Chapter 7 :: Microarchitecture

Quiz 5

1) False

2) True

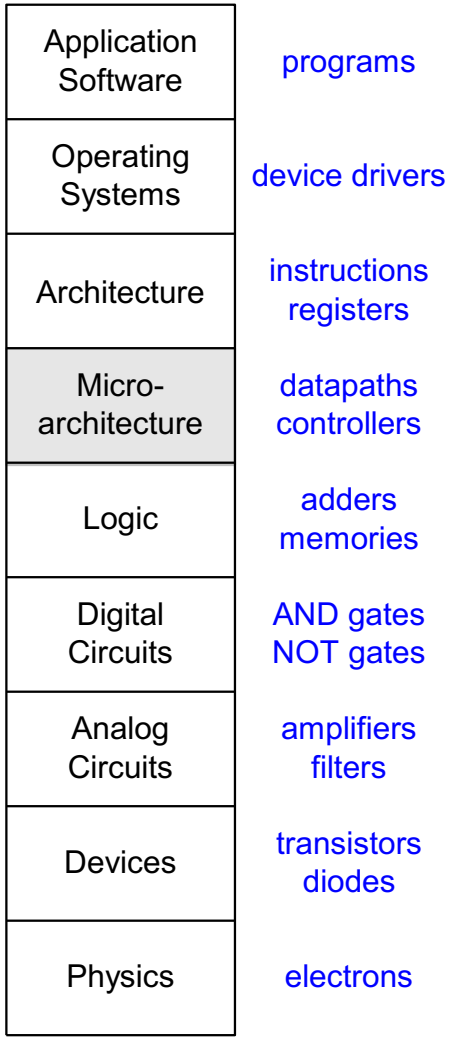
3) zero

Digital Design and Computer Architecture

Adapted from David Money Harris and Sarah L. Harris' book

Introduction

- Microarchitecture: how to implement an architecture in hardware
- Processor:
  - Datapath: functional blocks
  - Control: control signals



Microarchitecture

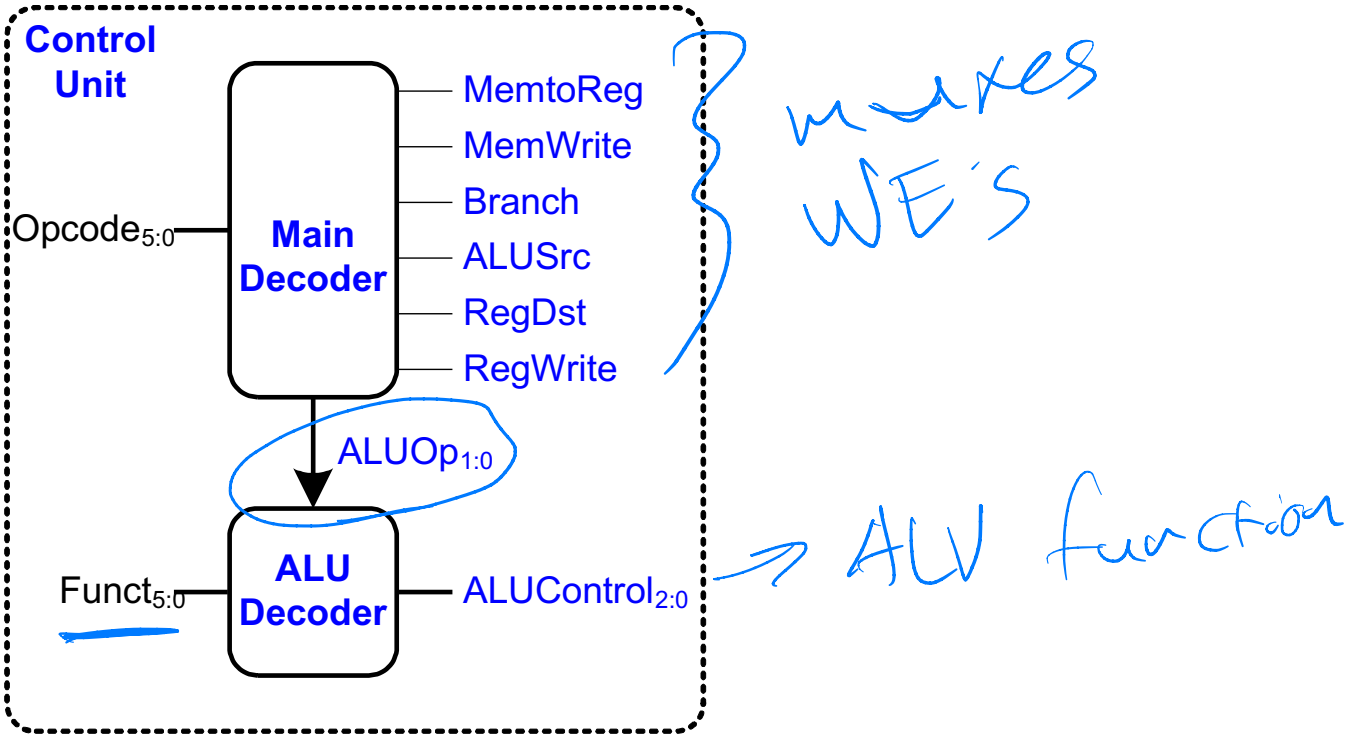
- Multiple implementations for a single architecture:
  - Single-cycle
    - Each instruction executes in a single cycle
  - Multicycle
    - Each instruction is broken up into a series of shorter steps
  - Pipelined
    - Each instruction is broken up into a series of steps
    - Multiple instructions execute at once.

7.3.2 Single Cycle Control Unit

Design

- Desired function:
  - Given an instruction word….
  - Generate control signals needed to execute instruction
- Implemented as a combinational logic function:
  - Inputs
    - Instruction word - `op` and `funct` fields
    - ALU status output - `Zero`
  - Outputs - processor control points
    - ALU control signals
    - Multiplexer control signals
    - Register File & memory control signal

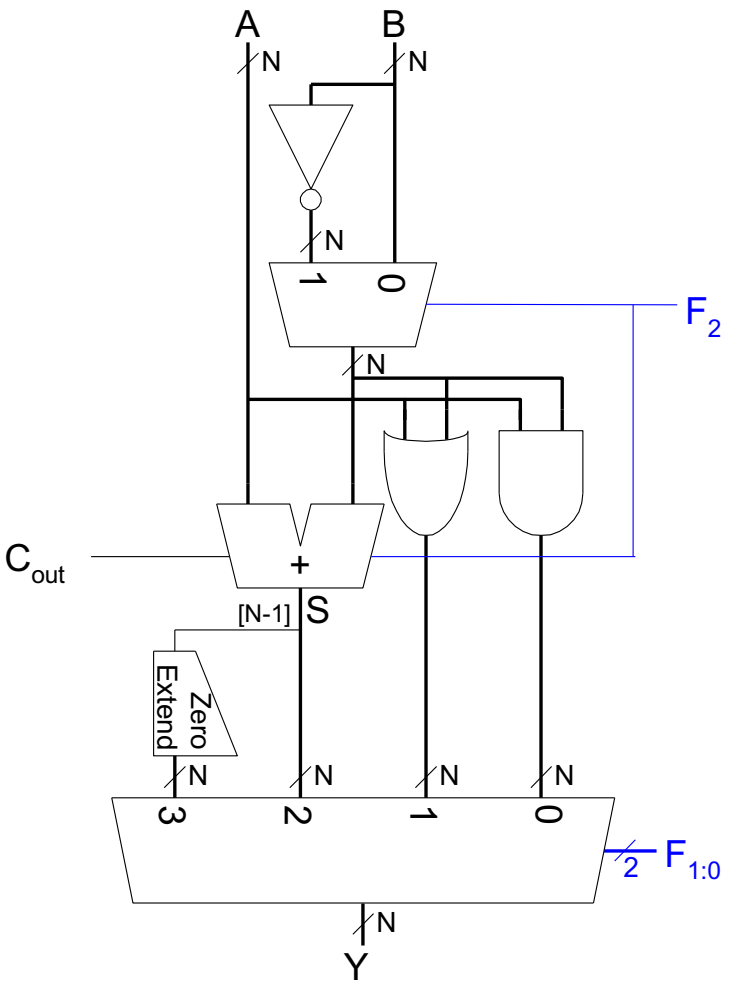
Control Unit



- 'funct' field only works for R type(addition, subtraction, etc). If lw, sw, beq need addition, we need create ALUOp1:0 to help.
- lw, sw, beq: need addition/subtraction,control information come s from *opcode, 31-26*
- R type: besides *opcode*, also needs *funct*.
- Simplify design by factoring the control unit into two block. Main decoder determines a 2-bit *ALUOp* signal. The ALU Decoder uses this *ALUOp* in conjunction with *funct* field to compute *ALUControl.*

Review from Ch5: ALU Design

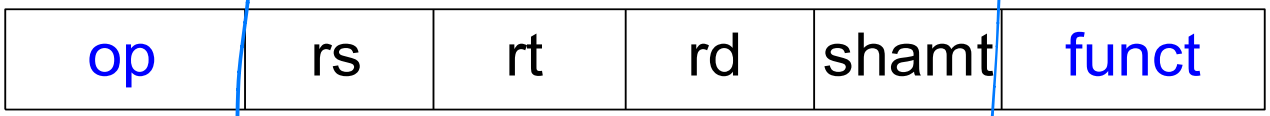
reminder



$F_{2:0}$	Function
000	A & B
001	A | B
010	*A + B*
011	not used
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Review: Instruction Formats

R-Type



I-Type

J-Type

Control Unit: ALU Decoder

$ALUOp_{1:0}$	Meaning
00	Add
01	Subtract
10	Look at Funct
11	Not Used

→ lw, sw } i-type
→ beq
← R-type

we decide the meaning

Control Unit: ALU Decoder

$ALUOp_{1:0}$	Meaning
00	Add
01	Subtract
10	Look at Funct *for R type*
11	Not Used, *to be used as don't care*

- Produce control signals in column 3 based on ALUOp and Funct

$ALUOp_{1:0}$	Funct	$ALUControl_{2:0}$ $_{F2:0}$ in ALU of Ch.5 Output	Instruction types
00	X	010 (Add)	lw,sw
X1	X	110 (Subtract)	beq
1X	100000 (add)	010 (Add)	R-type
1X	100010 (sub)	110 (Subtract)	
1X	100100 (and)	000 (And)	
1X	100101 (or)	001 (Or)	
1X	101010 (slt)	111 (SLT)	

Because *ALUOp* never uses 11, the truth table can use don't care's X1 and `1X to simply the design

Control Unit: Main Decoder

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01

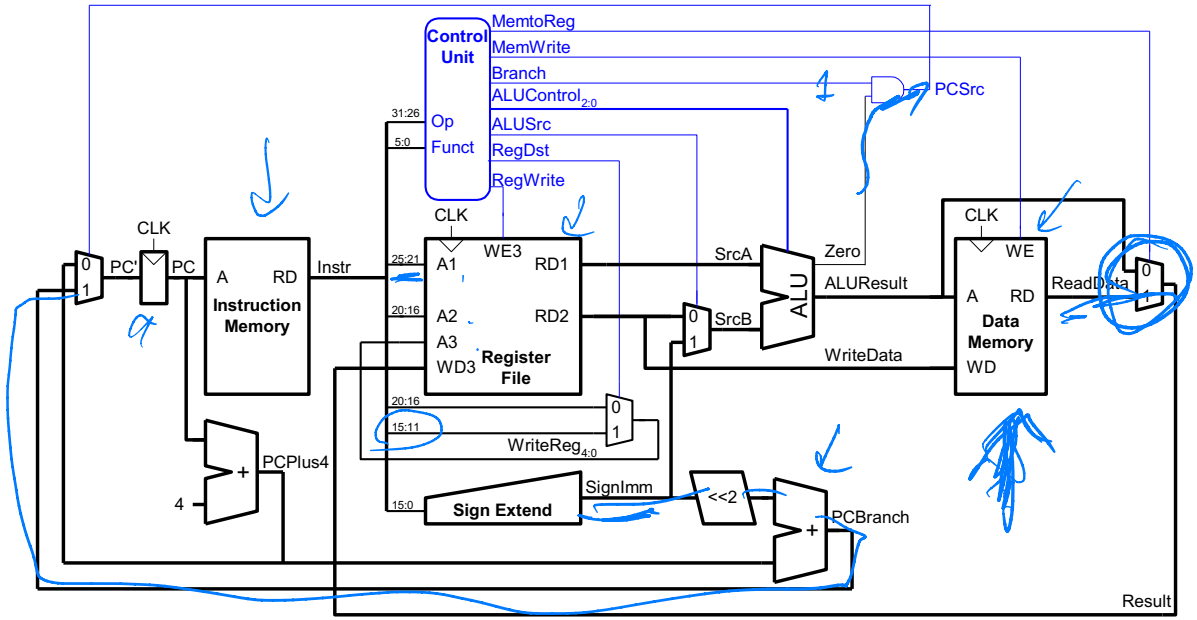
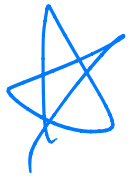


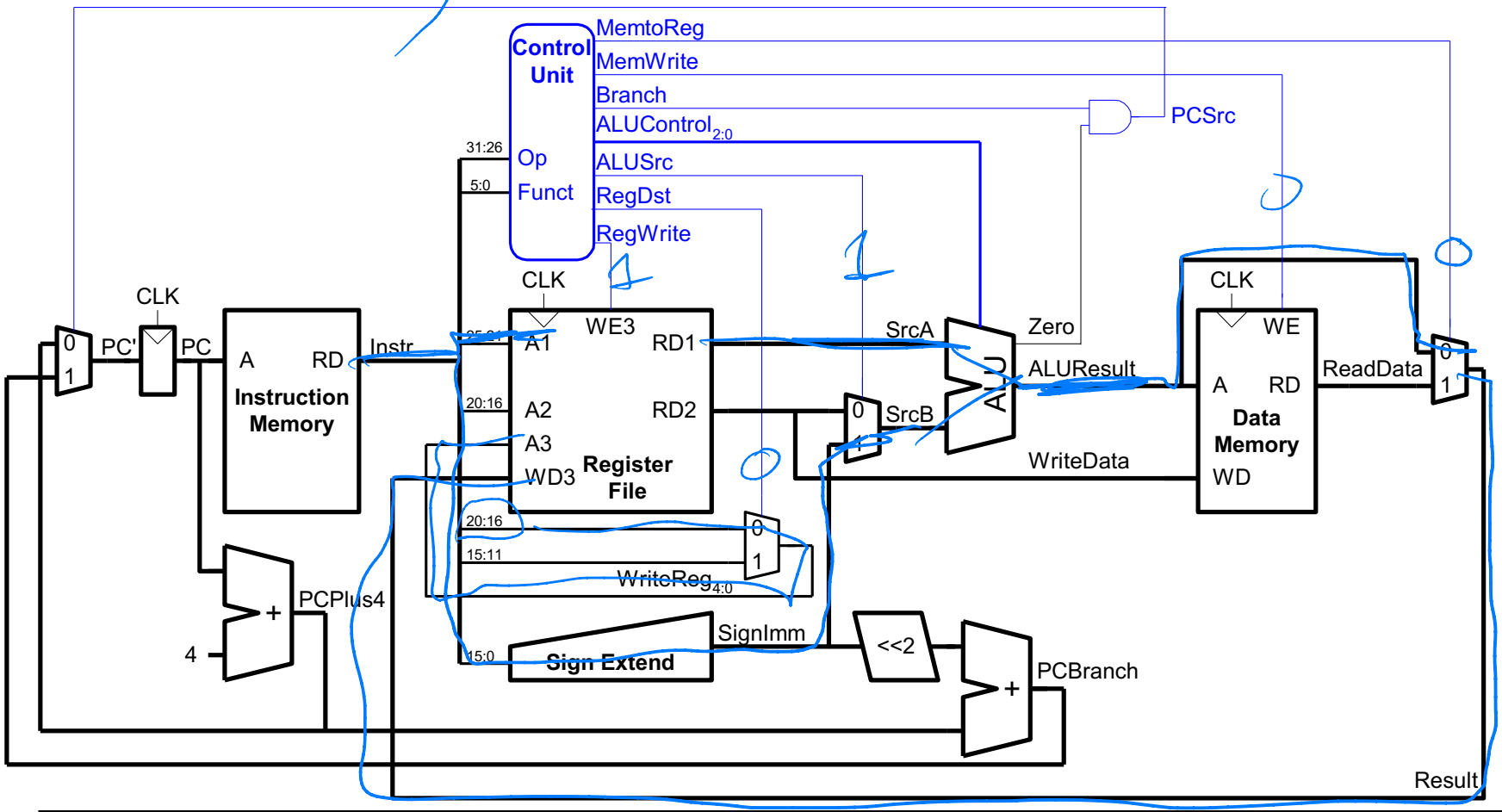
Table 7.3: Main decoder truth table(important)



Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	1	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01

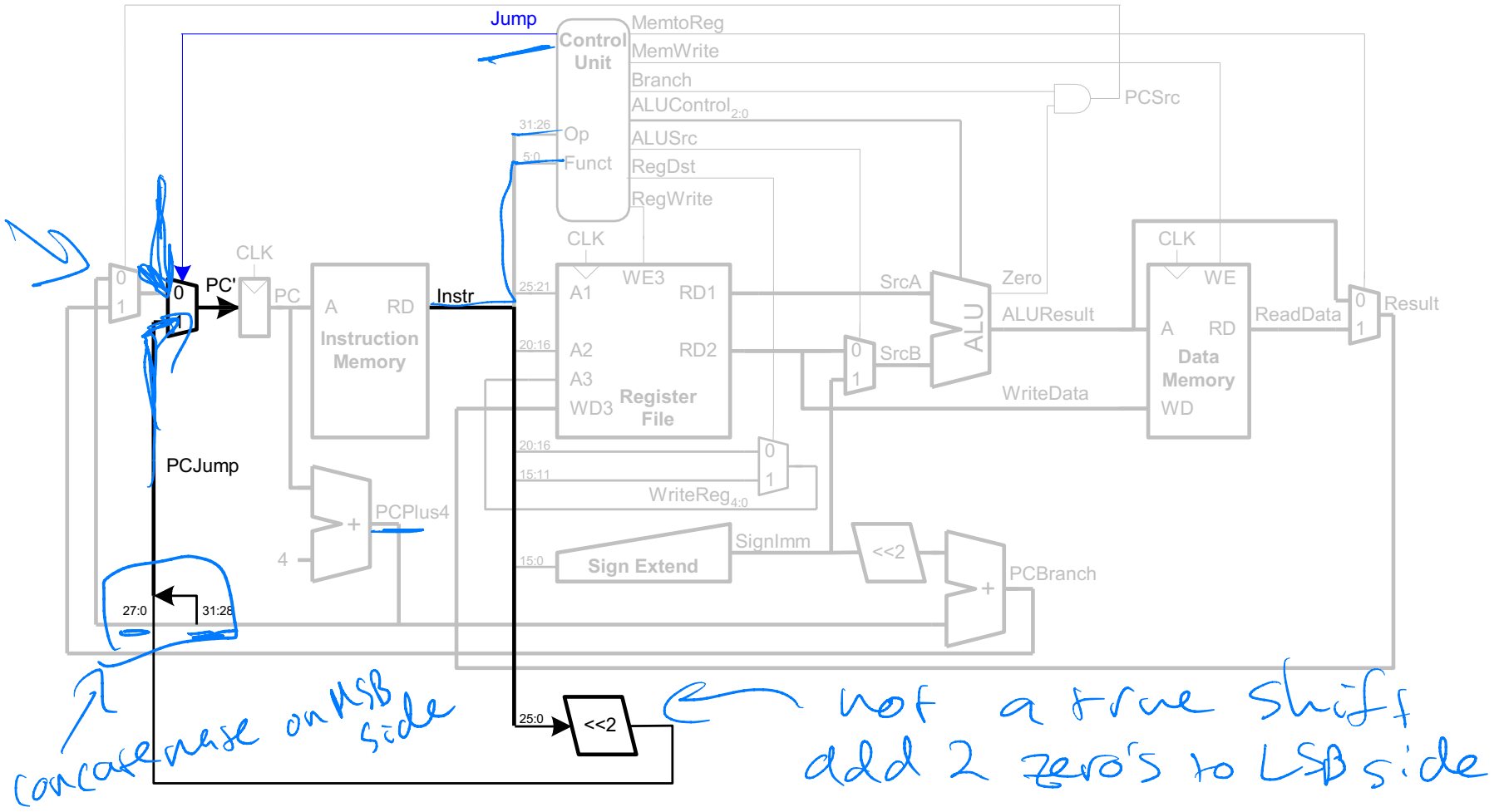
Extended Functionality: addi

- No change to datapath



Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
addi	001000	1	0	1	0	0	0	00
							7-<13>	

Extended Functionality: j



Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump
j	000010	0	X	X	X	0	X	XX	1

Chapter 7 :: Microarchitecture

Stop here 10/8

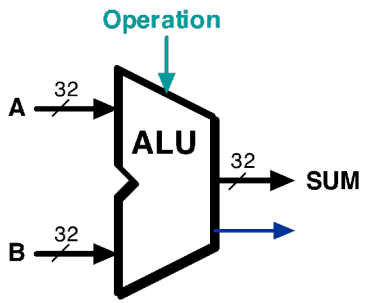
Digital Design and Computer Architecture

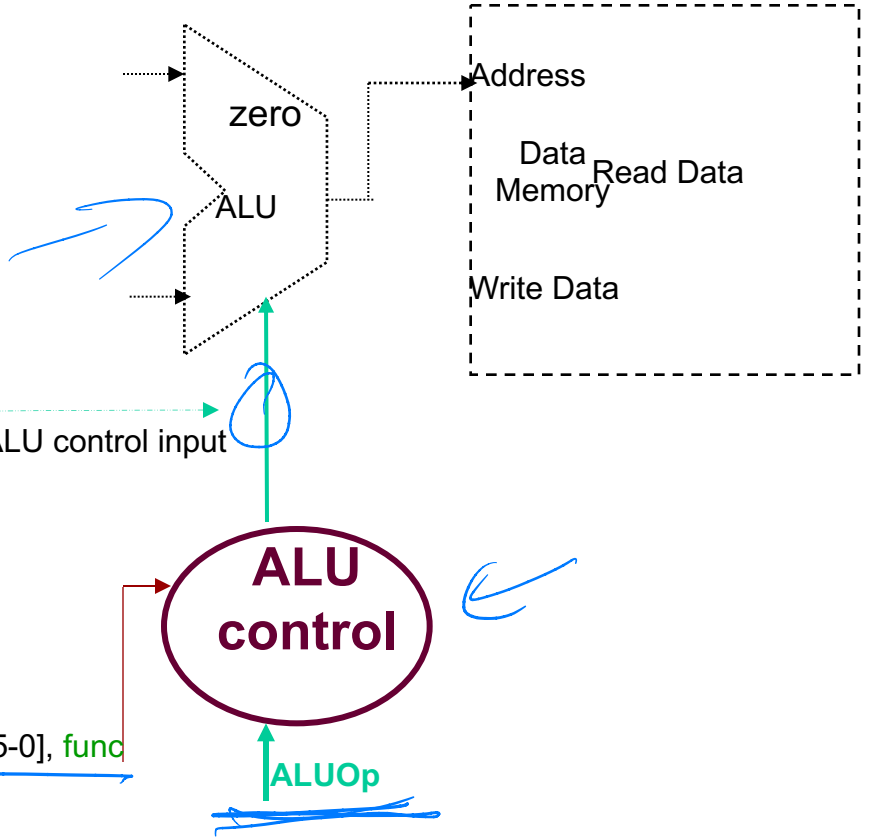
Adapted from David Money Harris and Sarah L. Harris' book

Appendix: from Hennessey and Patterson book:

Review : ALU Function

- Functions: Fig B.5.13 (also in Ch. 5 - p. 301)





ALU control input	Operation
000	AND
001	OR
010	add
110	subtract
111	set on less than

The only control signal that depends on the ***funct*** field is the ALU Operation signal
Idea: separate logic for ***ALU control***

Appendix: from Hennessey and Patterson book:

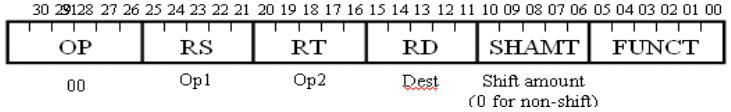
1. ALU Usage in Processor Design

- Usage depends on instruction type
  - Instruction type (specified by opcode)
  - `funct` field (r-type instructions only)
- Encode instruction (Register type)

Instr. type	Operation	funct

r-type	add	32
r-type	sub	34
r-type	and	36
r-type	or	37
r-type	slt	42

00: sll
02: srl
03: sra
04: sllv
06: srlv
07: srav
08: jr
24: mult
26: div
32: add
33: addu
34: sub
35: subu
36: and
37: or
38: xor
39: nor
42: slt



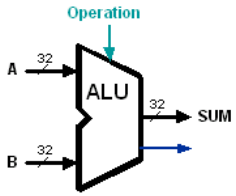
Appendix: from Hennessey and Patterson book:

ALU Usage in Processor Design

Encode instruction (Register type) in ALUOp signal/
Select ALUOp signal

Instr. type	Operation	funct	Desired Action	*ALUOp*	*ALU Ctl.*
r-type	add(32)	100000	add	*10*	*010*
r-type	Sub(34)	100010	subtract	*10*	*110*
r-type	And(36)	100100	and	*10*	*000*
r-type	Or(37)	100101	or	*10*	*001*
r-type	Slt(42)	101010	set on less than	*10*	*111*

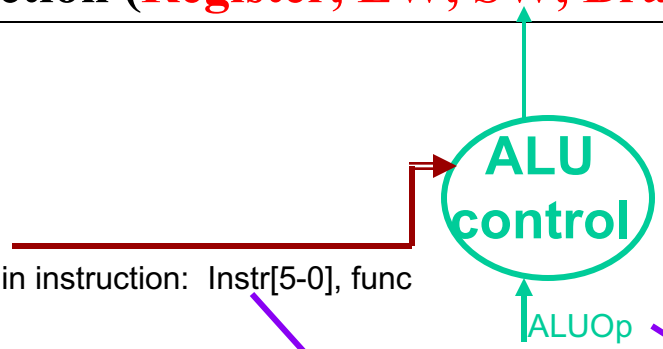
ALU control input	Function
000	AND
001	OR
010	add
110	subtract
111	set on less than



Appendix: from Hennessey and Patterson book:

ALU Usage in Processor Design

Encode instruction (Register, LW, SW, Branch) in ALUOp signal



ALUop is selected to indicates whether the operation shou
be add for loads and stores(lw, sw), subtract for beq,
or determined by the funct field (total 3 cases)

Indicats if it is addition or subtraction, etc

Instr. type	Operation	funct	Desired Action	ALUOp	ALU Ctl.
data transfer	*lw*	*XXXXXX*	*add*	*00*	*010*
data transfer	*sw*	*XXXXXX*	*add*	*00*	*010*
branch	*beq*	*XXXXXX*	*subtract*	*01*	*110*

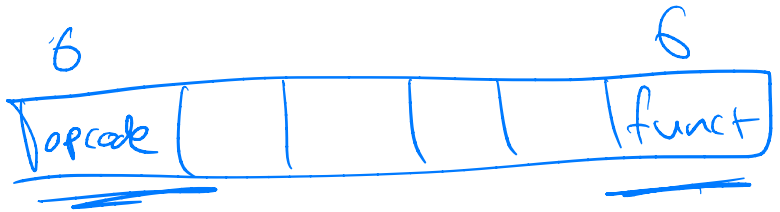
These two *add: compute memory address*

r-type	add	100000	add	10	*010*
r-type	sub	100010	subtract	10	*110*
r-type	and	100100	and	10	*000*
r-type	or	100101	or	10	*001*
r-type	slt	101010	set on less than	10	*111*

xxxxxx: don't care

Appendix: from Hennessey and Patterson book: To Design: ALU Control - Truth Table (Fig. 5-13)

- Choose columns of func, ALUop(inputs) and ALU Ctrl(output), and make a table
- Use don't care values to minimize length
  - Ignore F5, F4 (they are always "10")
  - Assume ALUOp never equals "11"



Corresponding to *lw, sw*. Two rows in previous slides merges here

func field

F5	F4	F3	F2	F1	F0	ALUOp1	ALUOp0	ALU Ctl.
X	X	X	X	X	X	0	0	010
X	X	X	X	X	X	X	1	110
X	X	0	0	0	0	1	X	010
X	X	0	0	1	0	1	X	110
X	X	0	1	0	0	1	X	000
X	X	0	1	0	1	1	X	001
X	X	1	0	1	0	1	X	111

Input

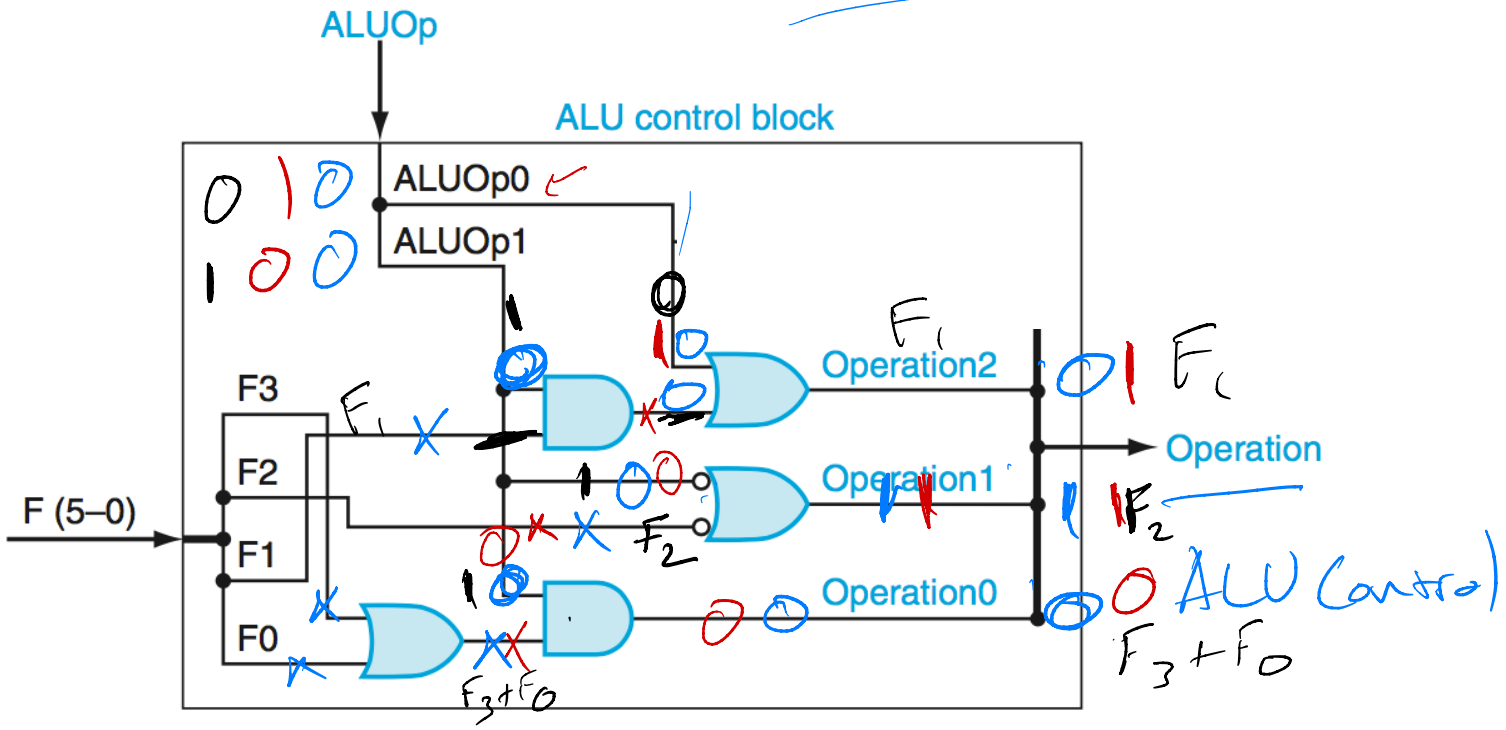
output

ALUop is the input to ALU control, we will explain how to produce ALUop later.

Appendix: from Hennessey and Patterson book:

ALU Control - Implementation

- Zybook, 12.2, figure 12.2.3 (No Dontcares)



Review: from truth table, what is the logic? What is the schematic?

x	y	output
0	0	0
0	1	0
1	0	0
1	1	1

Appendix: from Hennessey and Patterson book:
Control signals

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

FIGURE 5.16 The effect of each of the seven control signals. When the 1-bit control to a two-way multiplexor is asserted, the multiplexor selects the input corresponding to 1. Otherwise, if the control is deasserted, the multiplexor selects the 0 input. Remember that the state elements all have the clock as an implicit input and that the clock is used in controlling writes. The clock is never gated externally to a state element, since this can create timing problems. (See Appendix B for further discussion of this problem.)

Appendix: from Hennessey and Patterson book:
Control Unit Implementation

- Review: Opcodes for key instructions
- Control Unit Truth Table: (or see Fig. 5-18, p. 308)

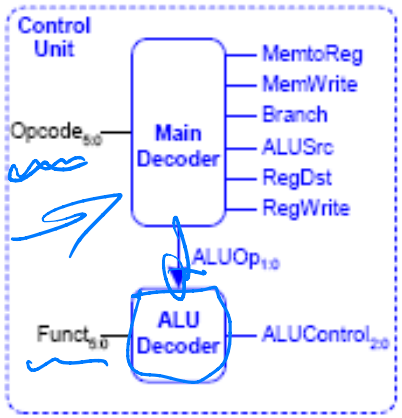


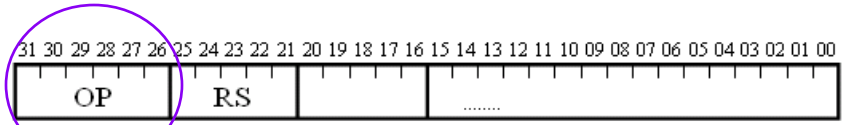
FIGURE 7.12 Control unit internal structure

Input

Various output

	OP5 ... OP1 OP0 Instruction	RegDst	ALUSrc	to-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R	000000	1	0	0	1	0	0	0	1	0
lw	100011	0	1	1	1	1	0	0	0	0
sw	101011	X	1	X	0	0	1	0	0	0

04: beq
05: bne
06: blez
07: bgtz
08: addi
09: addiu
10: slti
11: sltiu
12: andi
13: ori
14: xori
32: lb
35: lw
40: sb
43: sw



ALUop indicates whether the operation should be add for loads and stores, subtract for beq, or determined by the funct field

aluop	meaning
00	Add
01	Subtract
10	Look at funct field
11	n/a

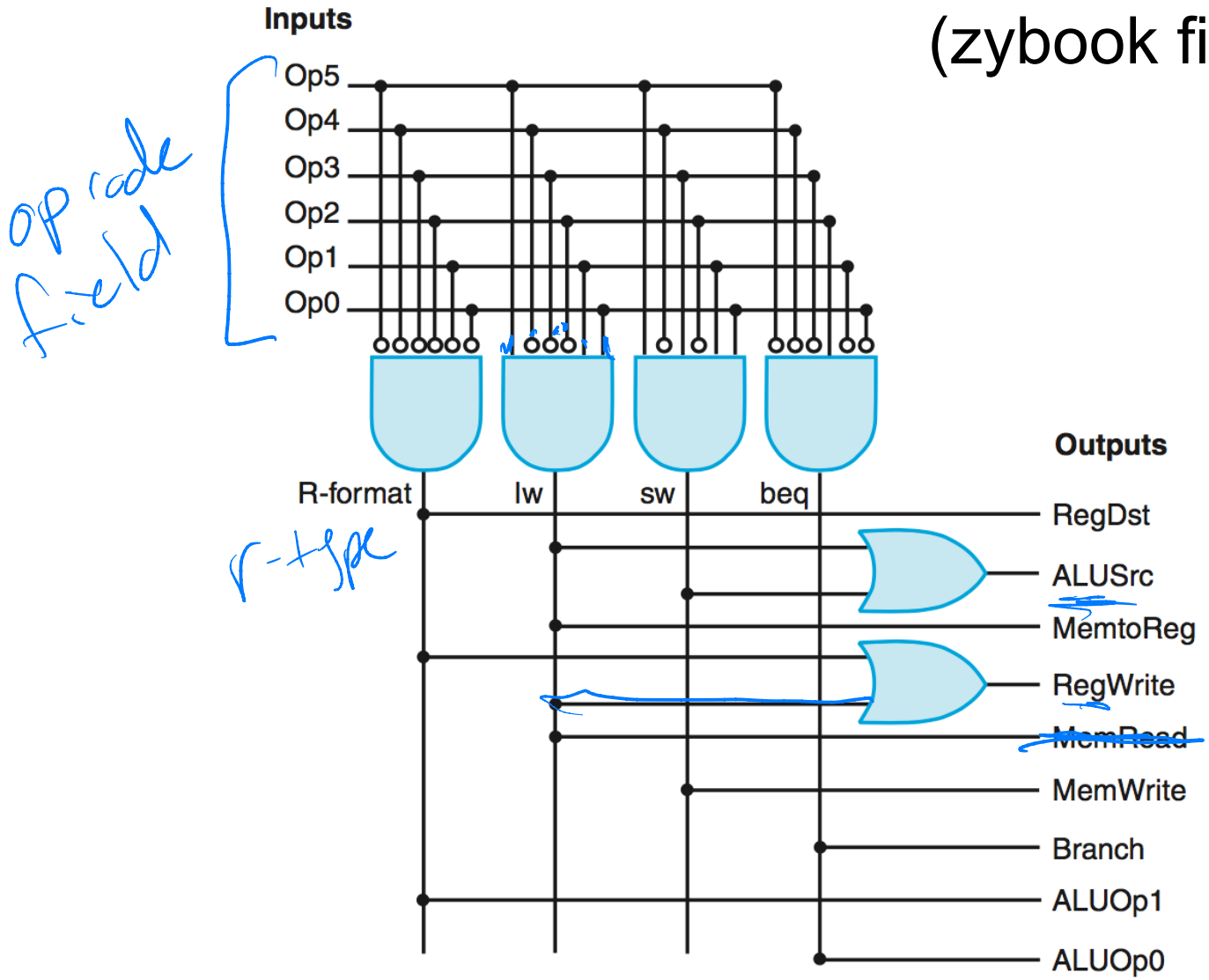
TABLE 7.1 *ALUOp* encoding

Appendix: from Hennessey and Patterson book:

Control Unit Implementation

- Implementation: Decoder + 2 Gates

(zybook figure 12.2.5)



Review: Processor Performance

1 for single cycle processor

Program Execution Time

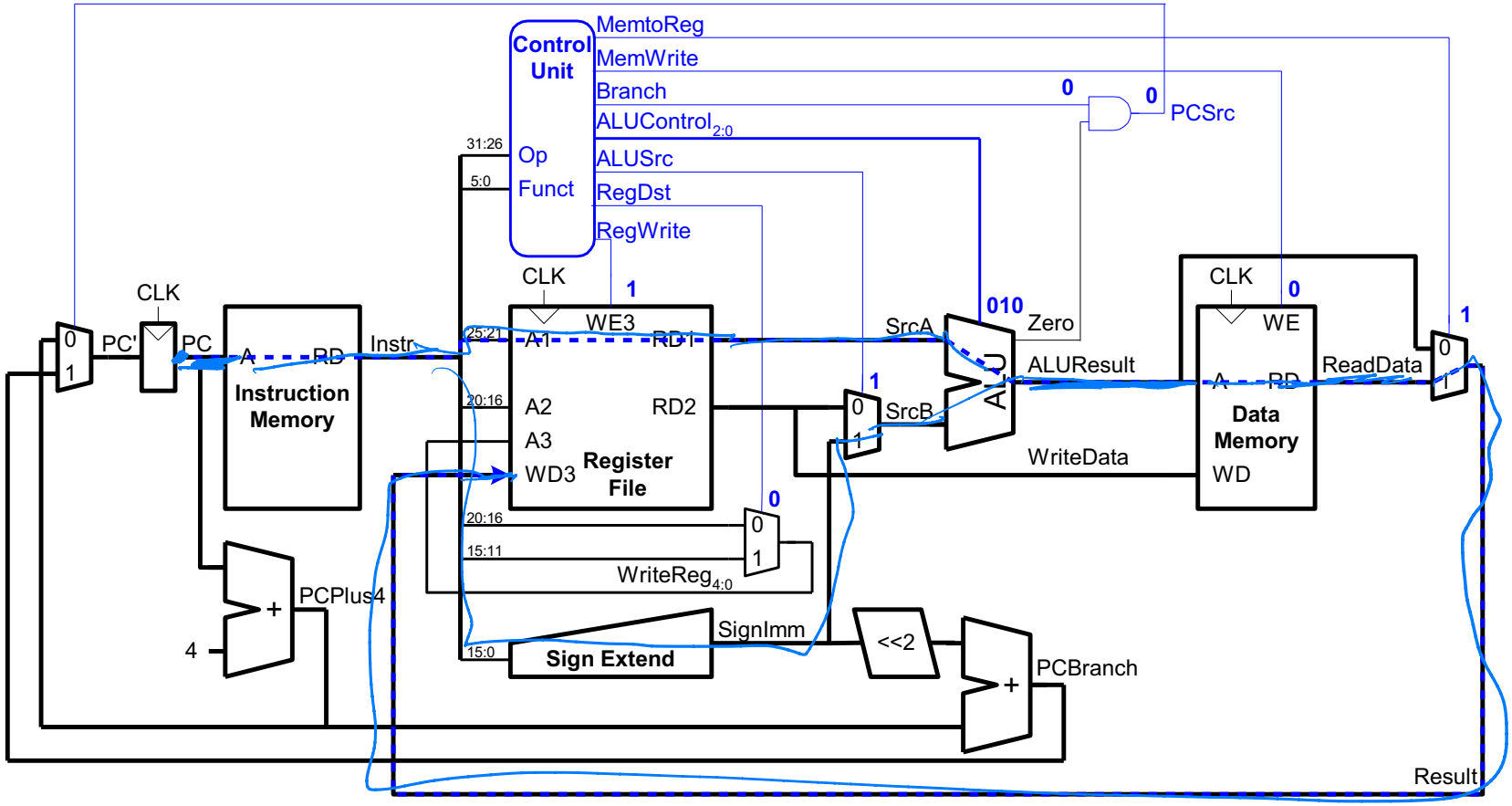
= (# instructions)(cycles/instruction)(seconds/cycle)

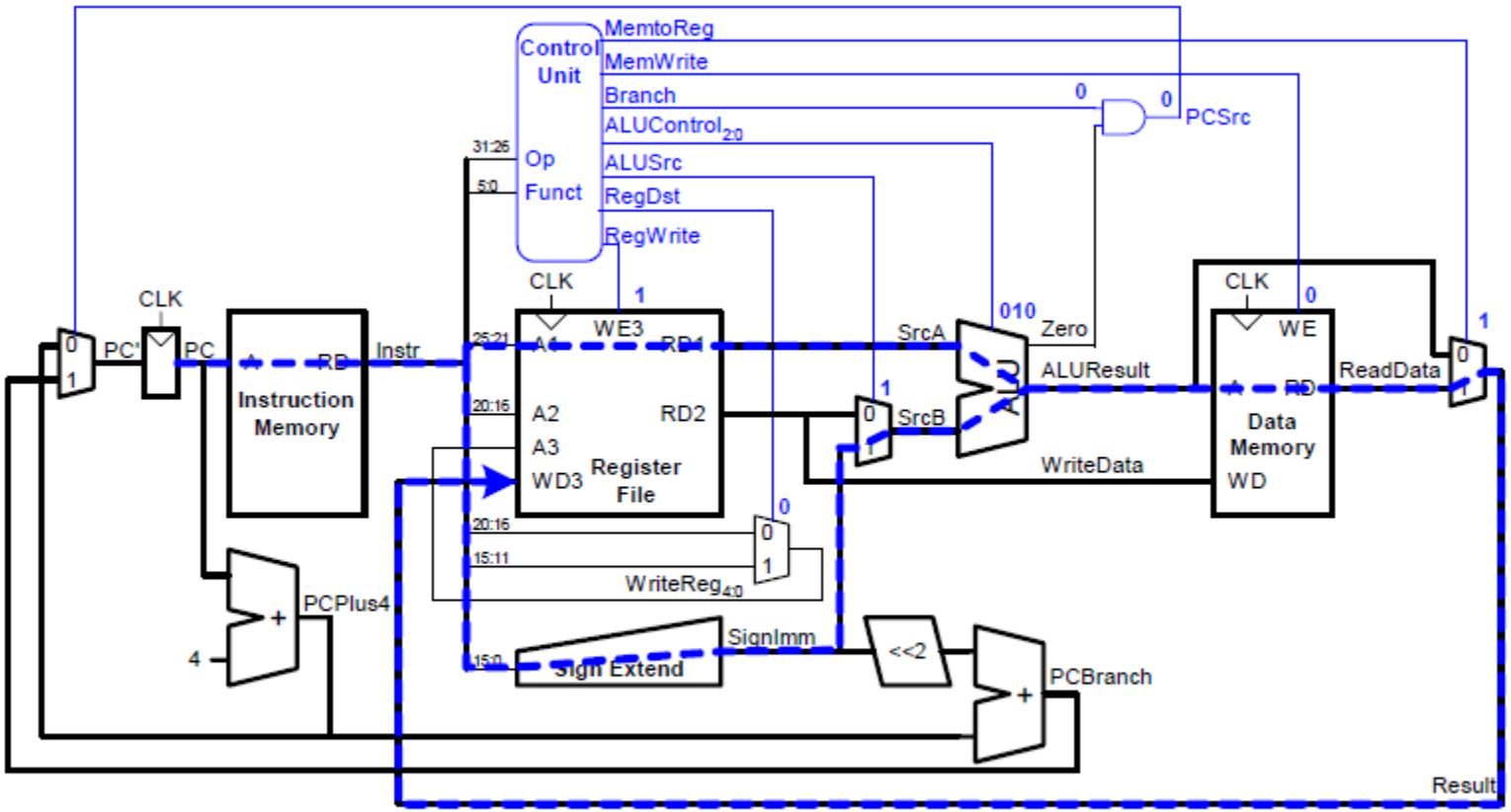
= # instructions x CPI x T_C

executed

Single-Cycle Performance

- T_C is limited by the critical path (`lw`) ⇒ longest instruction





Critical path for `lw` instruction

Single-Cycle Performance

- Single-cycle critical path:

$$T_c = t_{pcq\_PC} + t_{\text{mem}} + \max\left(t_{RF\text{read}}, t_{sext} + t_{\text{mux}}\right) + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$

not max [TRFread, Tset] + tmux

- In most implementations, limiting paths are:
  - memory, ALU, register file.
  - $T_c = t_{pcq\_PC} + 2t_{\text{mem}} + t_{RF\text{read}} + t_{\text{mux}} + t_{\text{ALU}} + t_{RF\text{setup}}$

Single-Cycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
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

how long until valid output after clock edge

$T_c =$

Single-Cycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
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

$$T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{RFsetup}$$
$$= [30 + 2(250) + 150 + 25 + 200 + 20]\ ps$$
$$= 925\ ps$$

Note that *lw* has longest path. There are two memory reads: read from instruction memory, read from data memory

Please note an errata IN FIRST EDITION : P 380 Equation 7.2: should be: Tc = tpcq_PC + tmem + max[tFRread, tsext + tmux] + tALU + tmem + tmux +tRFsetup.

This affects other equations and examples : Equation 7.3: should be "...+ tmux" (not 2tmux), Examples 7.4, 7.8, and 7.10: 95 seconds → 92.5 seconds, Example 7.11: 900 → 875 ps for the singlecycle propagation delay.

Single-Cycle Performance Example

- For a program with 100 billion instructions executing on a single-cycle MIPS processor,

Execution Time =

$$(100 \times 10^9) \times 1 \times (925 \times 10^{-12})$$

instr CPI

Single-Cycle Performance Example

- For a program with 100 billion instructions executing on a single-cycle MIPS processor,

Execution Time = # instructions x CPI x T_C

$= (100 \times 10^9)(1)(\mathbf{925 \times 10^{-12}\ s})$

$= 92.5$ seconds

baseline

Multicycle MIPS Processor

- Single-cycle microarchitecture:
  + simple
  - cycle time limited by longest instruction (`lw`)
  - two adders/ALUs and two memories
- Multicycle microarchitecture:
  + higher clock speed
  + simpler instructions run faster
  + reuse expensive hardware on multiple cycles
  - sequencing overhead paid many times
- Same design steps: datapath & control