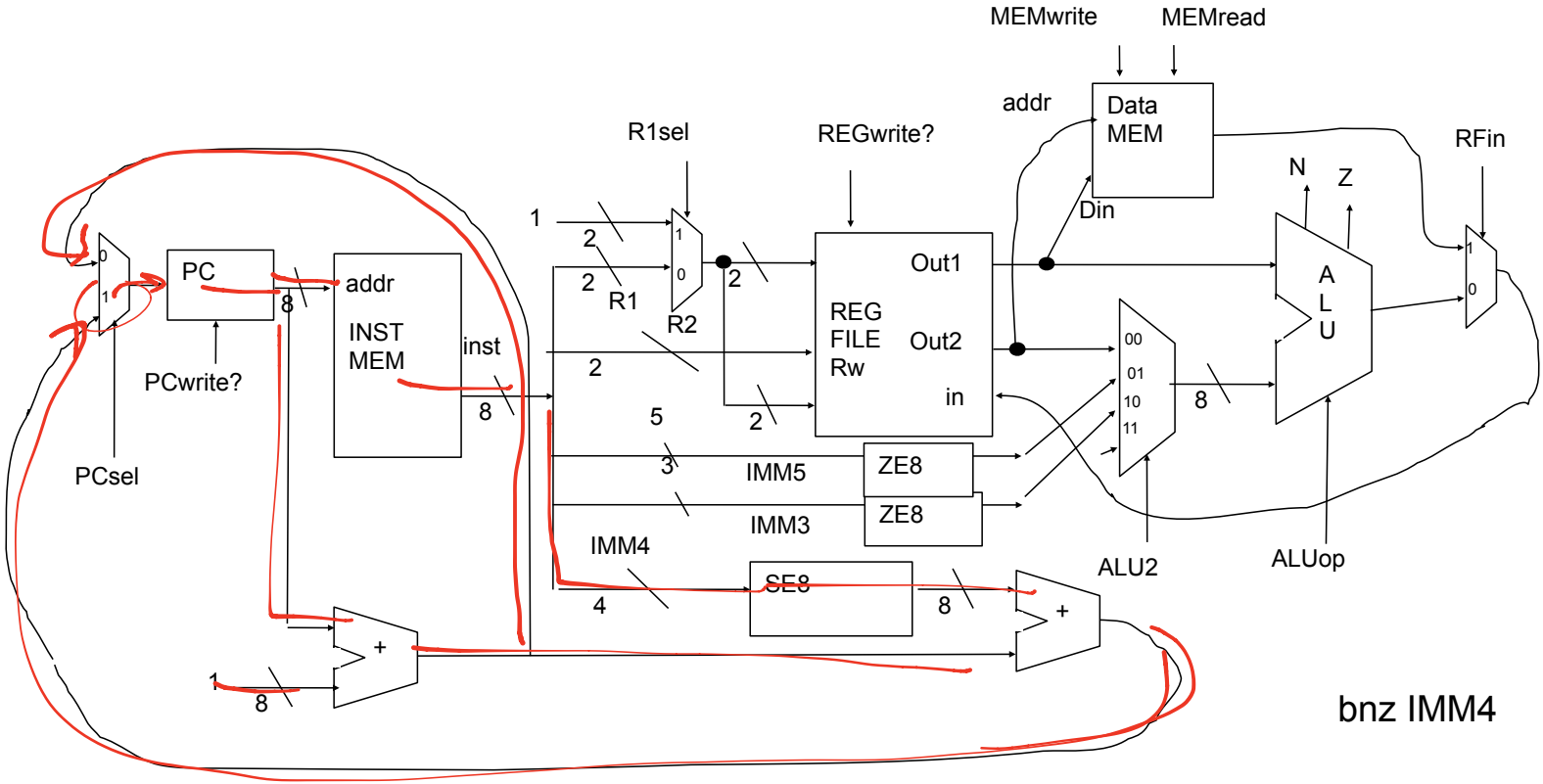

| | INPUTS | | | OUTPUTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **INST** | **Inst bits 3-0** | **N** | **Z** | **PC Sel** | **PC Write** | **Reg Write** | **Mem Read** | **R1Sel** | **Mem Write** | **ALU2** | **RFin** | **ALUop** |
| **ORI** | **X111** | **X** | **X** | 0 | 1 | 1 | 0 | 1 | 0 | 01 | 0 | 010 |



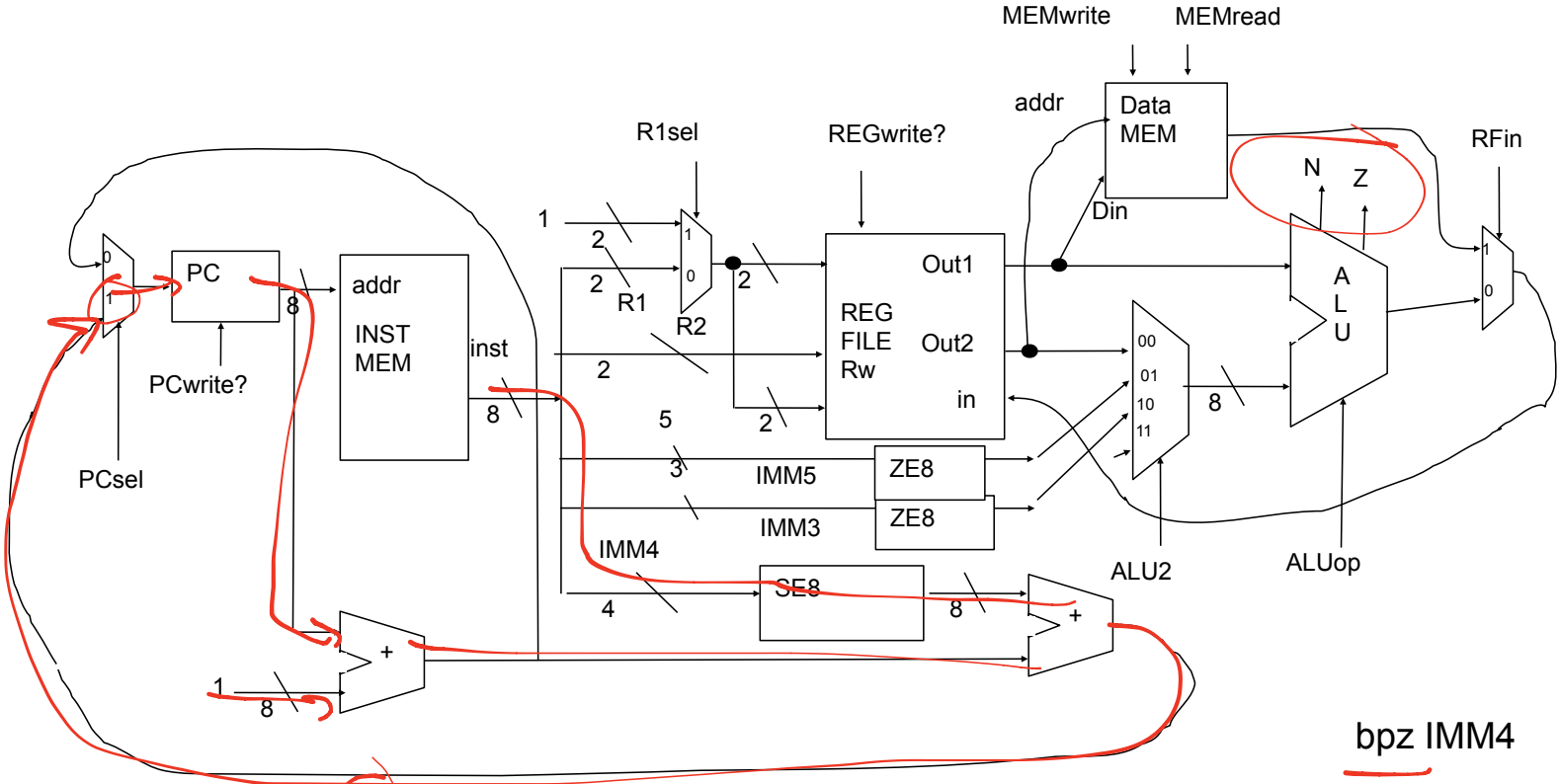

| | INPUTS | | | OUTPUTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **INST** | **Inst bits 3-0** | **N** | **Z** | **PC Sel** | **PC Write** | **Reg Write** | **Mem Read** | **R1Sel** | **Mem Write** | **ALU2** | **RFin** | **ALUop** |
| **SHIFT** | **X011** | **X** | **X** | 0 | 1 | 1 | 0 | 0 | 0 | 10 | 0 | 100 |




| INST | Inst bits 3-0 | N | Z | PC Sel | PC Write | Reg Write | Mem Read | R1 Sel | Mem Write | ALU2 | RFin | ALUop |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BZ | 0101 | X | 0 | 0 | 1 | 0 | 0 | X | 0 | XX | 1 | XXX |
| | 0101 | X | 1 | 1 | 1 | 0 | 0 | X | 0 | XX | X | XXX |




| INST | Inst bits 3-0 | N | Z | PC Sel | PC Write | Reg Write | Mem Read | R1 Sel | Mem Write | ALU2 | RFin | ALUop |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BNZ | 1001 | X | 0 | 1 | 1 | 0 | 0 | X | 0 | XX | X | XXX |
| | 1001 | X | 1 | 0 | 1 | 0 | 0 | X | 0 | XX | X | XXX |




| INST | Inst bits 3-0 | N | Z | PC Sel | PC Write | Reg Write | Mem Read | R1 Sel | Mem Write | ALU2 | RFin | ALUop |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BPZ | 1101 | 0 | X | 1 | 1 | 0 | 0 | X | 0 | x | x | xxx |
| | 1101 | 1 | X | 0 | 1 | 0 | 0 | X | 0 | xy | x | xxy |

# All Control Signals

| | INPUTS | | | OUTPUTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INST | Inst bits 3-0 | N | Z | PC Sel | PC Write | Reg Write | Mem Read | R1 Sel | Mem Write | ALU2 | RFin | ALUop |
| LOAD | 0000 | X | X | 0 | 1 | 1 | 1 | X | 0 | X | 1 | XXX |
| STORE | 0010 | X | X | 0 | 1 | 0 | 0 | 0 | 1 | X | X | XXX |
| ADD | 0100 | X | X | 0 | 1 | 1 | 0 | 0 | 0 | 00 | 0 | 000 |
| SUB | 0110 | X | X | 0 | 1 | 1 | 0 | 0 | 0 | 00 | 0 | 001 |
| NAND | 1000 | X | X | 0 | 1 | 1 | 0 | 0 | 0 | 00 | 0 | 011 |
| ORI | X111 | X | X | 0 | 1 | 1 | 0 | 1 | 0 | 01 | 0 | 010 |
| SHIFT | X011 | X | X | 0 | 1 | 1 | 0 | 0 | 0 | 10 | 0 | 100 |
| BZ | 0101 | X | 0 | 0 | 1 | 0 | 0 | X | 0 | X | X | XXX |
| | 0101 | X | 1 | 1 | 1 | 0 | 0 | X | 0 | X | X | XXX |
| BNZ | 1001 | X | 0 | 1 | 1 | 0 | 0 | X | 0 | X | X | XXX |
| | 1001 | X | 1 | 0 | 1 | 0 | 0 | X | 0 | X | X | XXX |
| BPZ | 1101 | 0 | X | 1 | 1 | 0 | 0 | X | 0 | X | X | XXX |
| | 1101 | 1 | X | 0 | 1 | 0 | 0 | X | 0 | X | X | XXX |

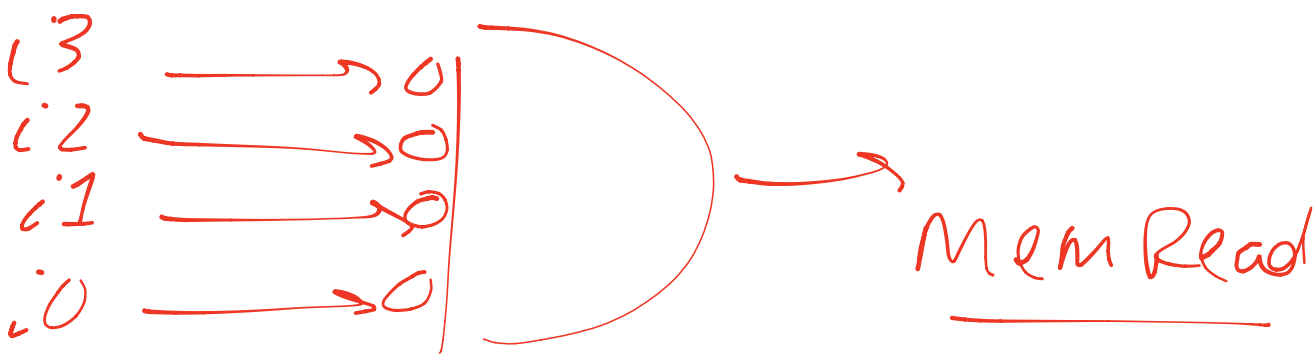
# Building Control Logic: MemRead

| | Load | Store | Add | Sub | Nand | Ori | Shift | Bz | | Bnz | | BPZ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| inst bits i3-i0 | 0000 | 0010 | 0100 | 0110 | 1000 | X111 | X011 | 0101 | 0101 | 1001 | 1001 | 1101 | 1101 |
| N | X | X | X | X | X | X | X | X | X | X | X | 0 | 1 |
| Z | X | X | X | X | X | X | X | 0 | 1 | 0 | 1 | X | X |
| Mem Read | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- MemRead = opcode (load)

  = !i3 & !i2 & !i1 & !i0

  Don't care about N & Z



# Building Control Logic: PCSel

| | Load | Store | Add | Sub | Nand | Ori | Shift | Bz | | Bnz | | BPZ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| inst bits i3-i0 | 0000 | 0010 | 0100 | 0110 | 1000 | X111 | X011 | 0101 | 0101 | 1001 | 1001 | 1101 | 1101 |
| N | X | X | X | X | X | X | X | X | X | X | X | 0 | 1 |
| Z | X | X | X | X | X | X | X | 0 | 1 | 0 | 1 | X | X |
| PCSel | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

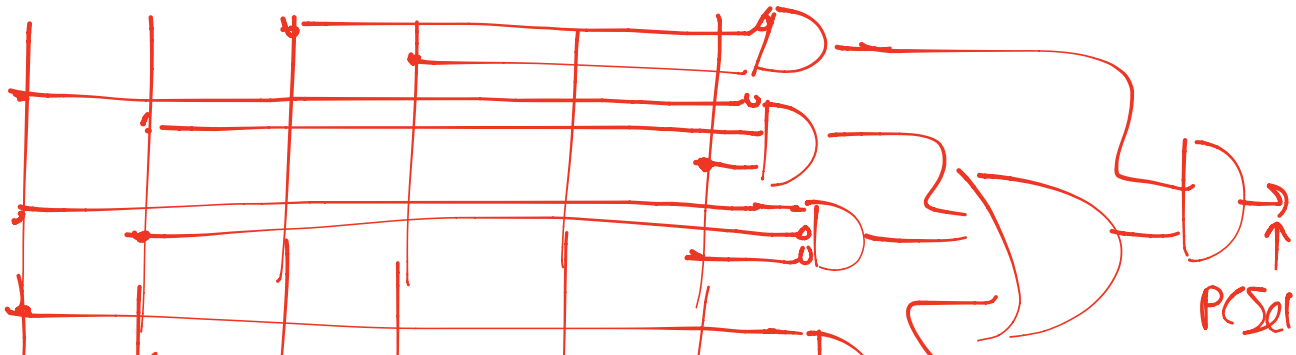
PCSel = (op=bz & z) | (op=bnz & !z) | (op=bpz & !n)

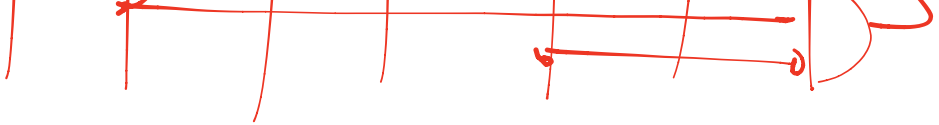
= (!i3 & i2 & !i1 & i0 & z) | (i3 & !i2 & !i1 & i0 & !z) |

(i3 & i2 & !i1 & i0 & !n)

= !i1 & i0 & (!i3 & i2 & z | i3 & !i2 & !z | i3 & i2 & !n)

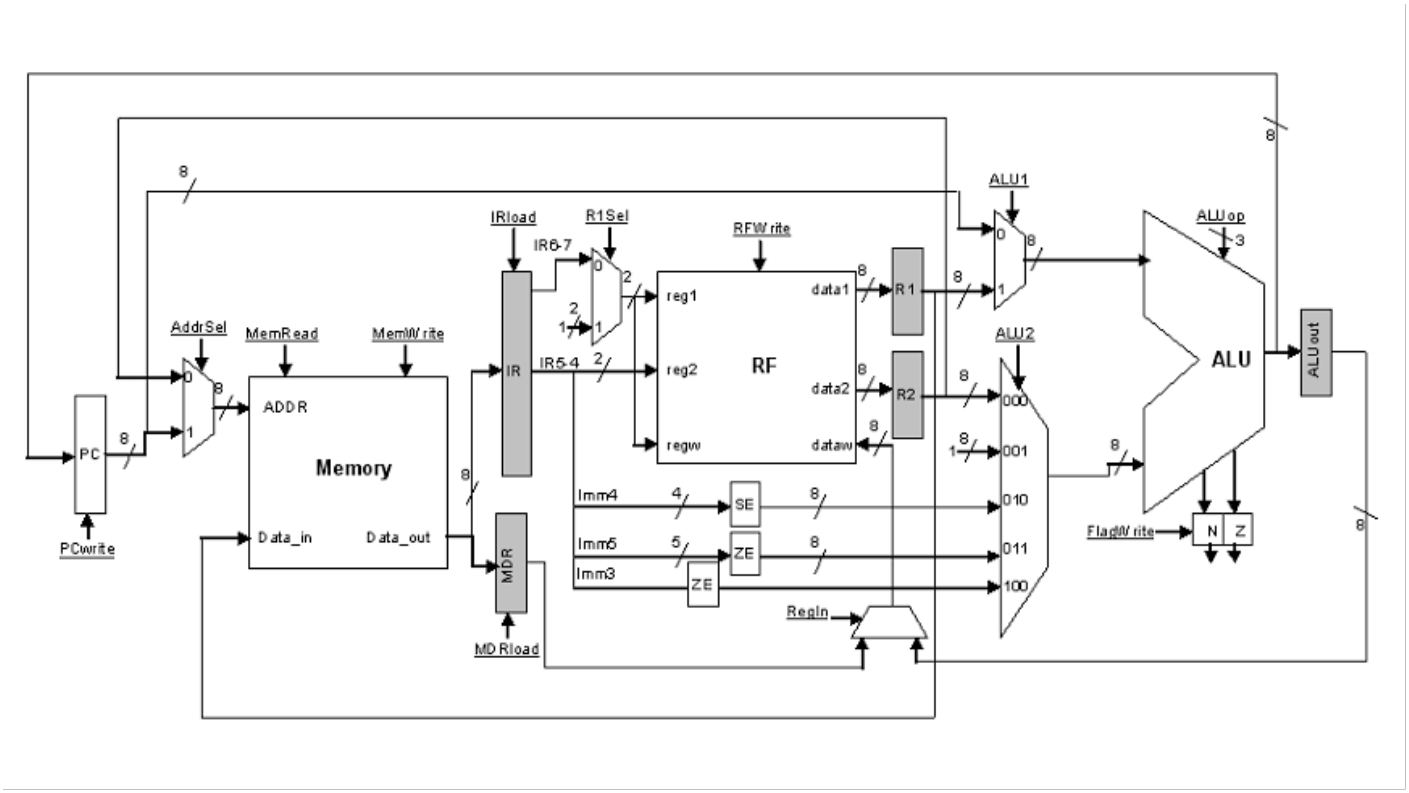
i3 i2 i1 i0 n z



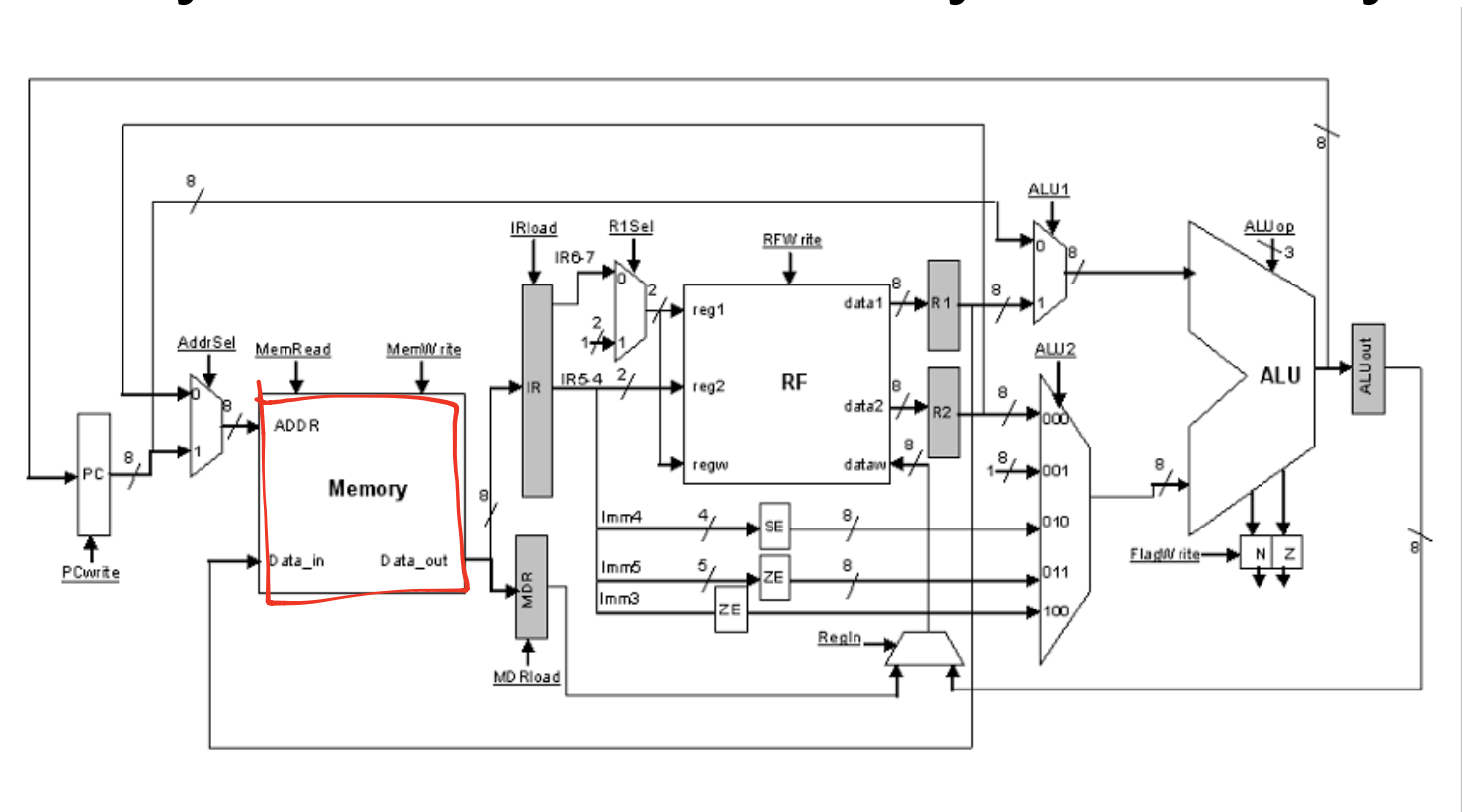


# CPU: Multicycle Implementation

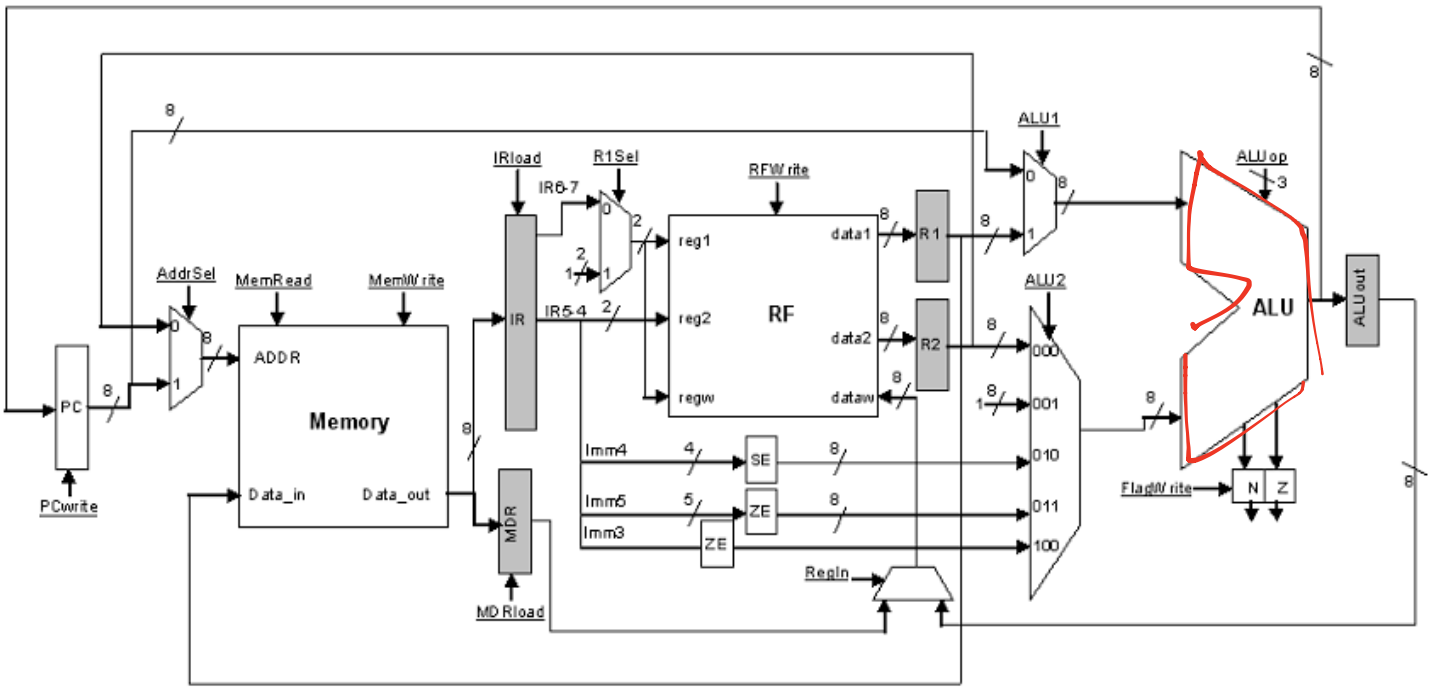
# A Multicycle Datapath



# Key Difference #1: Only 1 Memory

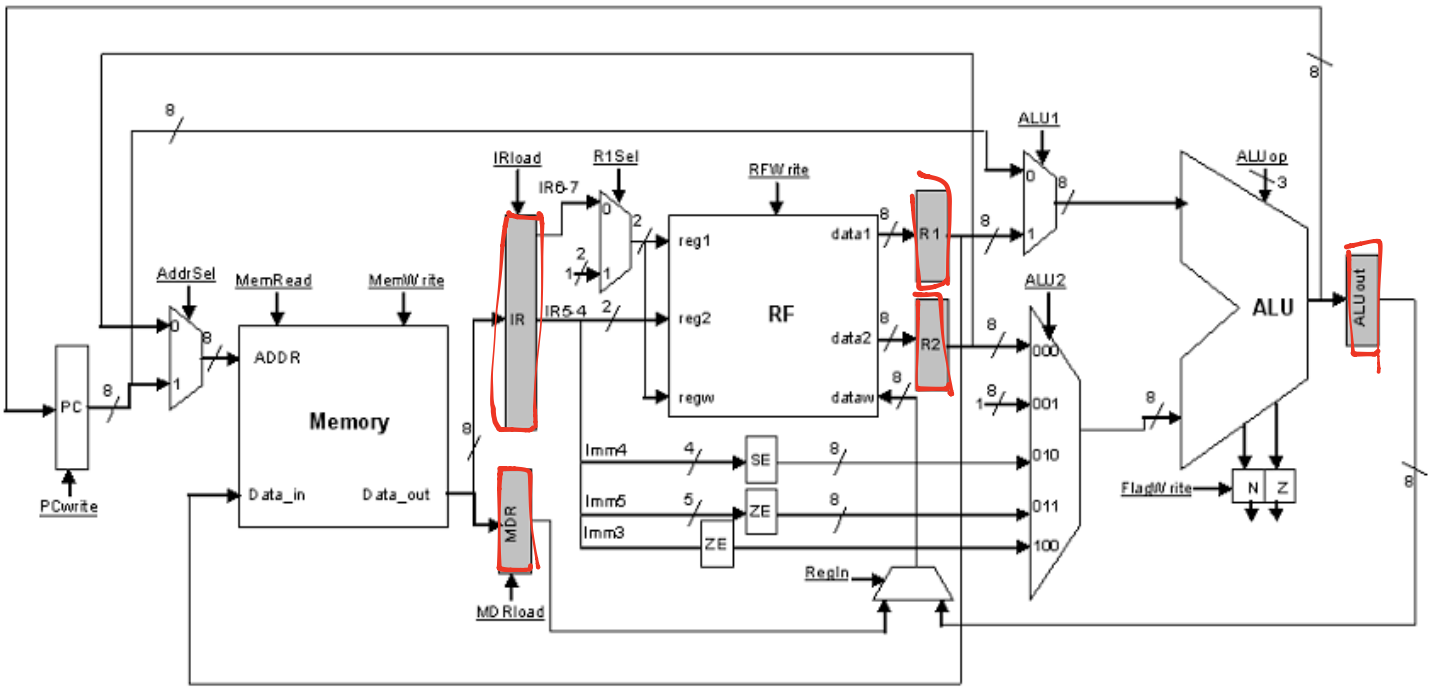


# Key Difference #2: Only 1 ALU



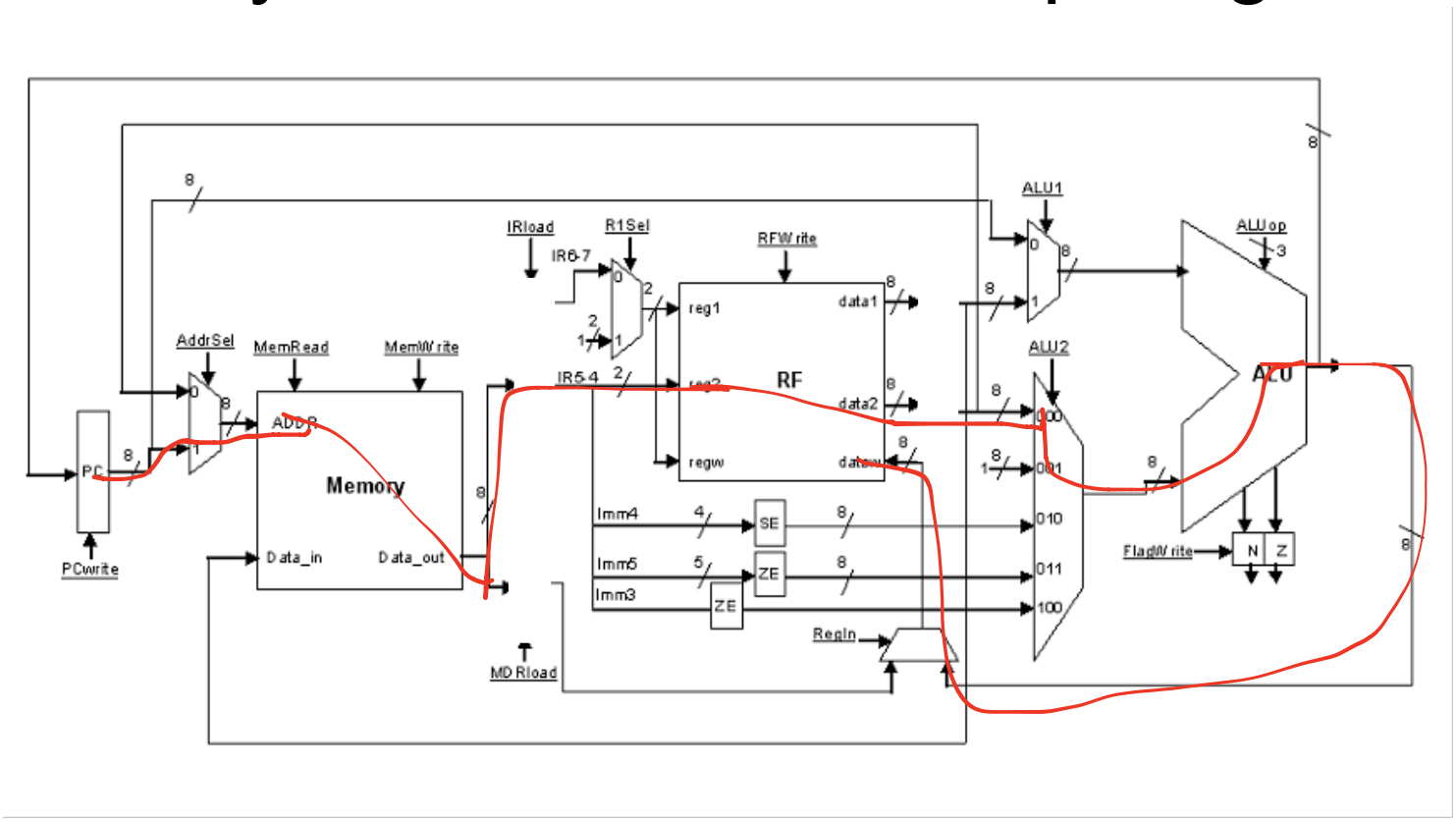

previously: 1 ALU + 2 Adders

# Key Difference #3: Temp Regs



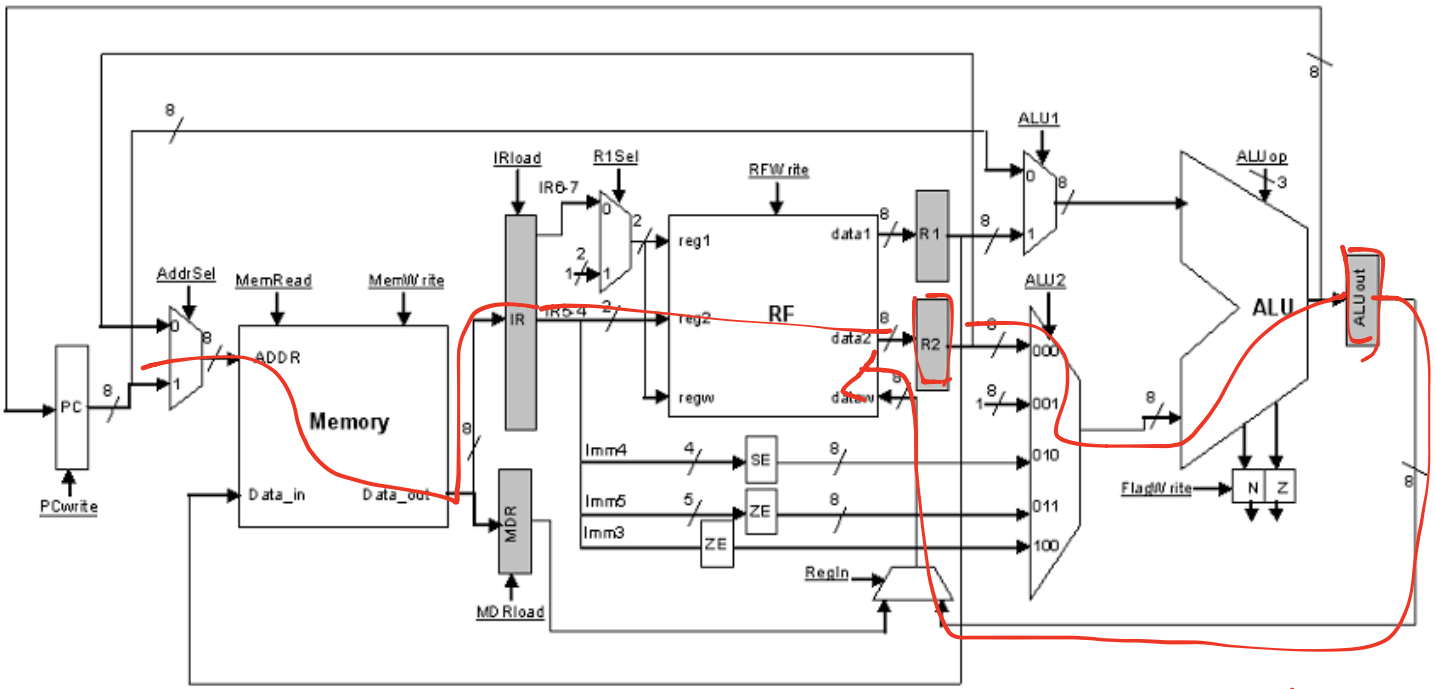

what benefit are temp regs/multicycle?

# Key Difference #3: Temp Regs




Critical path is long → large clock period!

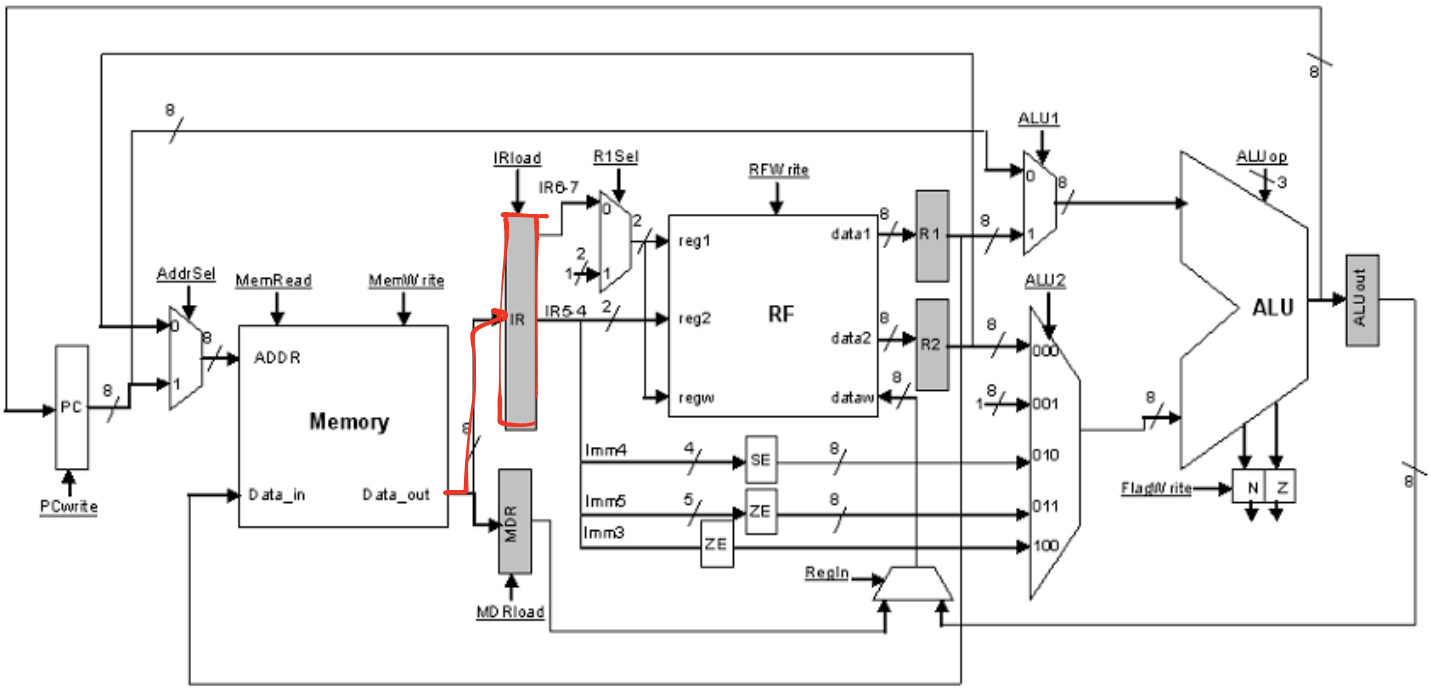
# Key Difference #3: Temp Regs




1 cycle

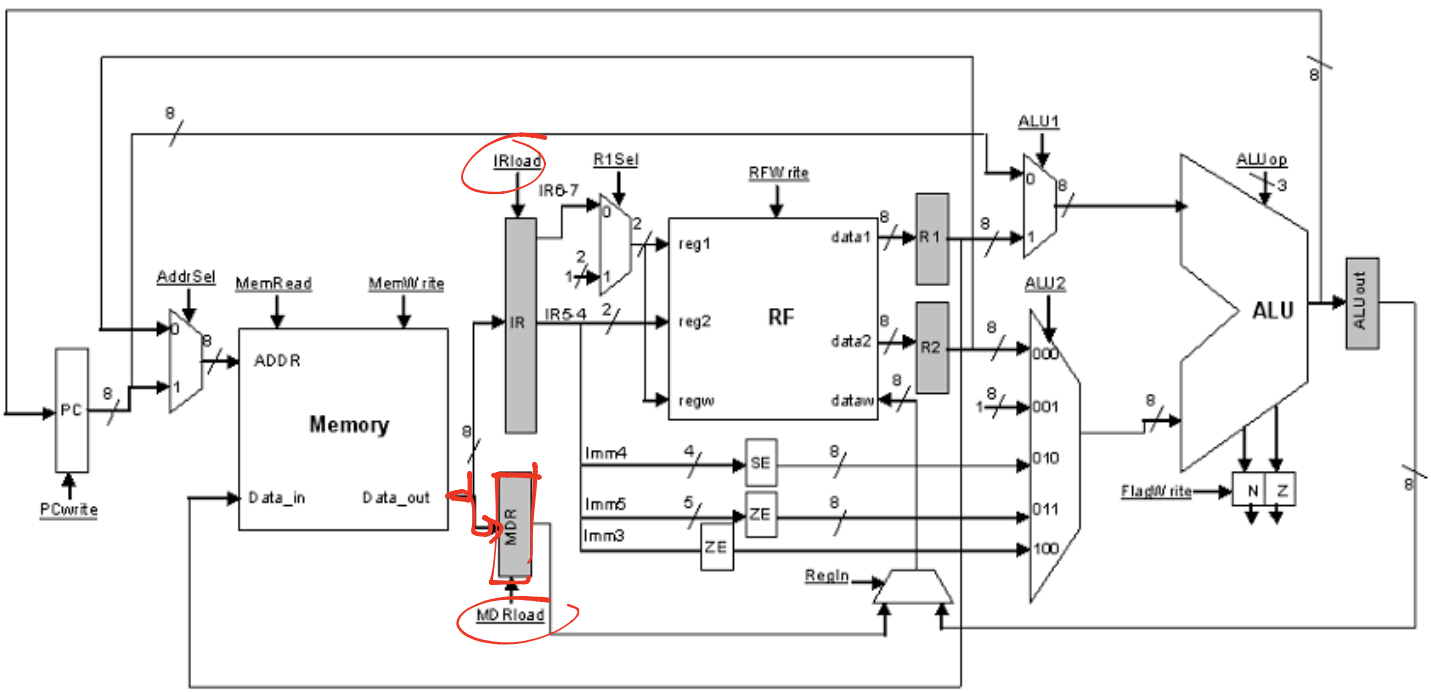
Smaller critical path → shorter clk period!

# IR: Instruction Register



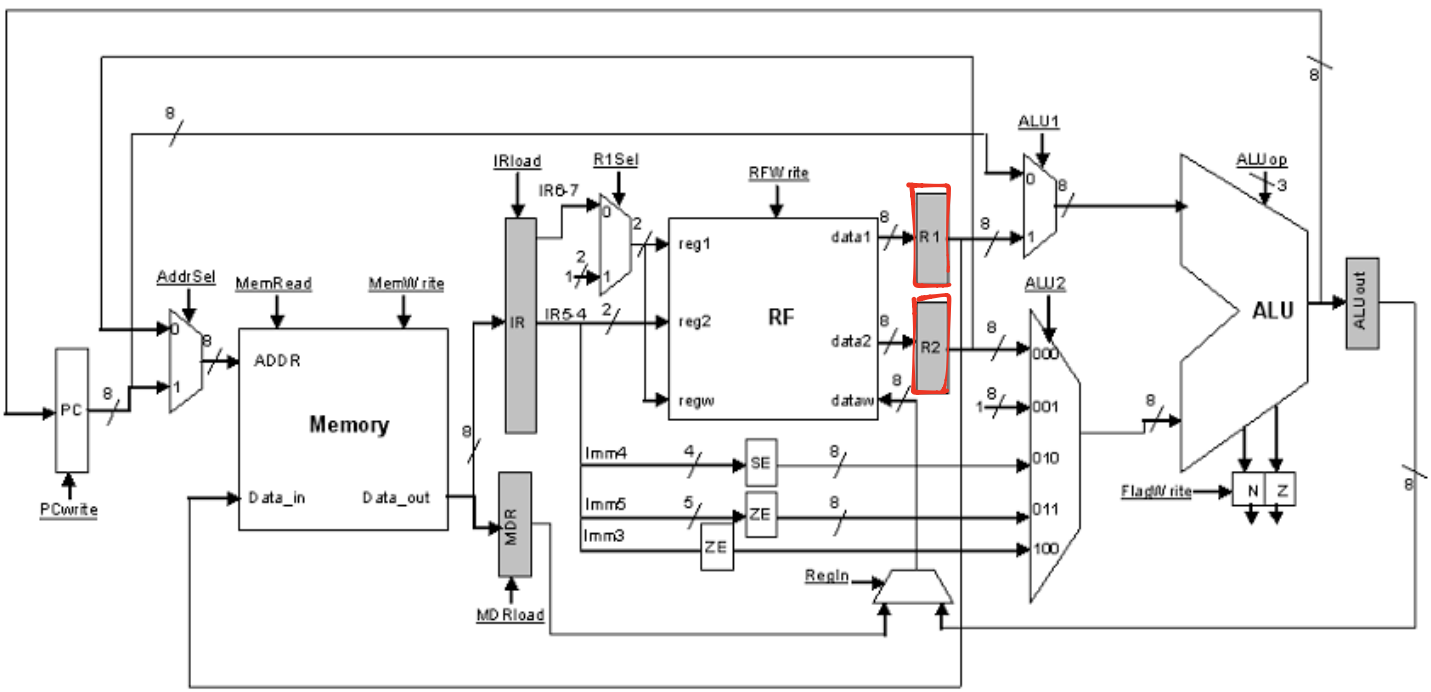

holds instruction encoding

# MDR: Memory Data Register



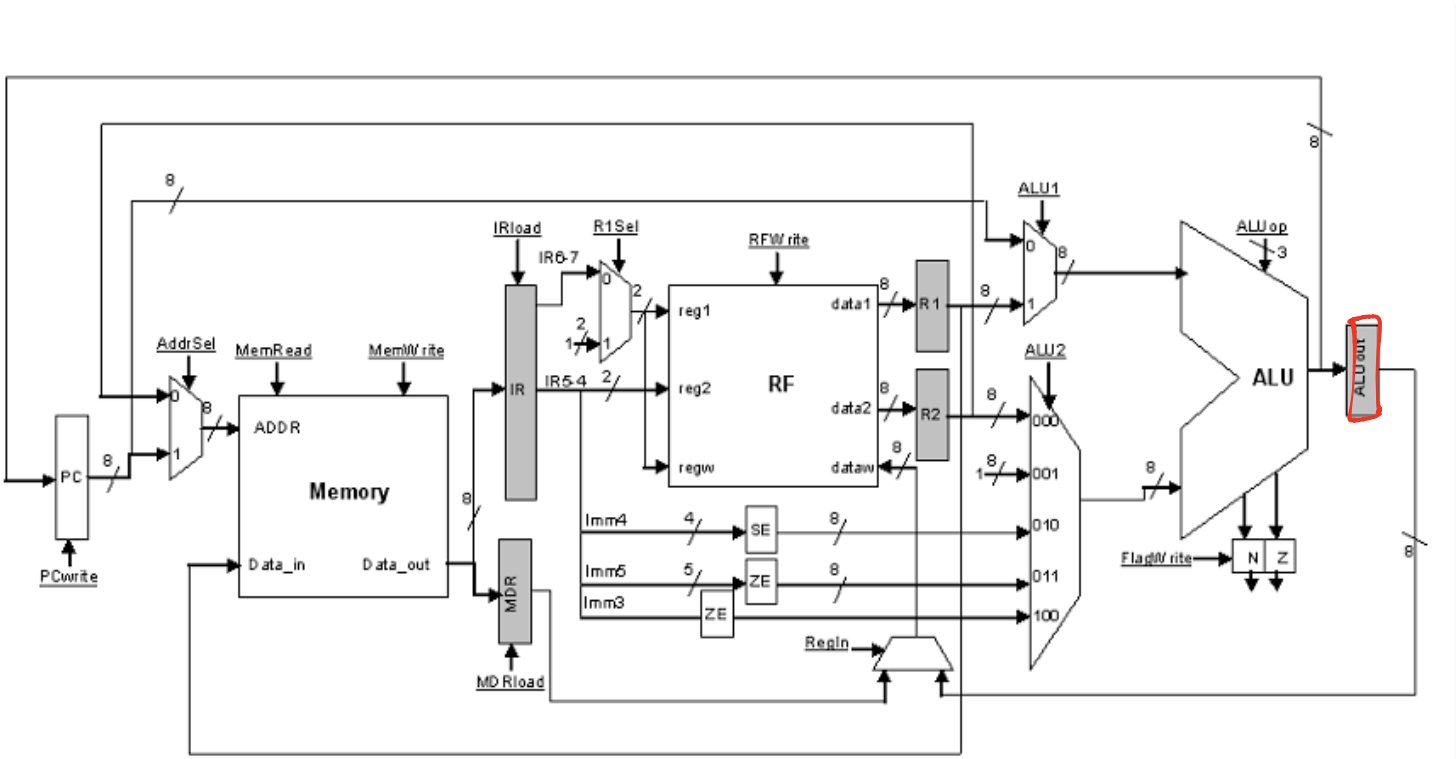

holds value returned from memory

# R1 and R2




hold values from reg file

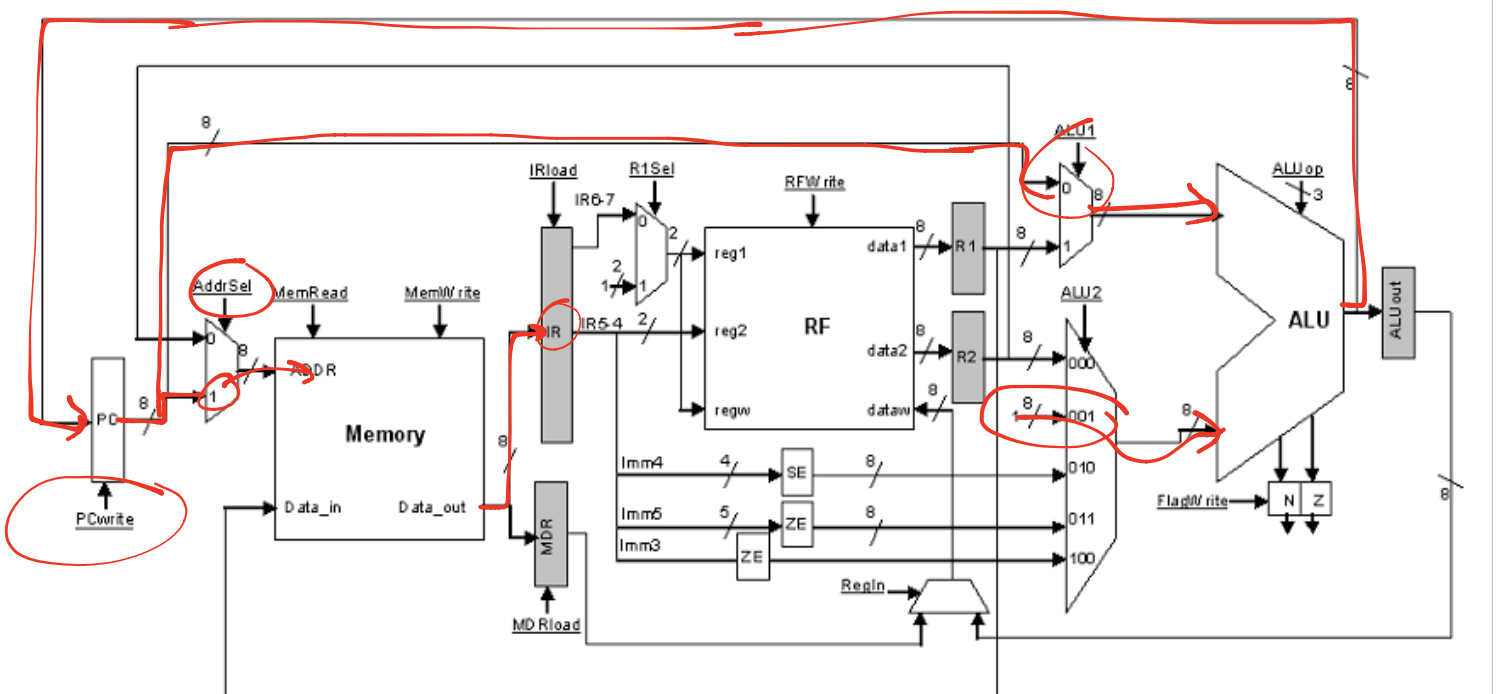
# ALUout




holds result calculated by ALU

# Cycle by Cycle Operation

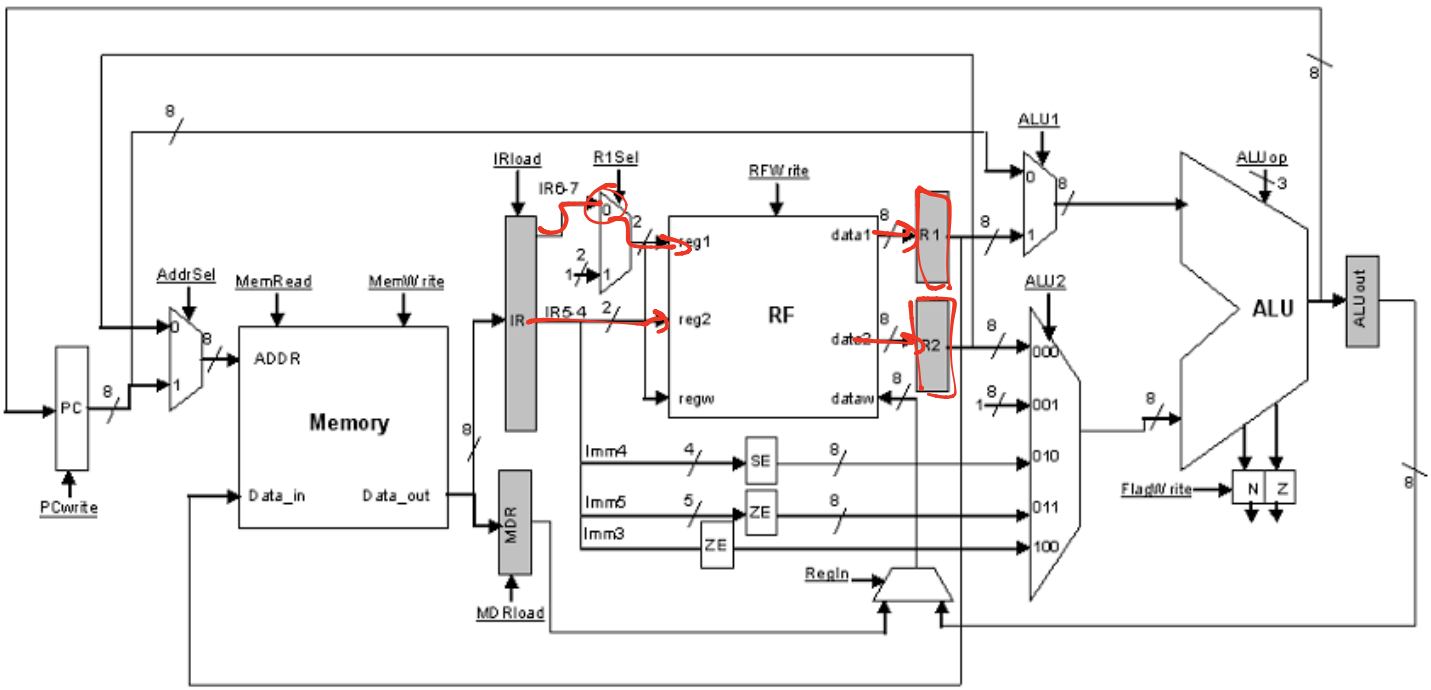
## All Insts Cycle1:
## Fetch and Increment PC




IR ← mem[PC]

PC ← PC+1

# All Insts Cycle2: Decoding Inst & Reading Reg File

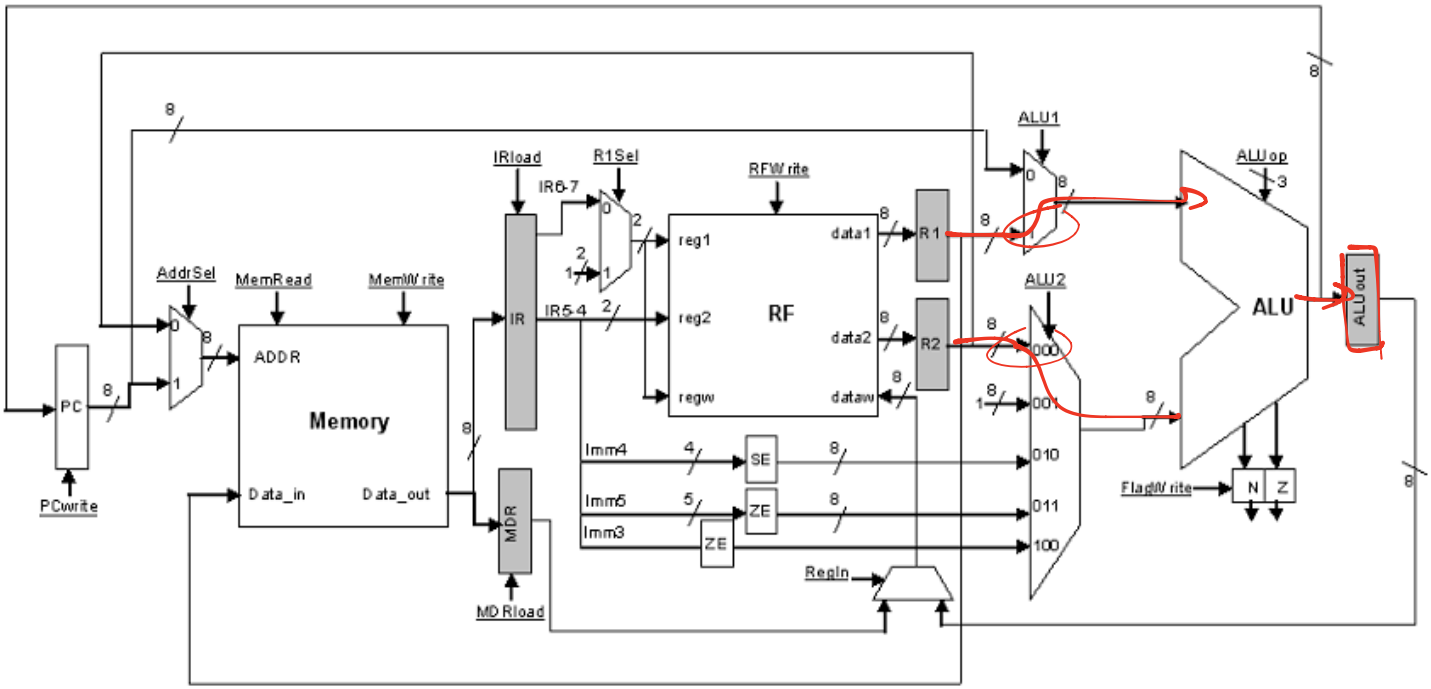



R1 ← KX (IR[7..6])

R2 ← Ky (IR[5..4])

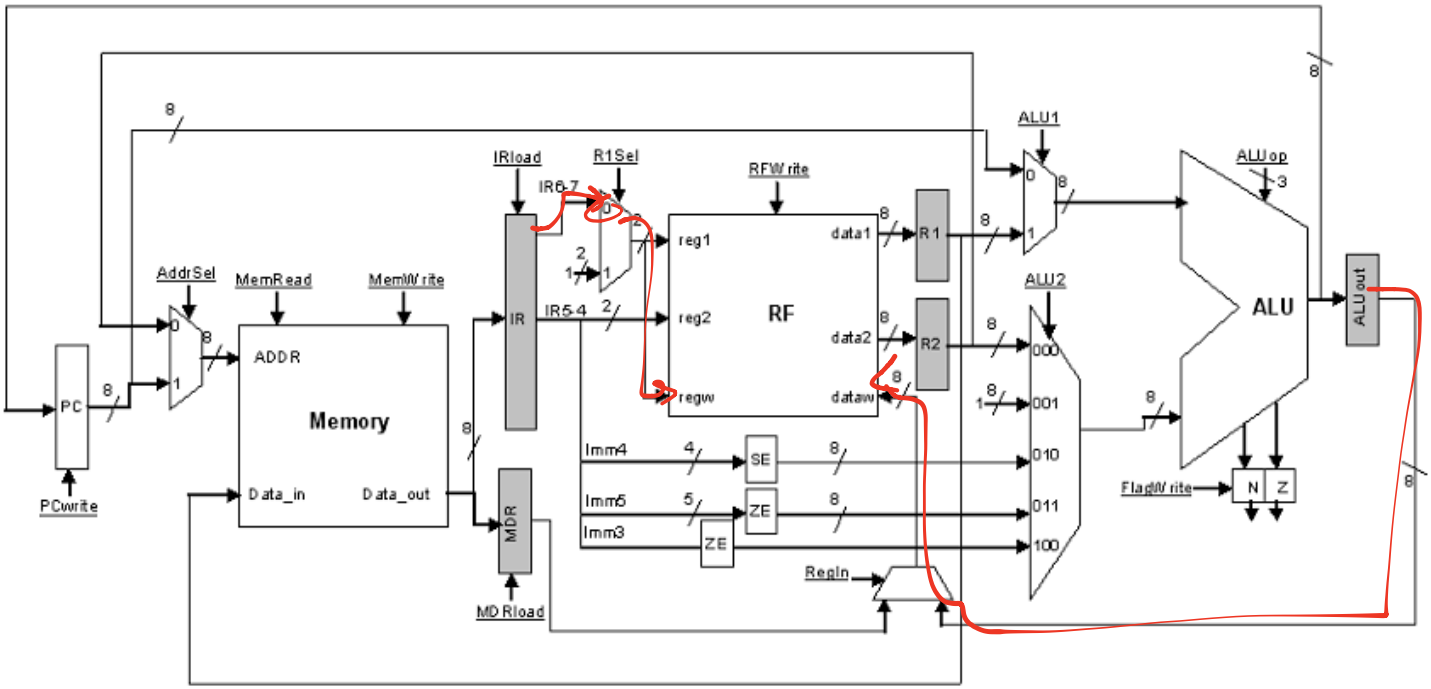
Note: not all instr need R1 & R2

# Add, Sub, Nand Cycle3: Calculate



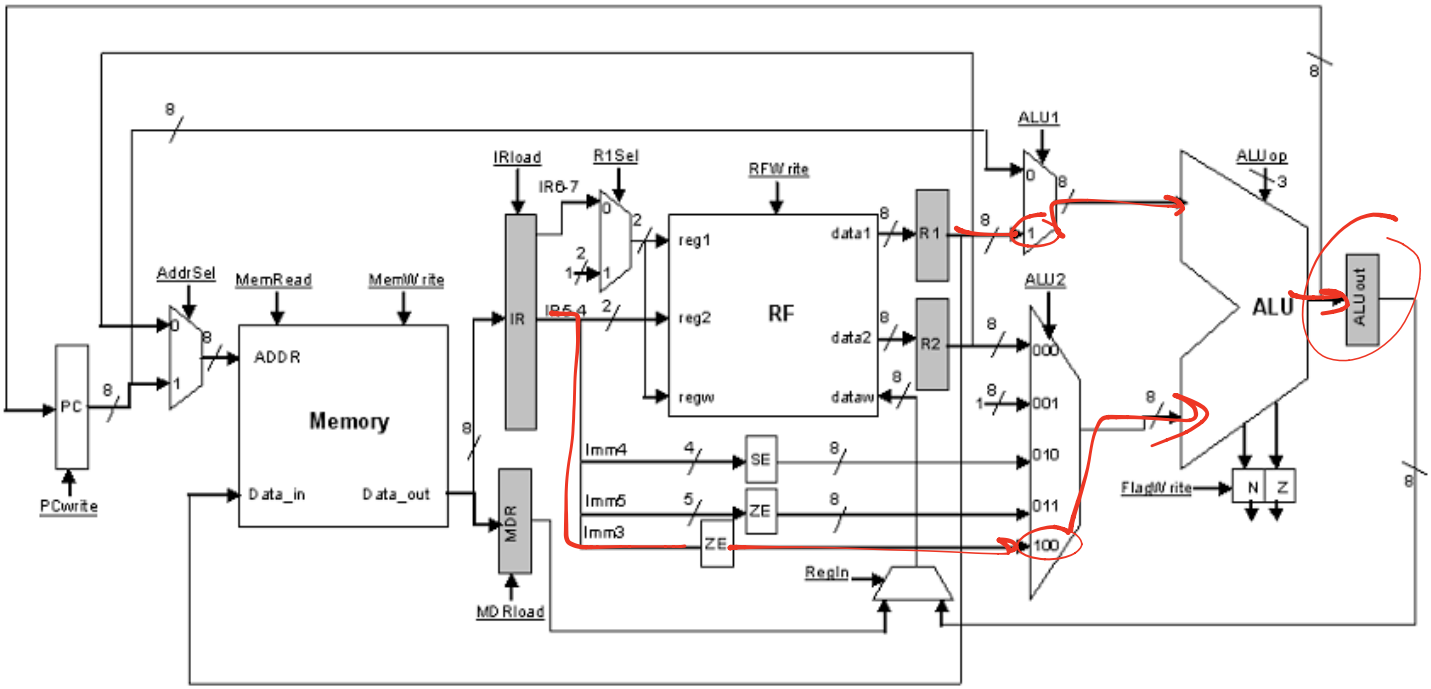

ALUout ← R1 op R2

# Add, Sub, Nand Cycle4: Write to Reg File



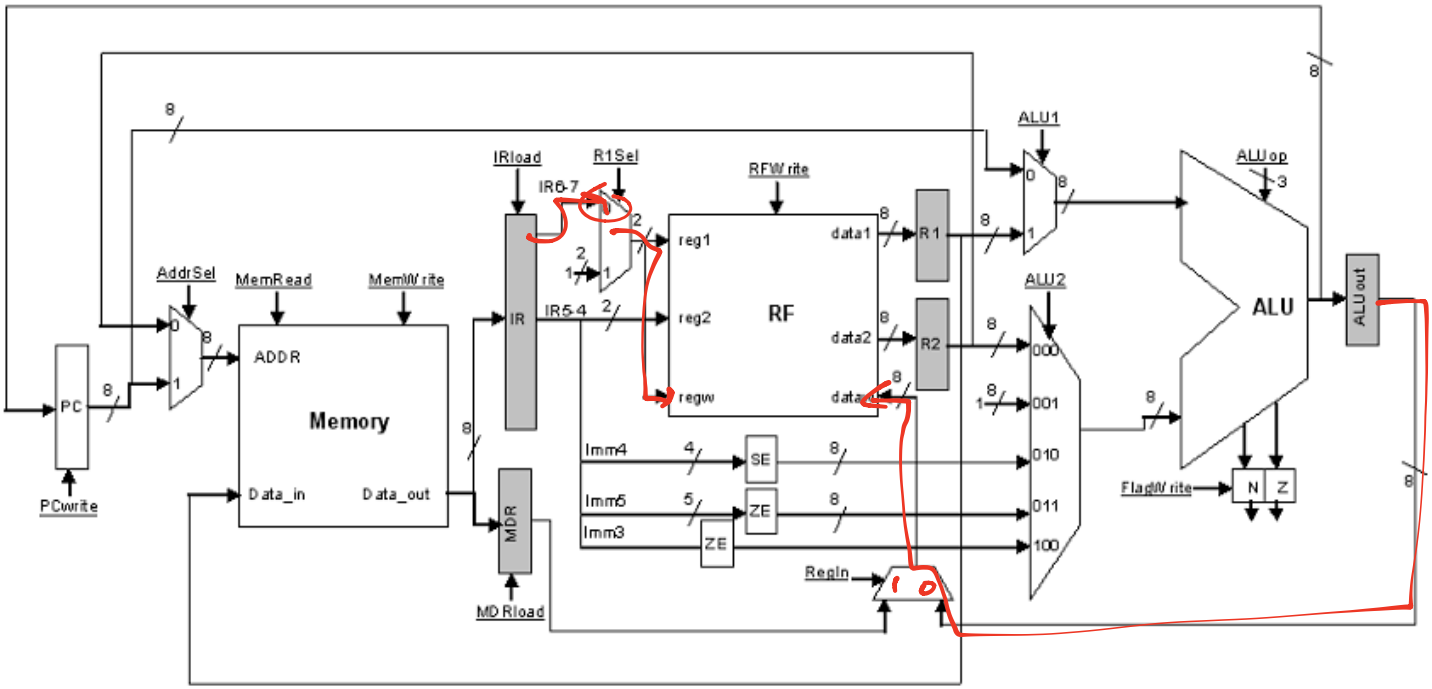

Rx ← ALUout

# Shift Cycle3: Calculate




ALUout = R1 op Imm3

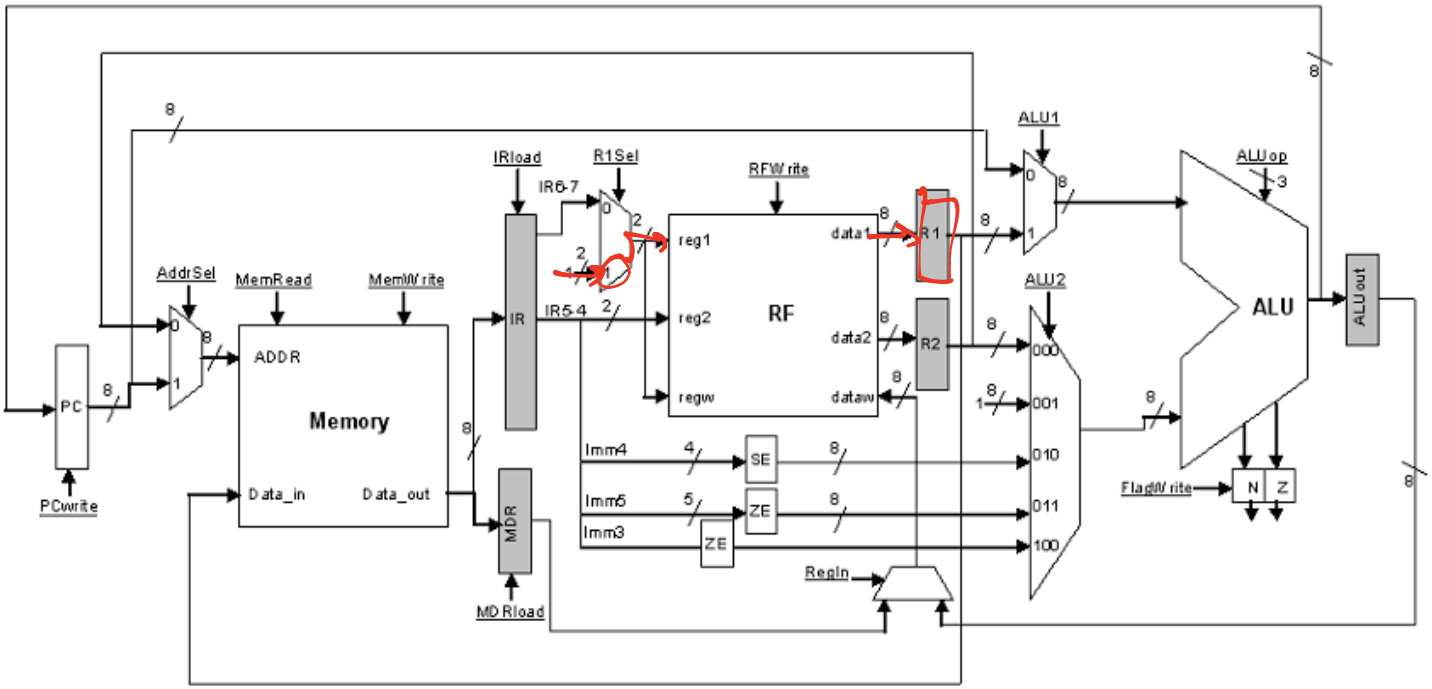
# Shift Cycle4: Write to Reg File




Rx ← ALUout

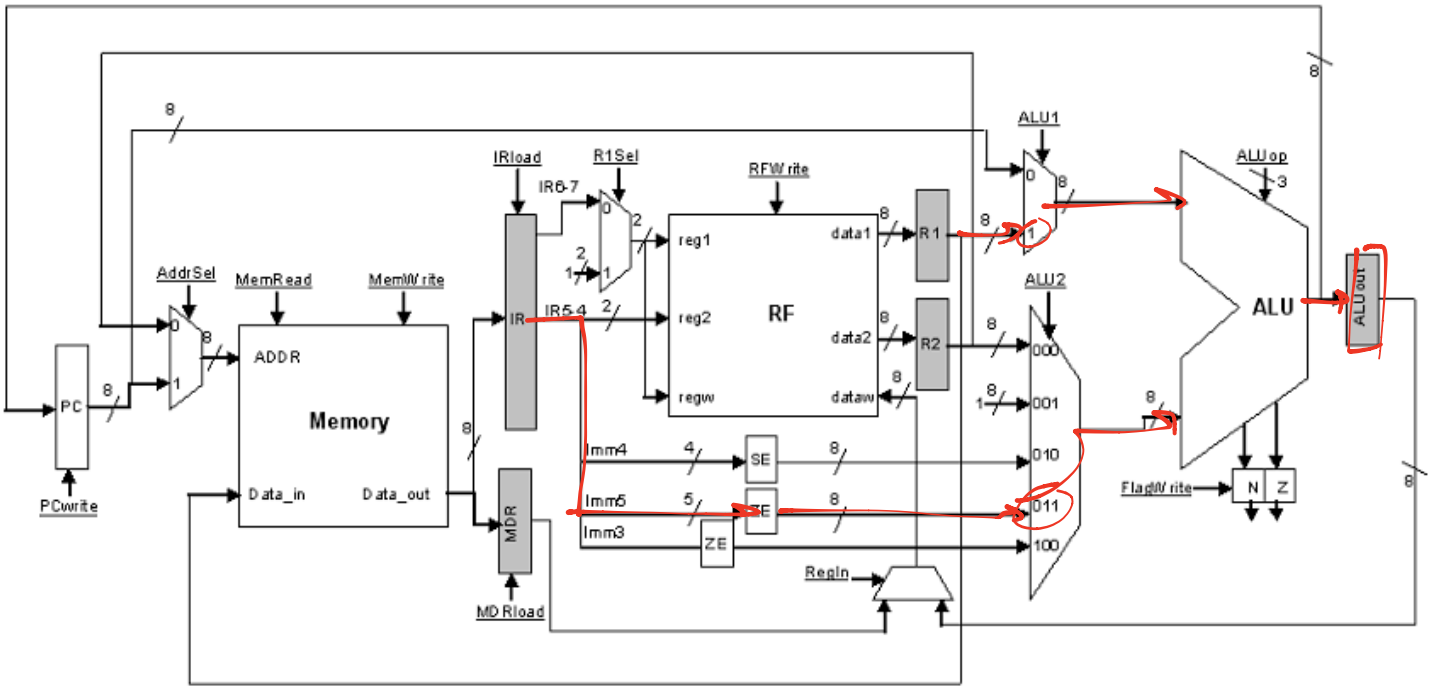
IR[7..6]

# ORI Cycle3: Read K1 from Reg File



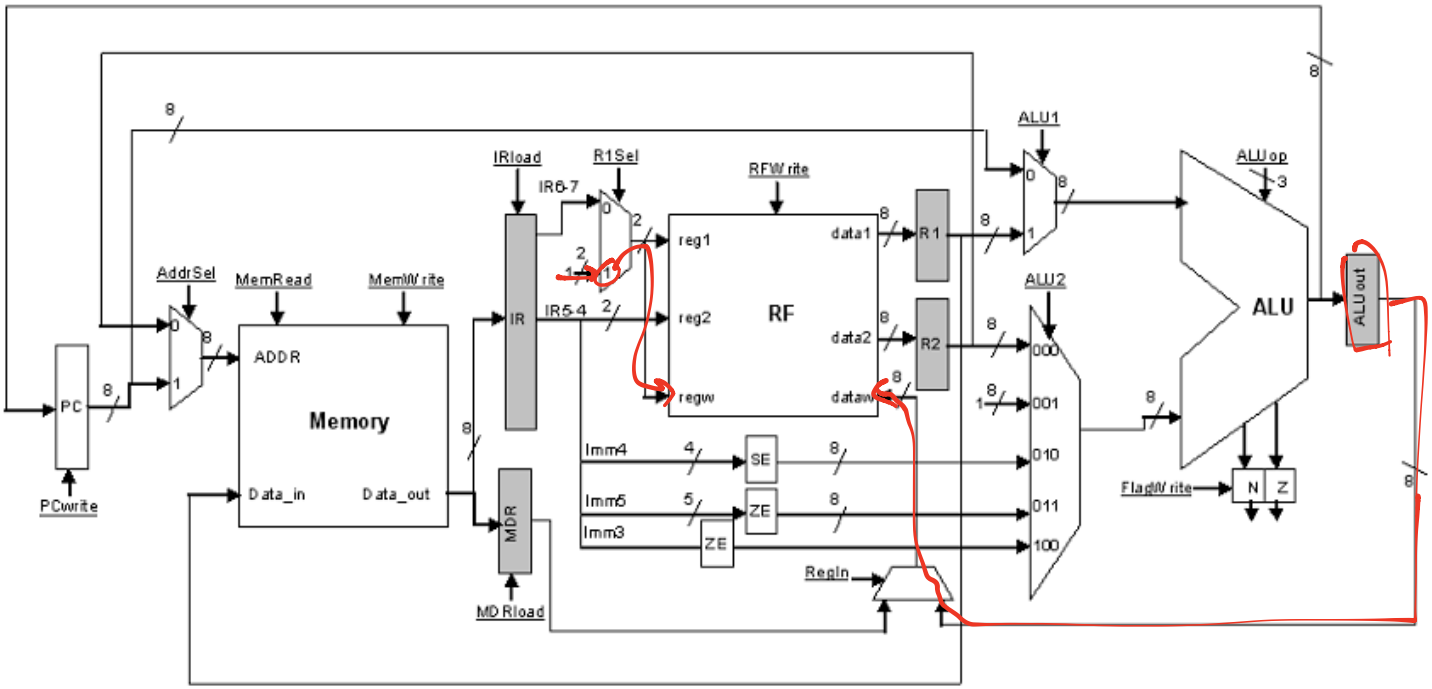

R1 ← K1

# ORI Cycle4: Calculate



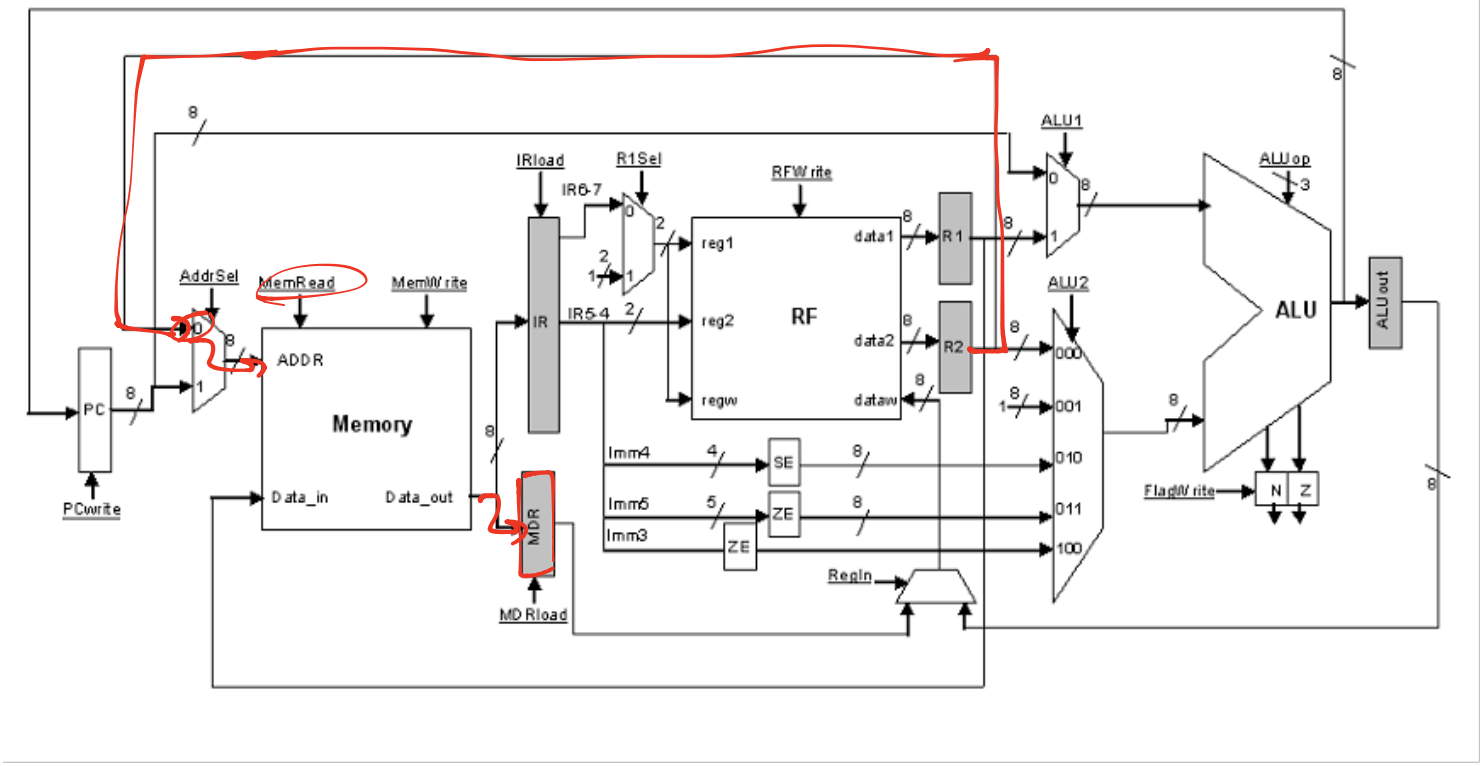

ALUout ← R1 op Imm5

# ORI Cycle5: Write to Reg File



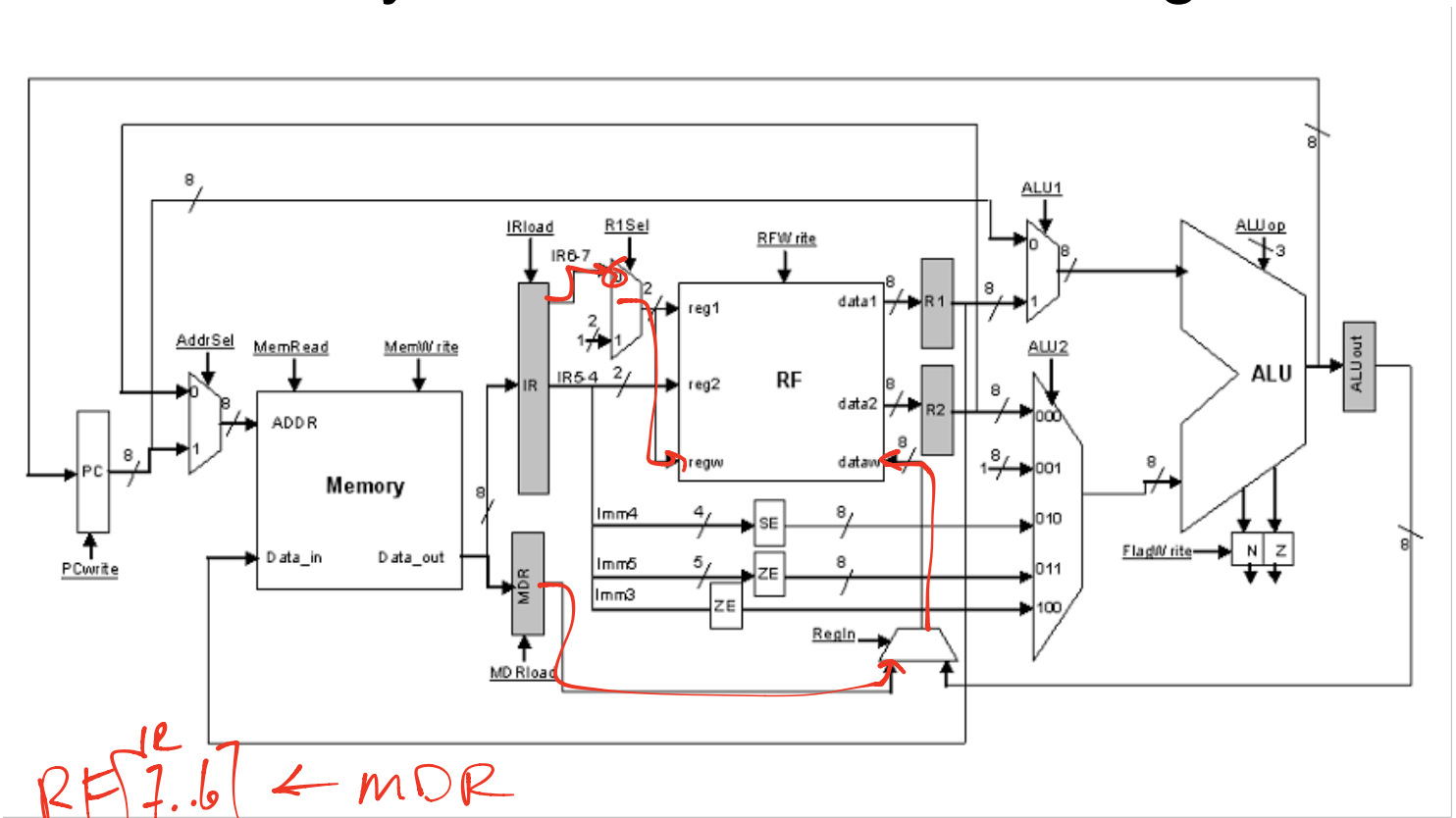

R1 ← ALUout

# Load Cycle3: addr to Mem, value into MDR

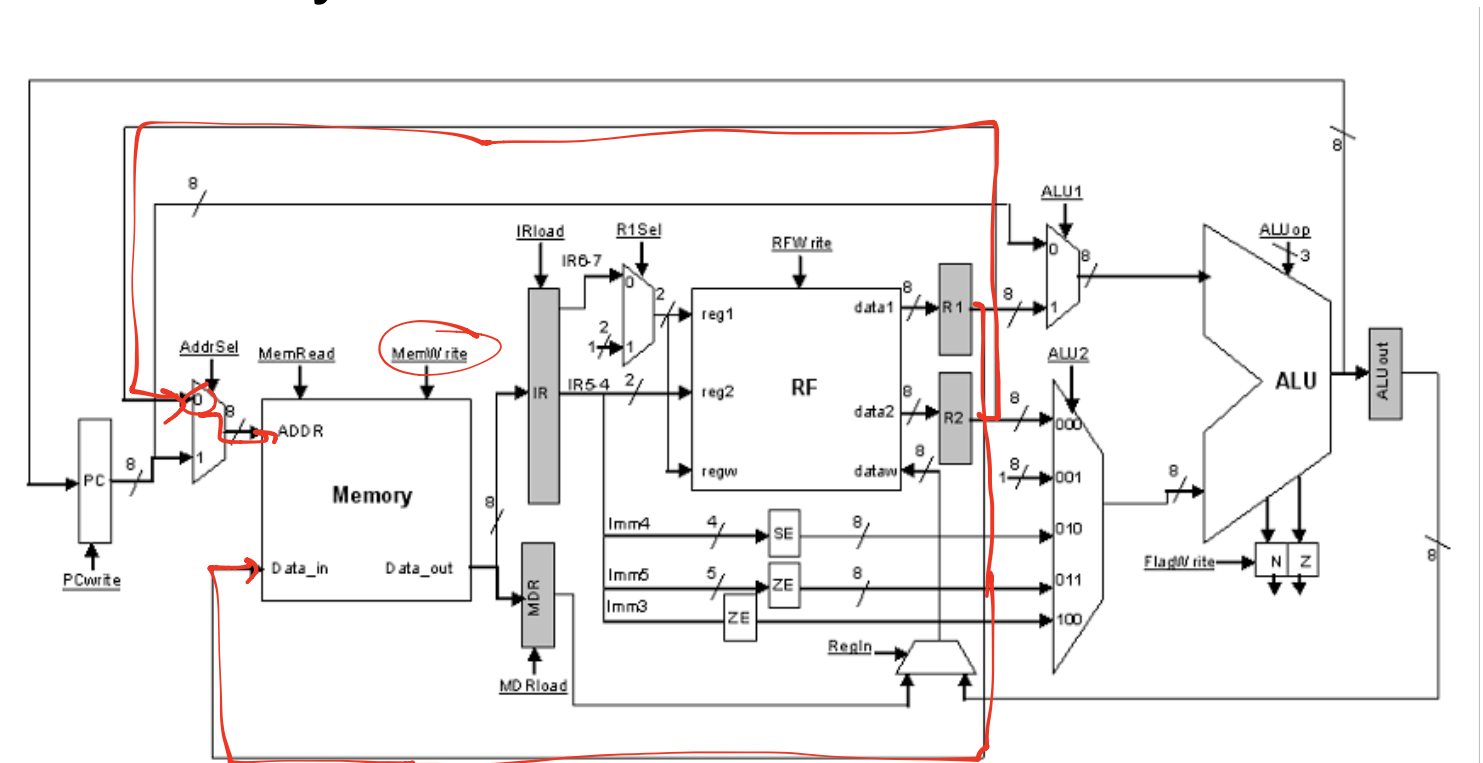



MDR ← mem[R2]

# Load Cycle4: write value into reg file

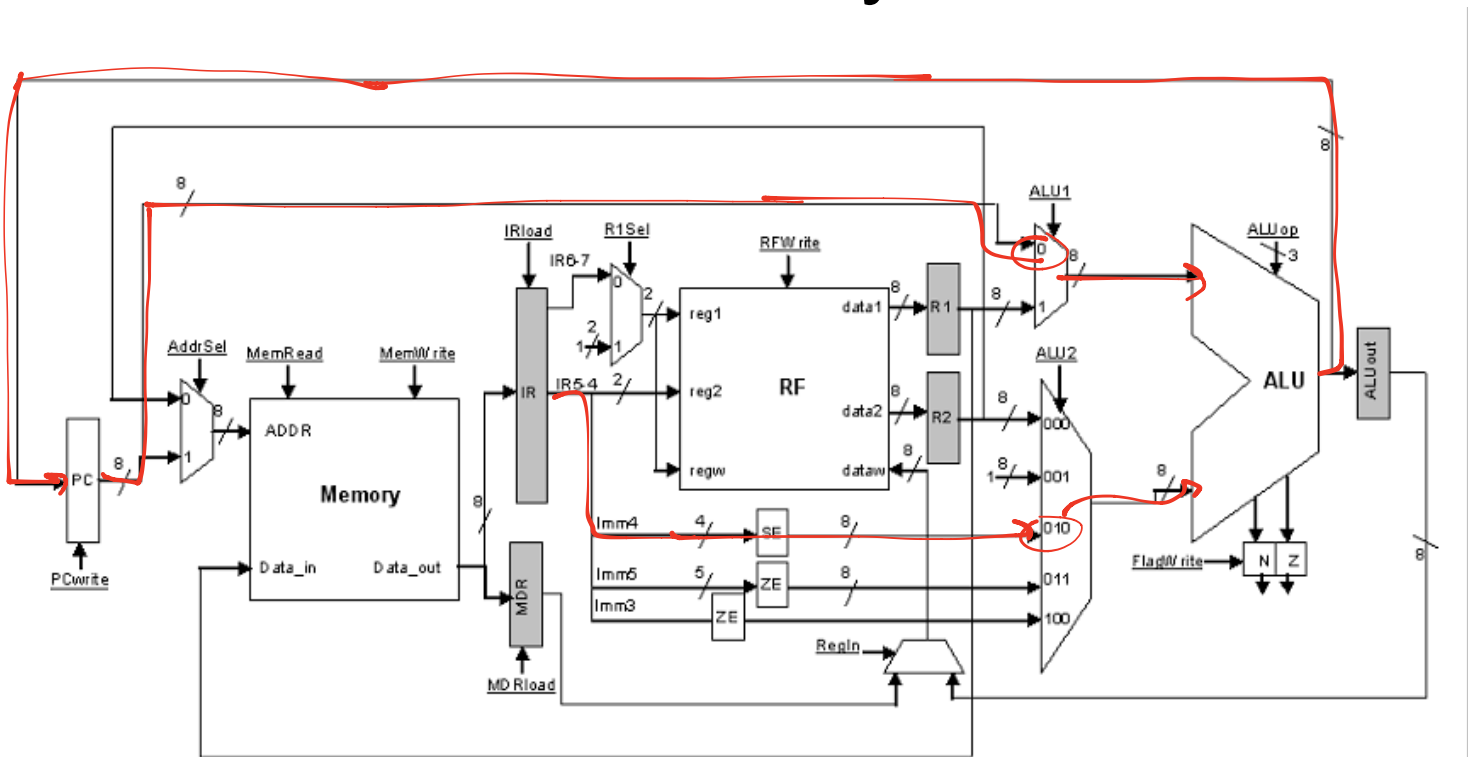


# Store Cycle3: addr to Mem, value to Mem




mem[R2] ← R1

# Branches Cycle3




PC ← PC + Imm4 (if branch taken)

# Summary

| Instructions | Single Cycle<br>Eg: 1 MHz | Multicycle<br>Eg: 4 MHz |
|---|---|---|
| Store, BZ, BNZ, BPZ | 1 cycle | 3 cycles |
| Add, Sub, Nand, Load, Shift | 1 cycle | 4 cycles |
| ORI | 1 cycle | 5 cycles |

Example: total time to execute one of each instruction:

Single cycle: $1 \times 4 + 1 \times 5 + 1 \times 1 = 10$ cycles / 1 MHz
$= 10$ us

multicycle: $3 \times 4 + 4 \times 5 + 5 \times 1 = 37$ cycles / 4 MHz
$= 9.25$ us

# Implementing Multicycle Control

| | Add, Sub, Nand | Shift | Ori | Load | Store | Bnz, Bz, Bpz |
|---|---|---|---|---|---|---|
| 1 | IR = Mem[PC]<br>PC = PC + 1 | | | | | |
| 2 | R1 = RF[IR[7..6]]<br>R2 = RF[IR[5..4]] | | | | | |
| 3 | ALUout = R1 op R2 | ALUout = R1 Shift Imm3 | R1 = RF[1] | MDR = mem[R2] | mem[R2] = R1 | PC = PC + SE(Imm4) |
| 4 | RF[IR[7..6]] = ALUout | | ALUout = R1 OR Imm8 | RF[IR[7..6]] = MDR | X | X |
| 5 | X | X | RF[1] = ALUout | X | X | X |

# Control: An FSM

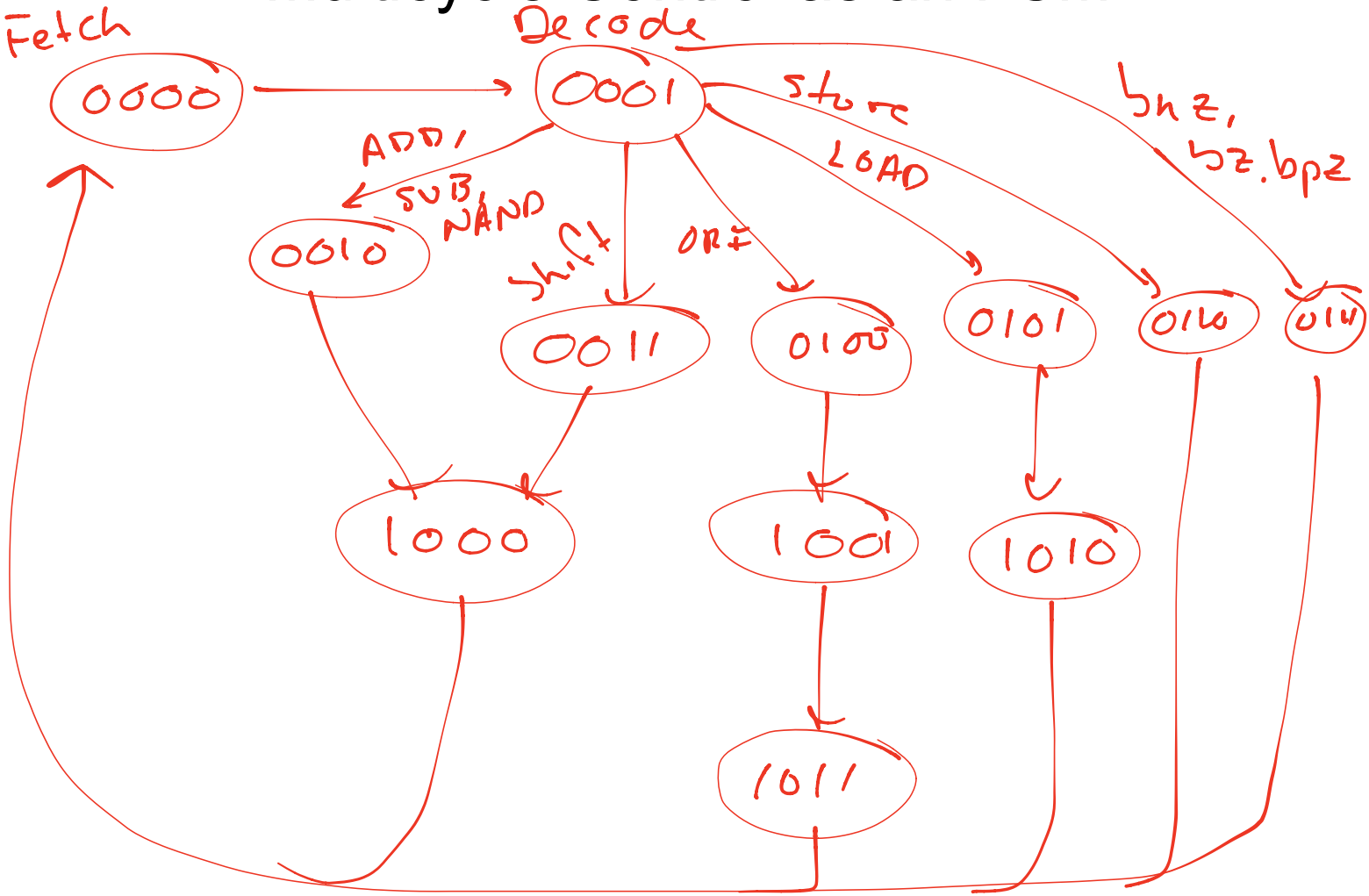
- need a state transition diagram
- how many states are there?

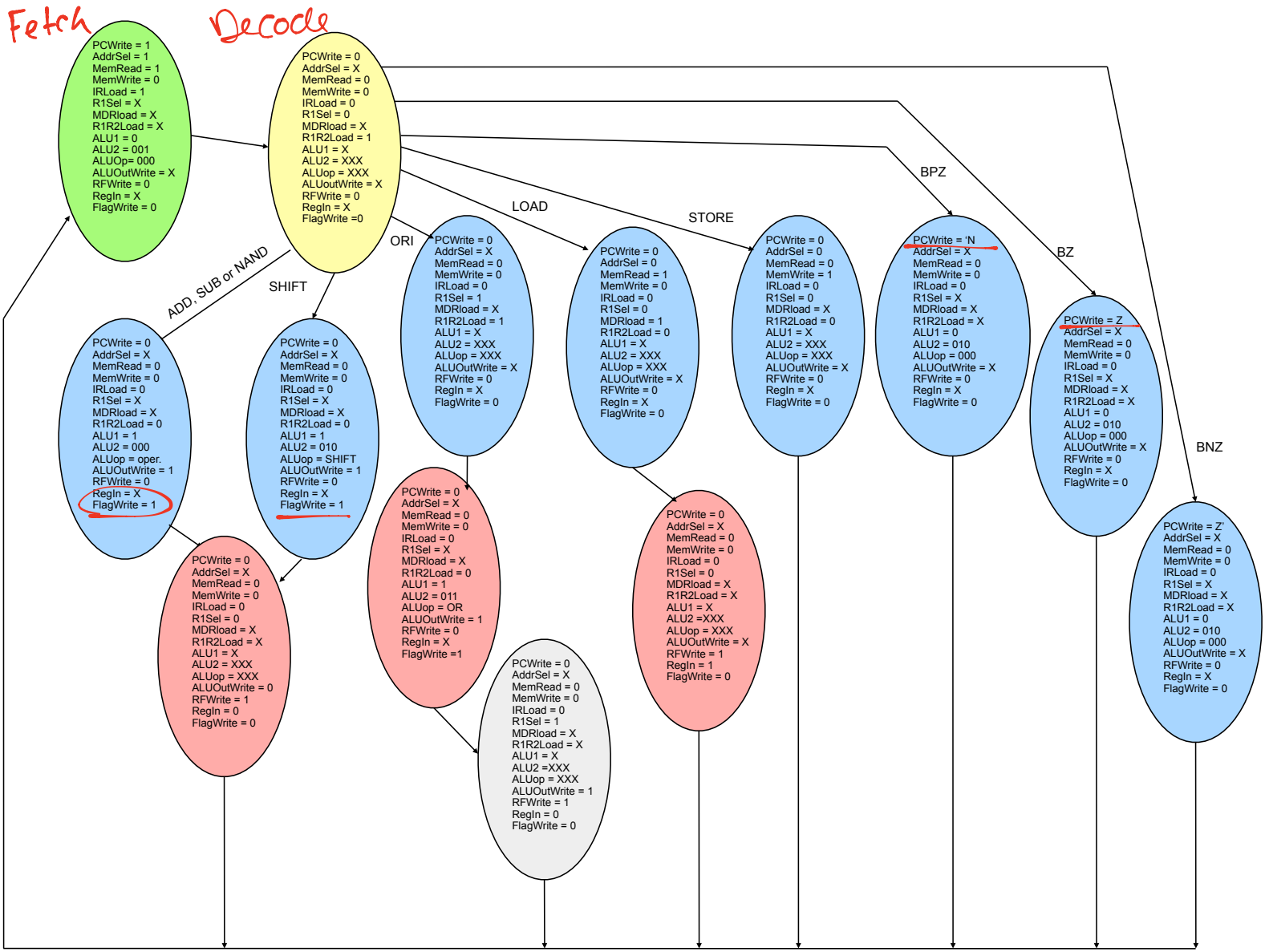
12

- how many bits to represent state?

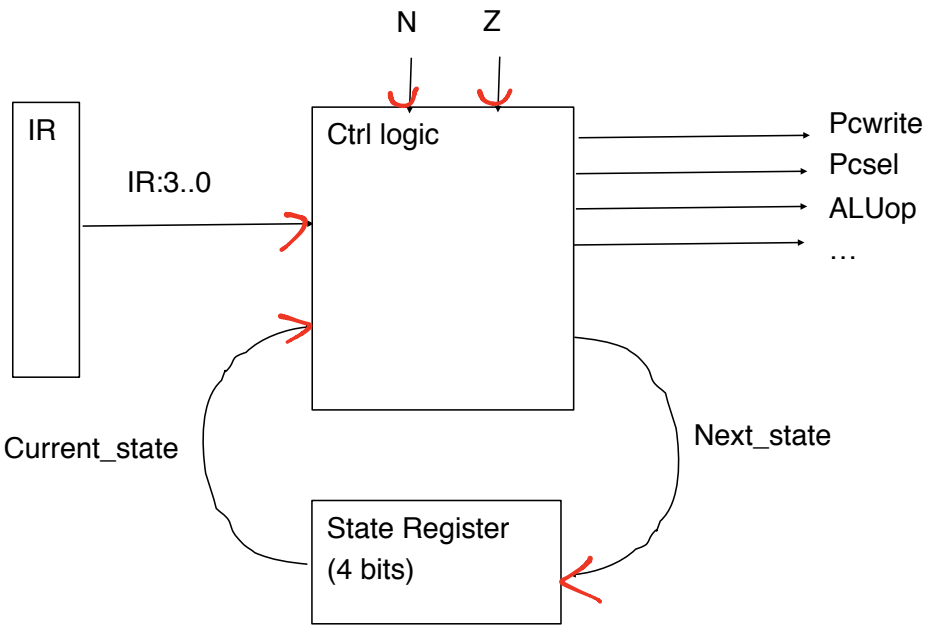
4 state bits

# Multicycle Control as an FSM




Fetch
PCWrite = 1
AddrSel = 1
MemRead = 1
MemWrite = 0
IRLoad = 1
R1Sel = X
MDRload = X
R1R2Load = X
ALU1 = 0
ALU2 = 001
ALUOp= 000
ALUOutWrite = X
RFWrite = 0
RegIn = X
FlagWrite = 0
Decode
PCWrite = 0
AddrSel = X
MemRead = 0
MemWrite = 0
IRLoad = 0
R1Sel = 0
MDRload = X
R1R2Load = 1
ALU1 = X
ALU2 = XXX
ALUop = XXX
ALUoutWrite = X
RFWrite = 0
RegIn = X
FlagWrite =0
ADD, SUB or NAND
SHIFT
ORI
LOAD
STORE
BPZ
BZ
BNZ
PCWrite = 0
AddrSel = X
MemRead = 0
MemWrite = 0
IRLoad = 0
R1Sel = X
MDRload = X
R1R2Load = 0
ALU1 = 1
ALU2 = 000
ALUop = oper.
ALUOutWrite = 1
RFWrite = 0
RegIn = X
FlagWrite = 1
PCWrite = 0
AddrSel = X
MemRead = 0
MemWrite = 0
IRLoad = 0
R1Sel = X
MDRload = X
R1R2Load = 0
ALU1 = 1
ALU2 = 010
ALUop = SHIFT
ALUOutWrite = 1
RFWrite = 0
RegIn = X
FlagWrite = 1
PCWrite = 0
AddrSel = X
MemRead = 0
MemWrite = 0
IRLoad = 0
R1Sel = 1
MDRload = X
R1R2Load = 1
ALU1 = X
ALU2 = XXX
ALUop = XXX
ALUOutWrite = X
RFWrite = 0
RegIn = X
FlagWrite = 0
PCWrite = 0
AddrSel = X
MemRead = 1
MemWrite = 0
IRLoad = 0
R1Sel = 0
MDRload = 1
R1R2Load = 0
ALU1 = X
ALU2 = XXX
ALUop = XXX
ALUOutWrite = X
RFWrite = 0
RegIn = X
FlagWrite = 0
PCWrite = 0
AddrSel = 0
MemRead = 0
MemWrite = 1
IRLoad = 0
R1Sel = 0
MDRload = X
R1R2Load = 0
ALU1 = X
ALU2 = XXX
ALUop = XXX
ALUOutWrite = X
RFWrite = 0
RegIn = X
FlagWrite = 0
PCWrite = 'N
AddrSel = X
MemRead = 0
MemWrite = 0
IRLoad = 0
R1Sel = X
MDRload = X
R1R2Load = X
ALU1 = 0
ALU2 = 010
ALUop = 000
ALUOutWrite = X
RFWrite = 0
RegIn = X
FlagWrite = 0
PCWrite = Z
AddrSel = X
MemRead = 0
MemWrite = 0
IRLoad = 0
R1Sel = X
MDRload = X
R1R2Load = X
ALU1 = 0
ALU2 = 010
ALUop = 000
ALUOutWrite = X
RFWrite = 0
RegIn = X
FlagWrite = 0
PCWrite = Z'
AddrSel = X
MemRead = 0
MemWrite = 0
IRLoad = 0
R1Sel = X
MDRload = X
R1R2Load = X
ALU1 = 0
ALU2 = 010
ALUop = 000
ALUOutWrite = X
RFWrite = 0
RegIn = X
FlagWrite = 0
PCWrite = 0
AddrSel = X
MemRead = 0
MemWrite = 0
IRLoad = 0
R1Sel = X
MDRload = X
R1R2Load = 0
ALU1 = 1
ALU2 = 011
ALUop = OR
ALUOutWrite = 1
RFWrite = 0
RegIn = X
FlagWrite =1
PCWrite = 0
AddrSel = X
MemRead = 0
MemWrite = 0
IRLoad = 0
R1Sel = 1
MDRload = X
R1R2Load = X
ALU1 = X
ALU2 =XXX
ALUop = XXX
ALUOutWrite = 1
RFWrite = 1
RegIn = 0
FlagWrite = 0
PCWrite = 0
AddrSel = X
MemRead = 0
MemWrite = 0
IRLoad = 0
R1Sel = 0
MDRload = X
R1R2Load = X
ALU1 = X
ALU2 =XXX
ALUop = XXX
ALUOutWrite = X
RFWrite = 1
RegIn = 1
FlagWrite = 0
PCWrite = 0
AddrSel = X
MemRead = 0
MemWrite = 0
IRLoad = 0
R1Sel = 0
MDRload = X
R1R2Load = X
ALU1 = X
ALU2 = XXX
ALUop = XXX
ALUOutWrite = 0
RFWrite = 1
RegIn = 0
FlagWrite = 0

# Multicycle Control Hardware




# CPU: Adding a New Instruction

# EXAMPLE QUESTION: ADDING A NEW INSTRUCTION

- **Implement a post-increment load:**
- **load r1, (r2)+**

  Does: RF[r1] = MEM[RF[r2]]
  RF[r2] = RF[r2] + 1

r2 is permanently changed to be r2+1

Implementing:
RF[r1] = MEM[RF[r2]]; RF[r2] = RF[r2] + 1

**Recall: load r1, (r2)**

IR = mem[PC], PC = PC + 1 - Fetch (same)

R1 = RF[r1], R2 = RF[r2] - Decode (same)

MDR = mem[R2], ALUout = R2 + 1

RF[r1] = MDR

RF[r2] = ALUout

# Modifying the Datapath

RF[r2] = RF[r2] + 1

ALUout = R2 + 1 ✓

RF[r2] = ALUout ✓