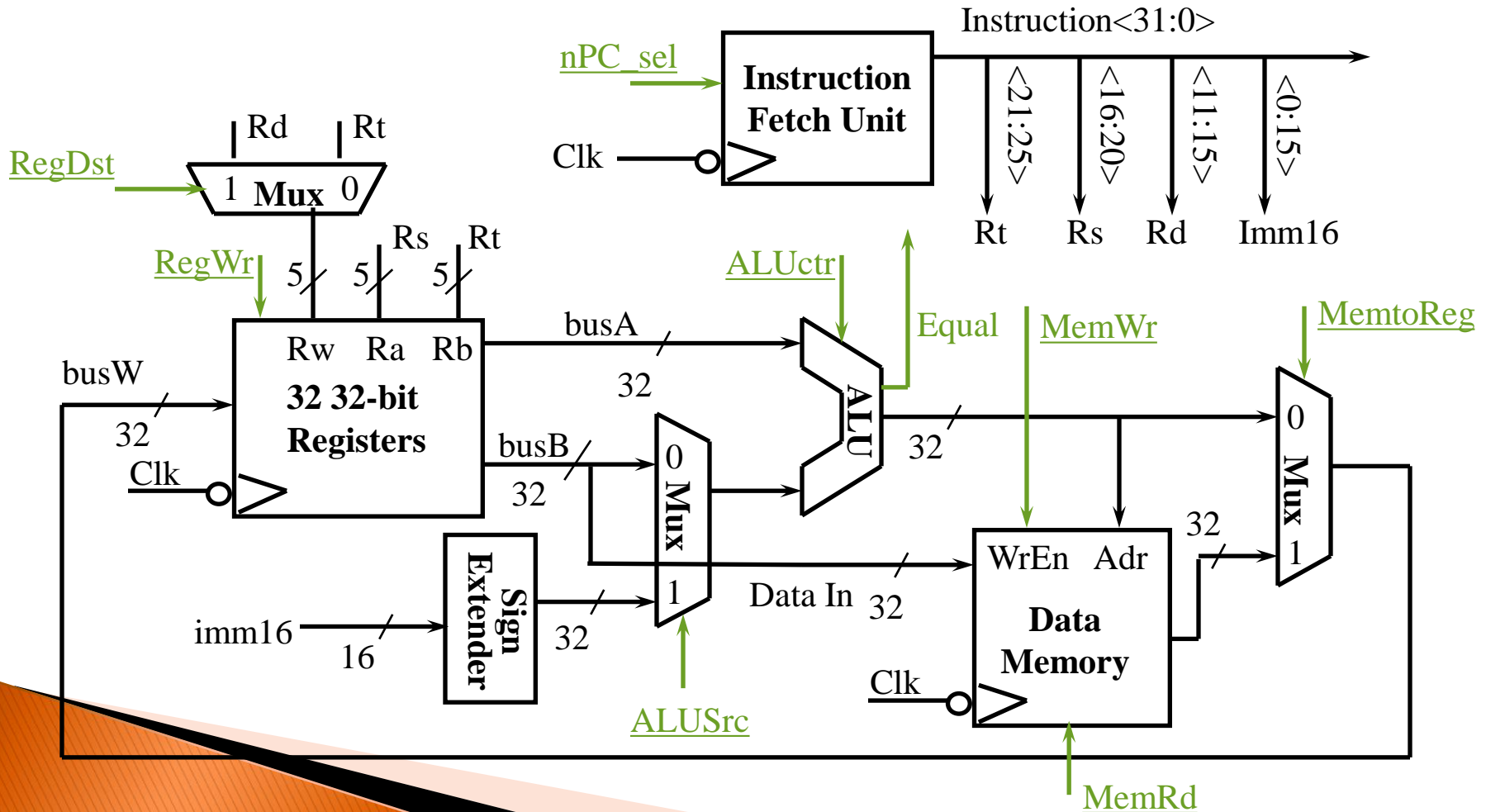


# CSE331 – Computer Organization

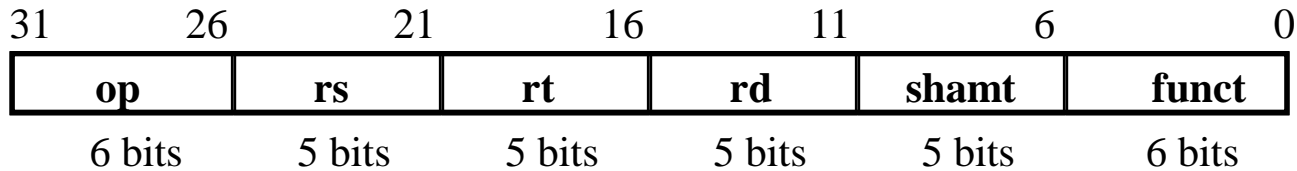
Lecture 7: Single Cycle Control

# A Single Cycle Datapath

- ▶ We have everything except control signals (underlined)
  - Today's lecture will look at how to generate the control signals



# RTL: R-Type Instructions

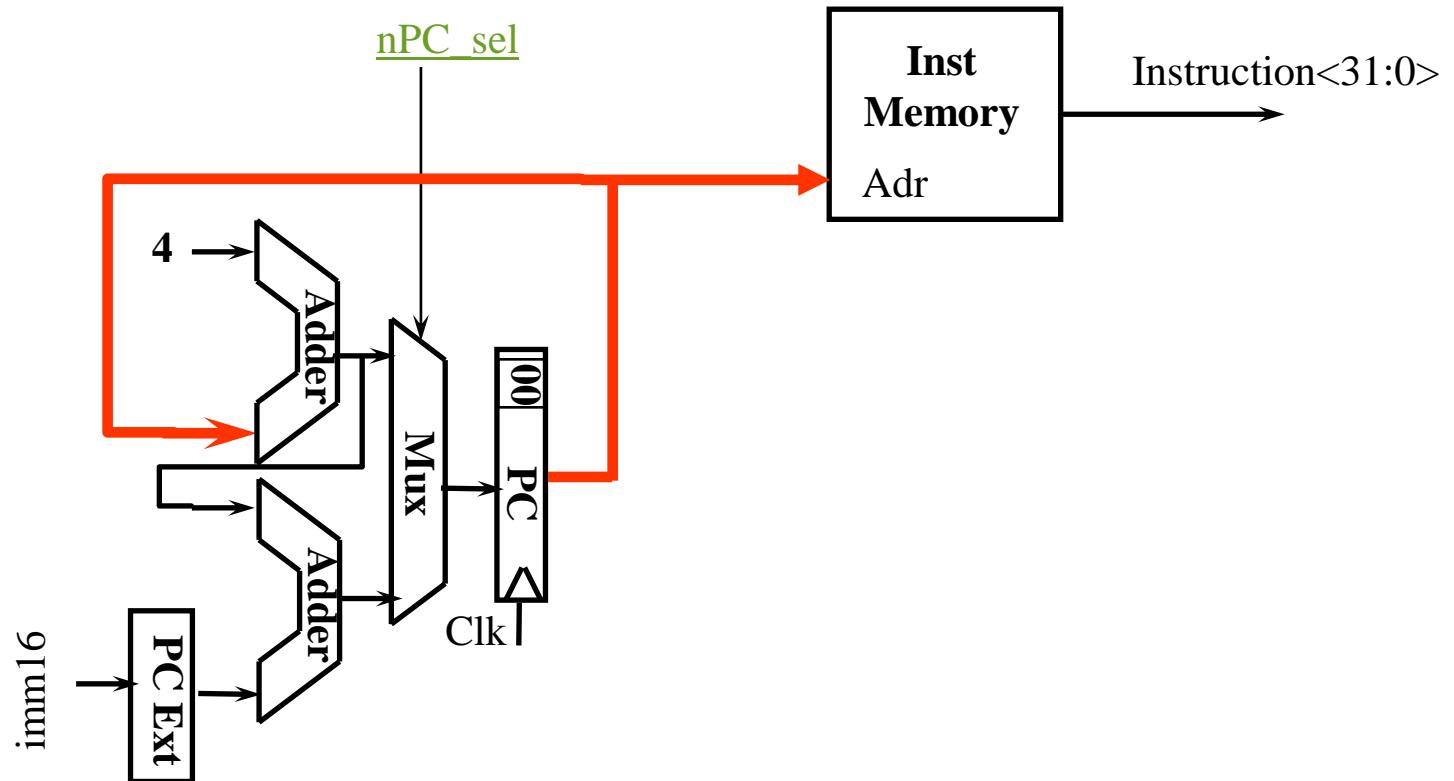


## ► Example: `add rd, rs, rt`

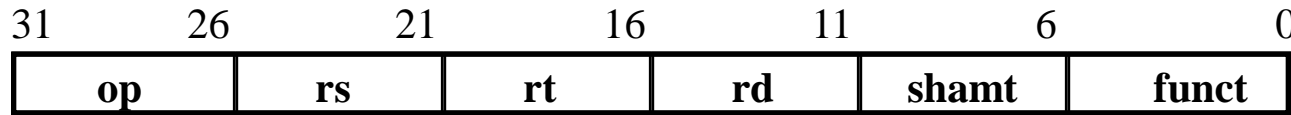
- `Mem[PC]`                      Fetch the instruction from memory
- $R[rd] \leftarrow R[rs] + R[rt]$       The actual operation
- $PC \leftarrow PC + 4$               Calculate the next instruction's address

# Instruction Fetch Unit at the Beginning of Add

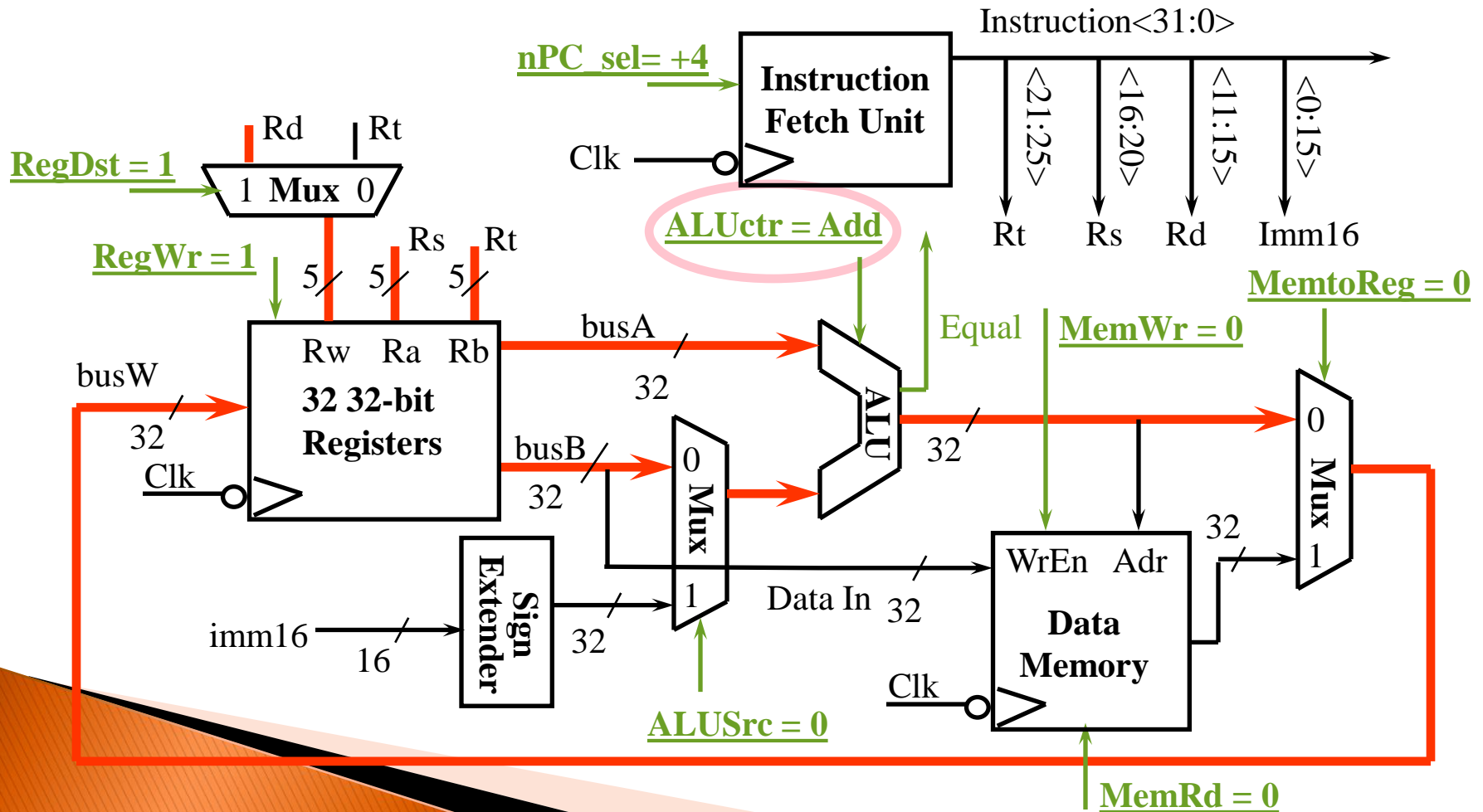
- Fetch the instruction from Instruction memory:  
 $\text{Instruction} \leftarrow \text{mem}[\text{PC}]$
- Done for all instructions => Don't need special control bits



# The Single Cycle Datapath during Add

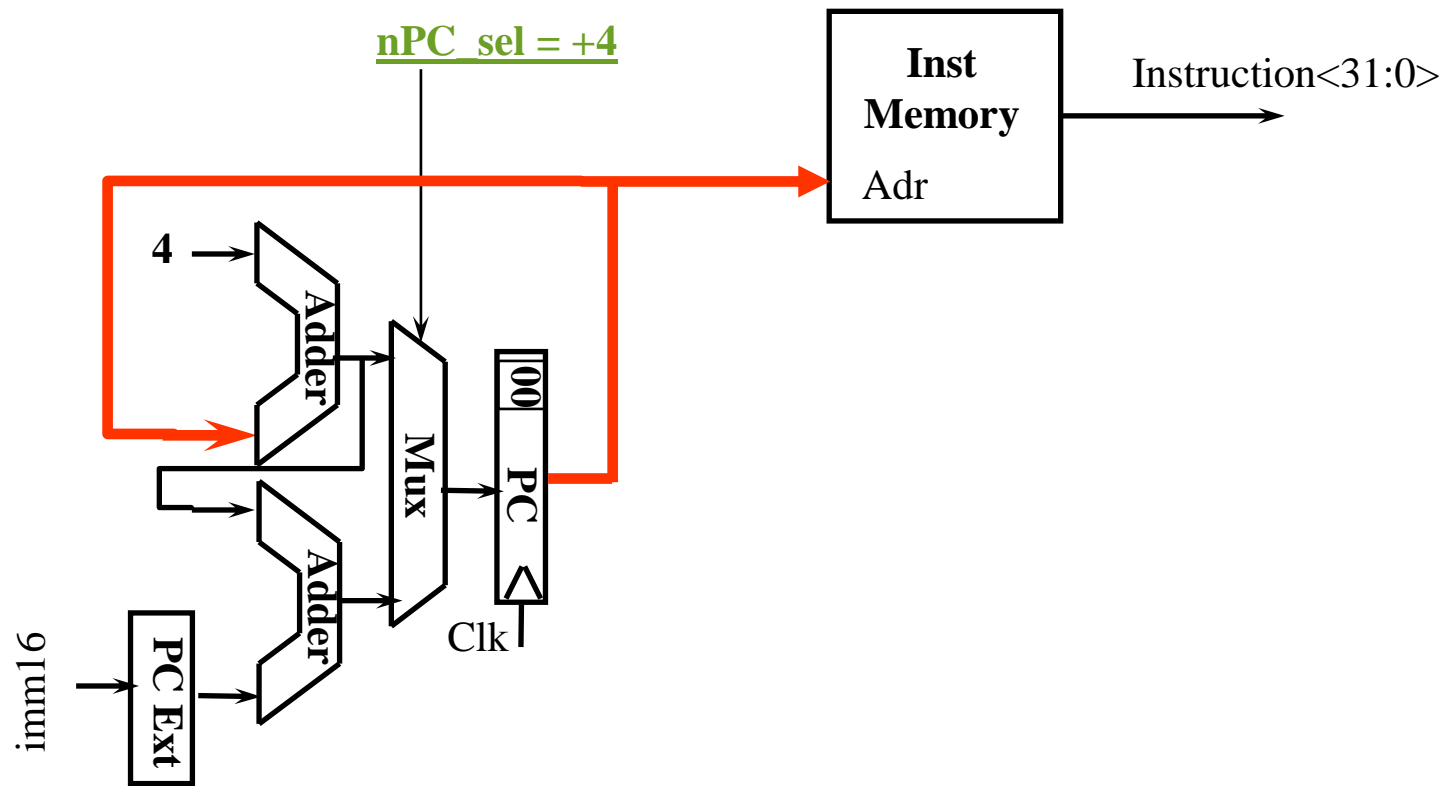


►  $R[rd] \leftarrow R[rs] \text{ op } R[rt]$

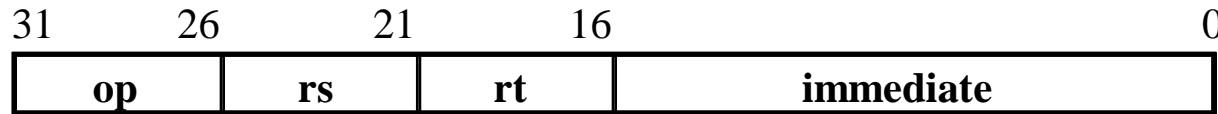


# Instruction Fetch Unit at the End of Add

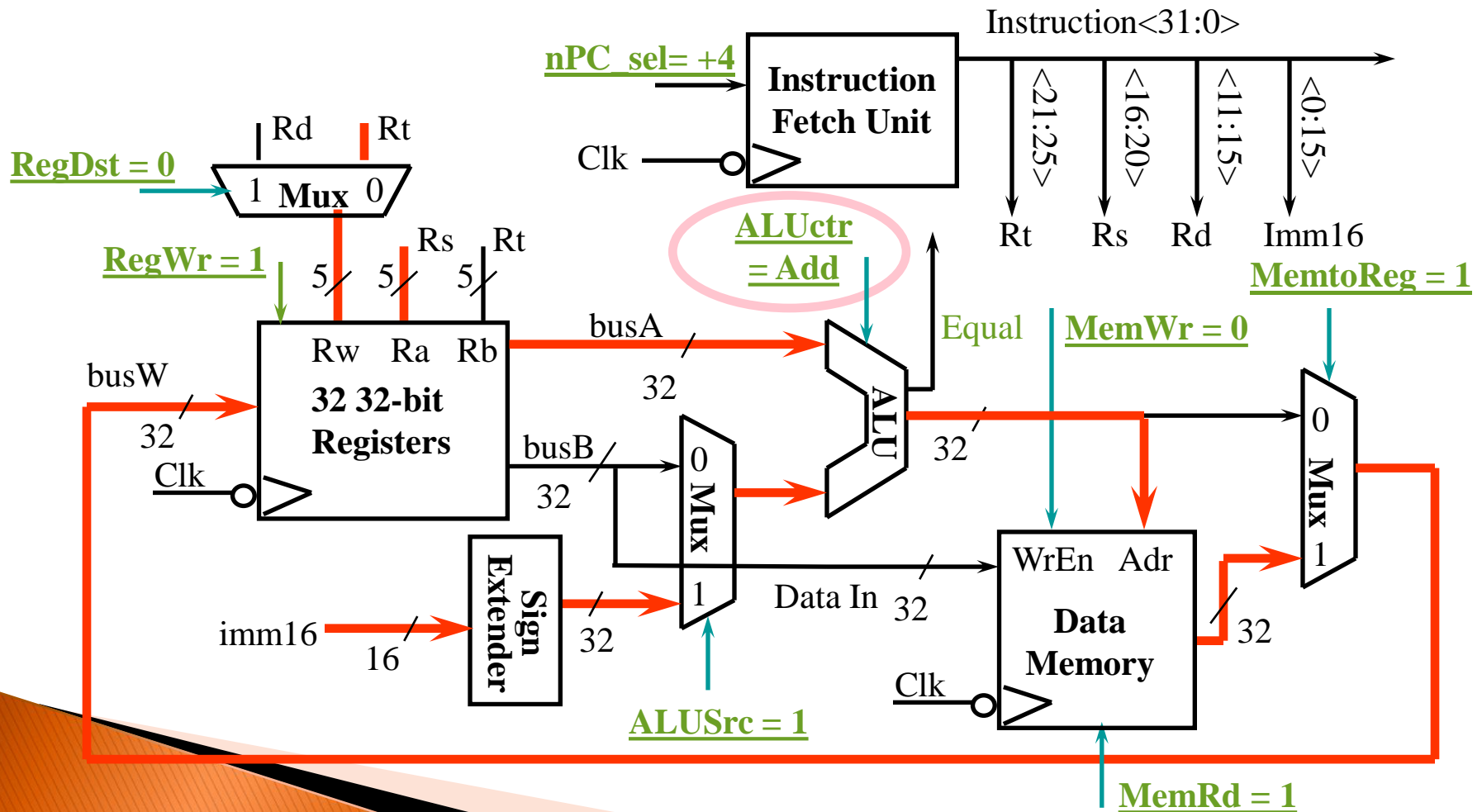
- ▶  $PC \leftarrow PC + 4$ 
  - This is the same for all instructions except Branch and Jump



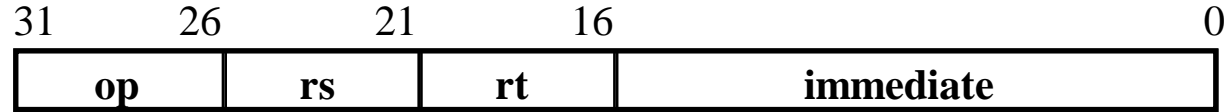
# The Single Cycle Datapath during Load



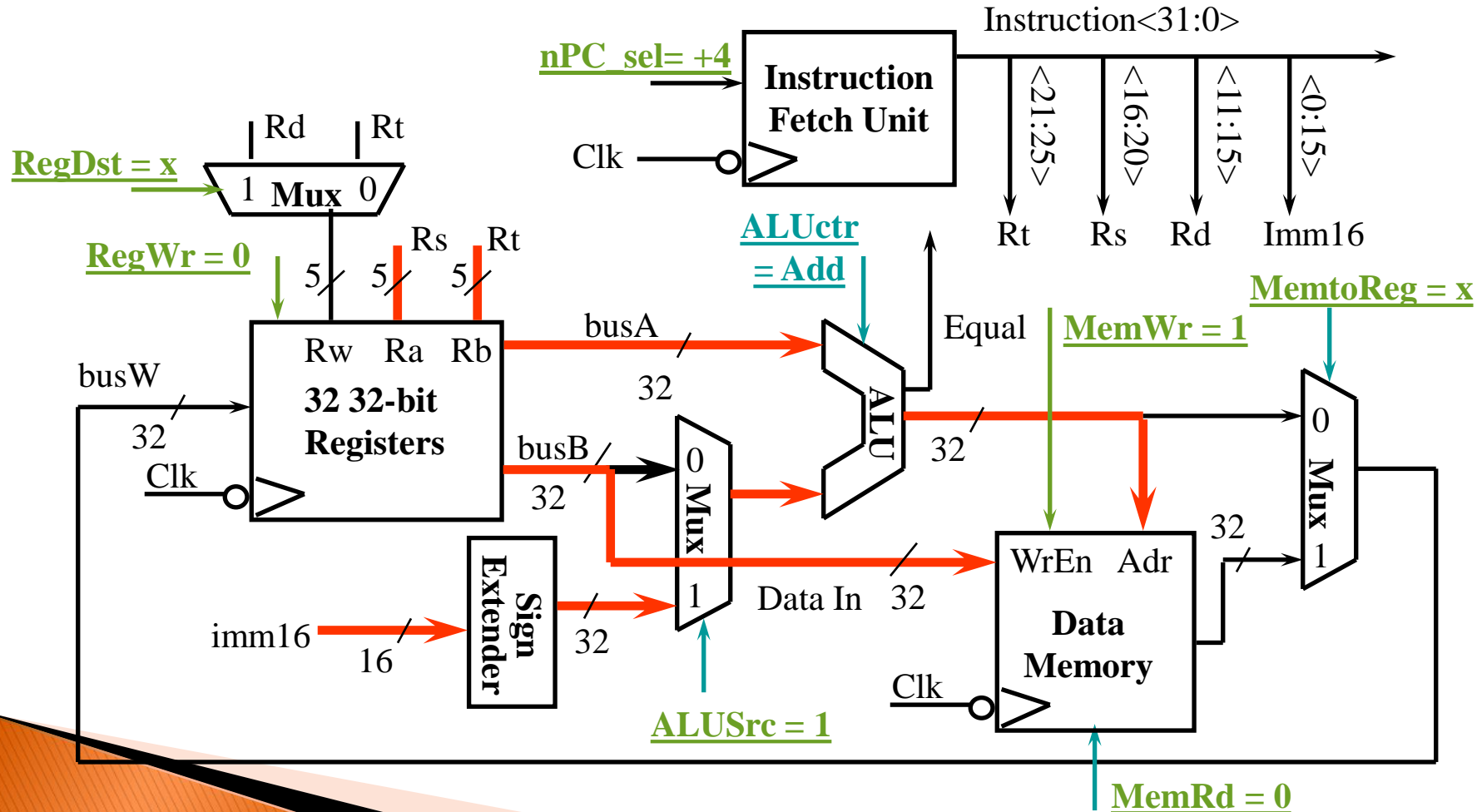
- ▶  $R[rt] \leftarrow \text{Data Memory } \{R[rs] + \text{SignExt}[imm16]\}$



# The Single Cycle Datapath during Store

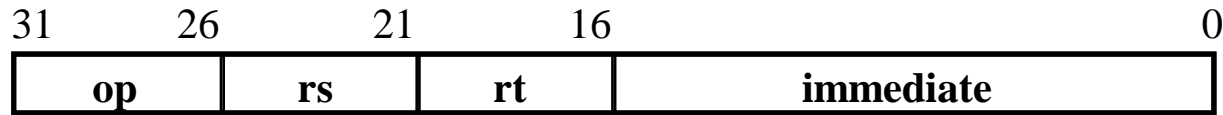


- ▶ Data Memory  $\{R[rs] + \text{SignExt}[imm16]\} \leftarrow R[rt]$

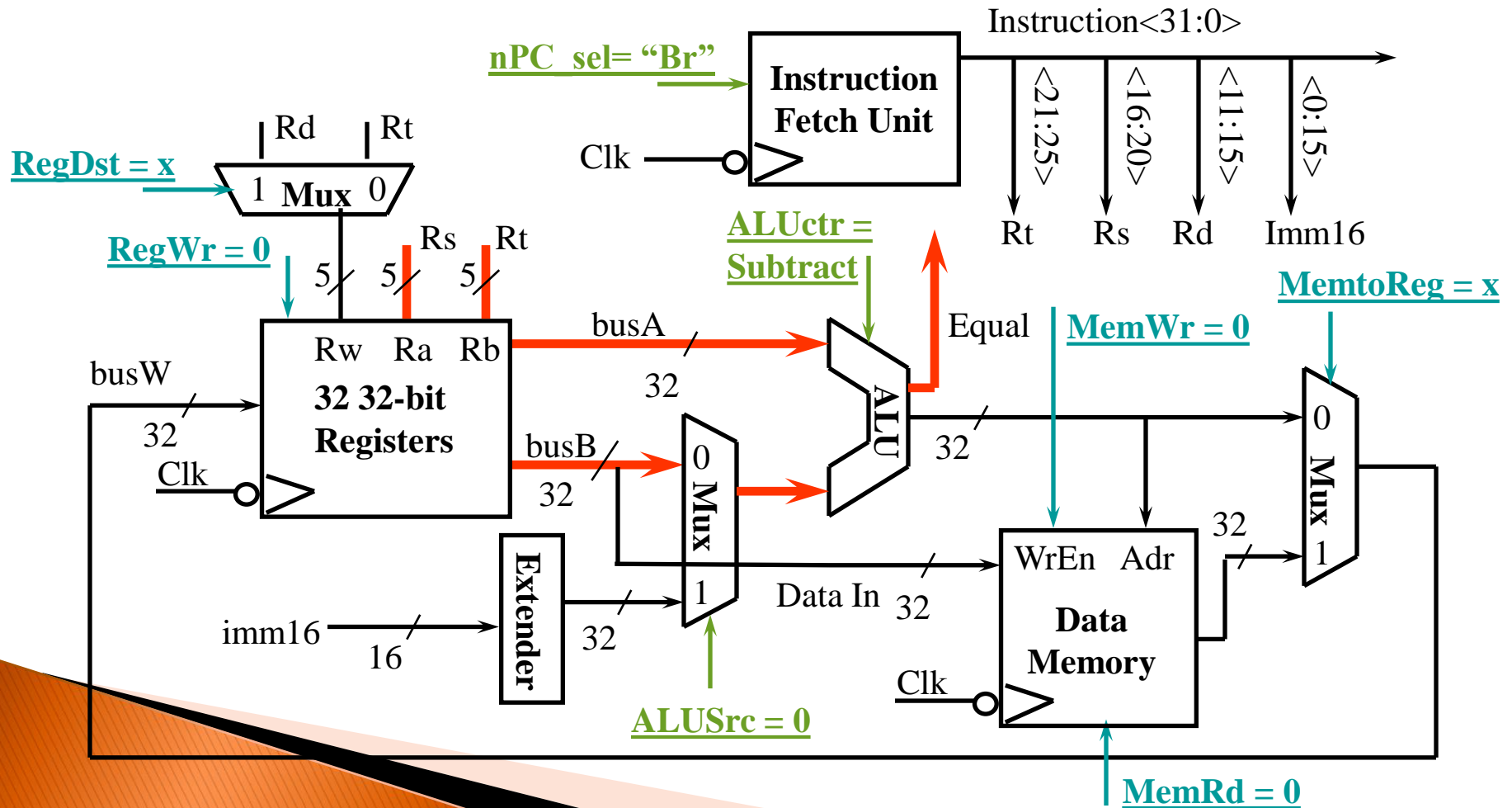




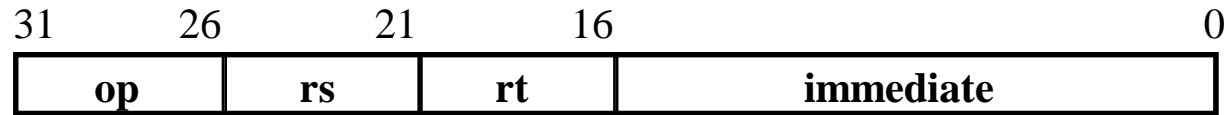
# The Single Cycle Datapath during Branch



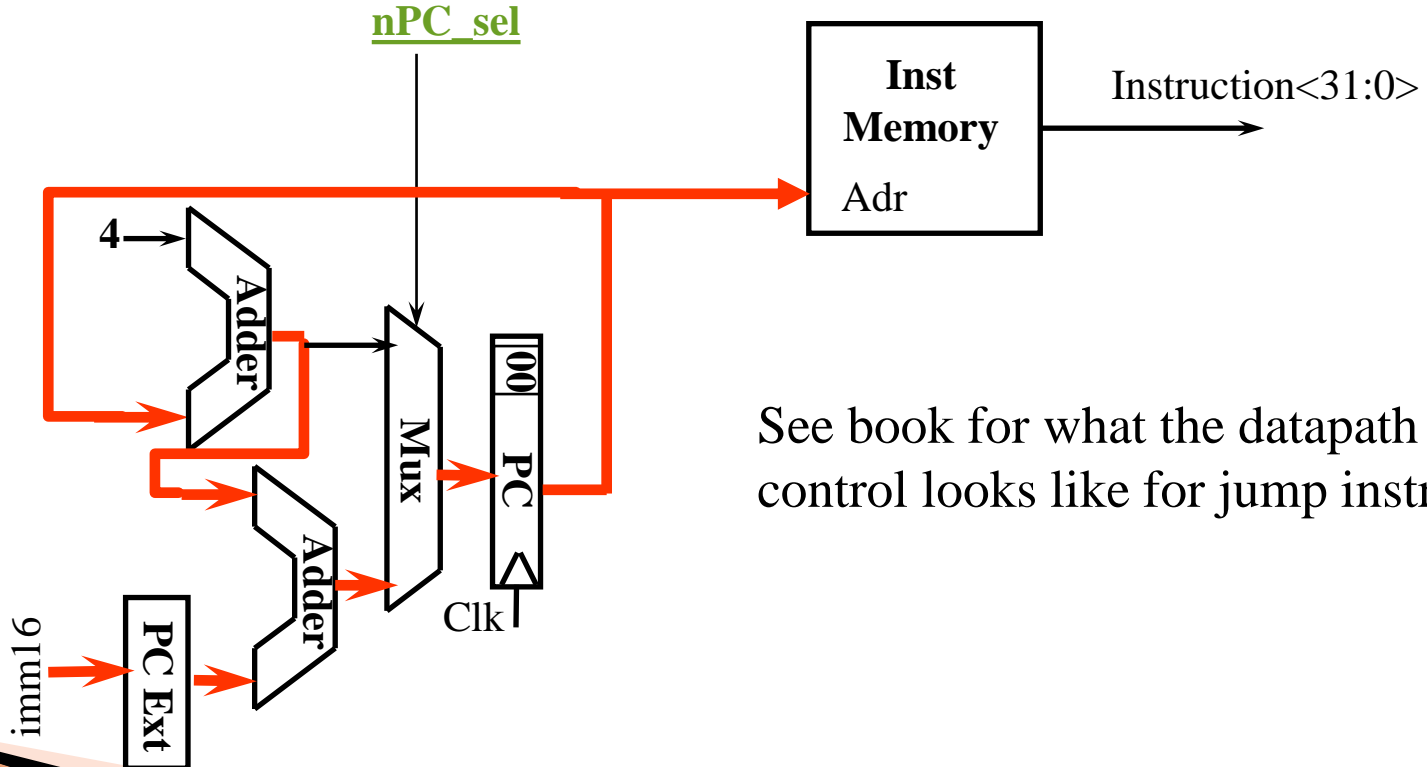
► if  $(R[rs] - R[rt] == 0)$  then  $Equal \leftarrow 1$  ; else  $Equal \leftarrow 0$



# Instruction Fetch Unit at the End of Branch

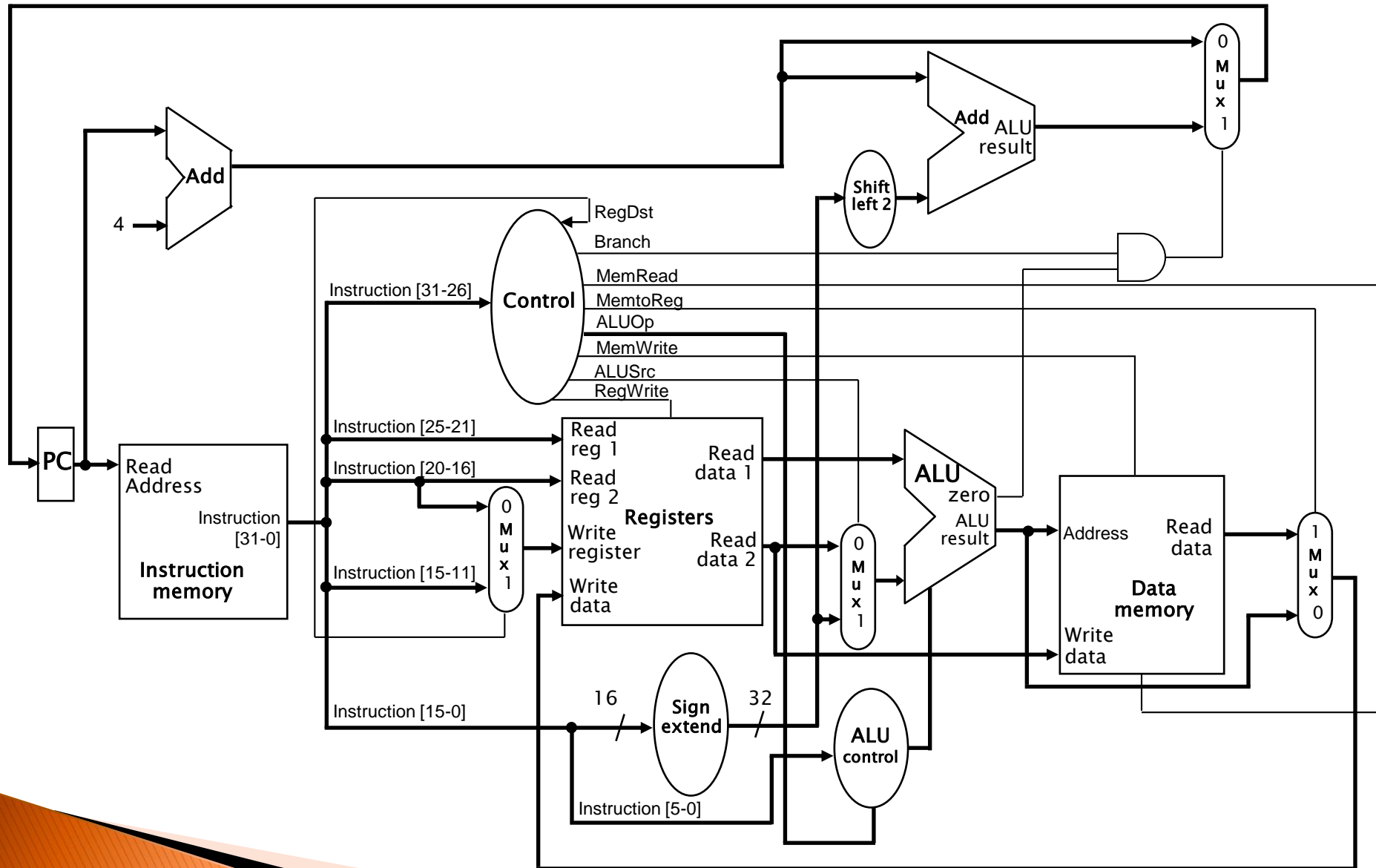


- if (Equal & Branch)  
then  $PC = PC + 4 + \text{SignExt}[\text{imm16}] * 4$  ;  
else  $PC = PC + 4$

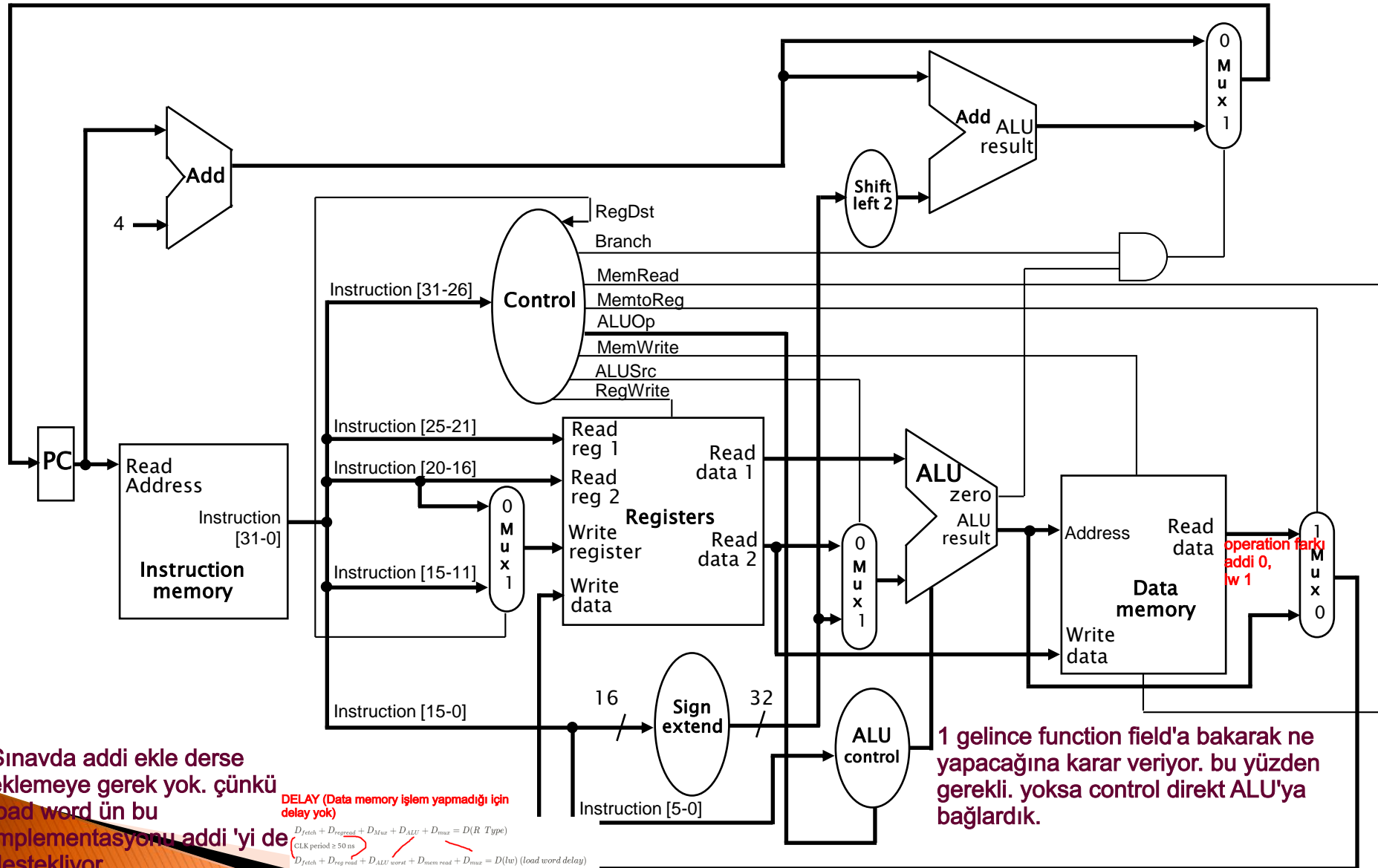


See book for what the datapath and control looks like for jump instructions.

# Different View of Same Implementation (From Book)



# Different View of Same Implementation (From Book)



Sınavda addi ekle derse eklemeye gerek yok. çünkü load word ün bu implementasyonu addi 'yi de destekliyor.  
- ALU Control'e operasyonu control unit söylüyor.

DELAY (Data memory işlem yapmadığı için delay yok)

$$D_{\text{fetch}} + D_{\text{regread}} + D_{\text{Mux}} + D_{\text{ALU}} + D_{\text{memr}} = D(R \text{ Type})$$

CLK period  $\geq 50 \text{ ns}$

$$D_{\text{fetch}} + D_{\text{regread}} + D_{\text{ALU}} + D_{\text{memr}} + D_{\text{Mux}} = D(\text{lw}) \text{ (load word delay)}$$

CLK period  $\geq 100 \text{ ns}$

Register veya sign extend+mux delaylarından hangisi fazlaysa ona dikkate al.

clock hızı

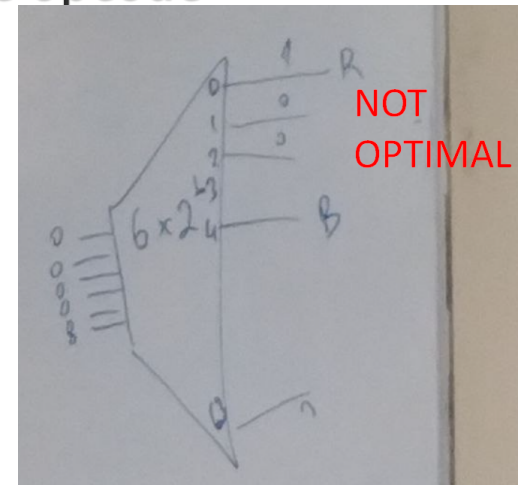
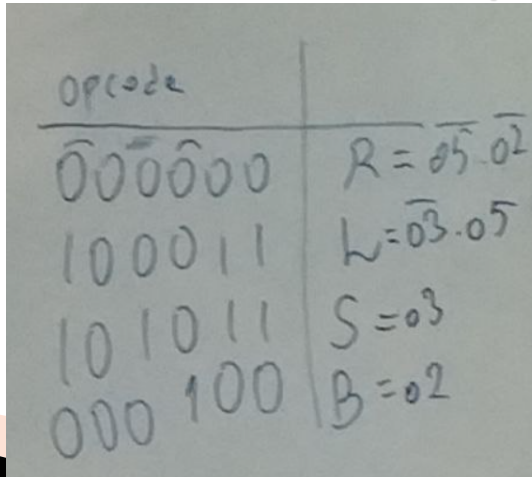
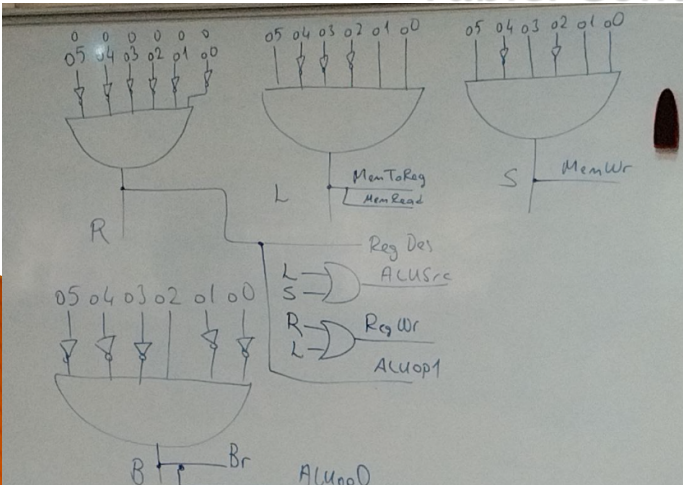
1 gelince function field'a bakarak ne yapacağına karar veriyor. bu yüzden gerekli. yoksa control direkt ALU'ya bağlardık.

örneğin andi eklemek istiyorsak şimdilik desteklemiyor. Ama aluop = 1 olunca andi yap diyebiliriz. Diyelimki orimmediate eklemek istiyorsak ALUOp'u 3 bite çıkarmak gerekiyor. (yeni komut için)

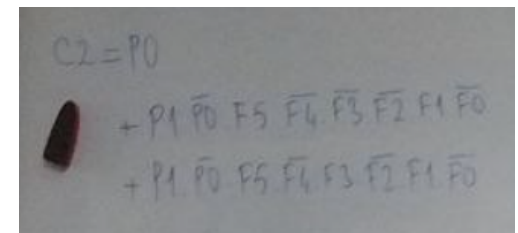
## Step 4: Given Datapath: RTL $\rightarrow$ Control

Instr.	Reg Des	ALUSrc	Memto Reg	Reg Wr	Mem Rd	Mem Wr	Branch	ALUop1	ALUop0
R-type	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

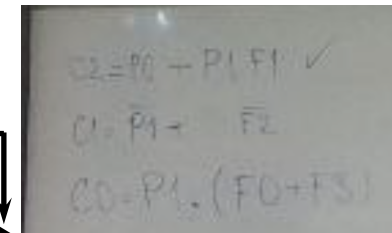
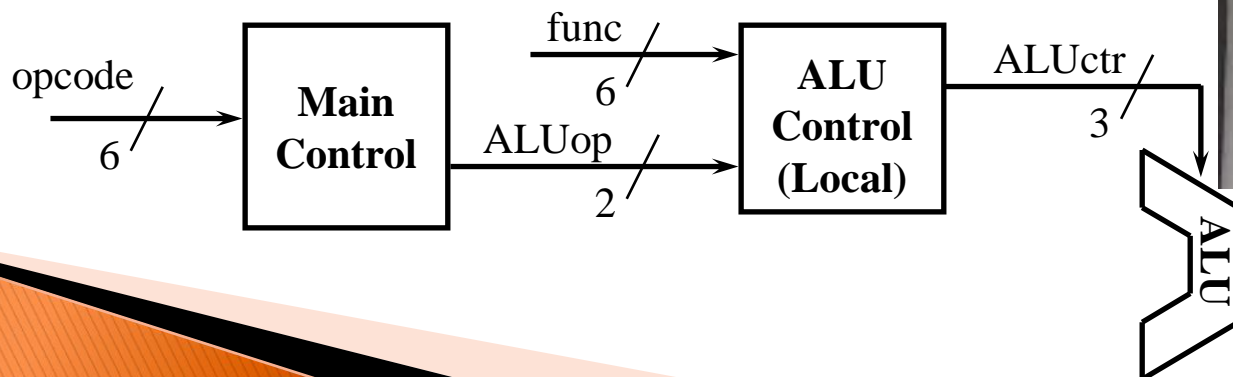
### Table: Control Lines Determined by the opcode



# ALU Control Bits



Instruction opcode	ALUop	Instruction operation	Function field	Desired ALU action	ALU control
LW	00	load word	XXXXXX	add	<b>010</b>
SW	00	store word	XXXXXX	add	<b>010</b>
beq	01	branch equal	XXXXXX	subtract	<b>110</b>
R-type	10	add	100000	add	<b>010</b>
R-type	10	subtract	100010	subtract	<b>110</b>
R-type	10	AND	100100	and	<b>000</b>
R-type	10	OR	100101	or	<b>001</b>
R-type	10	set on less than	101010	set on less than	<b>111</b>



# The Truth Table for the 3 ALU Control Bits

ALUop		Function filed						Operation
ALUop1	ALUop0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0	1	X	X	X	X	X	X	110
1	0	1	0	0	0	0	0	010
1	0	1	0	0	0	1	0	110
1	0	1	0	0	1	0	0	000
1	0	1	0	0	1	0	1	001
1	0	1	0	1	0	1	0	111

funct<5:0>	Instruction Operation
10 0000	add
10 0010	subtract
10 0100	and
10 0101	or
10 1010	set-on-less-than

# The Logic Equation for ALUctr<2>

ALUop		Function filed						Operation
ALUop1	ALUop0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0	1	X	X	X	X	X	X	110
1	0	1	0	0	0	0	0	010
1	0	1	0	0	0	1	0	110
1	0	1	0	0	1	0	0	000
1	0	1	0	0	1	0	1	001
1	0	1	0	1	0	1	0	111

▶  $ALUctr<2> = ALUop0 + (ALUop1 \& func<1>)$



# The Logic Equation for ALUctr<1>

ALUop		Function filed						Operation
ALUop1	ALUop0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0	1	X	X	X	X	X	X	110
1	0	1	0	0	0	0	0	010
1	0	1	0	0	0	1	0	110
1	0	1	0	0	1	0	0	000
1	0	1	0	0	1	0	1	001
1	0	1	0	1	0	1	0	111

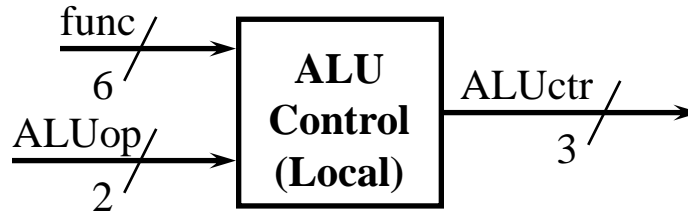
▶  $ALUctr<1> = !ALUop1 + !func<2>$

# The Logic Equation for ALUctr<0>

ALUop		Function filed						Operation
ALUop1	ALUop0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0	1	X	X	X	X	X	X	110
1	0	1	0	0	0	0	0	010
1	0	1	0	0	0	1	0	110
1	0	1	0	0	1	0	0	000
1	0	1	0	0	1	0	1	001
1	0	1	0	1	0	1	0	111

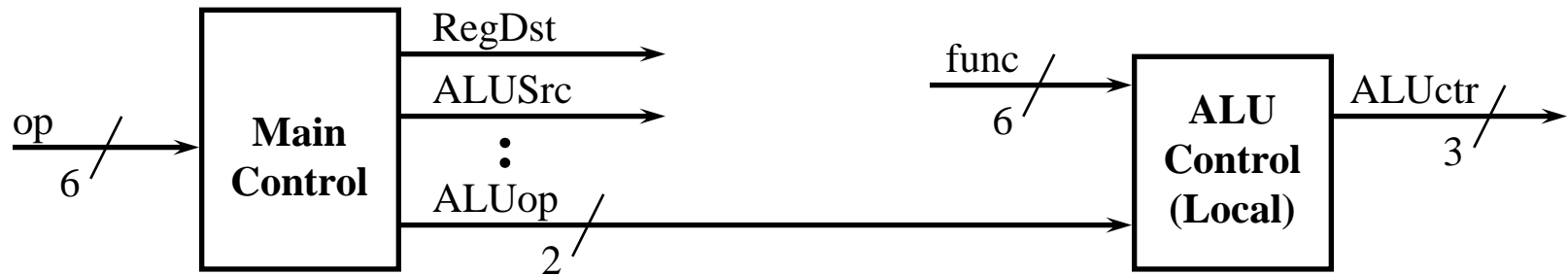
- $ALUctr<0> = ALUop1 \& (func<3> + func<0>)$

# The ALU Control Block



- ▶  $ALUctr<2> = ALUOp0 + (ALUOp1 \& func<1>)$
- ▶  $ALUctr<1> = !ALUOp1 + !func<2>$
- ▶  $ALUctr<0> = ALUOp1 \& (func<3> + func<0>)$

# The “Truth Table” for the Main Control

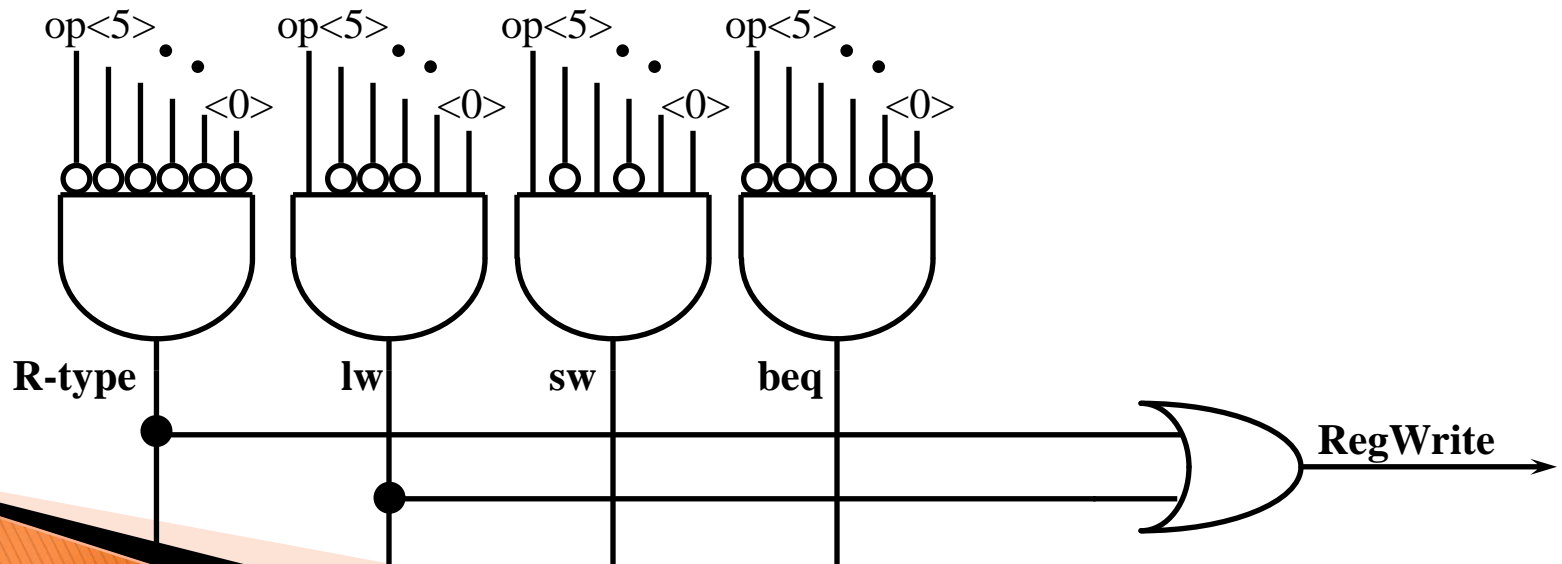


opcode	000000	100011	101011	000100
	R-type	lw	sw	beq
RegDst	1	0	x	x
ALUSrc	0	1	1	0
MemtoReg	0	1	x	x
<u>RegWrite</u>	<u>1</u>	<u>1</u>	<u>0</u>	<u>0</u>
MemRead	0	1	0	0
MemWrite	0	0	1	0
Branch	0	0	0	1
ALUOp (Symbolic)	“R-type”	Add	Add	Subtract
ALUOp1	1	0	0	0
ALUOp0	0	0	0	1

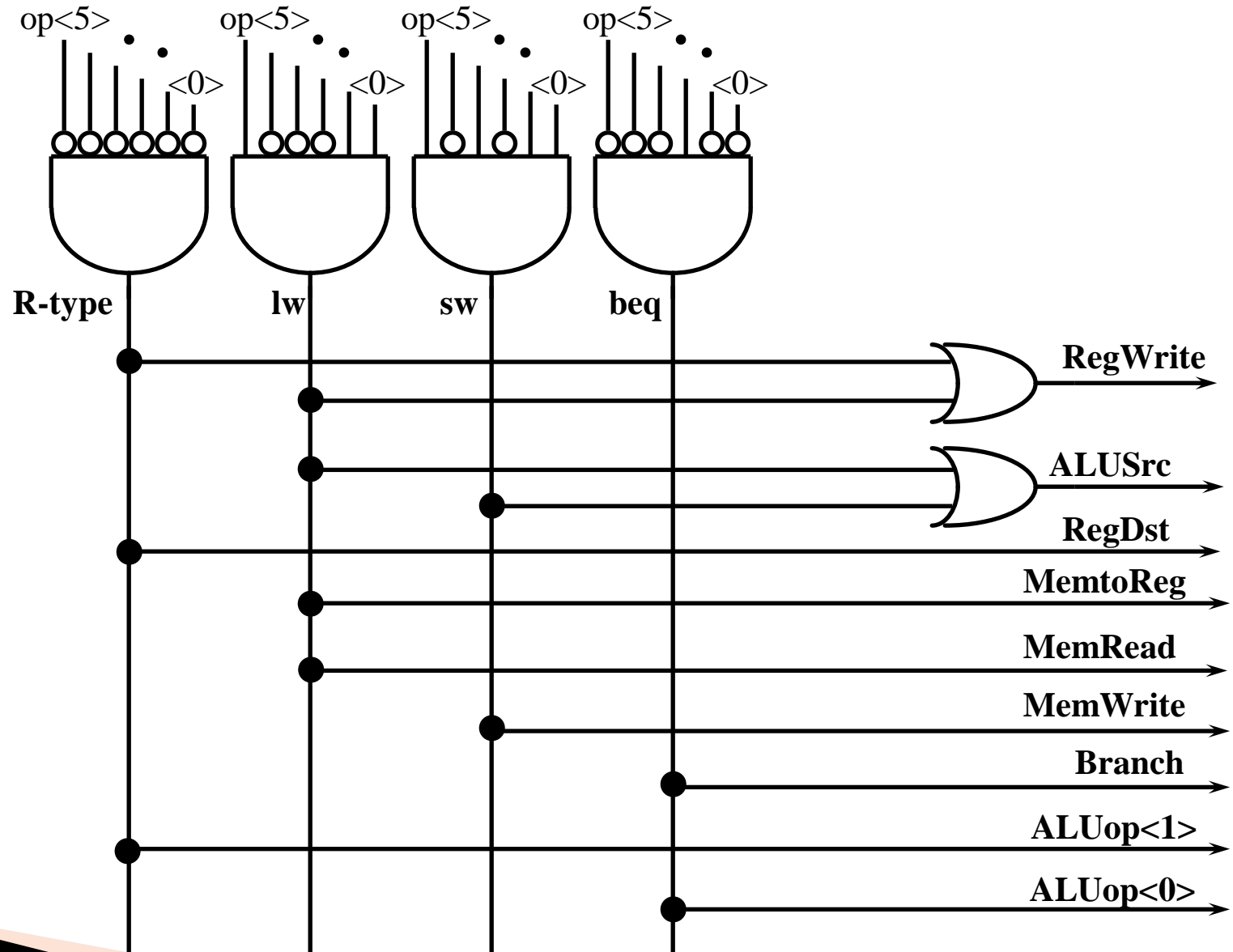
# The “Truth Table” for RegWrite

opcode	00 0000	10 0011	10 1011	00 0100
	<b>R-type</b>	<b>lw</b>	<b>sw</b>	<b>beq</b>
<b>RegWrite</b>	1	1	0	0

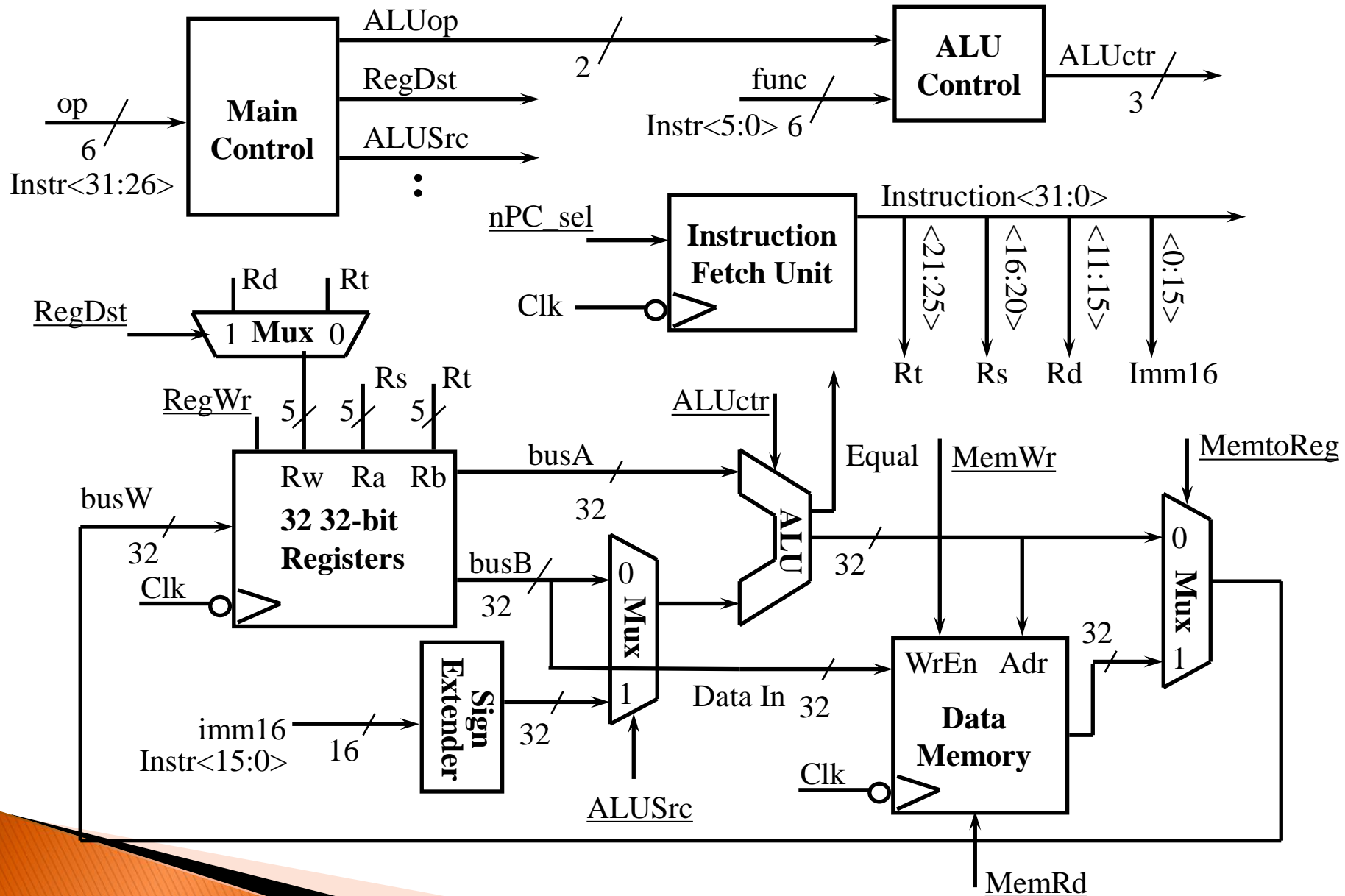
- ▶  $\text{RegWrite} = \text{R-type} + \text{lw}$   
 $= \neg \text{op}\langle 5 \rangle \ \& \ \neg \text{op}\langle 4 \rangle \ \& \ \neg \text{op}\langle 3 \rangle \ \& \ \neg \text{op}\langle 2 \rangle \ \& \ \neg \text{op}\langle 1 \rangle \ \& \ \neg \text{op}\langle 0 \rangle \ (\text{R-type})$   
 $+ \text{op}\langle 5 \rangle \ \& \ \neg \text{op}\langle 4 \rangle \ \& \ \neg \text{op}\langle 3 \rangle \ \& \ \neg \text{op}\langle 2 \rangle \ \& \ \text{op}\langle 1 \rangle \ \& \ \text{op}\langle 0 \rangle \ (\text{lw})$



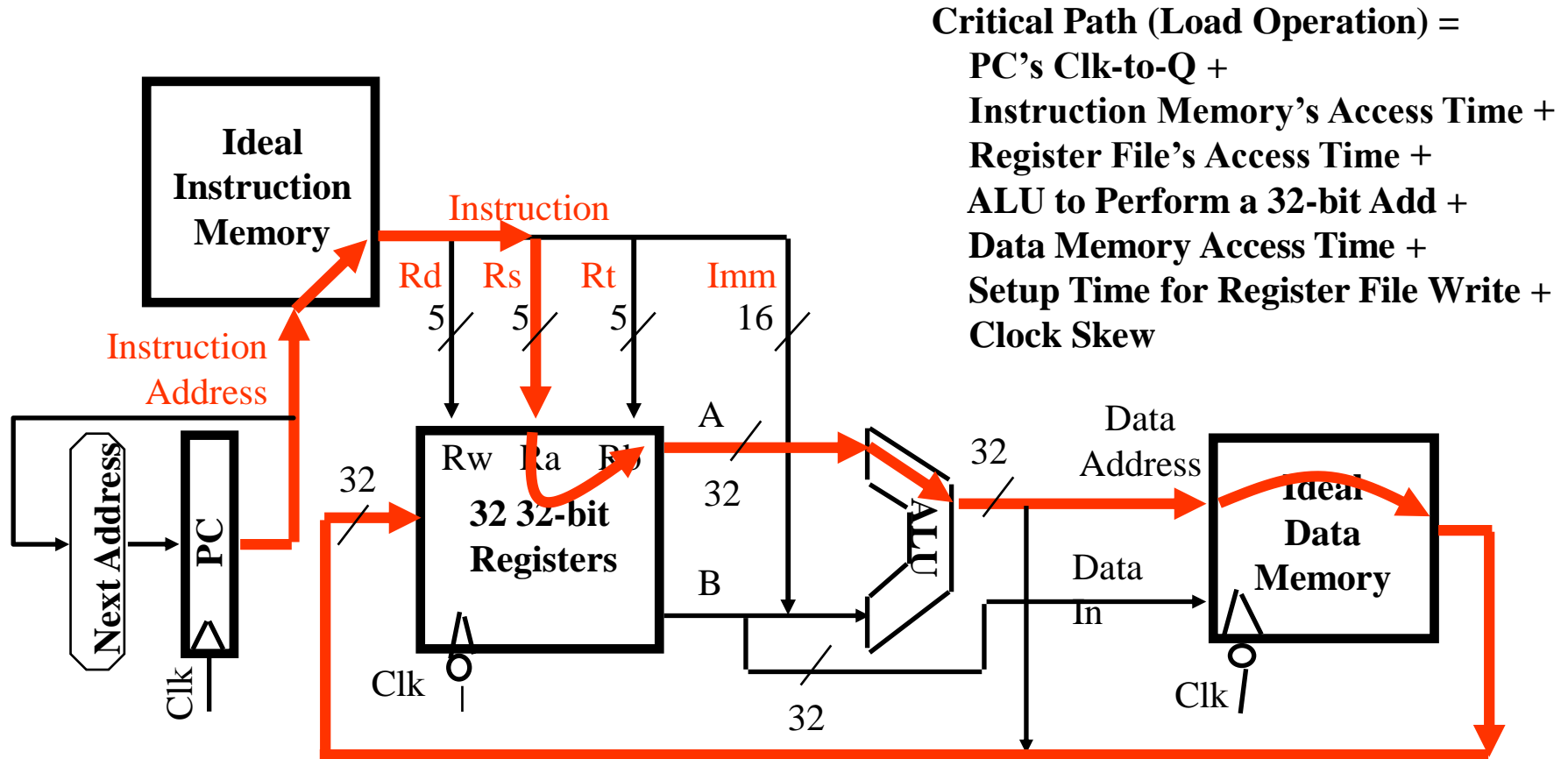
# Implementation of the Main Control



# Putting it All Together: A Single Cycle Processor



# An abstract view of the critical path – load instruction



Worst case delay for load is much longer than needed for all other instructions, yet this sets the cycle time.