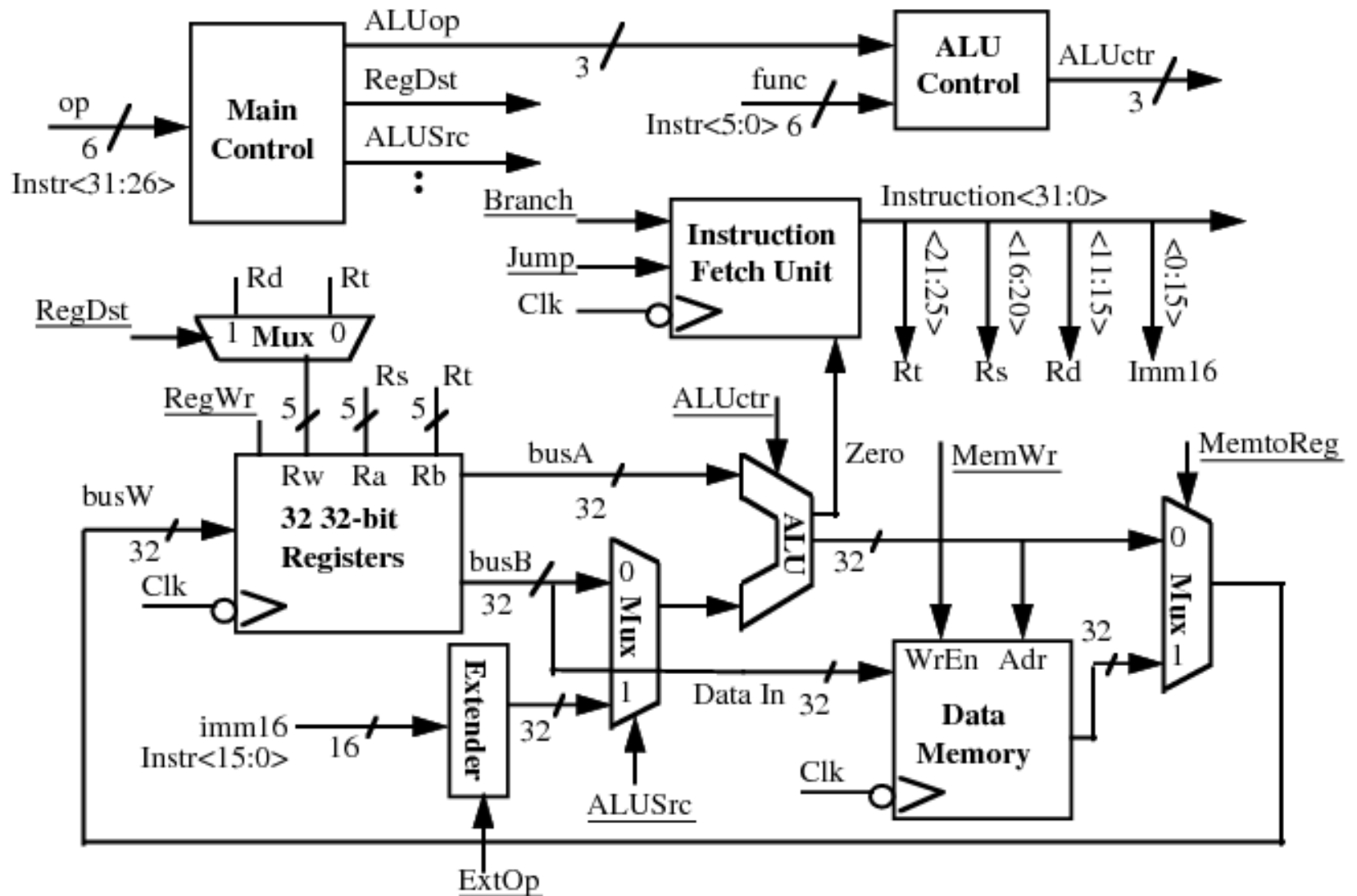Acknowledgment: Almost all of these slides are based on Dave Patterson's CS152 Lecture Slides at UC, Berkeyley.

COMPUTER SYSTEMS ORGANIZATION

Multi Cycle CPU Design -- Spring 2010 -- IIIT-H -- Suresh Purini

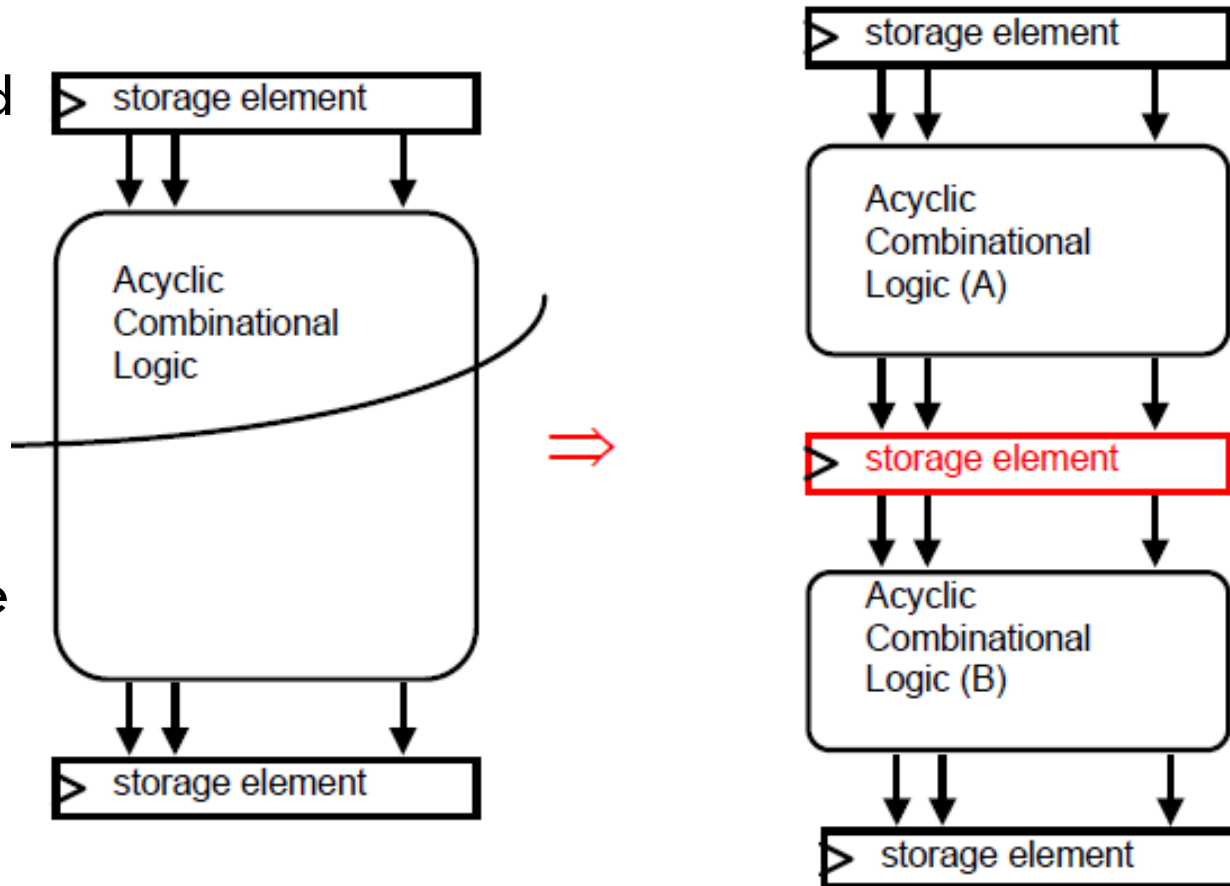# A Single Cycle Processor

Drawbacks of Single Cycle Processor

- Long cycle time: Cycle time must be long enough for the load instruction:
    - PC's Clock -to-Q +
    - Instruction Memory Access Time +
    - Register File Access Time +
    - ALU Delay (address calculation) +
    - Data Memory Access Time +
    - Register File Setup Time
- Cycle time is much longer than needed for all other instructions.
    - R-type instructions do not require data memory access
    - Jump does not require ALU operation nor data memory access

Overview of a Multiple Cycle Implementation

- The root of the single cycle processor's problems:
  - The cycle time has to be long enough for the slowest instruction
- Solution:
  - Break the instruction into smaller steps
  - Execute each step (instead of the entire instruction) in one cycle
    - Cycle time: time it takes to execute the longest step
    - Keep all the steps to have similar length
  - This is the essence of the multiple cycle processor

Reducing Cycle Time

- Cut combinational dependency graph and insert register / latch

- Do same work in two fast cycles, rather than one slow one

- May be able to short-circuit path and remove some components for some instructions!

Advantages of Multiple Cycle Processor

❑ Cycle time is much shorter

❑ Different instructions take different number of cycles to complete

- ❑ Load takes five cycles

- ❑ Jump only takes three cycles

❑ Allows a functional unit to be used more than once per instruction

The Big Picture: Performance Perspective

- Performance of a machine was determined by
  - Instruction Count
  - Clock cycle Time
  - Clock cycles per instruction
- Processor Design (data path and control) will determine
  - Clock cycle time
  - Clock cycles per instruction

The Five Steps of a Load Instruction
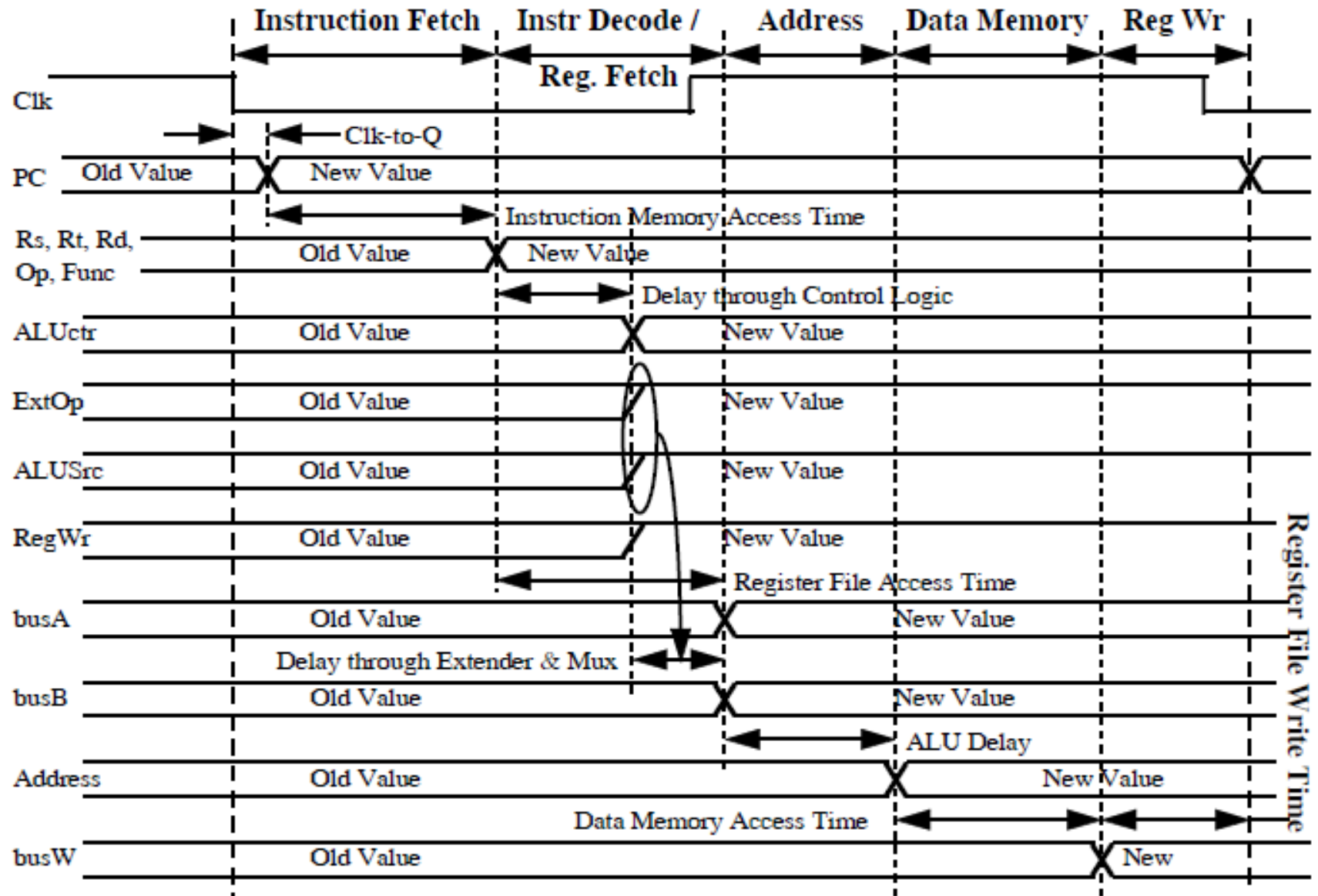
Memory Write Timing: Ideal Vs Reality

- In the Single Cycle Processor memory module is simplified
    - Write happens at the clock tick
    - Address, data, and write enable must be stable one "set-up" time before the clock tick
- In real life memory module has no clock input
    - The write path is a combinational logic delay path:
    - Write enable goes to 1 and Din settles down
    - Memory write access delay
    - Din is written into mem[address]
- Important: Address and Data must be stable BEFORE Write Enable goes to 1
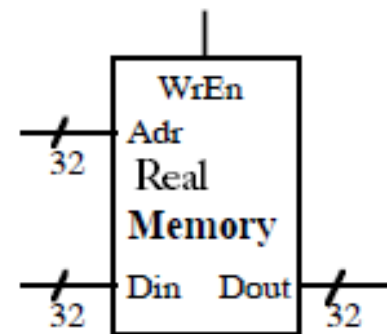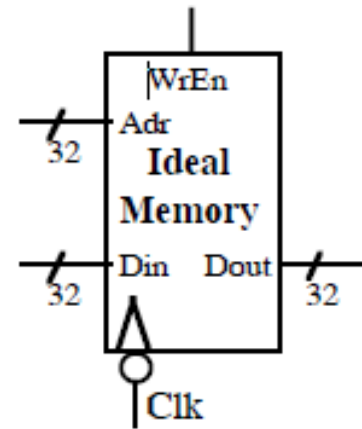
Race Condition Between Address and Write Enable

- The "real" (no clock input) memory may not work reliably in the single cycle processor because:
  - We cannot guarantee Address will be stable BEFORE WrEn = 1
  - There is a race between Adr and WrEn
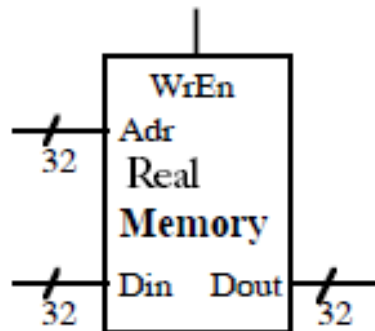
How to Avoid this Race Condition?

❑ Solution for the multiple cycle implementation:

    ❑ Make sure Address is stable by the end of Cycle N

    ❑ Assert Write Enable signal ONE cycle later at Cycle (N + 1)

    ❑ Address cannot change until Write Enable is deasserted.
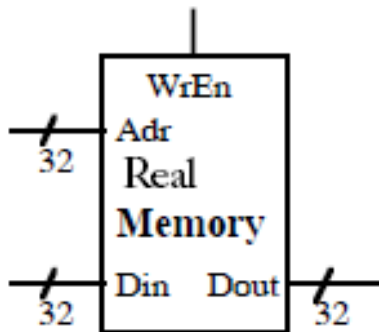
We can make use the same logic if we are using unclocked Register File to build our multiple cycle processor.

Multiple Cycle Data Path
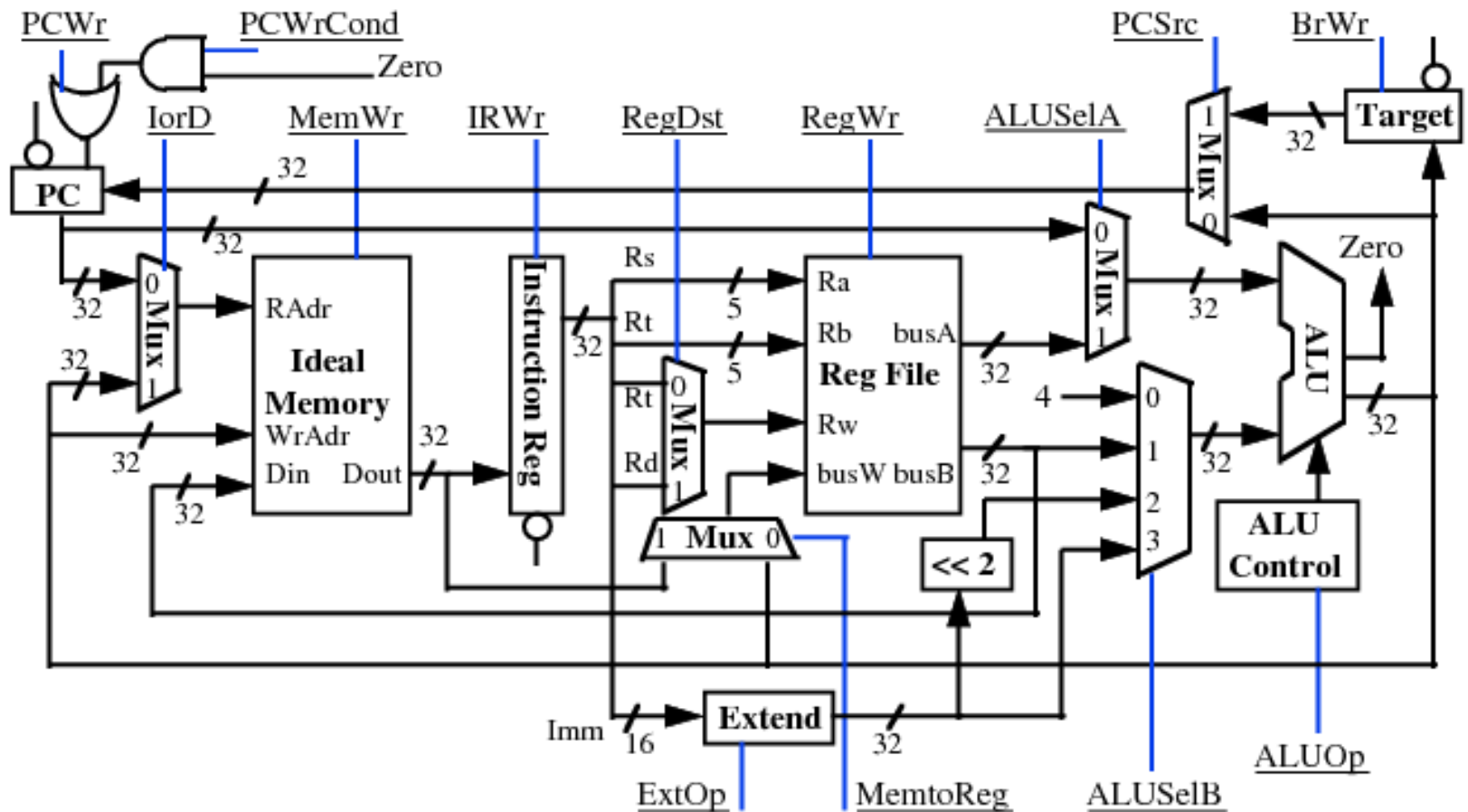
Multiple Cycle Data Path
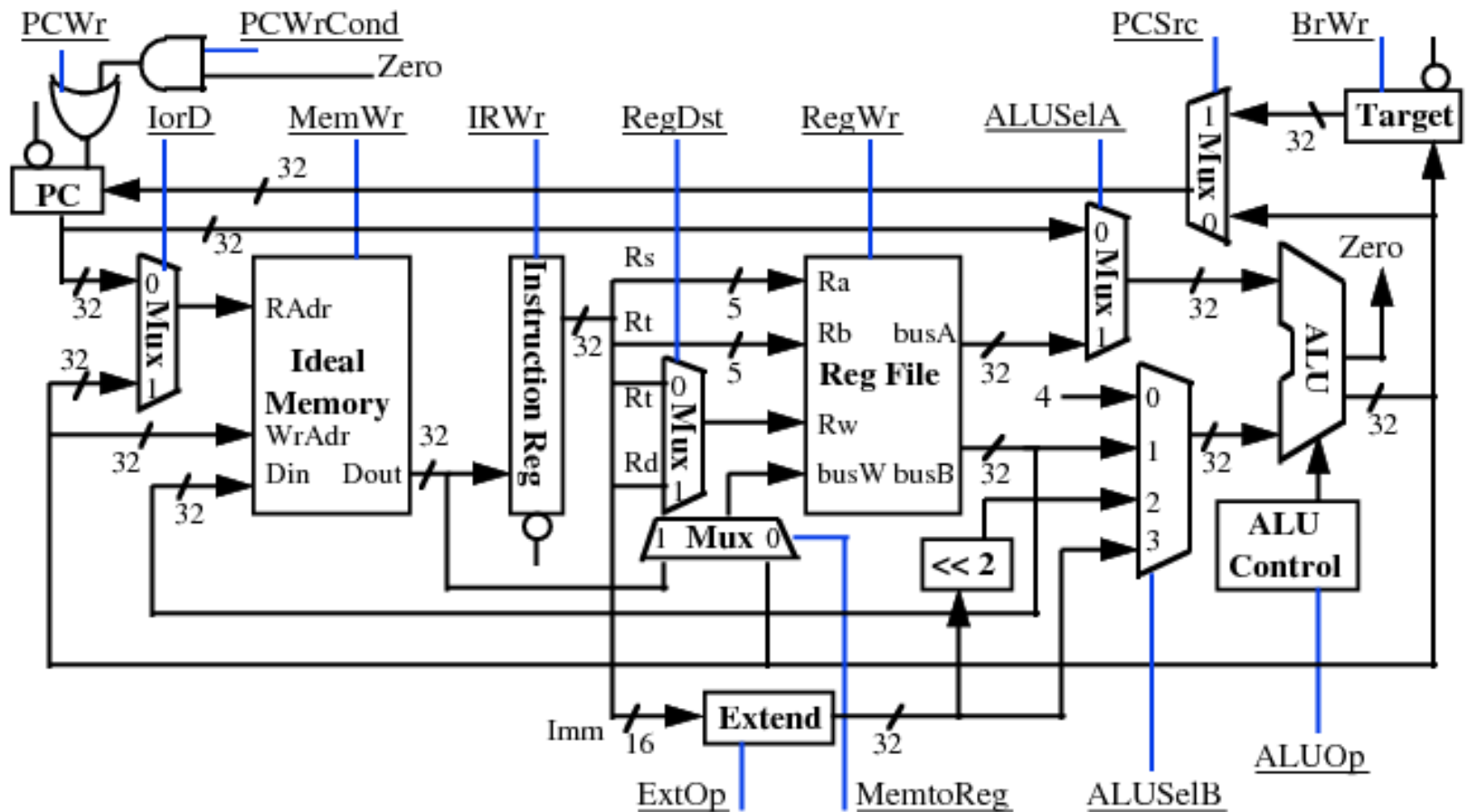
Instruction Fetch Phase

PCWr	IorD	MemWr	IrWr	RegDst	RegWr	ALUSelA
1	0	0	1	x	0	0
PCWrCond	**PCSrc**	**BrWr**	**ExtOp**	**MemtoReg**	**ALUOp**	**ALUSelB**
x	0	0	x	x	ADD	0

Ifetch

ALUOp=Add

1: PCWr, IRWr

x: PCWrCond RegDst, Mem2R

Others: 0s
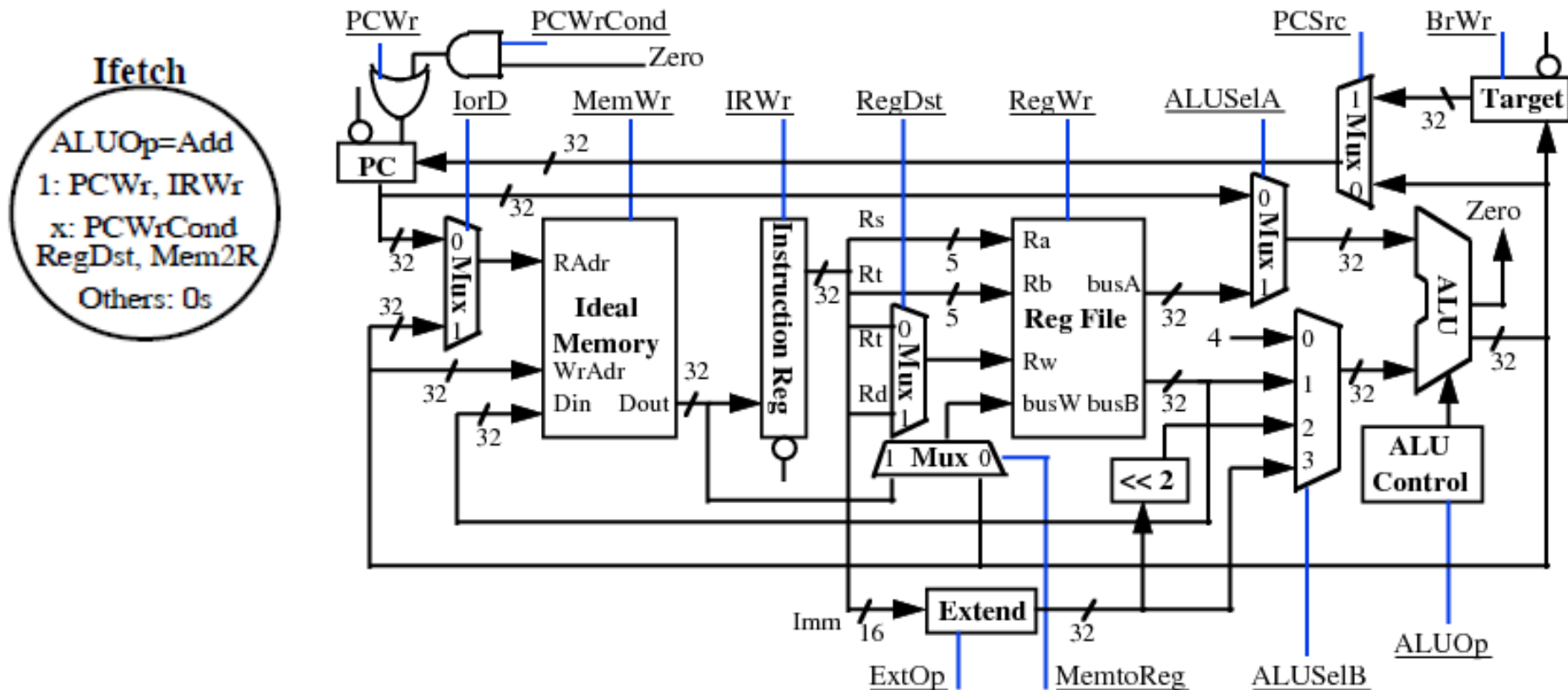
Instruction Fetch Cycle: In the Beginning

° **Every cycle begins right AFTER the clock tick:**

  • **mem[PC]   PC<31:0> + 4**

You are here!
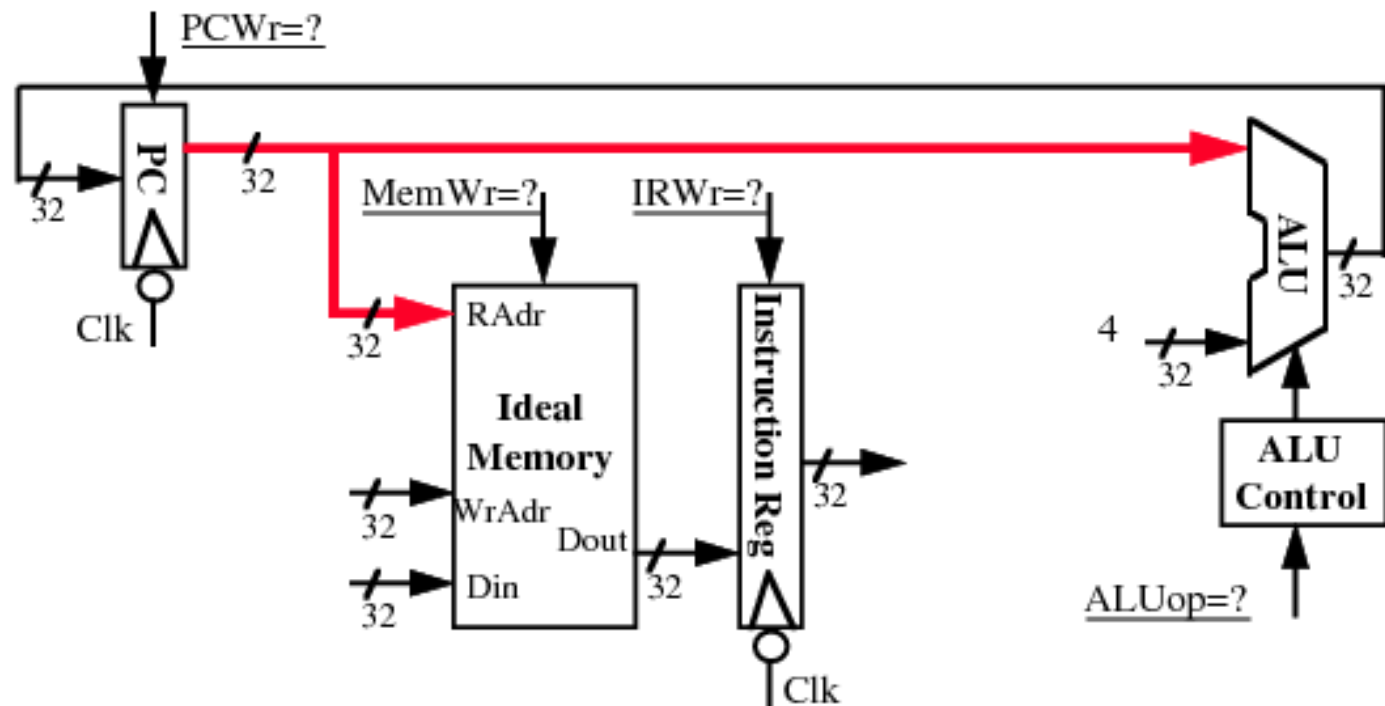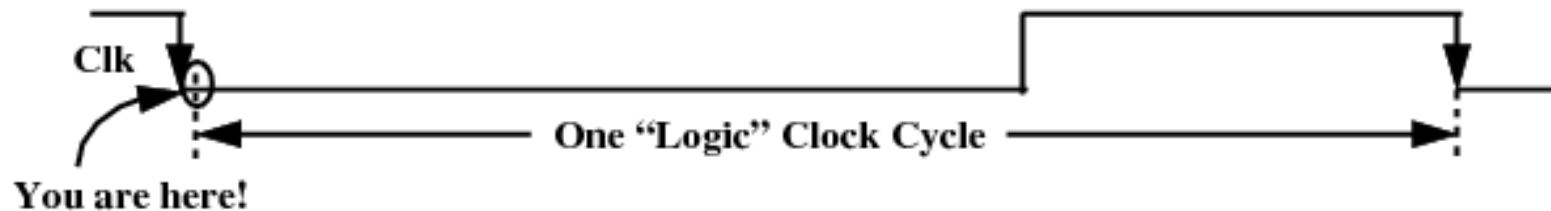
Instruction Fetch Cycle: The End

° **Every cycle ends AT the next clock tick (storage element updates):**

- IR <-- mem[PC]   PC<31:0> <-- PC<31:0> + 4
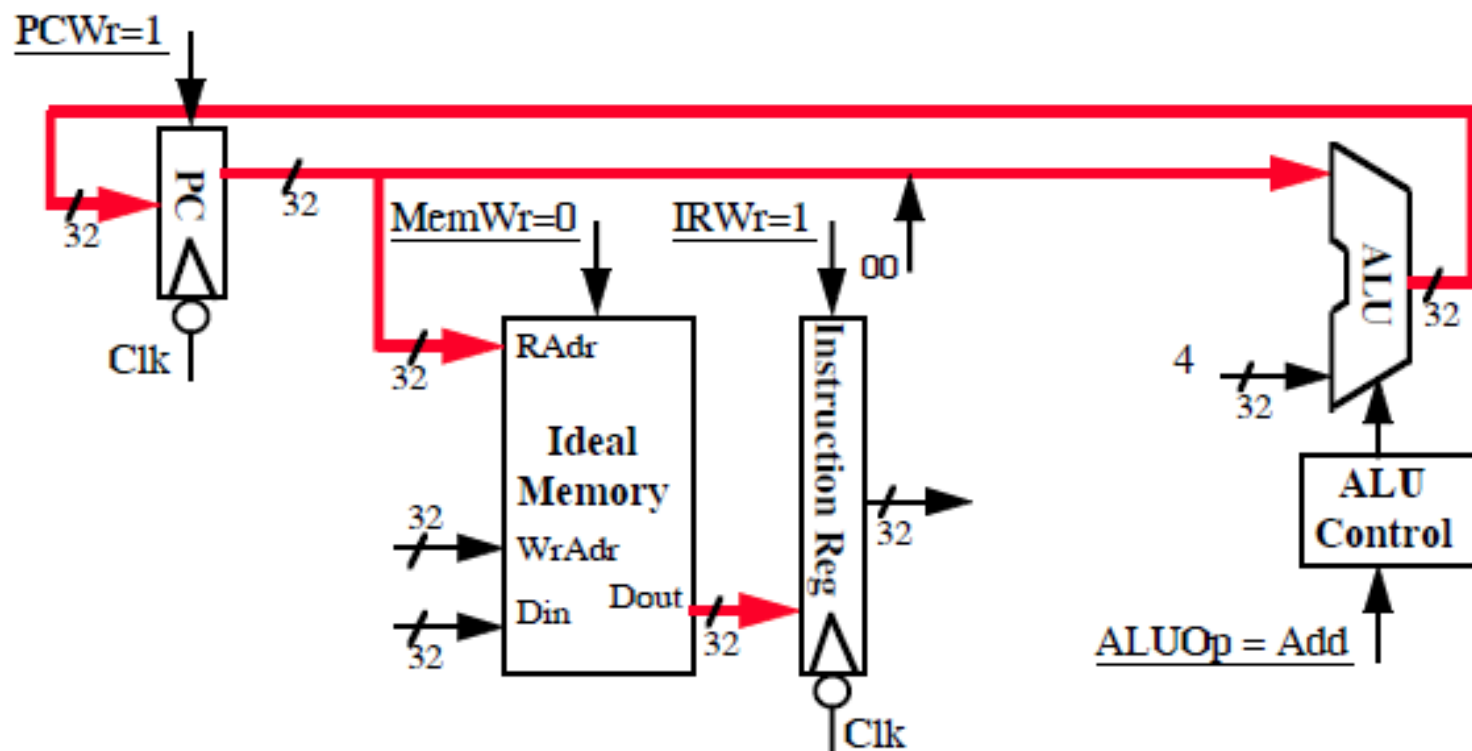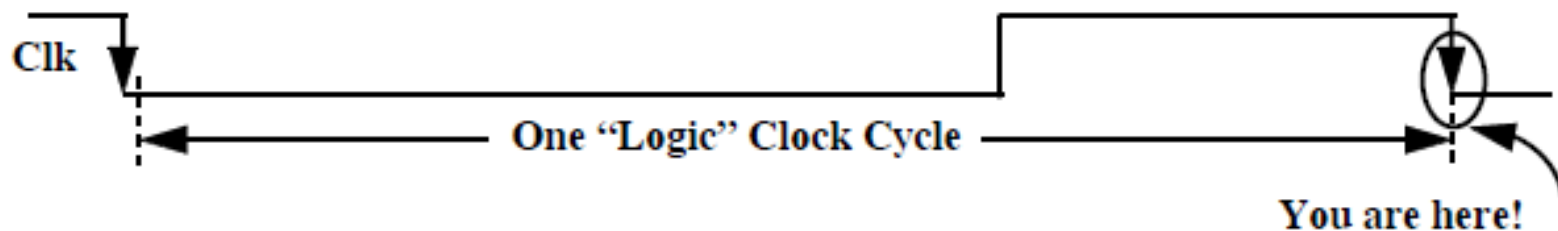
Instruction Fetch Cycle: Overall Picture

Ifetch

ALUOp=Add
1: PCWr, IRWr
x: PCWrCond
RegDst, Mem2R
Others: 0s

PCWr=1 PCWrCond=x
Zero

PCSrc=0 BrWr=0

IorD=0 MemWr=0 IRWr=1

ALUSelA=0

Target

PC

32

32

Mux 1

ALUSelB=00

ALUOp=Add

32

RAdr

Ideal
Memory

WrAdr

Din Dout

32

32

32

Instruction Reg

32

Mux 1

busA

32

4

busB 32

0
1
2
3

32

ALU

Zero

32

ALU
Control
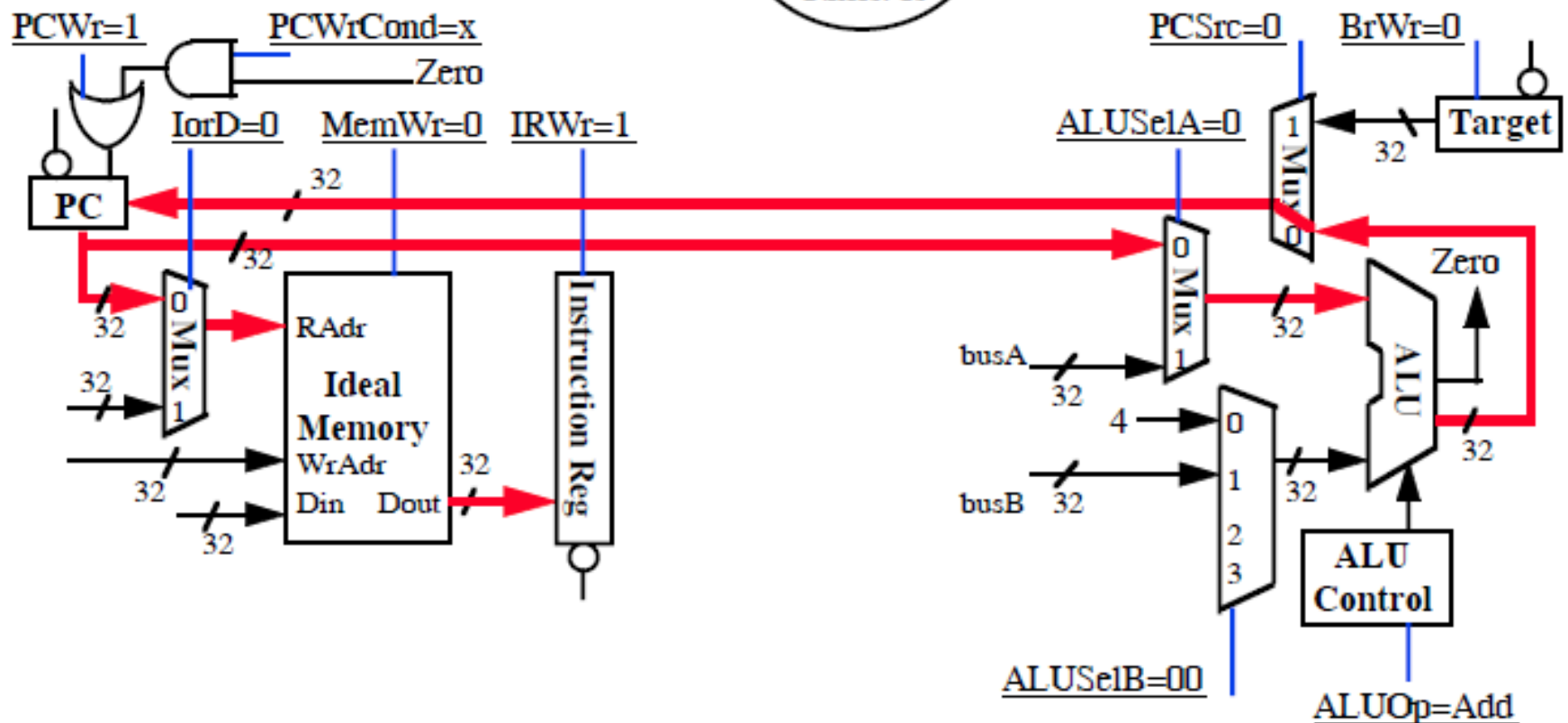
Register Fetch / Instruction Decode

° busA <- RegFile[rs] ; busB <- RegFile[rt] ;

° ALU is not being used: ALUctr = xx
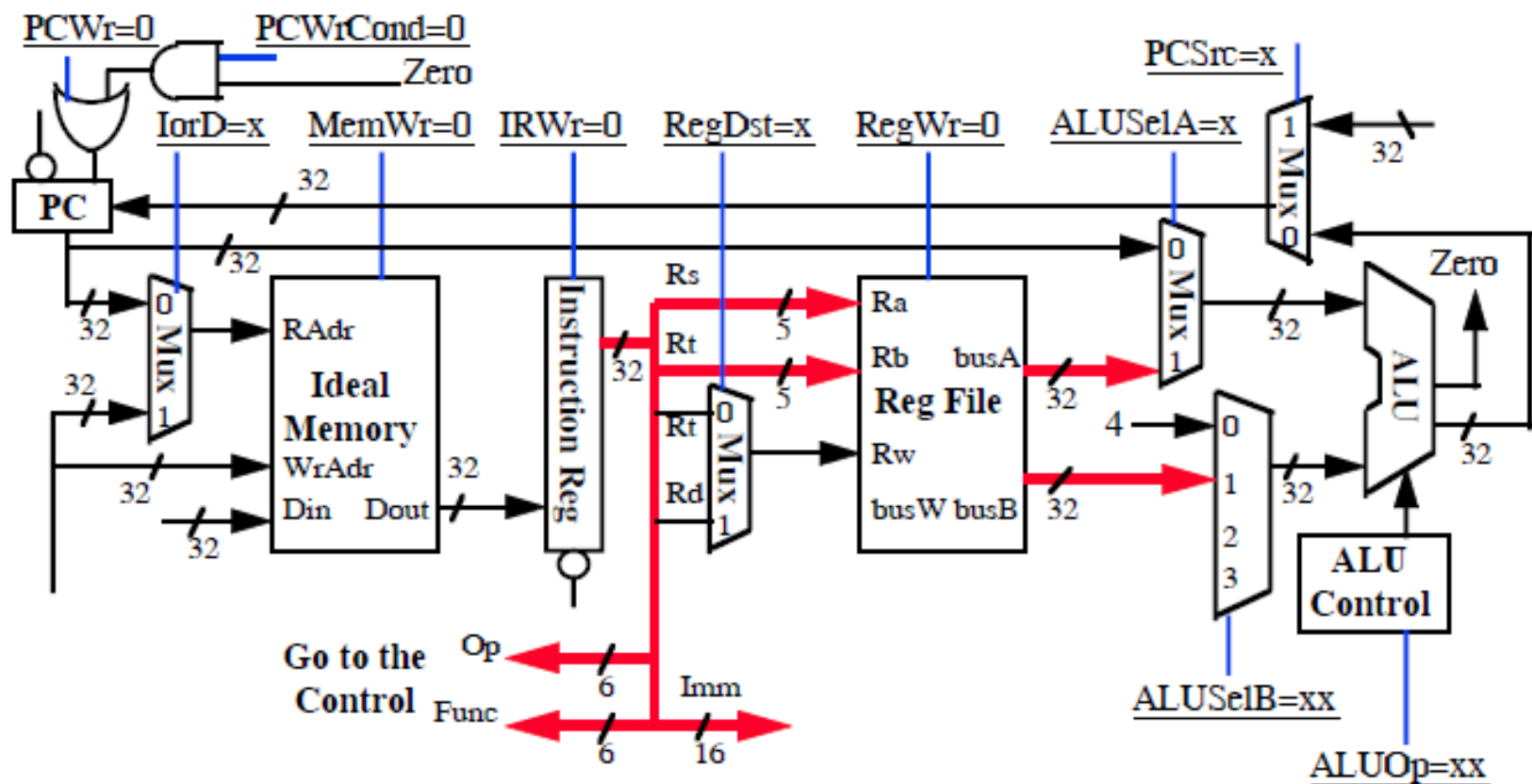
Register Fetch / Instruction Decode (Continue)

° **busA <- Reg[rs] ; busB <- Reg[rt] ;**

° **Target <- PC + SignExt(Imm16)*4**

Rfetch/Decode

ALUOp=Add

1: BrWr, ExtOp

ALUSelB=10

x: RegDst, PCSrc
IorD, MemtoReg

Others: 0s

PCWr=0   PCWrCond=0

Zero

PCSrc=x   BrWr=1

IorD=x   MemWr=0   IRWr=0   RegDst=x   RegWr=0   ALUSelA=0

PC

32

32

Target

32

Zero

0 Mux 1

32

32

RAdr

**Ideal
Memory**

WrAdr

Din   Dout

32

32

32

32

Instruction Reg

Rs

Rt

32

Rt

Rd

0 Mux 1

5

5

Ra

Rb   busA

Reg File

Rw

busW busB

32

32

1 Mux 0

0 Mux 1

32

ALU

4

0

1

2

3

32

32

<< 2

**ALU
Control**

Beq

Rtype

Ori

Memory

Control

Op

Func

6

6

Imm

16

Extend

32

ALUSelB=10

ALUOp=Add

:
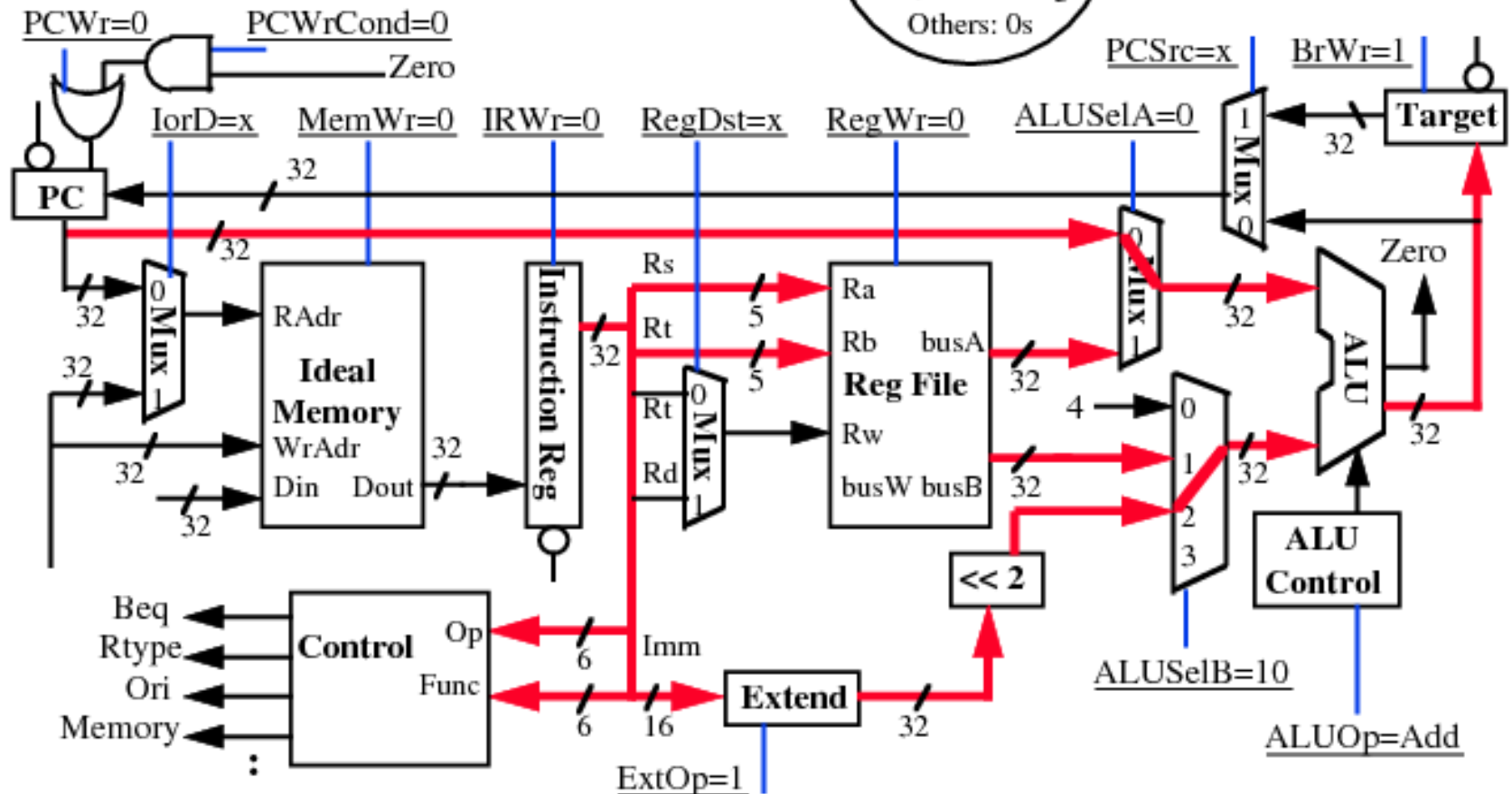
ExtOp=1

Branch Completion

° **if (busA == busB)**

  • **PC <- Target**

BrComplete

ALUOp=Sub

ALUSelB=01

x: IorD, Mem2Reg
RegDst, ExtOp

1: PCWrCond
ALUSelA
PCSrc
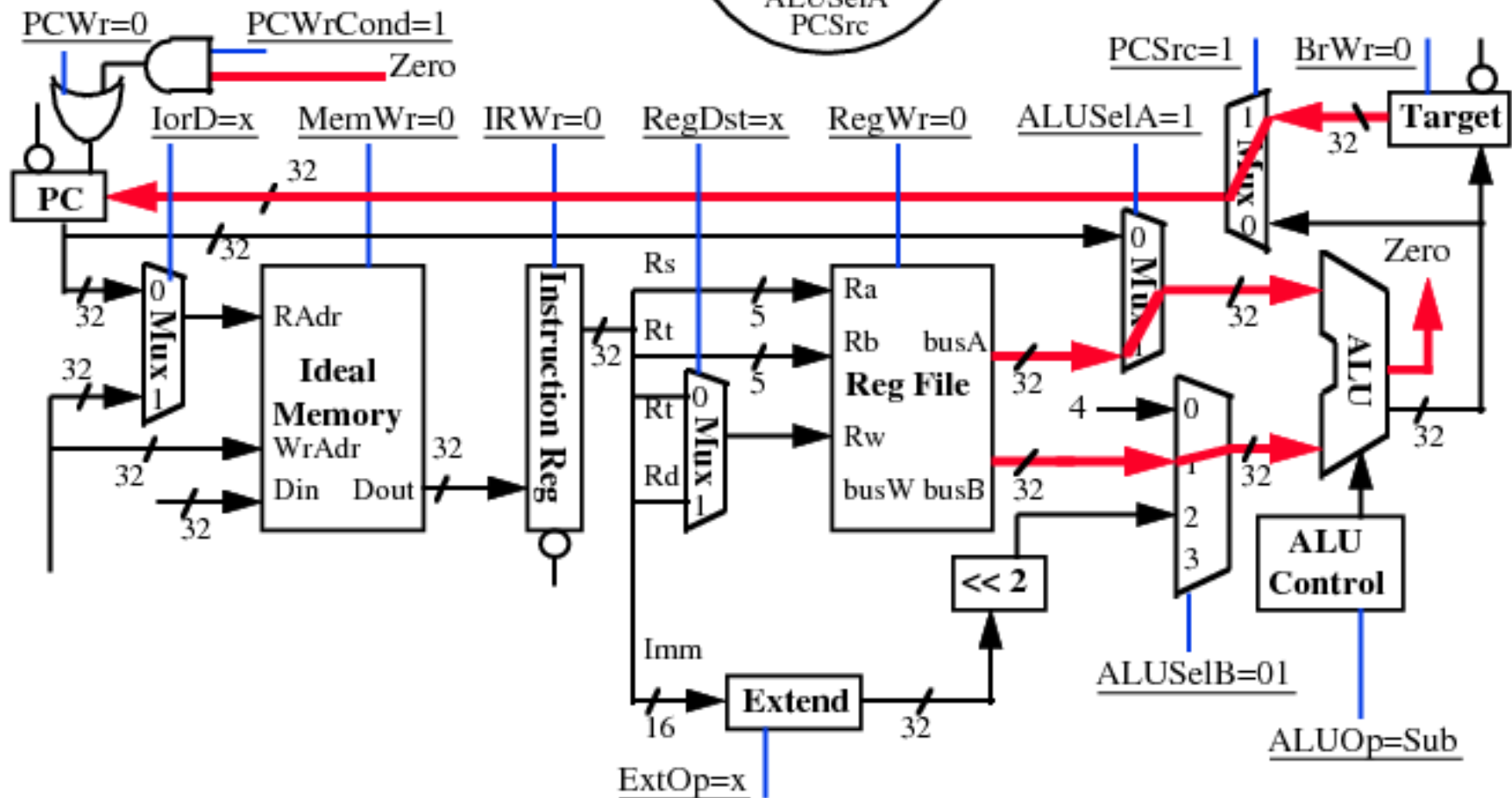
Instruction Decode: We have a R-type!

° **Next Cycle: R-type Execution**
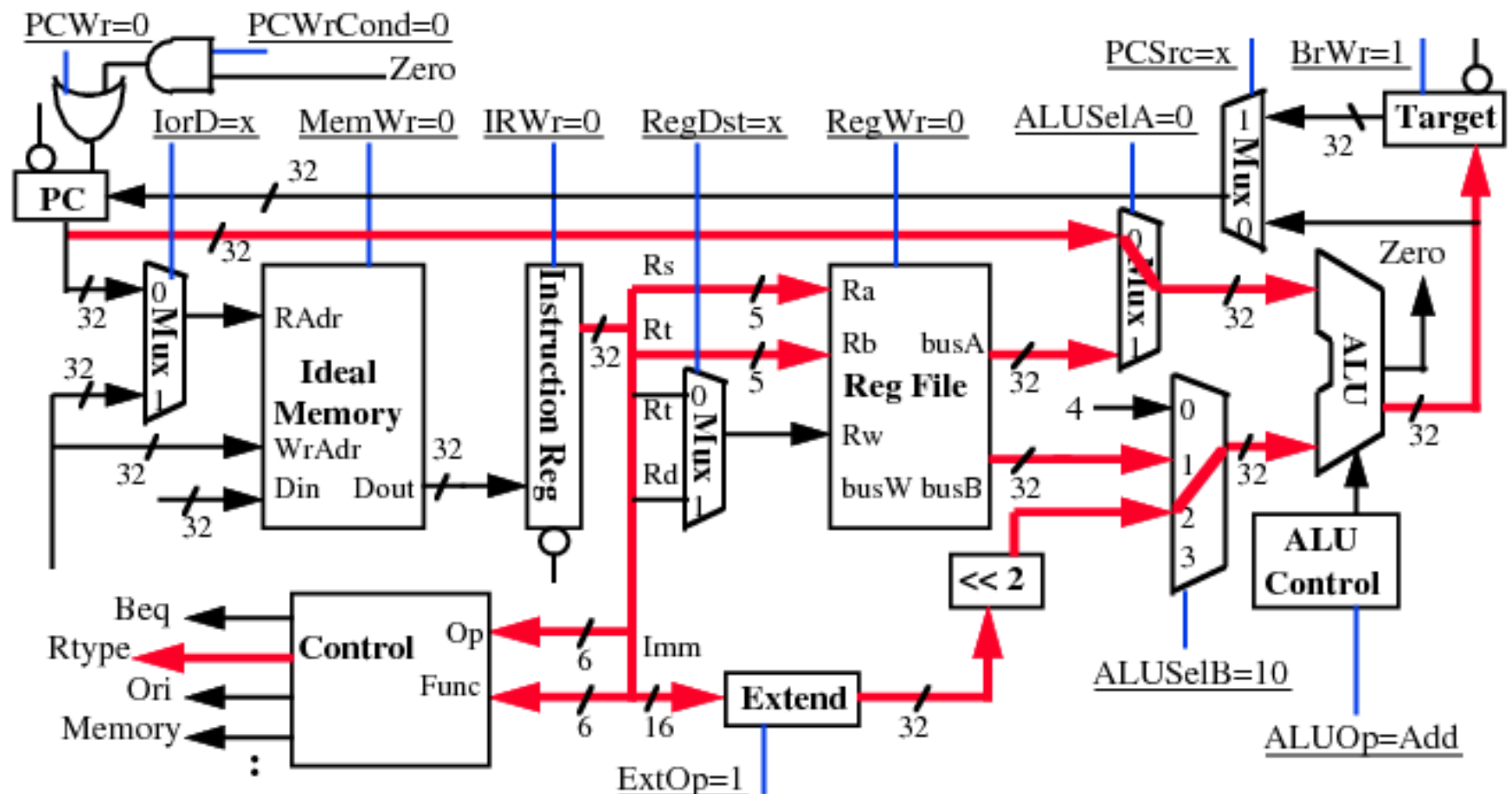
R-type Execution

° **ALU Output <- busA op busB**
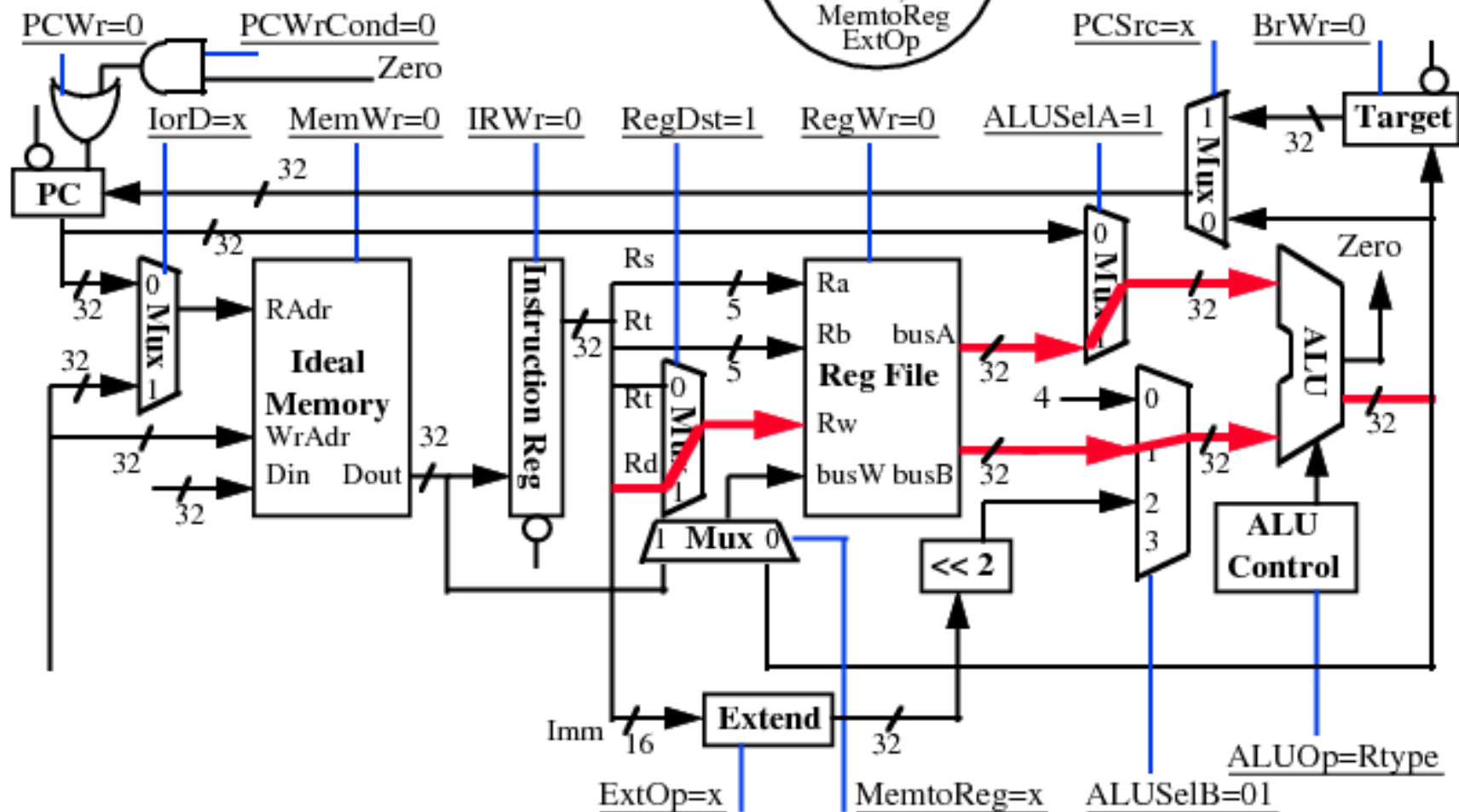
R-type Completion

° R[rd] <- ALU Output

Rfinish

ALUOp=Rtype
1: RegDst, RegWr
ALUselA
ALUSelB=01
x: IorD, PCSrc
ExtOp

PCWr=0   PCWrCond=0

Zero

IorD=x   MemWr=0   IRWr=0   RegDst=1   RegWr=1   ALUSelA=1   PCSrc=x   BrWr=0

Target

32

PC

32

32

Mux 0
1

32

RAdr

32

Mux 1
0

Ideal
Memory

WrAdr

Din   Dout

32

32

Instruction Reg

Rs

Rt

32

5

5

Mux 0
1

Rt

Rd

Ra

Rb   busA

Reg File

Rw

busW busB

32

Mux 1
0

32

32

Zero

ALU

32

4

0
1
2
3

32

ALU
Control

<< 2

Imm   16   Extend   32

ExtOp=x   MemtoReg=0   ALUSelB=01   ALUOp=Rtype

Mux
1   0
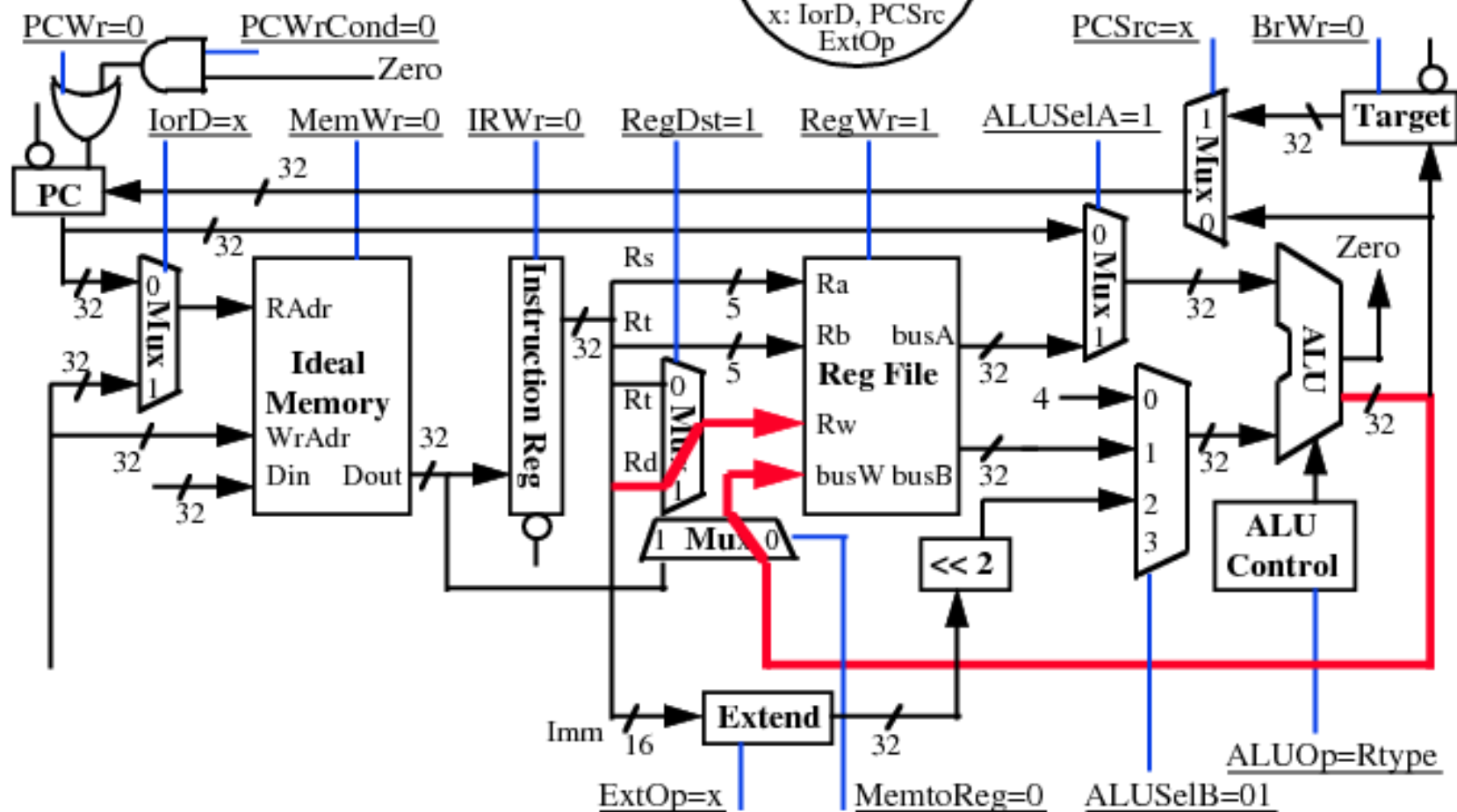
Instruction Decode: We have an Ori!

° **Next Cycle: Ori Execution**
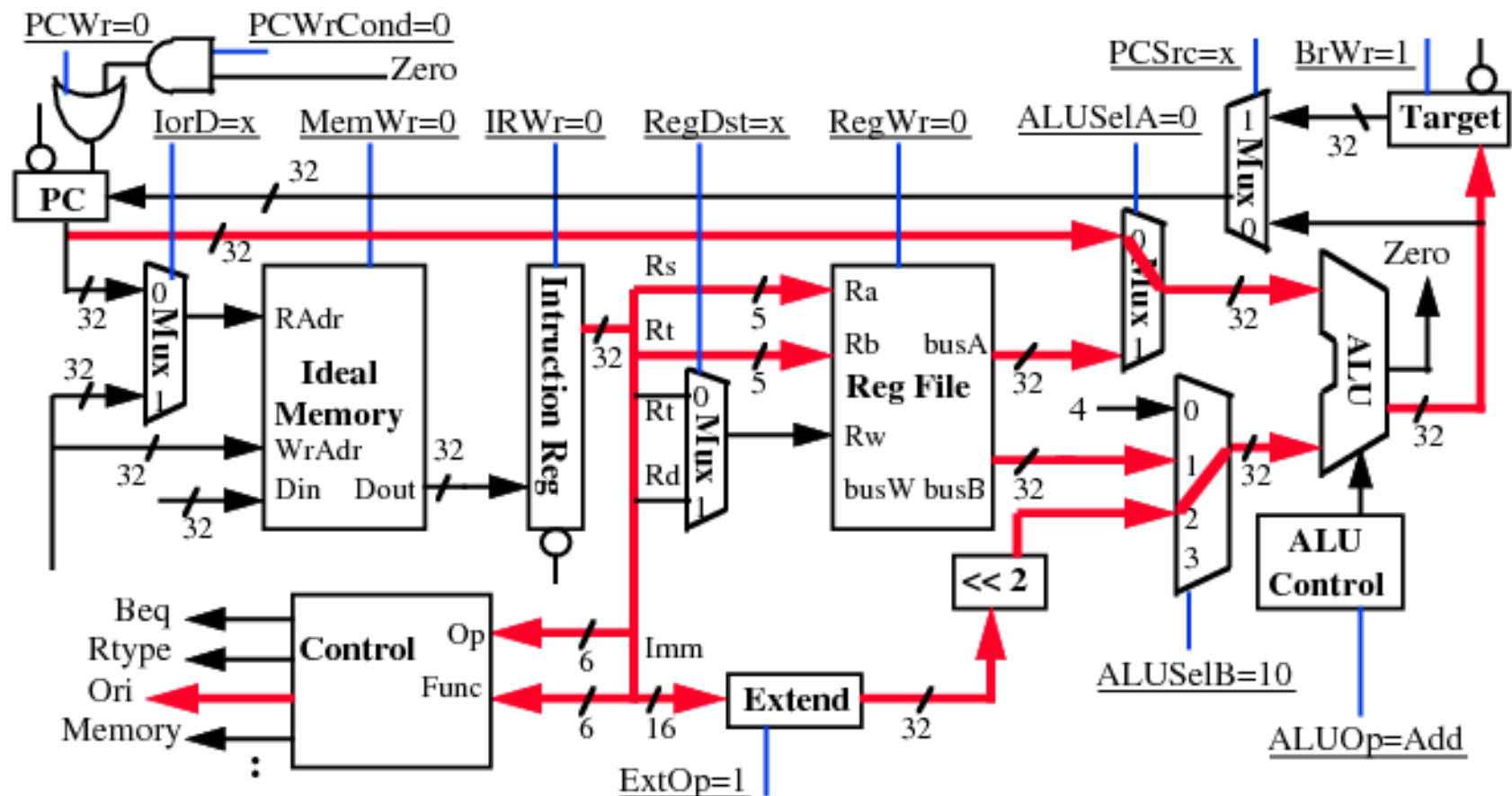
Ori Execution

° **ALU output <- busA or ZeroExt[Imm16]**
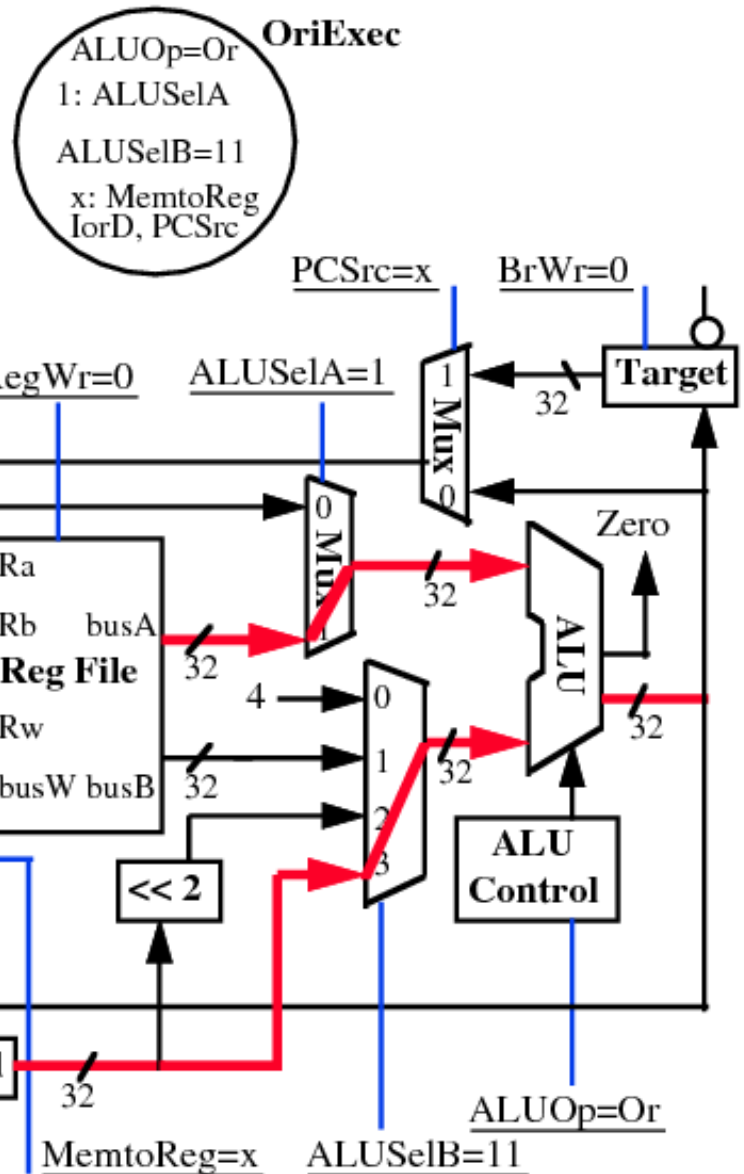
Ori Completion

° **Reg[rt] <- ALU output**

OriFinish

ALUOp=Or

x: IorD, PCSrc
ALUSelB=11
1: ALUSelA
RegWr
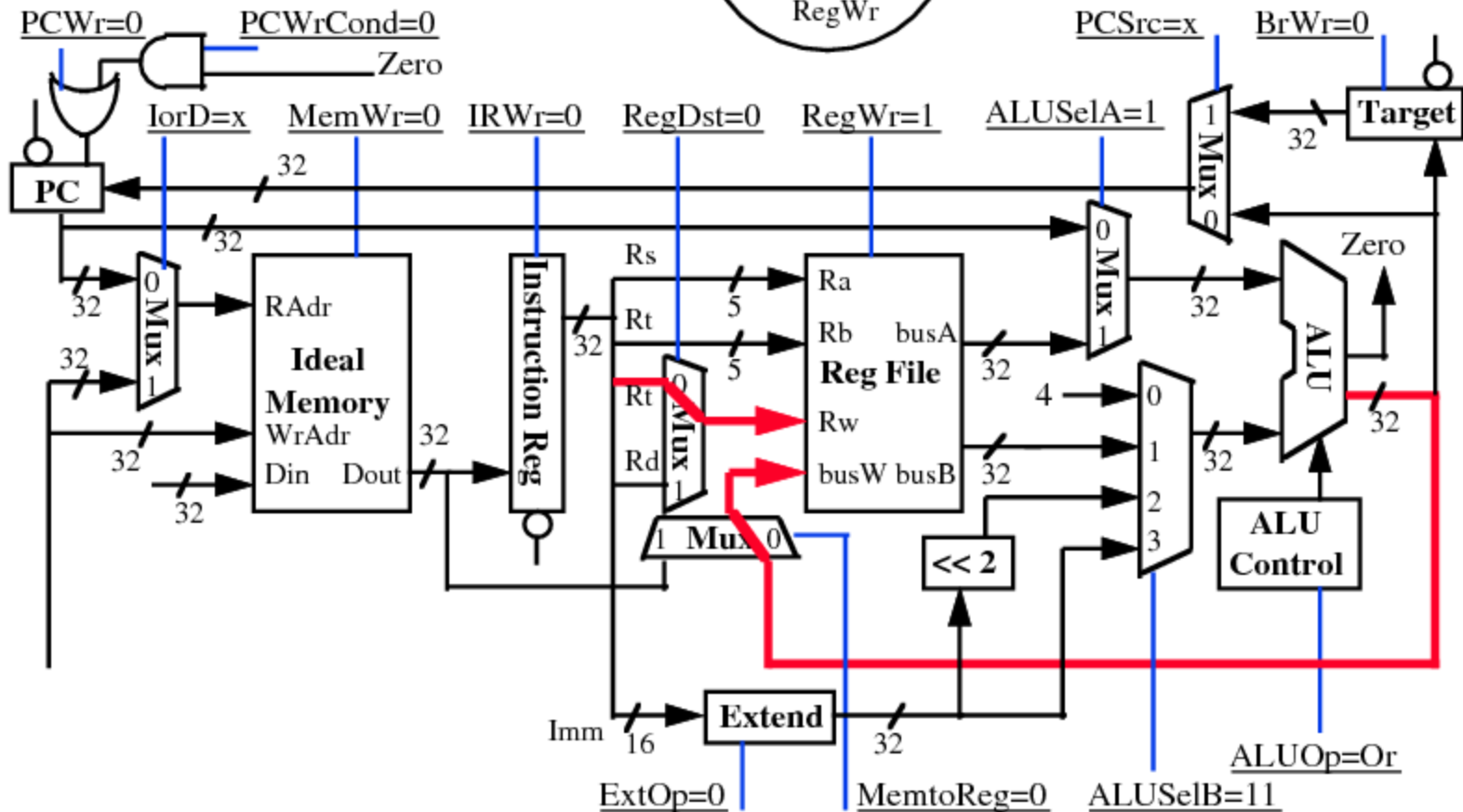
Instruction Decode: We have a Memory Access!

° **Next Cycle: Memory Address Calculation**
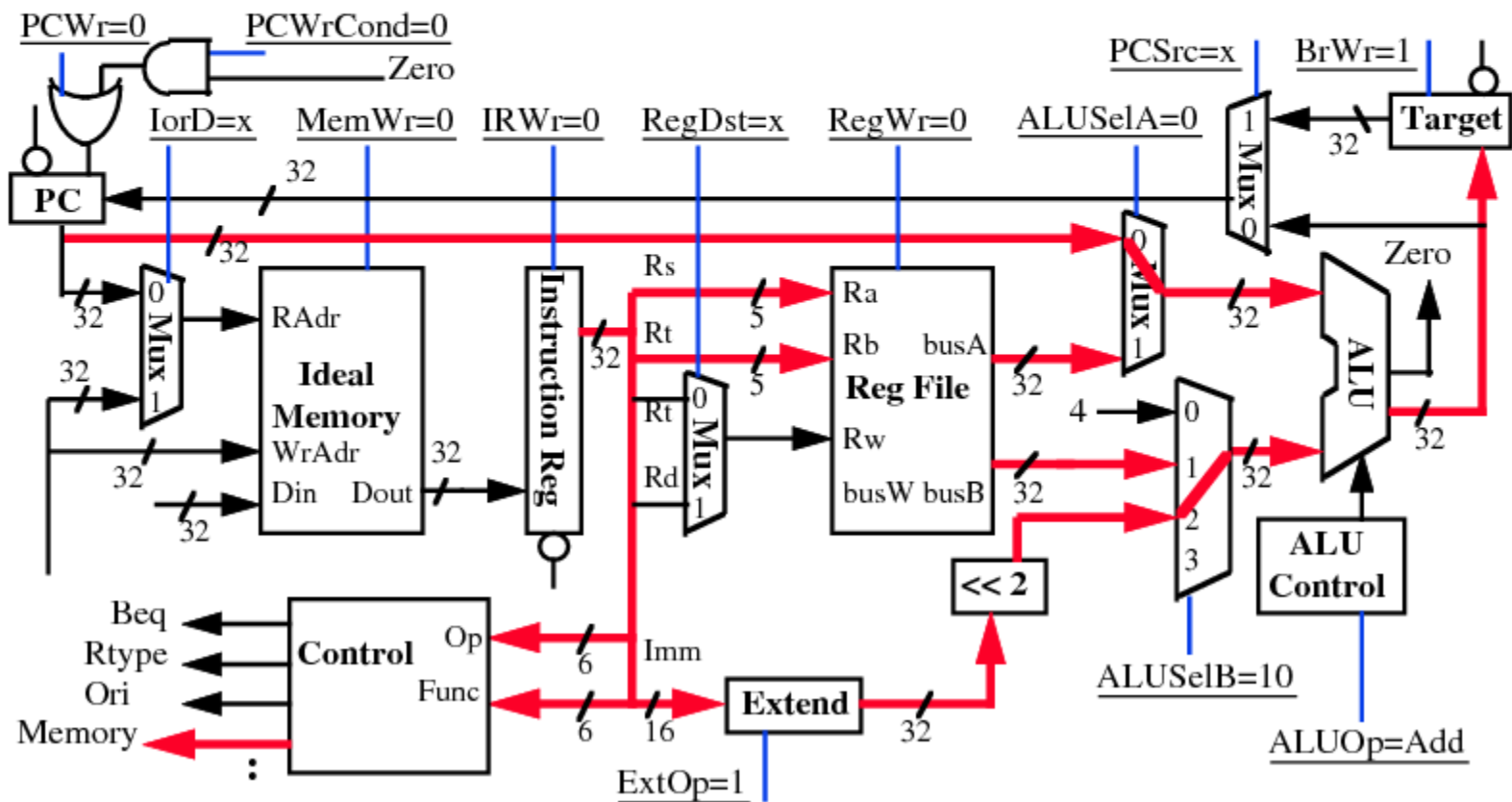
Memory Address Calculation

○ **ALU output <- busA + SignExt[Imm16]**

AdrCal

1: ExtOp
ALUSelA
ALUSelB=11
ALUOp=Add
x: MemtoReg
PCSrc
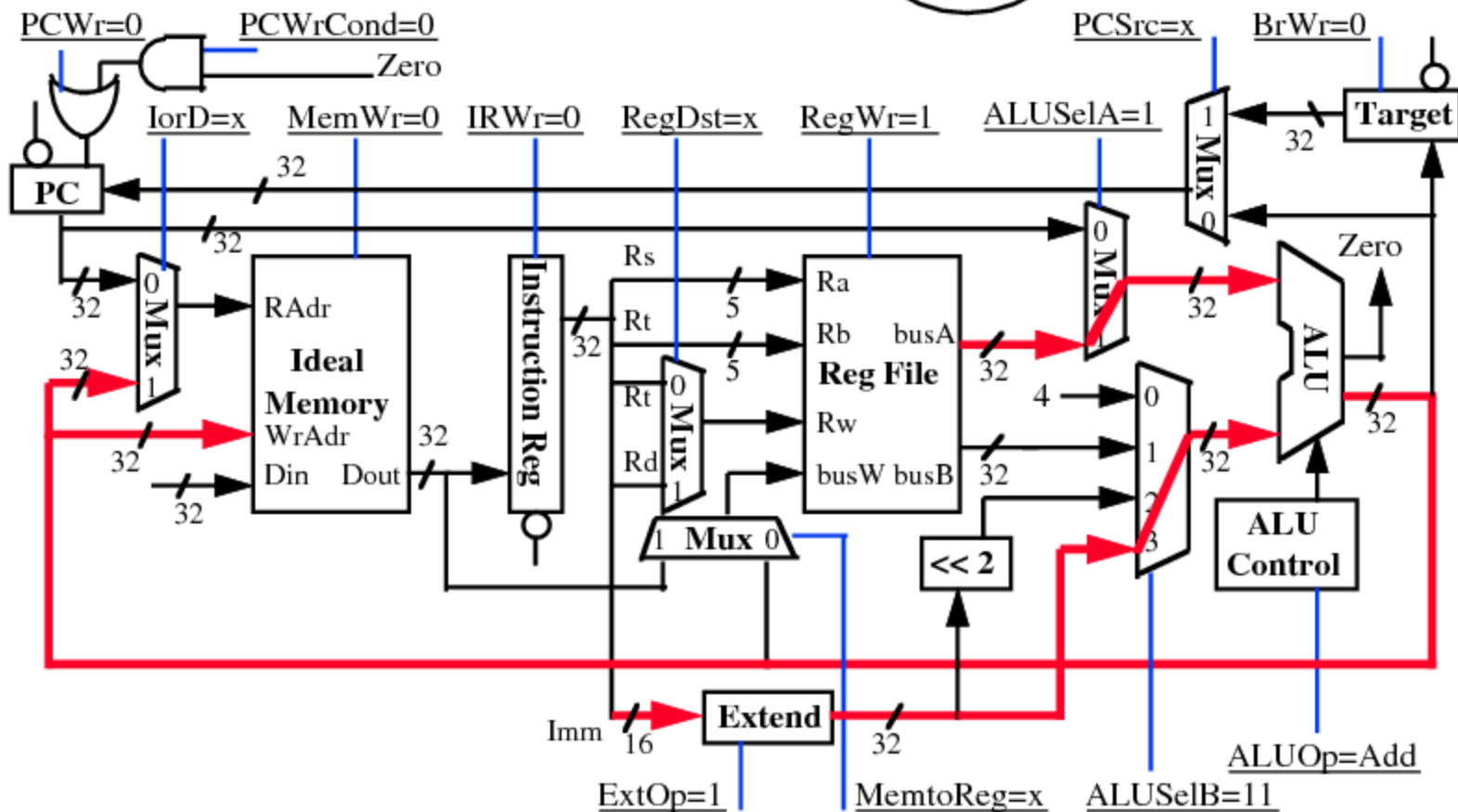
Memory Access for Load

° **Mem Dout <- mem[ALU output]**
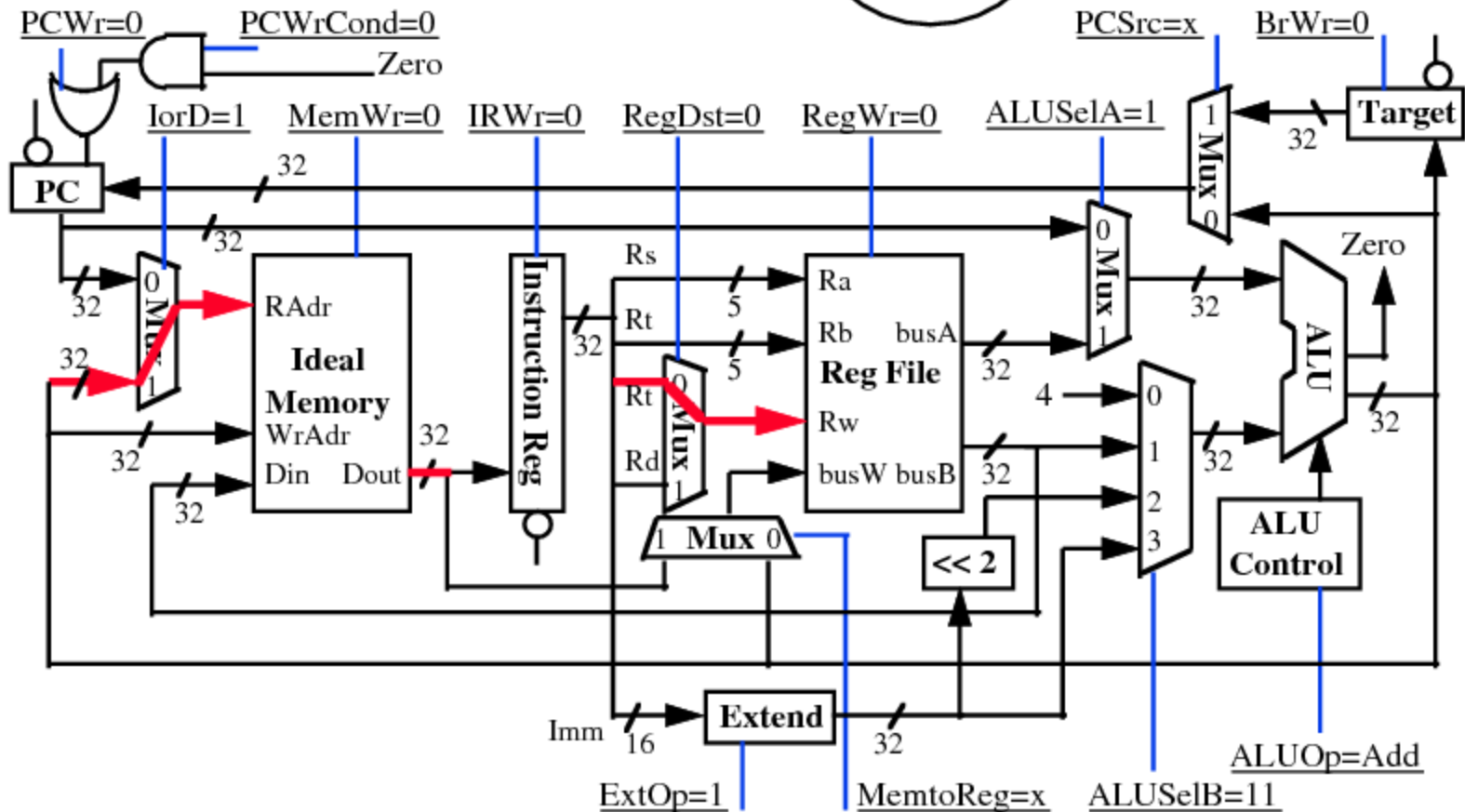
Write Back for Load

° **Reg[rt] <- Mem Dout**

LWwr

1: ALUSelA
RegWr, ExtOp
MemtoReg

ALUSelB=11
ALUOp=Add

x: PCSrc
IorD

PCWr=0   PCWrCond=0
                    Zero

IorD=x   MemWr=0   IRWr=0   RegDst=0   RegWr=0   ALUSelA=1   PCSrc=x   BrWr=0

PC   32   Target   32

Rs
Rt
Ra
Rb   busA

Ideal
Memory
RAdr

Instruction Reg

Rt
Rd
Rw
busW busB

Reg File

Mux

WrAdr
Din   Dout

<< 2

ALU
Control

ALU   Zero

Imm   Extend   32

ExtOp=1   MemtoReg=1   ALUSelB=11   ALUOp=Add
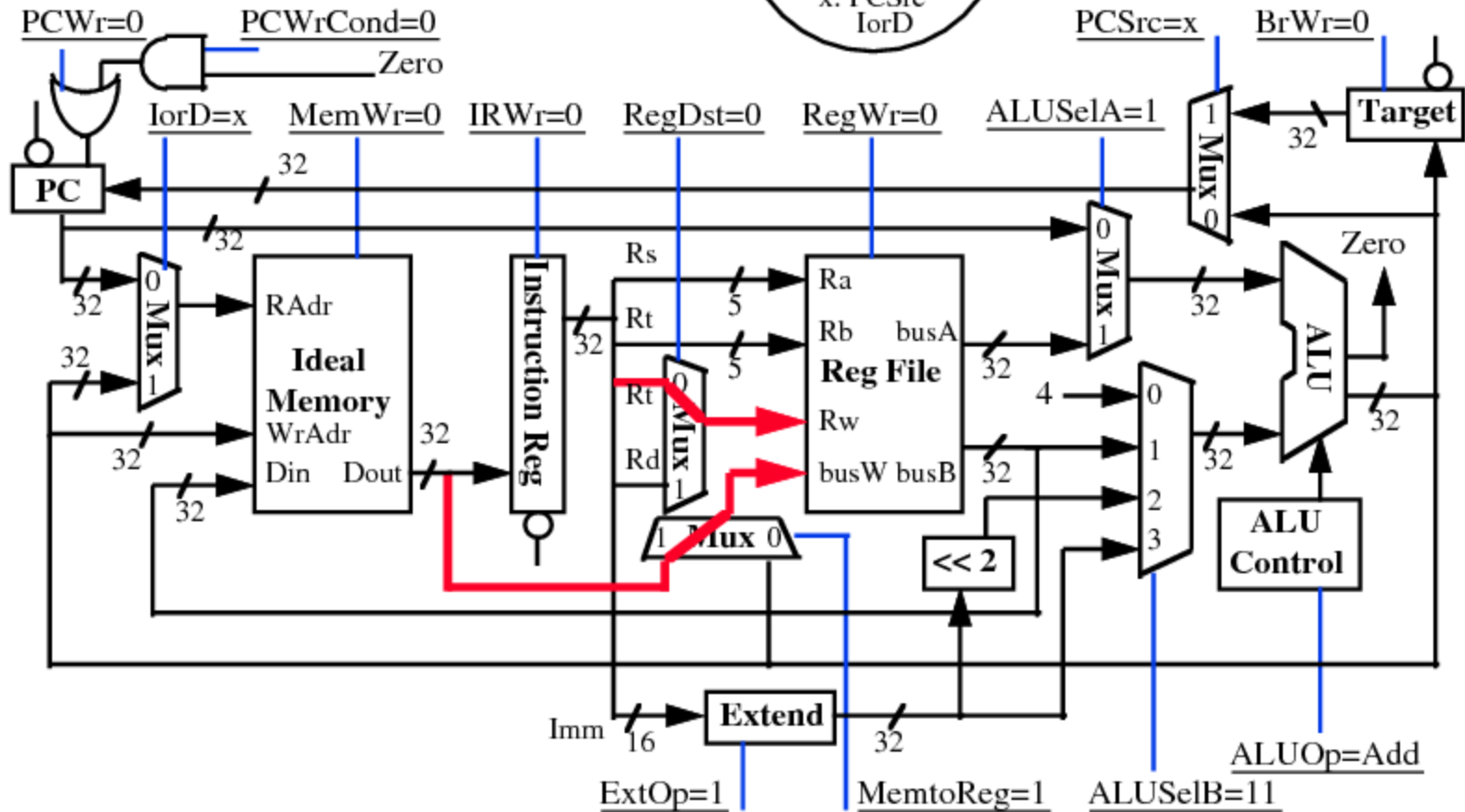
Memory Access for Store

° **mem[ALU output] <- busB**

SWmem
- 1: ExtOp
- MemWr
- ALUSelA
- ALUSelB=11
- ALUOp=Add
- x: PCSrc,RegDst
- MemtoReg

PCWr=0   PCWrCond=0
Zero

PCSrc=x   BrWr=0

IorD=x   MemWr=1   IRWr=0   RegDst=x   RegWr=0   ALUSelA=1

Target

PC   32

32

Mux 0 / Mux 1

32

Rs   Ra
Rt   Rb   busA
Rt   Rw
Rd   busW busB

Ideal Memory
RAdr
WrAdr
Din   Dout

Instruction Reg

Reg File

Mux   0 Mux 1

ALU   Zero

ALU Control

<< 2

Extend
Imm   16   32

ExtOp=1   MemtoReg=x   ALUSelB=11   ALUOp=Add

5   5   4   0 1 2 3   32
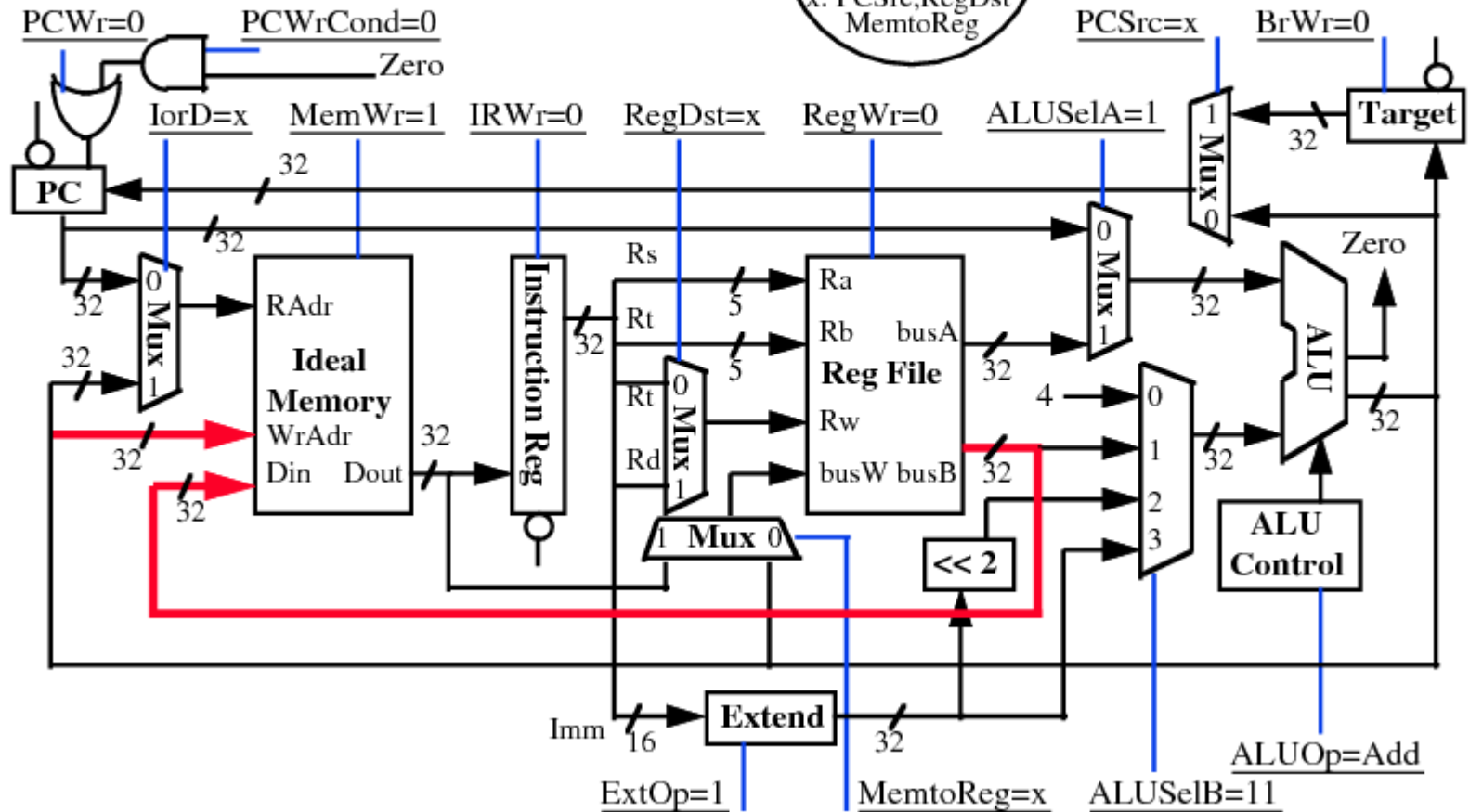
Putting it all together: Multiple Cycle Datapath
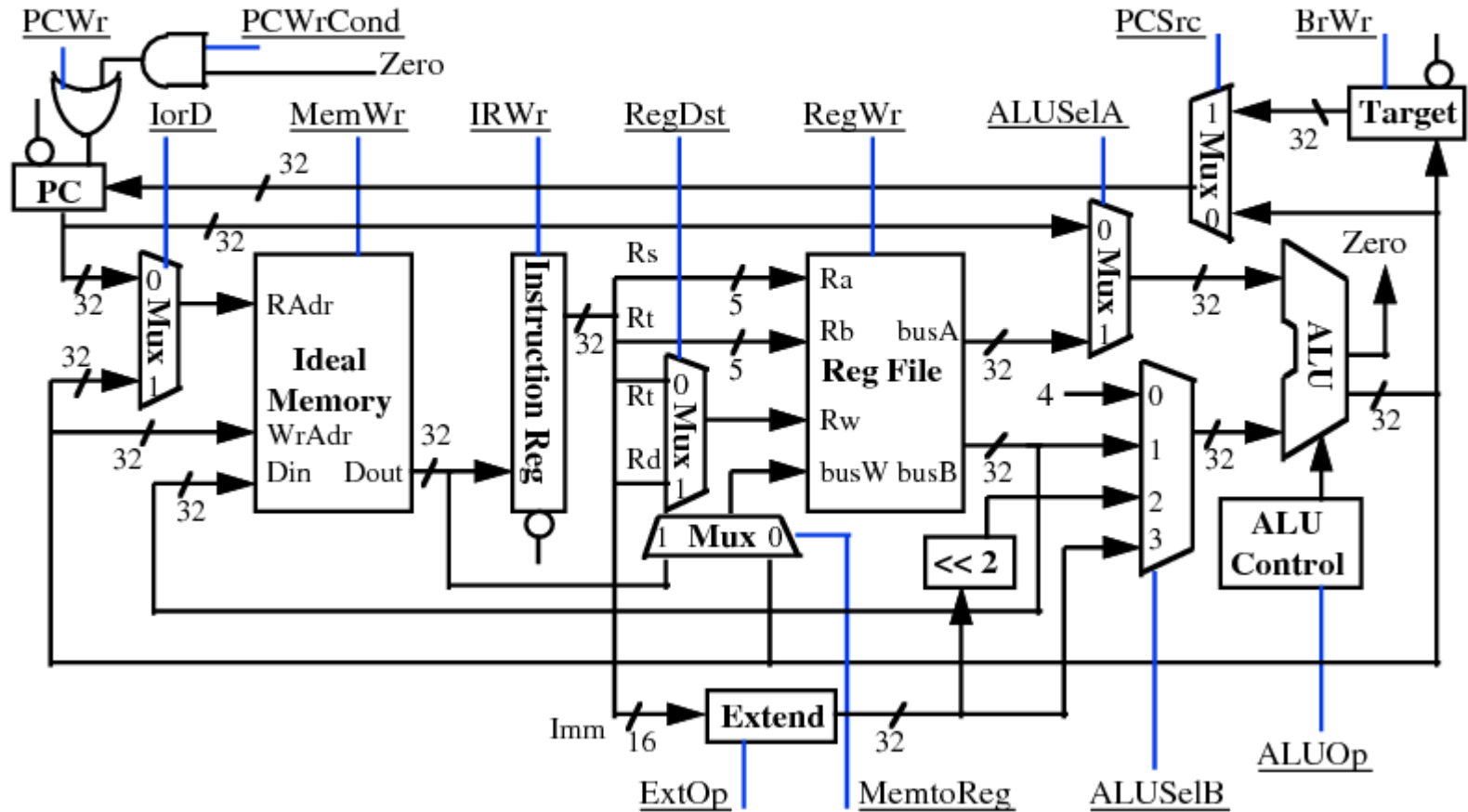
Putting it all together: Control State Diagram

Ifetch
ALUOp=Add
1: PCWr, IRWr
x: PCWrCond
RegDst, Mem2R
Others: 0s

Rfetch/Decode
ALUOp=Add
1: BrWr, ExtOp
ALUSelB=10
x: RegDst, PCSrc
IorD, MemtoReg
Others: 0s

BrComplete
ALUOp=Sub
ALUSelB=01
x: IorD, Mem2Reg
RegDst, ExtOp
1: PCWrCond
ALUSelA
PCSrc

beq

AdrCal
1: ExtOp
ALUSelA
ALUSelB=11
ALUOp=Add
x: MemtoReg
PCSrc

lw or sw

lw

sw

Rtype

Ori

LWmem
1: ExtOp
ALUSelA, IorD
ALUSelB=11
ALUOp=Add
x: MemtoReg
PCSrc

SWMem
1: ExtOp
MemWr
ALUSelA
ALUSelB=11
ALUOp=Add
x: PCSrc,RegDst
MemtoReg

RExec
1: RegDst
ALUSelA
ALUSelB=01
ALUOp=Rtype
x: PCSrc, IorD
MemtoReg
ExtOp

OriExec
ALUOp=Or
1: ALUSelA
ALUSelB=11
x: MemtoReg
IorD, PCSrc

LWwr
1: ALUSelA
RegWr, ExtOp
MemtoReg
ALUSelB=11
ALUOp=Add
x: PCSrc
IorD

Rfinish
ALUOp=Rtype
1: RegDst, RegWr
ALUselA
ALUSelB=01
x: IorD, PCSrc
ExtOp

OriFinish
ALUOp=Or
x: IorD, PCSrc
ALUSelB=11
1: ALUSelA
RegWr

Control Logic in the Form of Finite State Diagram

0 Ifetch
ALUOp=Add
1: PCWr, IRWr
x: PCWrCond
RegDst, Mem2R
Others: 0s

1 Rfetch/Decode
ALUOp=Add
1: BrWr, ExtOp
ALUSelB=10
x: RegDst, PCSrc
IorD, MemtoReg
Others: 0s

8 BrComplete
ALUOp=Sub
ALUSelB=01
x: IorD, Mem2Reg
RegDst, ExtOp
1: PCWrCond
ALUSelA
PCSrc

2 AdrCal
1: ExtOp
ALUSelA
ALUSelB=11
ALUOp=Add
x: MemtoReg
PCSrc

lw or sw

Ori

Rtype

10 OriExec
ALUOp=Or
1: ALUSelA
ALUSelB=11
x: MemtoReg
IorD, PCSrc

lw

sw

3 LWmem
1: ExtOp
ALUSelA, IorD
ALUSelB=11
ALUOp=Add
x: MemtoReg
PCSrc

5 SWMem
1: ExtOp
MemWr
ALUSelA
ALUSelB=11
ALUOp=Add
x: PCSrc,RegDst
MemtoReg

6 RExec
1: RegDst
ALUSelA
ALUSelB=01
ALUOp=Rtype
x: PCSrc, IorD
MemtoReg
ExtOp

11 OriFinish
ALUOp=Or
x: IorD, PCSrc
ALUSelB=11
1: ALUSelA
RegWr

4 LWwr
1: ALUSelA
RegWr, ExtOp
MemtoReg
ALUSelB=11
ALUOp=Add
x: PCSrc
IorD

7 Rfinish
ALUOp=Rtype
1: RegDst, RegWr
ALUselA
ALUSelB=01
x: IorD, PCSrc
ExtOp

Sequencing Control: Explicit Next State Function
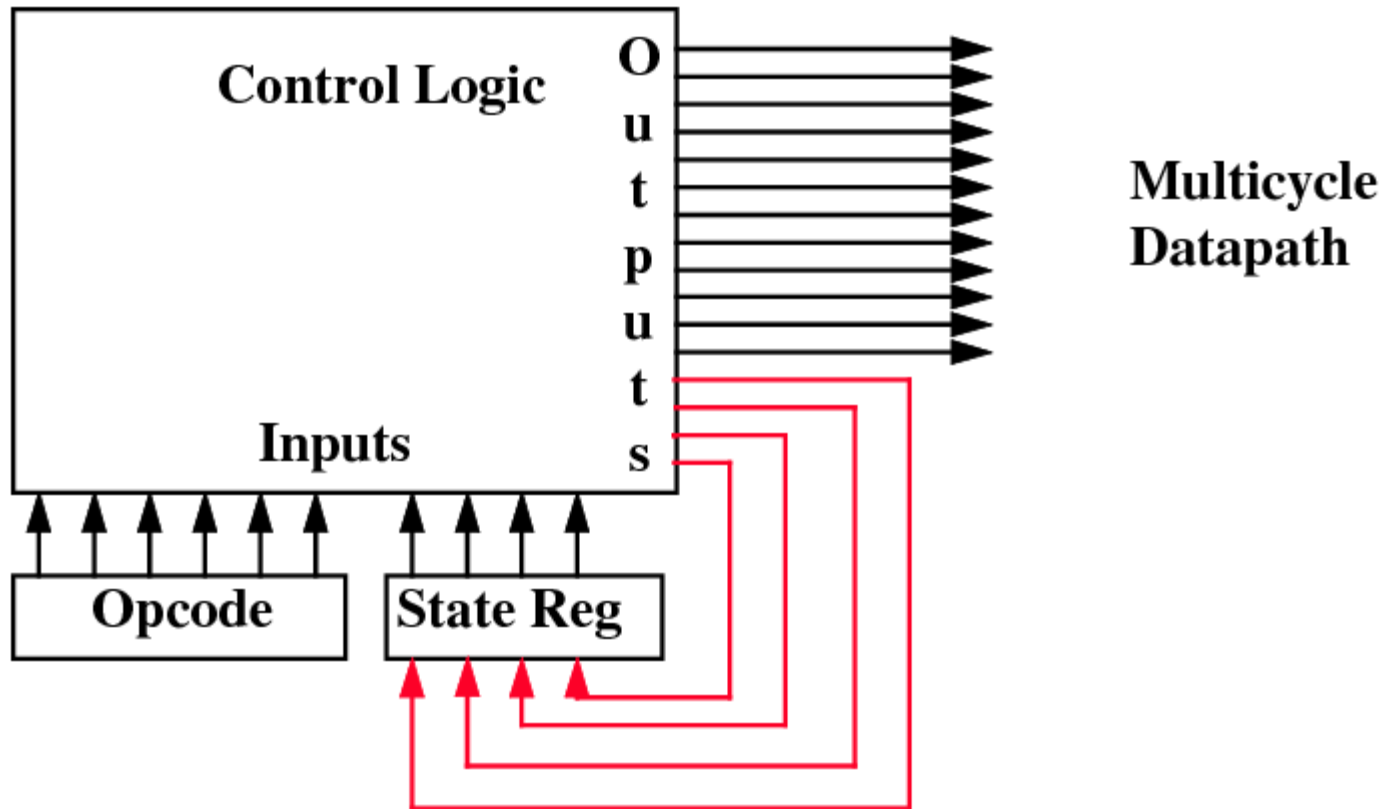
Control Logic

Outputs

Inputs

Opcode

State Reg

Multicycle Datapath

The Big Picture: Performance Perspective

- Performance of a machine was determined by
  - Instruction Count
  - Clock cycle Time
  - Clock cycles per instruction
- Processor Design (data path and control) will determine
  - Clock cycle time
  - Clock cycles per instruction
- We shall first design a Single Cycle Processor
  - Advantage: One clock cycle per instruction
  - Disadvantage: Long cycle time

Summary

- Disadvantages of the Single Cycle Processor
  - Long cycle time
  - Cycle time is too long for all instructions except load
- Multiple cycle processor
  - Divide the instruction into smaller steps
  - Execute each step (instead of the entire instruction) in one cycle