

# Fundamentals of Computer Systems

## A Single Cycle MIPS Processor

Stephen A. Edwards

Columbia University

Summer 2021

Illustrations Copyright © 2007 Elsevier

## The Datapath

- The lw Instruction

- The sw Instruction

- R-Type Instructions

- The beq Instruction

## The Controller

- Instruction Encoding

- The ALU Decoder

- The Main Decoder

- The j Instruction

## Processor Performance

- The Critical Path

# Let's Build a Simple Processor

Supported instructions:

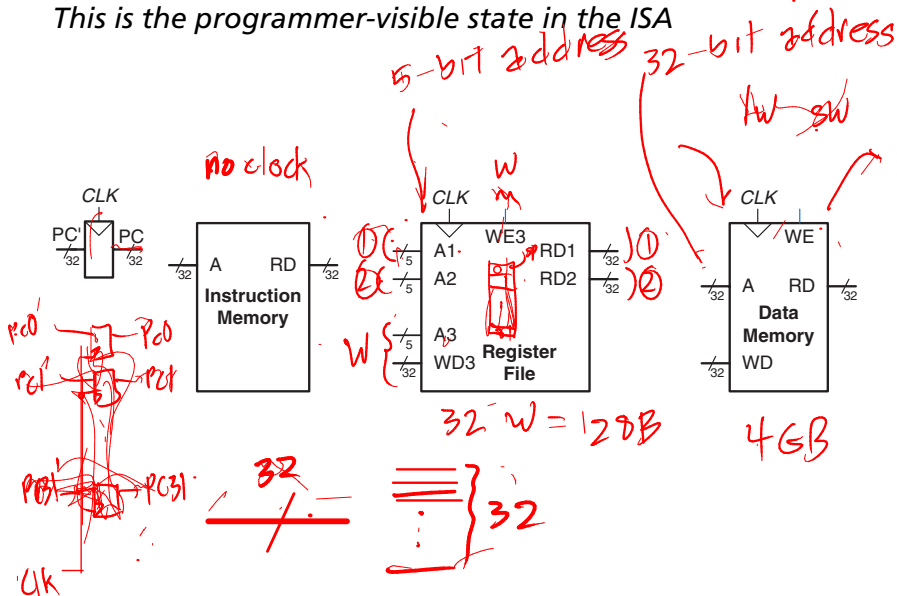
- ▶ R-type: and, or, addu, subu, slt
- ▶ Memory instructions: lw, sw
- ▶ Branch instructions: beq

Version 2.0:

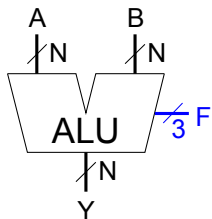
- ▶ I-type: addiu
- ▶ J-type: j

# MIPS State Elements

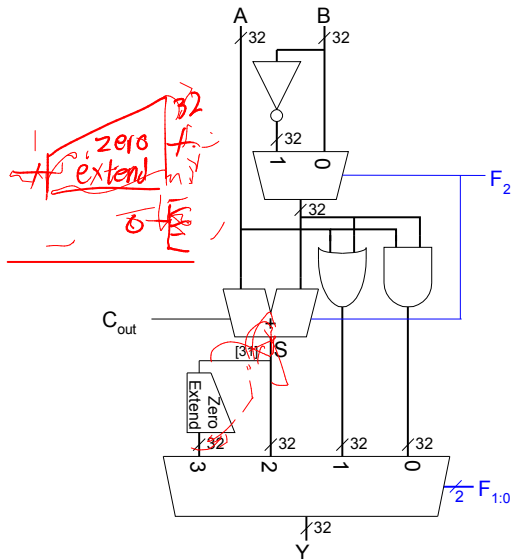
This is the programmer-visible state in the ISA



# ALU Interface and Implementation



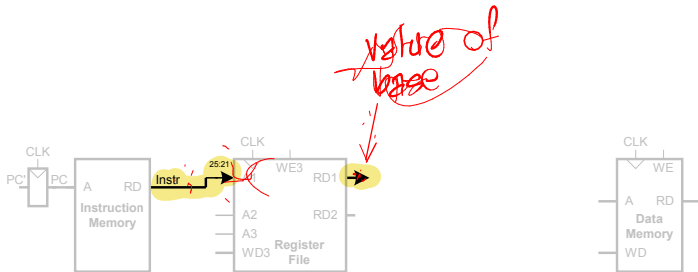
F2	F1	F0	Func.
0	0	0	A & B
0	0	1	A   B
0	1	0	A + B
0	1	1	
1	0	0	A & $\bar{B}$
1	0	1	A   $\bar{B}$
1	1	0	A - B
1	1	1	A < B (slt)





# Datapath Elements for the lw Instruction

Read the base register

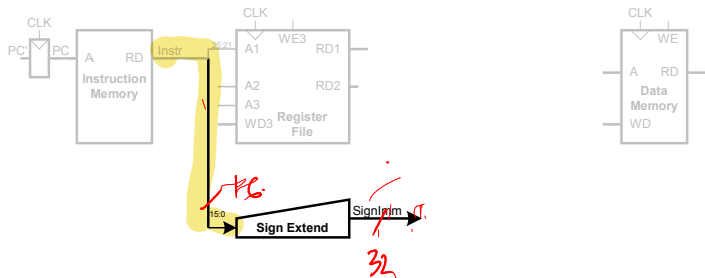


lw rt, offset(base)



# Datapath Elements for the lw Instruction

Sign-extend the immediate



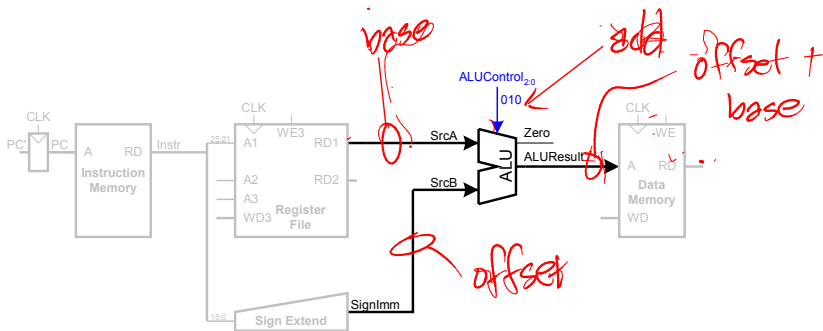
lw rt, offset(base)





# Datapath Elements for the lw Instruction

Add the base register and the sign-extended immediate to compute the data memory address

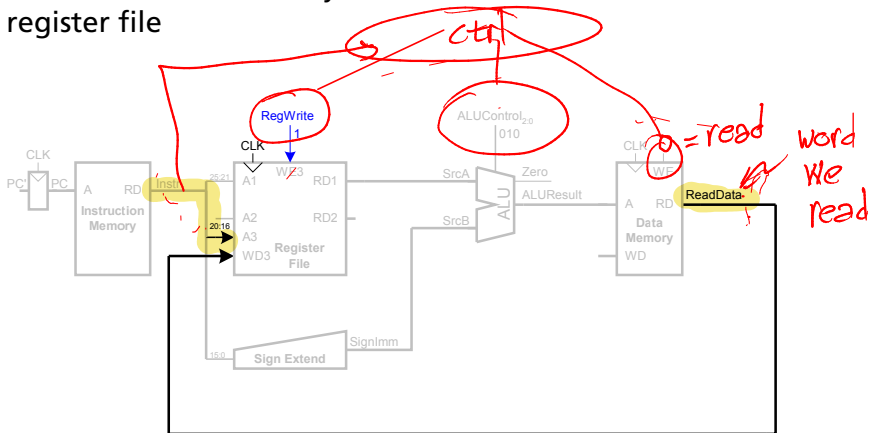


`lw rt, offset(base)`

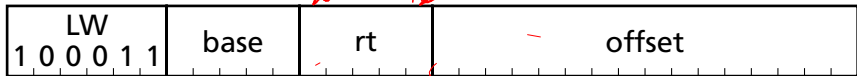


# Datapath Elements for the lw Instruction

Read data from memory and write it back to rt in the register file

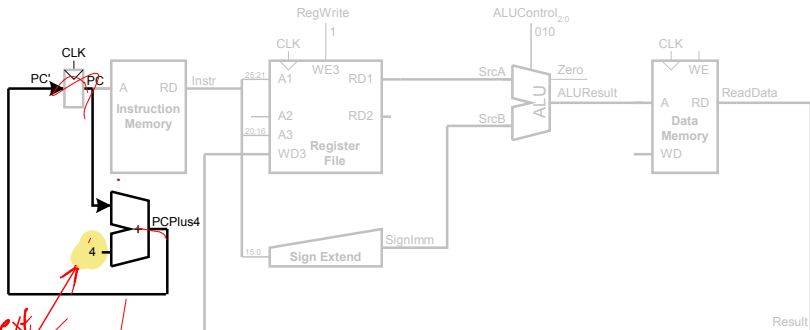


`lw rt, offset(base)`



# Datapath Elements for the lw Instruction

Add four to the program counter to determine address of the the next instruction to execute

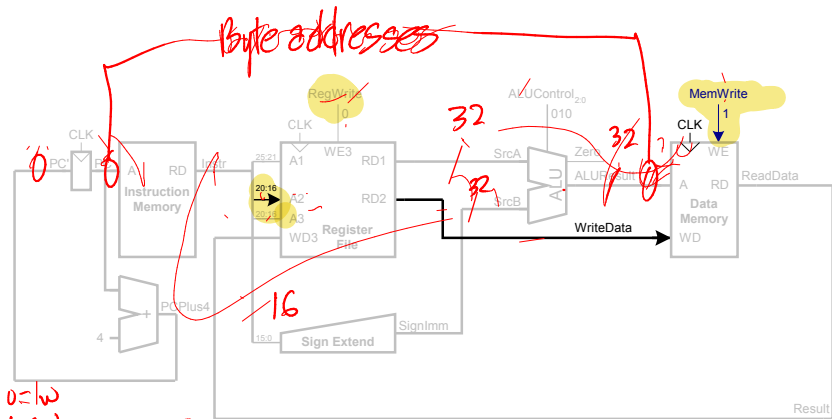


`lw rt, offset(base)`

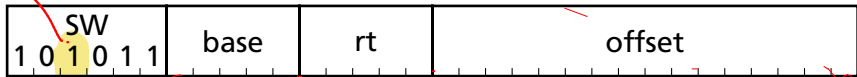


# Additional Elements for sw

Read *rt* from the register file and write it to data memory



sw *rt*, offset(base) 20 T6



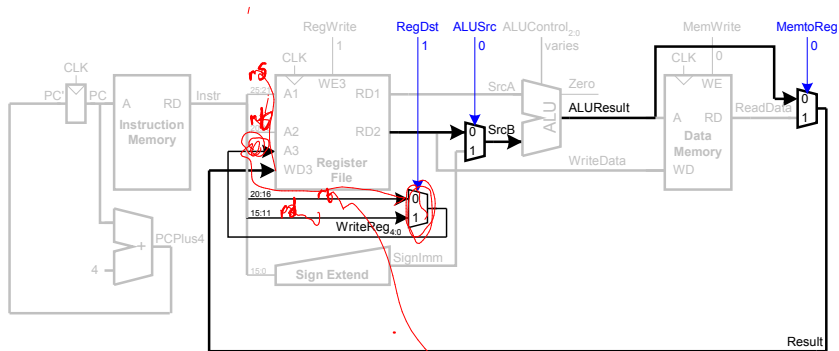
31

0

# Additional Elements for R-Type Instructions

Read from rs and rt

Write ALUResult to rd (instead of rt)



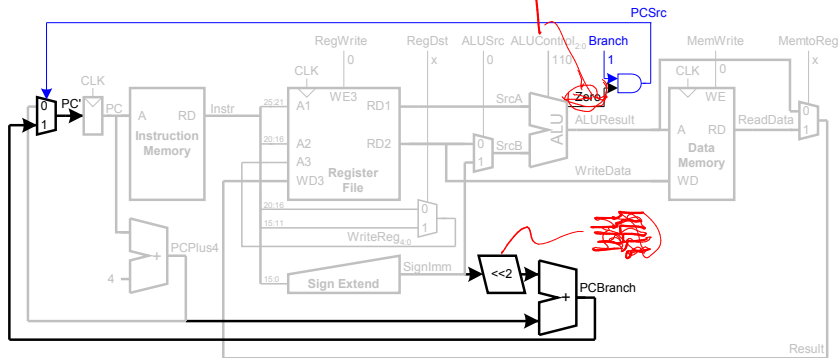
addu rd, rs, rt



# Additional Elements for beq

Determine whether rs and rt are equal

Calculate branch target address



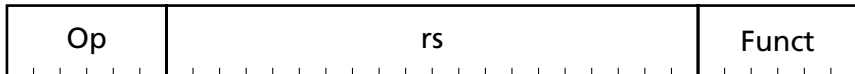
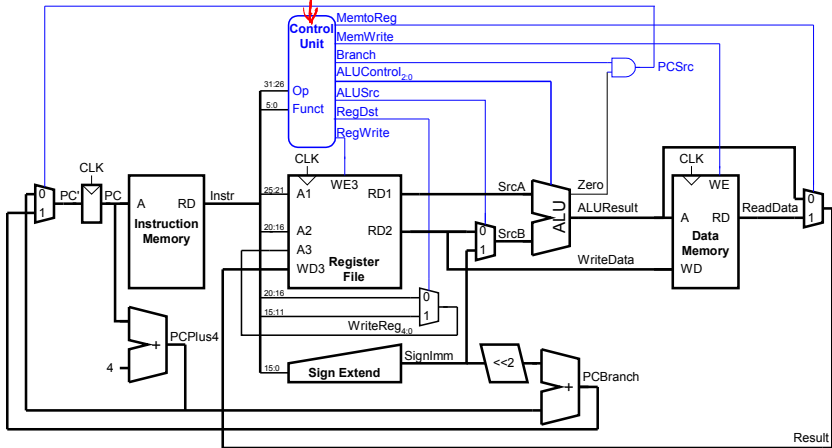
beq rs, rt, offset

~~# of words~~



# Add a controller to complete it

*Instruction decoder*



## R-Type Instruction Encoding

addu rd, rs, rt

SPECIAL	rs	rt	rd	000000	ADDU
000000					100001

subu rd, rs, rt

SPECIAL	rs	rt	rd	000000	SUBU
000000					100011

and rd, rs, rt

SPECIAL	rs	rt	rd	000000	AND
000000					100100

or rd, rs, rt

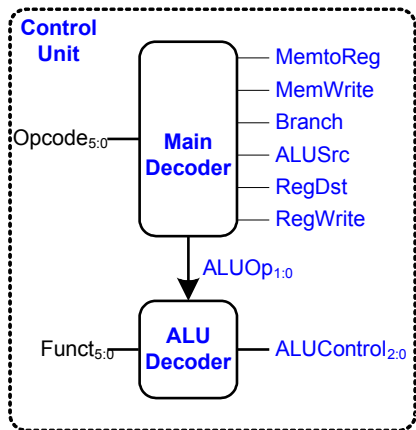
SPECIAL	rs	rt	rd	000000	OR
000000					100101

slt rd, rs, rt

SPECIAL	rs	rt	rd	000000	SLT
000000					101010



# The ALU Decoder



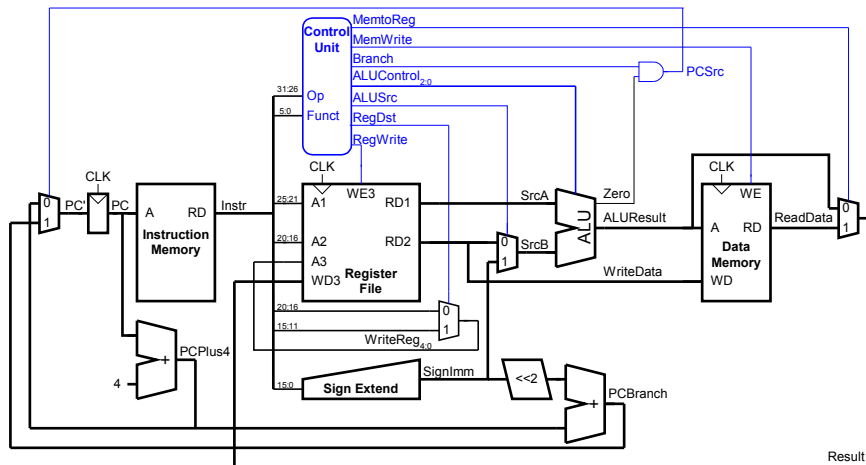
Part of the control unit responsible for implementing the opcode *Func* field.

ALU Op	Func	ALU Ctrl.	ALU Function
00	—	010	Add
-1	—	110	Subtract
1-	100001	010	Add
1-	100011	110	Subtract
1-	100100	000	AND
1-	100101	001	OR
1-	101010	111	Slt

*Handwritten notes:* "even if the line" (next to the first two rows) and "r-1000" (next to the AND row).

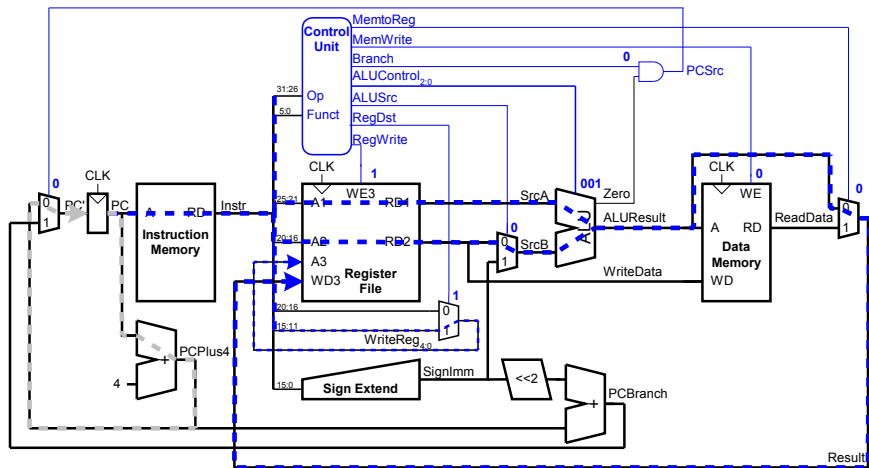
# The Main Decoder

Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp
R-type	000000							
lw	100011							
sw	101011							
beq	000100							



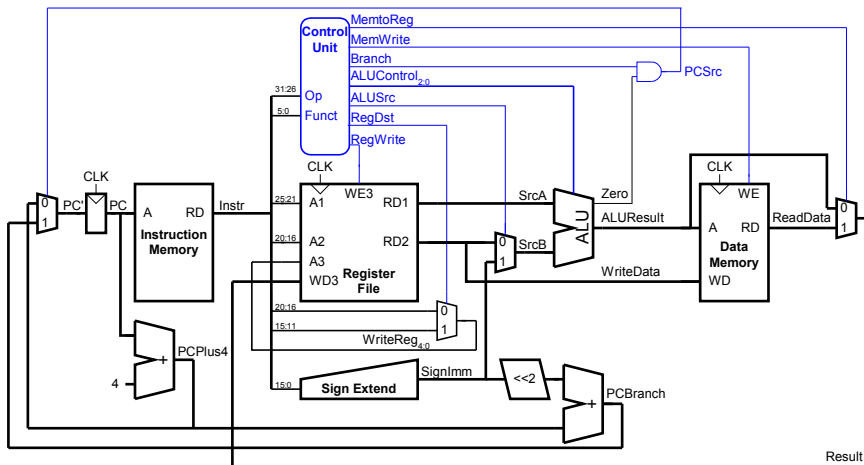
# The Main Decoder

Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp
R-type	000000	1	1	0	0	0	0	1-
lw	100011							
sw	101011							
beq	000100							



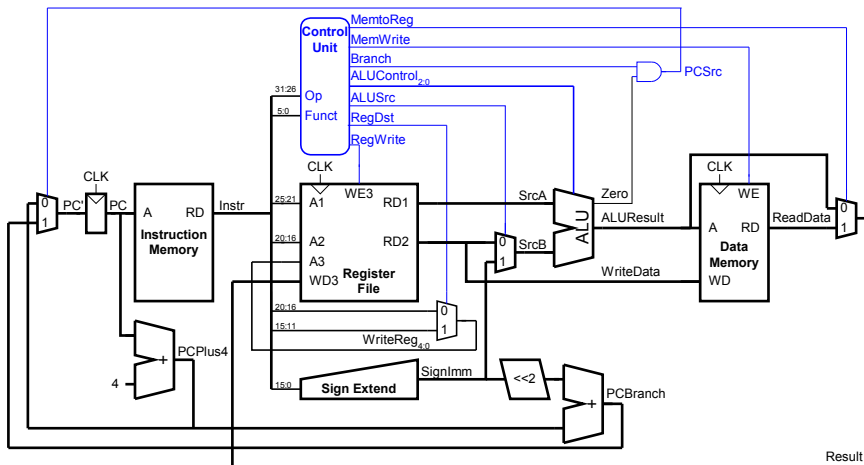
# The Main Decoder

Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp
R-type	000000	1	1	0	0	0	0	1-
lw	100011	1	0	1	0	0	1	00
sw	101011							
beq	000100							



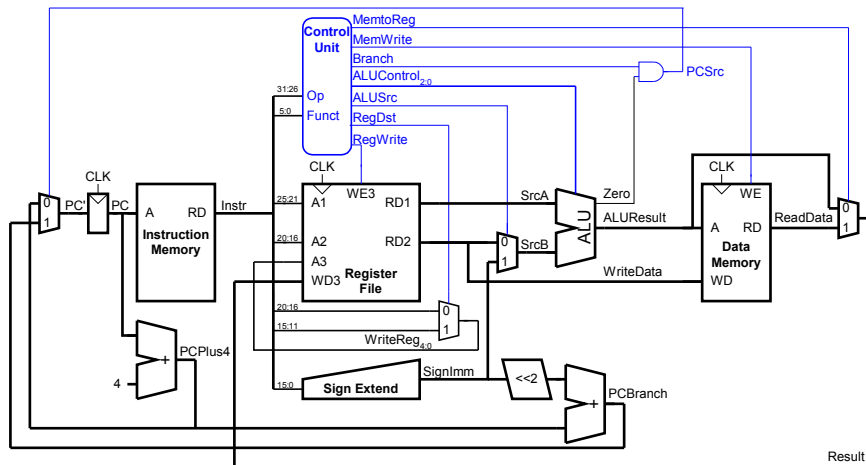
# The Main Decoder

Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp
R-type	000000	1	1	0	0	0	0	1-
lw	100011	1	0	1	0	0	1	00
sw	101011	0	-	1	0	1	-	00
beq	000100	-	-	-	1	-	-	-



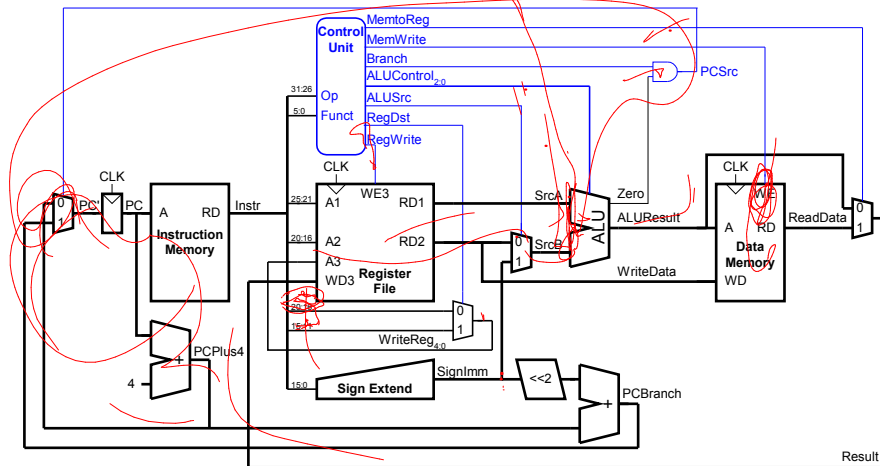
# The Main Decoder

Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp
R-type	000000	1	1	0	0	0	0	1-
lw	100011	1	0	1	0	0	1	00
sw	101011	0	-	1	0	1	-	00
beq	000100	0	-	0	1	0	-	01



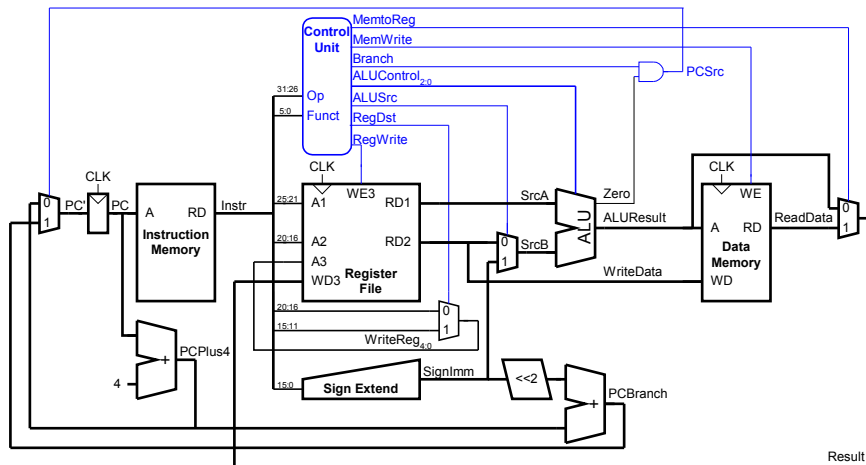
# The Main Decoder

Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp
R-type	000000	1	1	0	0	0	0	1-
lw	100011	1	0	1	0	0	1	00
sw	101011	0	-	1	0	1	-	00
beq	000100	0	-	0	1	0	-	01
addiu	001001	1	0	1	0	0	0	00 = Add



# The Main Decoder

Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp
R-type	000000	1	1	0	0	0	0	1-
lw	100011	1	0	1	0	0	1	00
sw	101011	0	-	1	0	1	-	00
beq	000100	0	-	0	1	0	-	01
addiu	001001	1	0	1	0	0	0	00



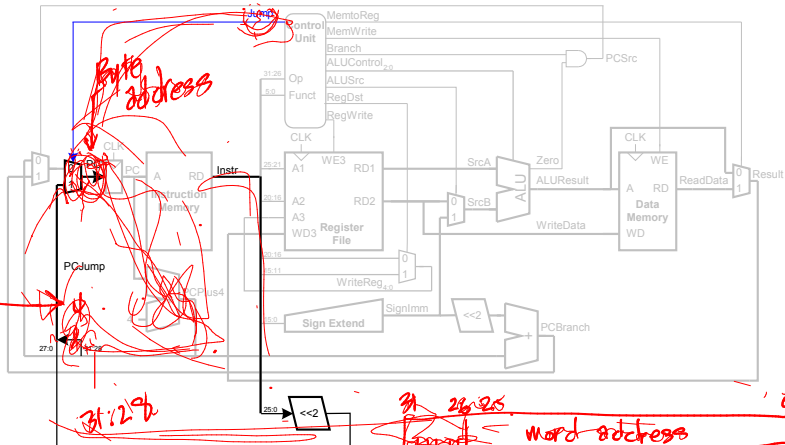


# Additional Elements for the j Instruction

Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp	Jump
R-type	000000	1	1	0	0	0	0	1-	0
lw	100011	1	0	1	0	0	1	00	0
sw	101011	0	-	1	0	1	-	00	0
beq	000100	0	-	0	1	0	-	01	0
addiu	001001	1	0	1	0	0	0	00	0
j	000010	0	-	-	-	0	-	--	1

*Don't jump*

*Use top 4 of old PC*



*31 26-25 word address 0 00*

# Processor Performance

*Dynamic* ← *The algorithm*  
1

$$\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$

$\frac{\text{Seconds}}{\text{Program}}$

How long you have to wait

$\frac{\text{Instructions}}{\text{Program}}$

Number that must execute to complete the task

$\frac{\text{Clock Cycles}}{\text{Instruction}}$

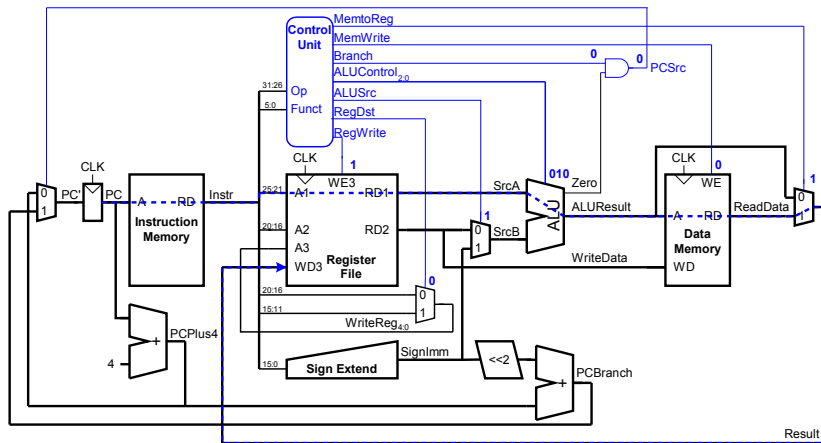
CPI: Cycles per instruction

$\frac{\text{Seconds}}{\text{Clock Cycle}}$

The clock period (1/frequency)

~~technology~~  
technology  
topology of  
circuit

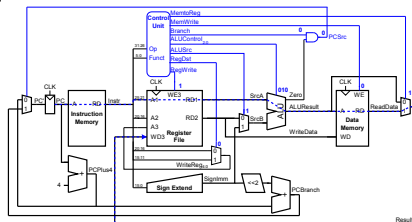
# The Critical Path Here: Load from Memory



Instruction Memory to Register File to ALU to Data Memory to Register File

# The Critical Path Dictates the Clock Period

Element	Delay
Register clk-to-Q	$t_{pcq-PC}$ 30 ps
Register setup	$t_{setup}$ 20
Multiplexer	$t_{mux}$ 25
ALU	$t_{ALU}$ 200
Memory Read	$t_{mem}$ 250
Register file read	$t_{RFread}$ 150
Register file setup	$t_{RFsetup}$ 20



*Optimistic*

$$\begin{aligned} T_C &= t_{pcq-PC} + t_{mem-I} + t_{RFread} + t_{ALU} + t_{mem-D} + t_{mux} + t_{RFsetup} \\ &= (30 + 250 + 150 + 200 + 250 + 25 + 20) \text{ ps} \\ &= 925 \text{ ps} \\ &= 1.08 \text{ GHz} \end{aligned}$$

## Execution Time for Our Single-Cycle Processor

For a 100 billion-instruction task on our single-cycle processor with a 925 ps clock period,

$$\begin{aligned}\frac{\text{Seconds}}{\text{Program}} &= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}} \\ &= 100 \times 10^9 \times 1 \times 925 \text{ ps} \\ &= 92.5 \text{ seconds}\end{aligned}$$