The Processor

COS375 / ELE375

David I. August

Acknowledgements:
Mohammad Shahrad
Margaret Martonosi

Fall 2022

Prof. David August (august@princeton.edu)
Office Hours: After class and by appointment

Course Staff: Your Other Instructor

(at age 15)

1999-Present:

- Professor of Computer Science at Princeton
- Compiler and Computer Architecture Research (more later)
- Lead Liberty Research Group

1993-1999:

- M.S. and Ph.D. Electrical Engineering from University of Illinois
- Ph.D. Thesis Topic: Predication (will discuss briefly later in semester)

Ongoing consulting and research relationships with companies like Intel, Apple, Google, Samsung, Sony, etc.

GodBolt.Org

COMPILER EXPLORER Add... More Templates

C source #1

```
1  int FromLecture3(int h, int *A) {
2      A[12] = h + A[8];
3      return A[12];
4  }
```

Not Optimized

```
RISC-V (64-bits) gcc 15.2.0 (Editor #1)
RISC-V (64-bits) gcc 15.
Compiler options...

1   FromLecture3:
2           addi    sp,sp,-32
3           sd      ra,24(sp)
4           sd      s0,16(sp)
5           addi    s0,sp,32
6           mv      a5,a0
7           sd      a1,-32(s0)
8           sw      a5,-20(s0)
9           ld      a5,-32(s0)
10          addi    a5,a5,32
11          lw      a4,0(a5)
12          ld      a5,-32(s0)
13          addi    a5,a5,48
14          lw      a3,-20(s0)
15          addw    a4,a3,a4
16          sext.w  a4,a4
17          sw      a4,0(a5)
18          ld      a5,-32(s0)
19          lw      a5,48(a5)
20          mv      a0,a5
21          ld      ra,24(sp)
22          ld      s0,16(sp)
23          addi    sp,sp,32
24          jr      ra
```

Optimized

```
RISC-V (64-bits) gcc 15.2.0 (Editor #1)
RISC-V (64-bits) gcc 15.
-O2

1   FromLecture3:
2           lw      a5,32(a1)
3           addw    a0,a5,a0
4           sw      a0,48(a1)
5           ret
```

Let's Build a RISC-V Processor!

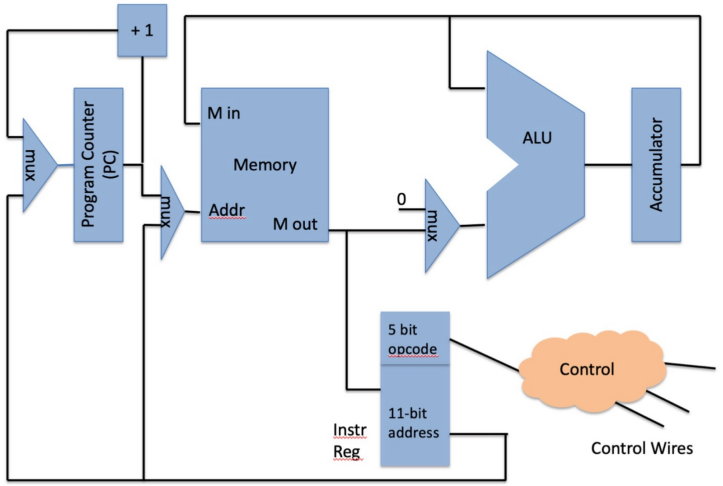
Support a subset of instructions:

- Arithmetic/logical (R-type)
- Memory: lw, sw
- Control flow: beq

Datapath, then control

Datapath

1949 Datapath:



Datapath

"The component of the processor that performs arithmetic operations" – P&H p. 20

datapath The component of the processor that performs arithmetic operations.

Datapath

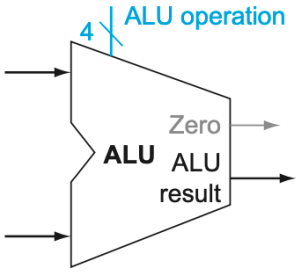
"The collection of state elements, computation elements, and interconnections that transform data in the processor during execution." – Me

Datapath Computational Elements

Blue vs. Black?

ALU:

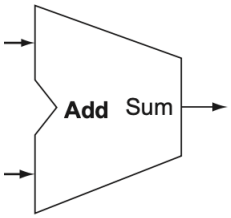
- Supports different operations
- In RISC-V, ALU has an output for zero equality. Why?



ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract

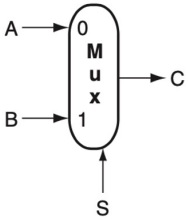
Adder:

- Not an ALU, just add
- Why useful in RISC-V?



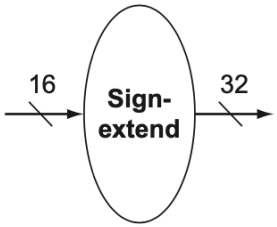
The Magical Mux:

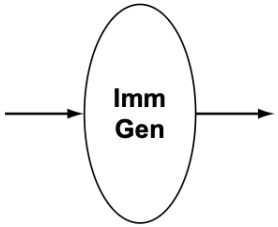
- Mux selects an input based on the control line.
- Why useful in any datapath?



Sign-Extender and Imm-Gen:

- This sign-extender replicates bit [15] 16 times.
- What does Imm Gen do in RISC-V?





I	imm[11:0]		rs1	funct3	rd	Opcode
S	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode

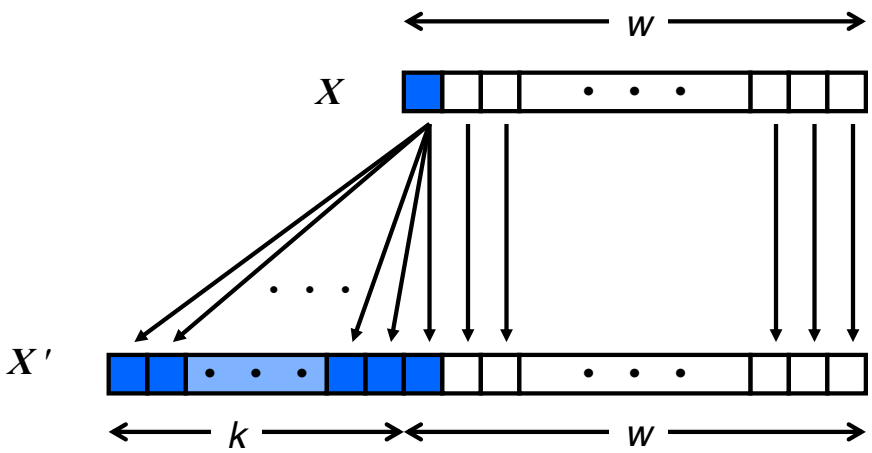
ALL COMBINATIONAL

Sign Extension vs. Zero Extension

```
char minusFour = -4;
short moreBits;
moreBits = (short)minusFour;
```

Given w bit signed integer,
return equivalent w+k bit signed integer.

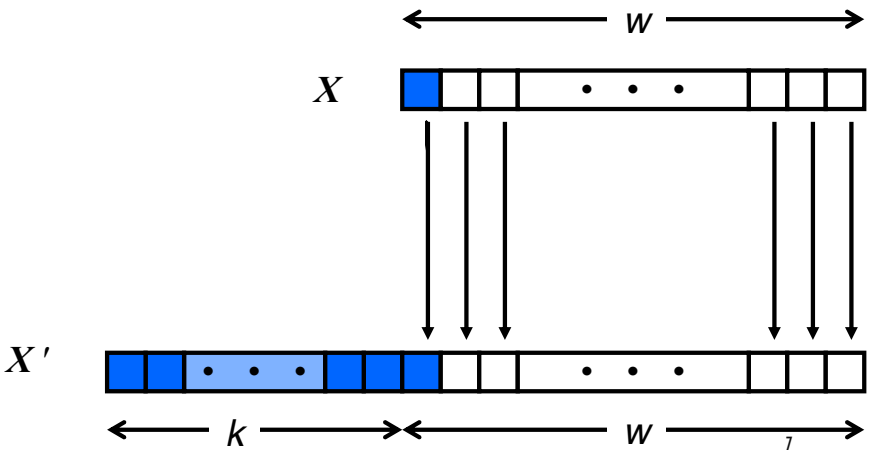
Sign Extend:




```
unsigned char Four = 4;
unsigned short moreBits;
moreBits = (unsigned short)Four;
```

Given w bit unsigned integer,
return equivalent w+k bit unsigned integer.

Sign Extend:



REVIEW SLIDES

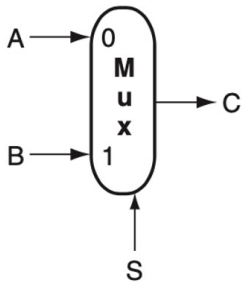
Combinational Circuits

- The ALU, Adder, Sign-Extender, and Mux are all Combinational Circuits.
- A Combinational Circuit is a digital circuit whose outputs depend solely on the present combination of the circuit input's values. No memory (i.e., no state).
- A combinational circuit implements a Boolean function.
- A Boolean function can be represented by enumerating the function output value for all possible input values: A Truth Table

Mux Example:

Inputs: s, i0, i1

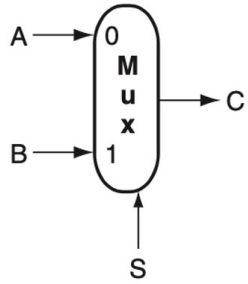
Outputs: out



S	A	B	C
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

S	A	B	C

Computation Element: Mux Boolean Expression



- A Boolean function can be represented by enumerating the function output value for all possible input values: A Truth Table
- A Boolean function can be represented by Boolean Algebra.
- Truth Table to Sum-Of-Products Boolean Expression Conversion:

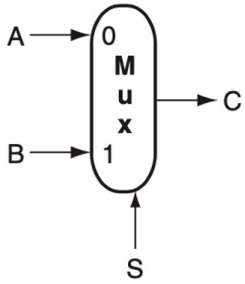
S	A	B	C	
0	0	0	0	
0	0	1	0	
0	1	0	1	S'AB'
0	1	1	1	S'AB
1	0	0	0	
1	0	1	1	SA'B
1	1	0	0	
1	1	1	1	SAB

S'AB'+ S'AB + SA'B + SAB

S	A	B	C	S'A'	S'A	SB'	SB	S'A+SB
0	0	X	0	1	0	0	0	0
0	1	X	1	0	1	0	0	1
1	X	0	0	0	0	1	0	0
1	X	1	1	0	0	0	1	1

S'A+SB

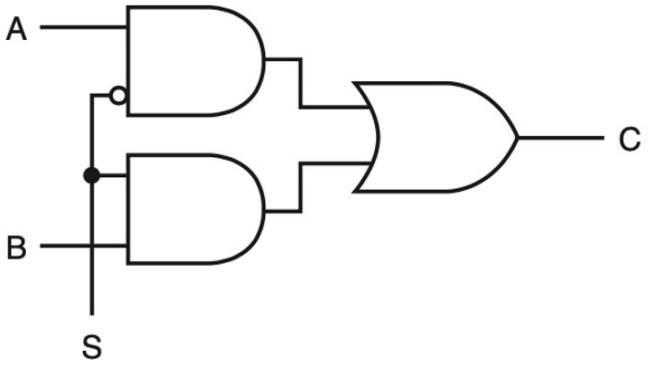
Computation Element: Mux Circuit!



- A Boolean function can be represented by enumerating the function output value for all possible input values: A Truth Table
- A Boolean function can be represented by Boolean Algebra.
- A Boolean function can be represented and implemented by a Circuit.

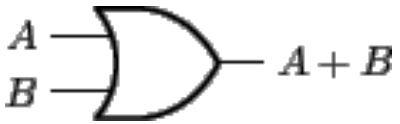
S	A	B	C
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1

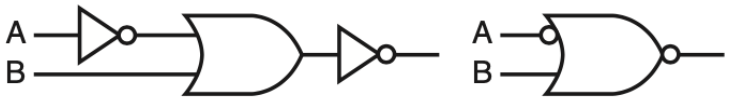
$C = S'A + SB$



Symbology:

THIS IS AS LOW AS WE GO...

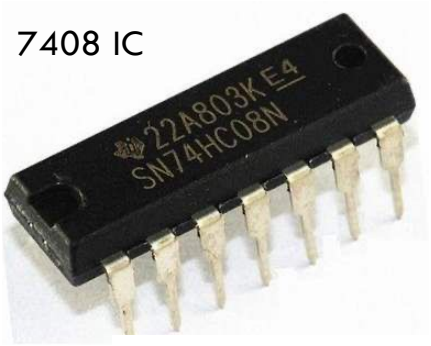




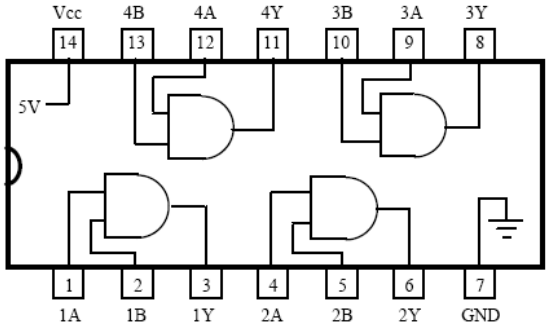
You can physically build these circuits!

Dedicated Integrated Circuits (IC)

7408 IC

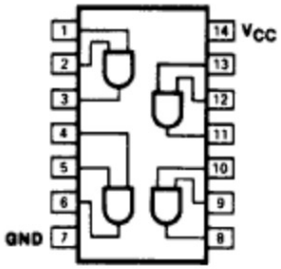


Pinout of the 7408 IC

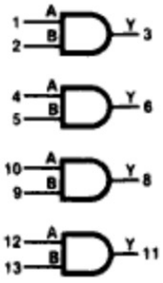


7408 • 74S08 • 74LS08
QUAD 2-INPUT AND GATE

PIN ASSIGNMENT



LOGIC DIAGRAM



TRUTH TABLE

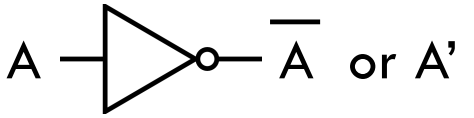
INPUTS		OUTPUT
A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

H = HIGH voltage level
L = LOW voltage level

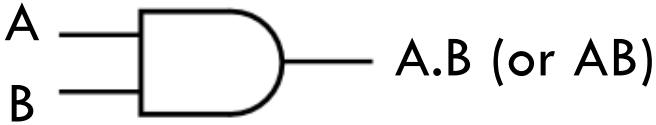
Boolean Algebra

George Boole

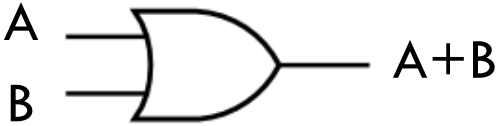
- Introduced by George Boole in his 1847 book
- Fundamental operators:
  - NOT (negation) [0→1, 1→0]



  - AND (output is 1 if both inputs are 1)



  - OR (output is 1 if any input is 1)



These three operators are Boolean Functions.

These three simple Boolean Functions are universal (e.g., they can build all other functions).

Boolean Algebra Laws

MUX:

S'AB'+ S'AB + SA'B + SAB = S'A+SB

S'AB'+ S'AB + SA'B + SAB = S'A(B'+B) + S(A'+A)B (Distribution)

S'A(B'+B) + S(A'+A)B = S'A+SB (Inversion)

Convince yourself these are true with truth tables: $\overline{A+B} = \overline{A}\,.\,\overline{B}$

A	B	A+B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$
0	0					
0	1					
1	0					
1	1					

Identity: $A\,.\,1 = A, \quad A + 0 = A$

Zero and one: $A\,.\,0 = 0, \quad A + 1 = 1$

Inversion: $A\,.\,\overline{A} = 0, \quad A + \overline{A} = 1$

Idempotence: $A\,.\,A = A \quad A + A = A$

Commutativity: $A\,.\,B = B\,.\,A, \qquad A + B = B + A$

Associativity: $A\,.\,(B\,.\,C) = (A\,.\,B).\,C, \qquad A + (B + C) = (A\ +\ B) + C$

Distribution: $A\,.\,(B + C) = A\,.\,B + A\,.\,C, \qquad A + (B\,.\,C) = (A\ +\ B).\,(A + C)$

DeMorgan's theorems: $\overline{A\,.\,B} = \overline{A} + \overline{B}, \qquad \overline{A + B} = \overline{A}\,.\,\overline{B}$

(You'll find this one very useful.)

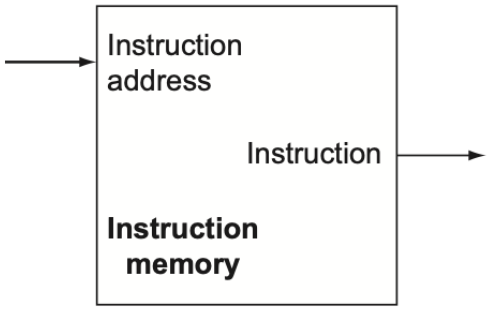
END REVIEW SLIDES

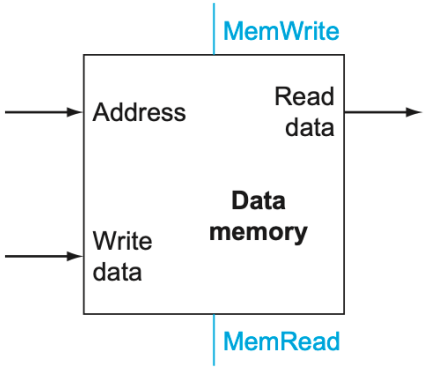
Datapath State Elements

RISC-V Register File:

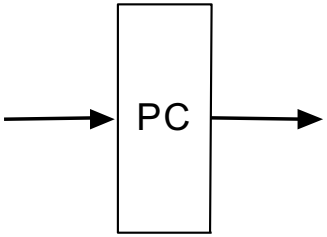
- Why "Read register 1", "Read register 2" and "Write register" inputs?
- Why 5 bits each?

Instruction and Data Memory (for now and again much later):

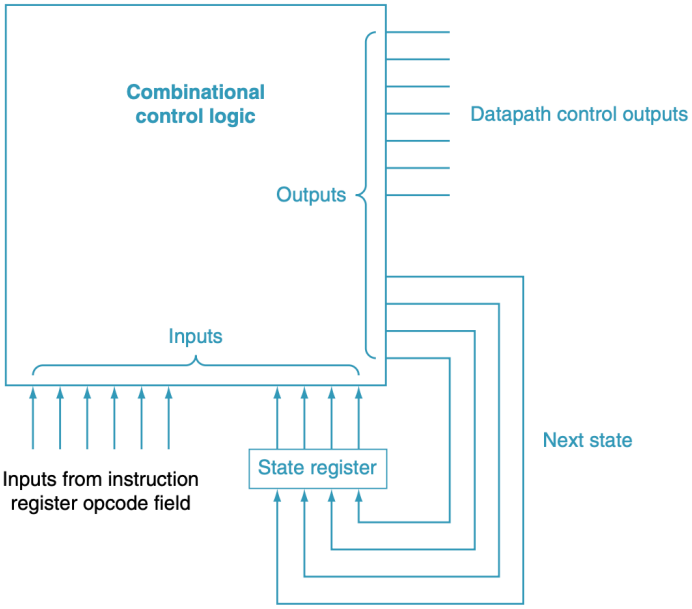




Program Counter (PC):

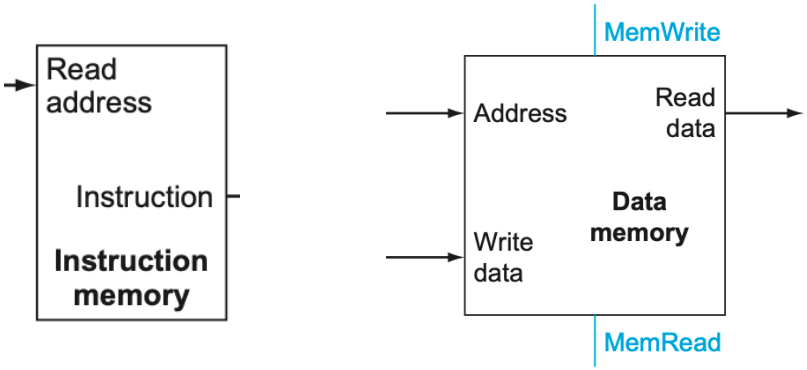


REVIEW: State Elements



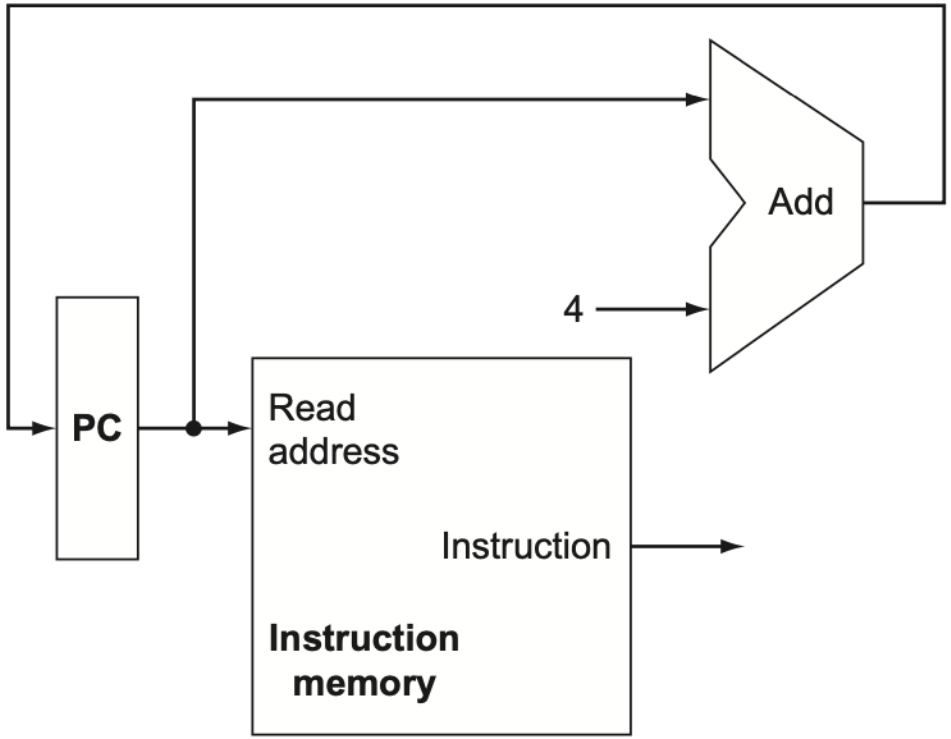
- State Elements are Sequential Circuits, not Combinational Circuits.
- Sequential Circuit output is a function of state (and often also input).
- How does one design and build Sequential Circuits?
- We will see when we build ADVANCED Control. For now, let's build the Datapath...

Combinational?



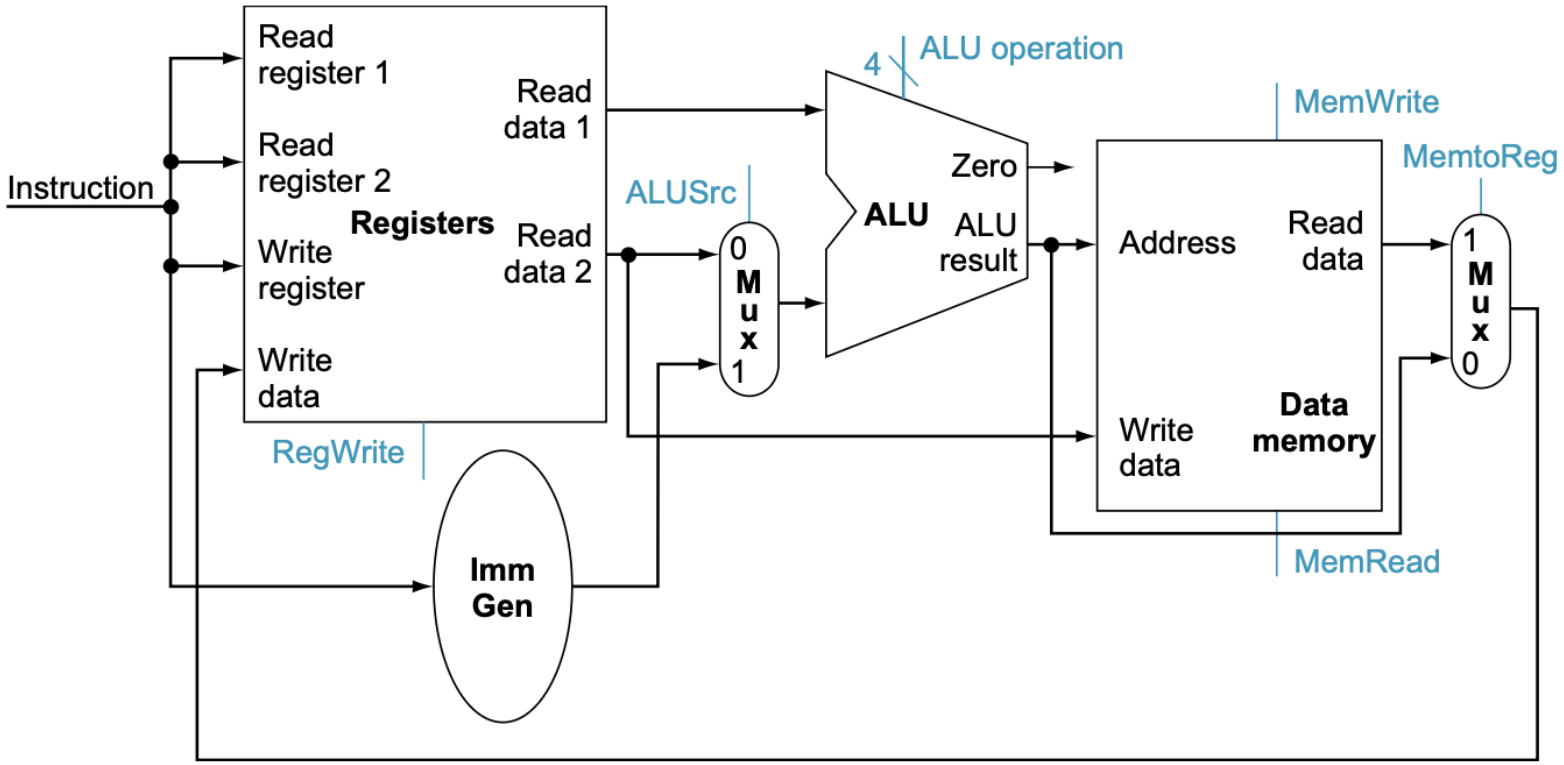
Let's start building our processor, datapath first!

Fetching Instructions (no branching)

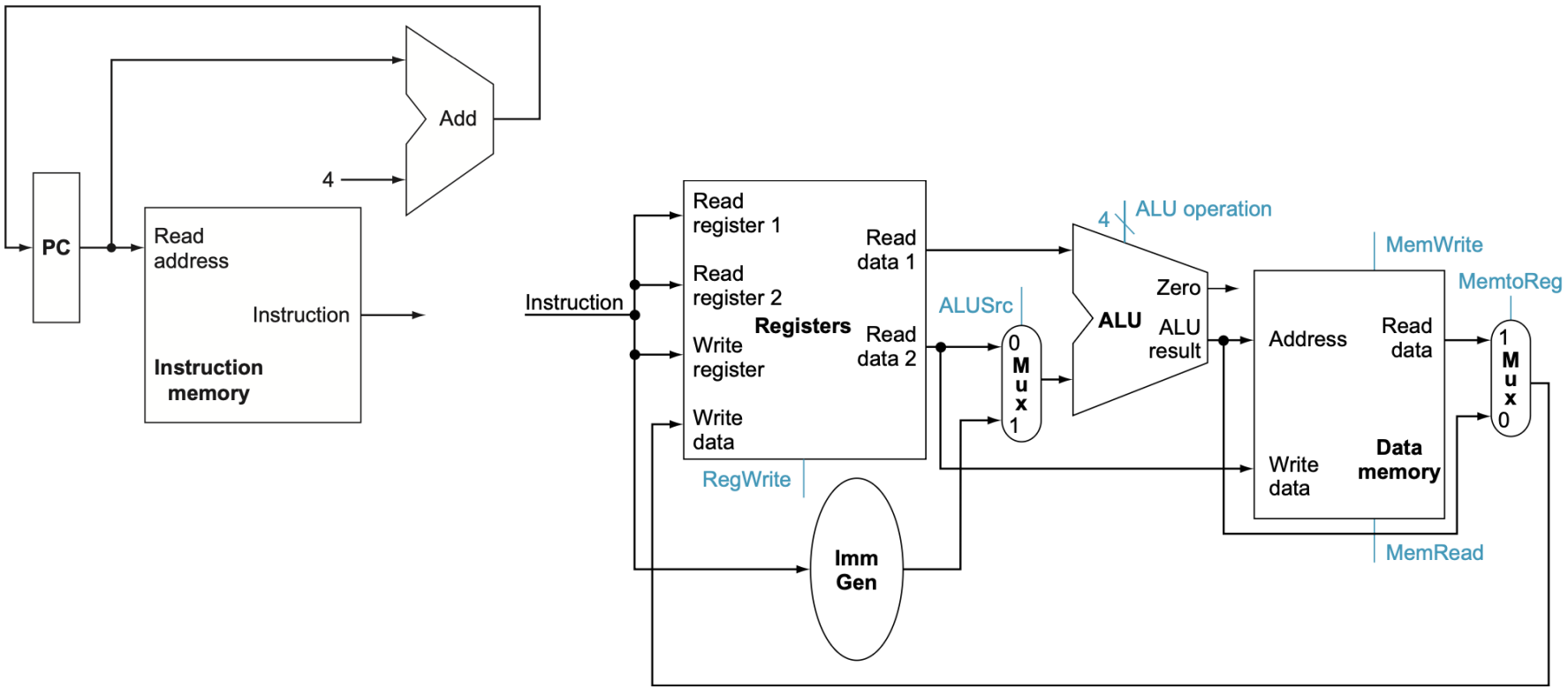


This part looks like EDSAC fetch

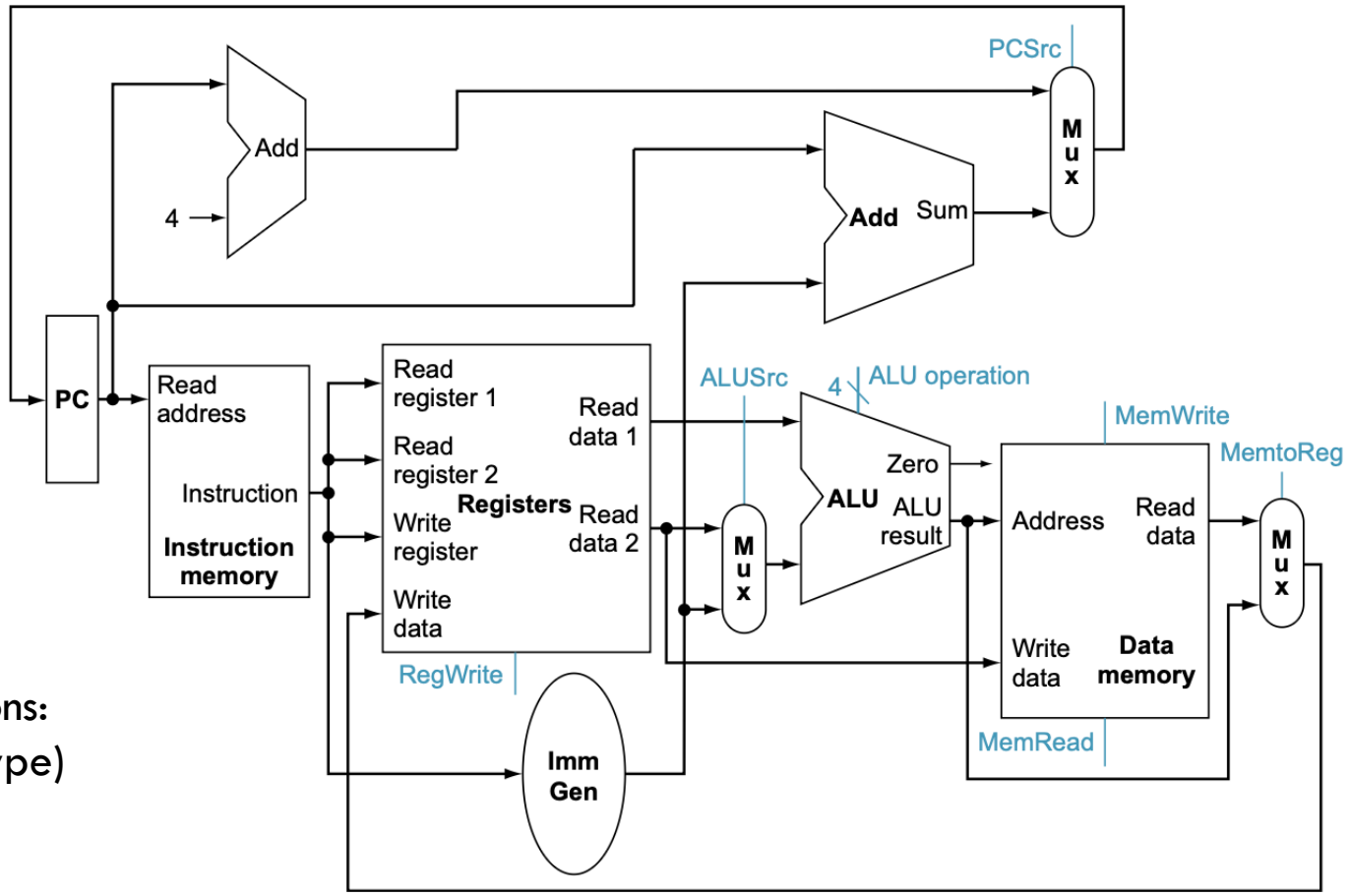
Memory and R-Type Datapath



Bring Fetch and Mem/R-Type Datapaths Together:



Adding Instruction Fetch With Branches



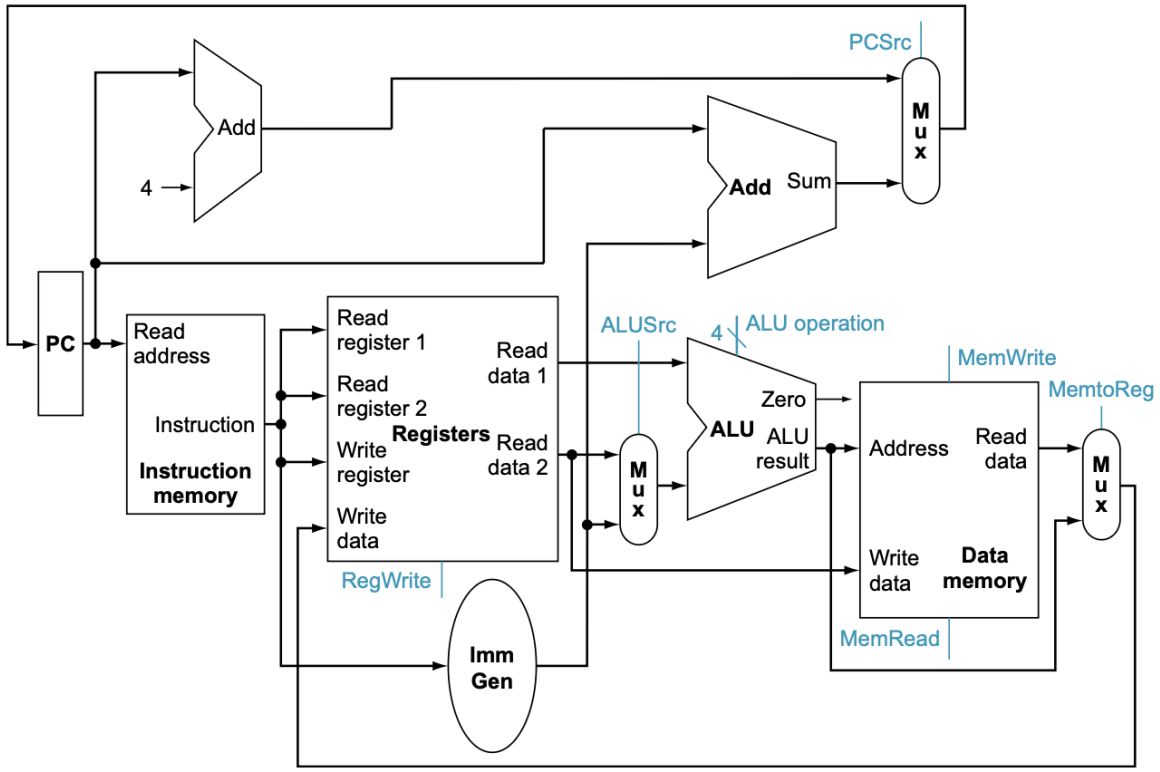
Supports a subset of instructions:

- Arithmetic/logical (R-type)
- Memory: lw, sw
- Control flow: beq

Adding Instruction Fetch With Branches

R-type:
lw:
sw:
beq:

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode



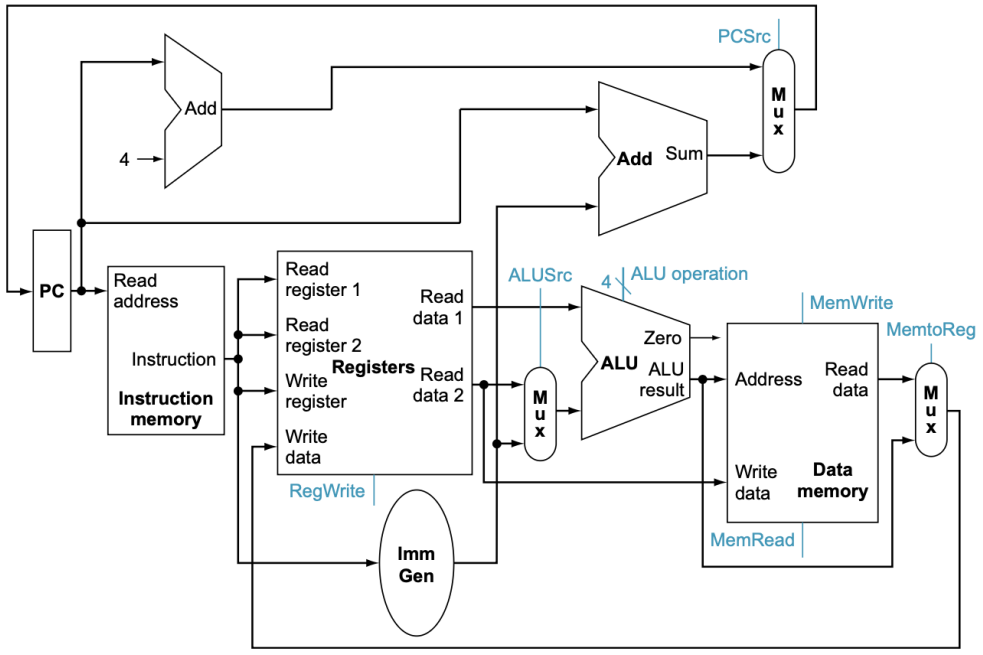
Imm Gen?

Adding Instruction Fetch With Branches

R-type:
lw:
sw:
beq:
jal:

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

jal	UJ Jump & Link	R[rd] = PC+4; PC = PC + {imm,1b'0}



Can this datapath support jal?

"Architecture" Coined (1964)

> •The term *architecture* is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation.

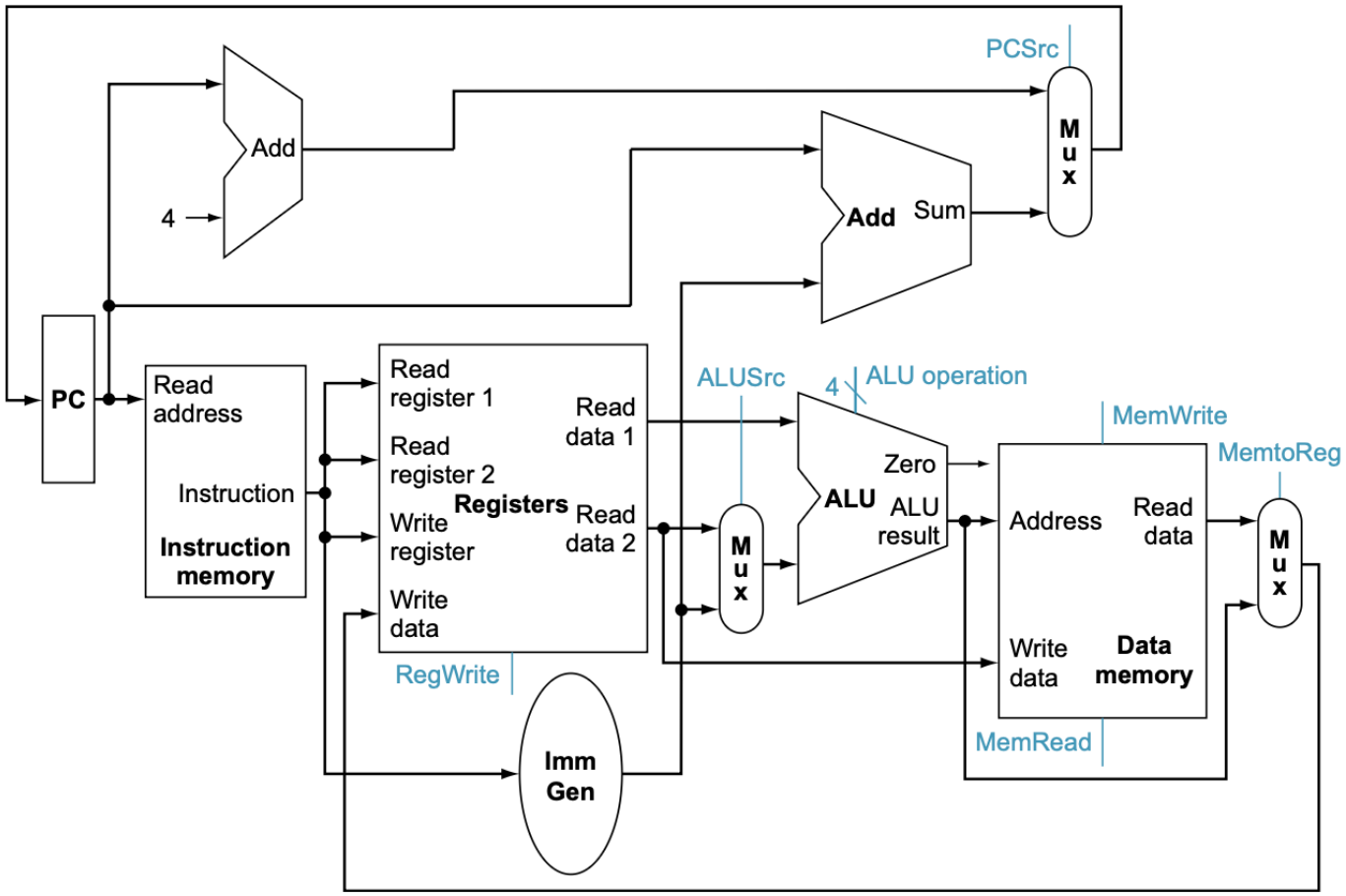
Architecture or Instruction Set Architecture (ISA):

- A "contract" between HW and SW
- Examples: ARM, x86, MIPS, RISC-V, PowerPC, …

Implementation or Microarchitecture:

- A physical instance of an architecture
- Apple M1 vs. M2, Intel Core i5-1038NG7 vs Xeon ES-2620

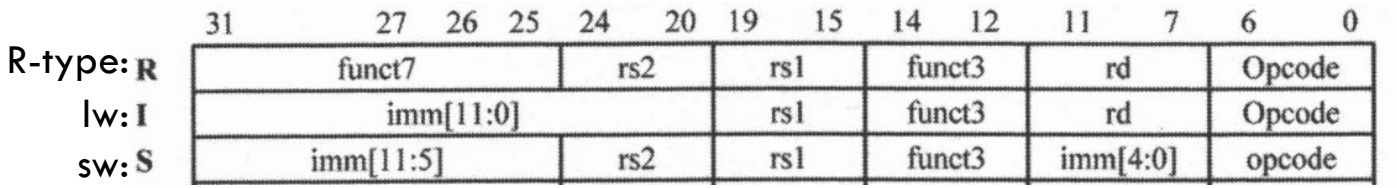
Full Datapath: Architectural/Microarchitectural State



MIPS Instruction Quirk

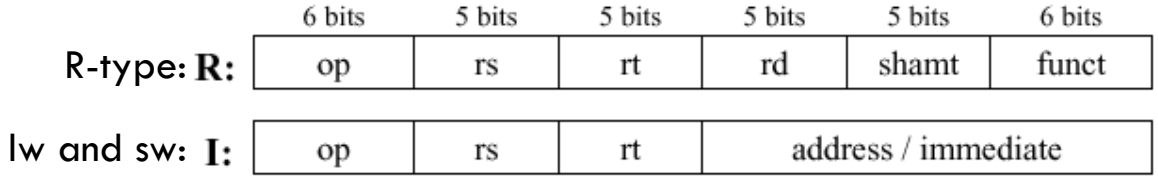
What is MIPS?

RISC-V:

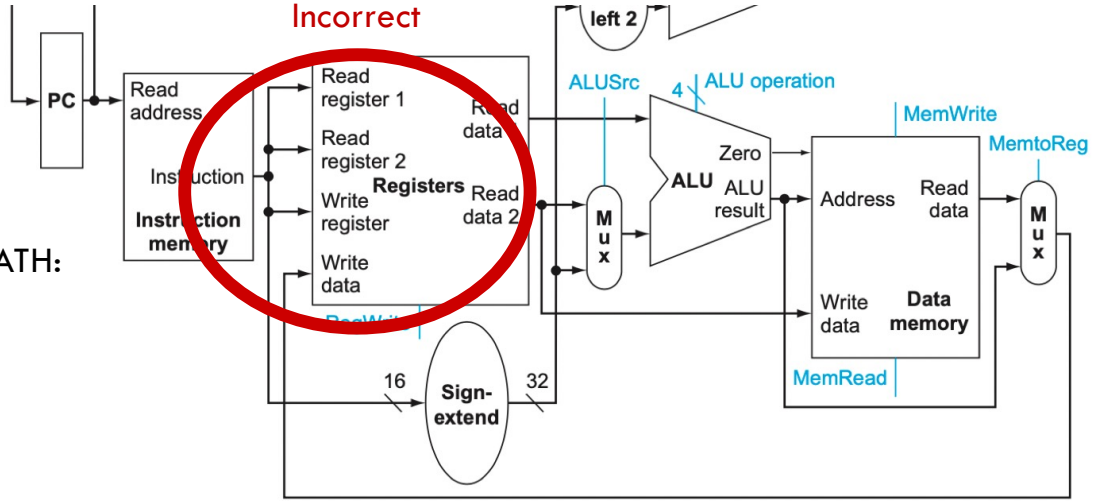


In MIPS, the Destination Register may be in different locations:

- All R-Types use `rd` (bits 11-15)
- Loads use `rt` (bits 16-20)



BROKEN MIPS DATAPATH:

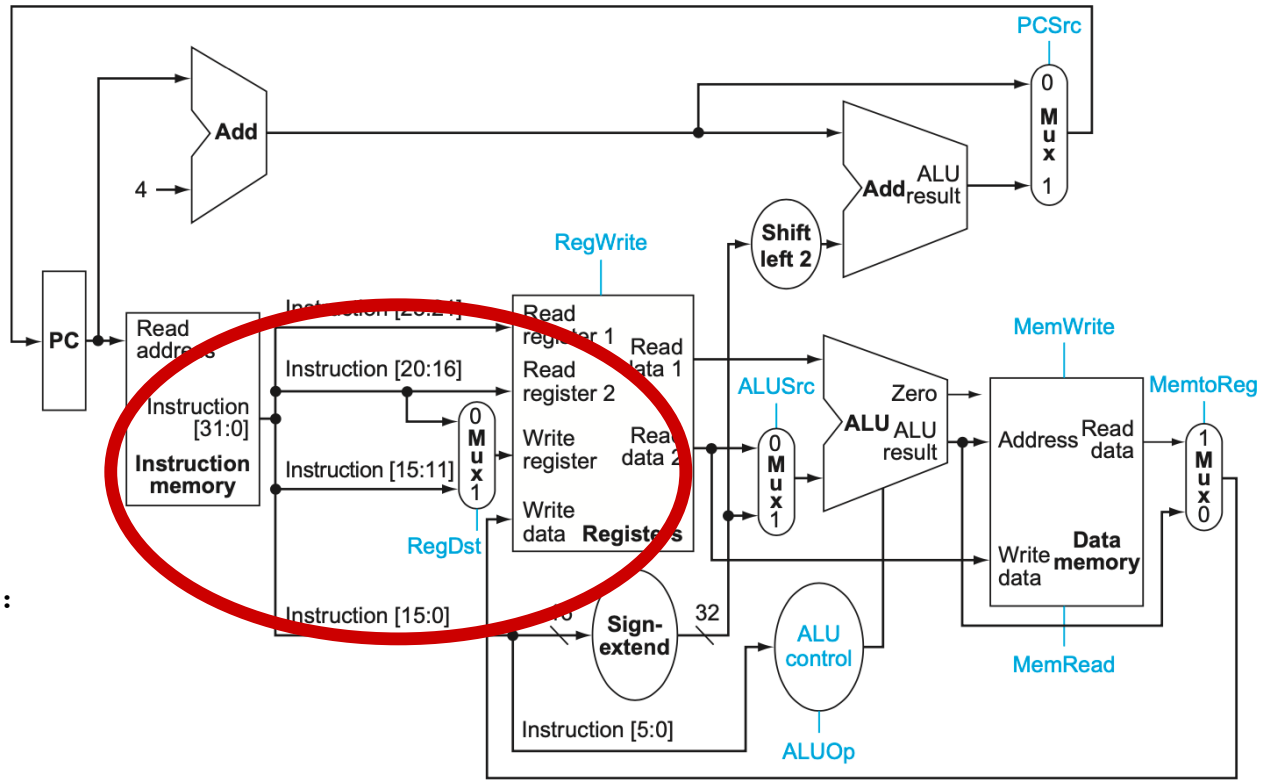


The magic of the Mux!

MIPS:

	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R:	op	rs	rt	rd	shamt	funct
I:	op	rs	rt	address / immediate		

FIXED MIPS DATAPATH:



ICQ: Why do we use different ISAs rather than have a monolithic version?

Control

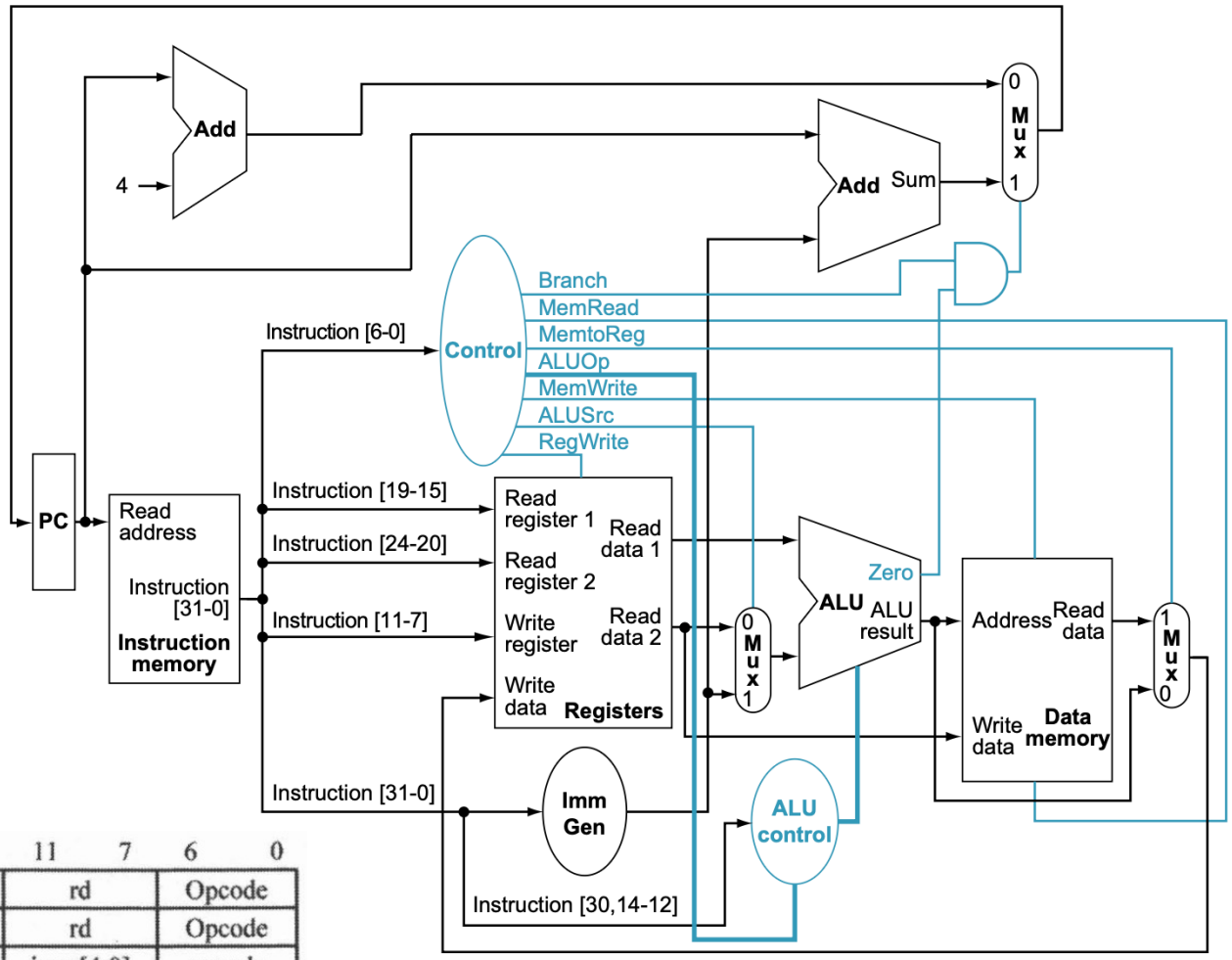
control The component of the processor that commands the datapath, memory, and I/O devices according to the instructions of the program.

"The component of the processor that commands the datapath, memory, and I/O devices according to the instructions of the program." – P&H p. 21

I prefer: "The component of the processor that commands the datapath according to the instructions of the program."

Full RISC-V Datapath with Control

- Supports a subset of instructions:
  - Arithmetic/logical (R-type)
  - Memory: lw, sw
  - Control flow: beq
- Control generates signals: Branch, MemRead, MemtoReg, ALUOp, MemWrite, ALUSrc, RegWrite
- Control uses opcode: Instruction[6-0]



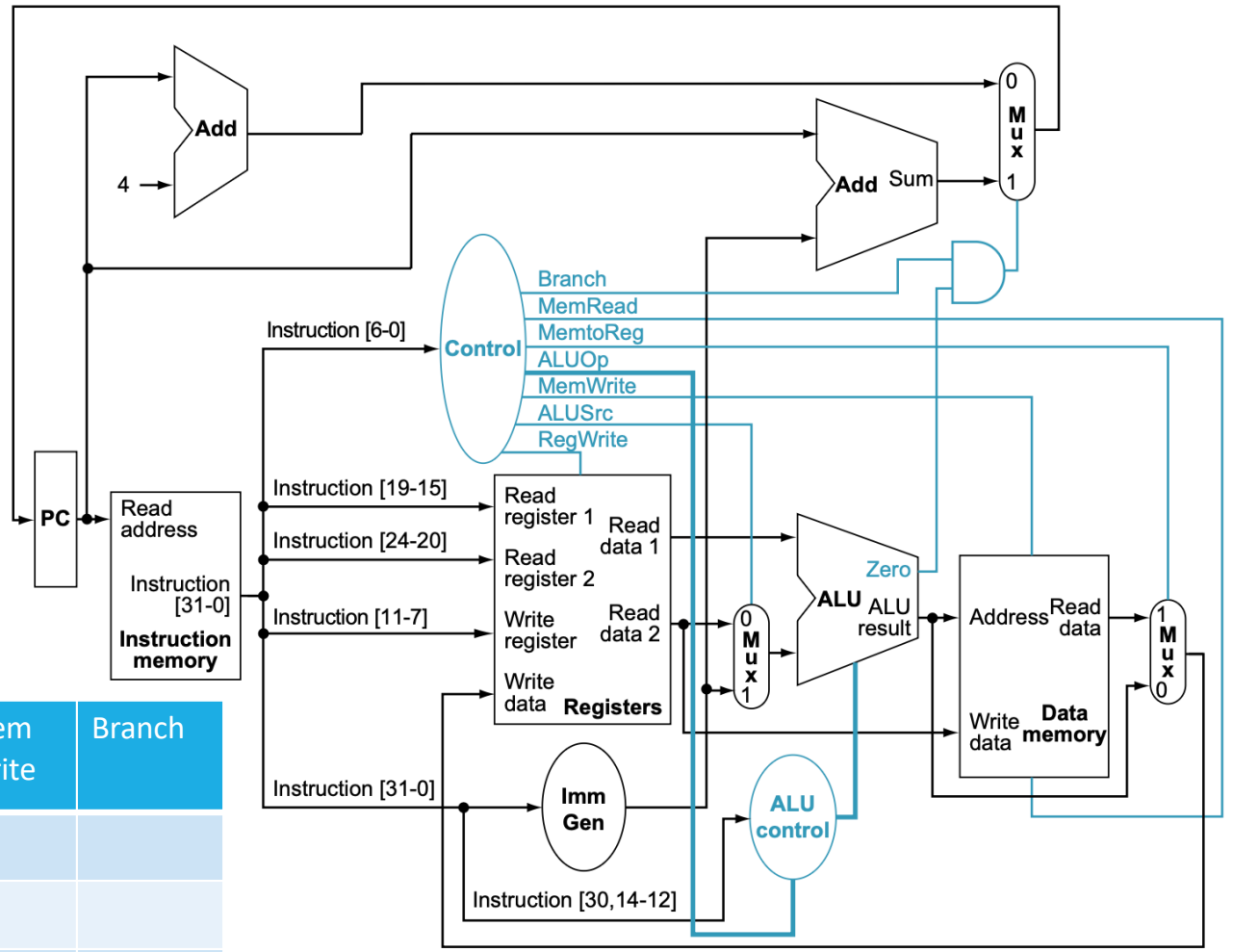
	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode

Remember: THIS IS A LITTLE OFF.

Control Signals in Full Datapath

Signal name	Effect when deasserted	Effect when asserted
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, 12 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.

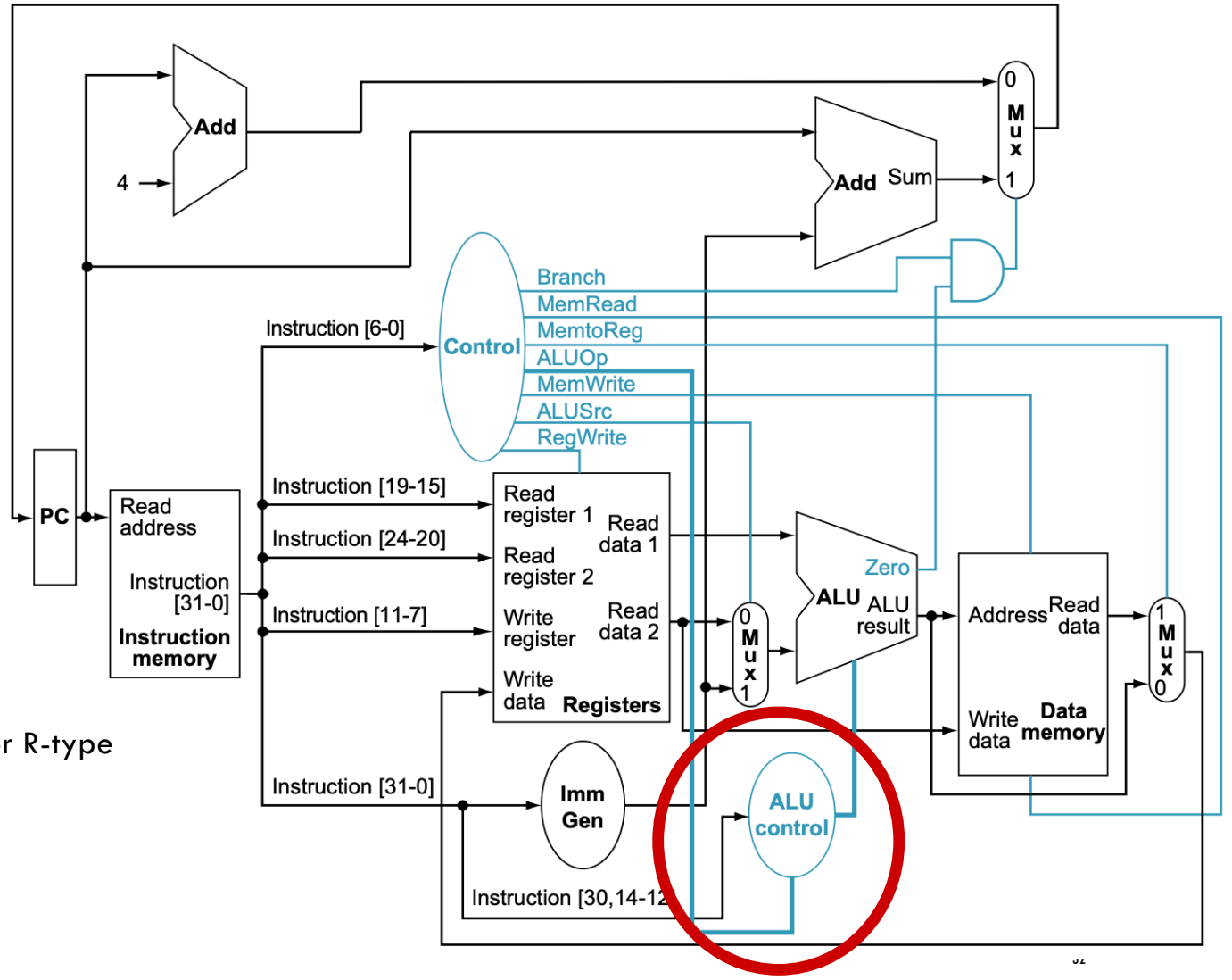
Practice:
Fill in this table!



Inst.	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch
R-fmt						
lw						
sw						
beq						

What is going on here?

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract



ALUOp should have ALU control chose:

- **add** for lw/sw
- **subtract** for beq
- **AND/OR/add/subtract** operation for R-type instructions based on its funct fields

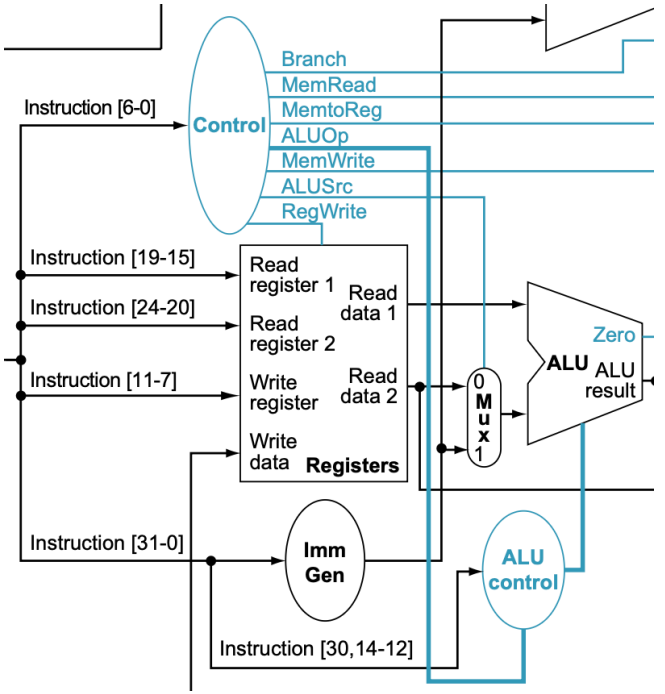
[at least needs two bits]

This is what is going on!

From Before: Only a function of the opcode:

- **add** for lw/sw
- **subtract** for beq
- **AND/OR/add/subtract** for R-type

Instruction	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
lw	1	1	1	1	0	0	0	0
sw	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1



Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
lw	00	load word	XXXXXXX	XXX	add	0010
sw	00	store word	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract

SB	imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode

Control Signals: Branch and PCSrc Mux

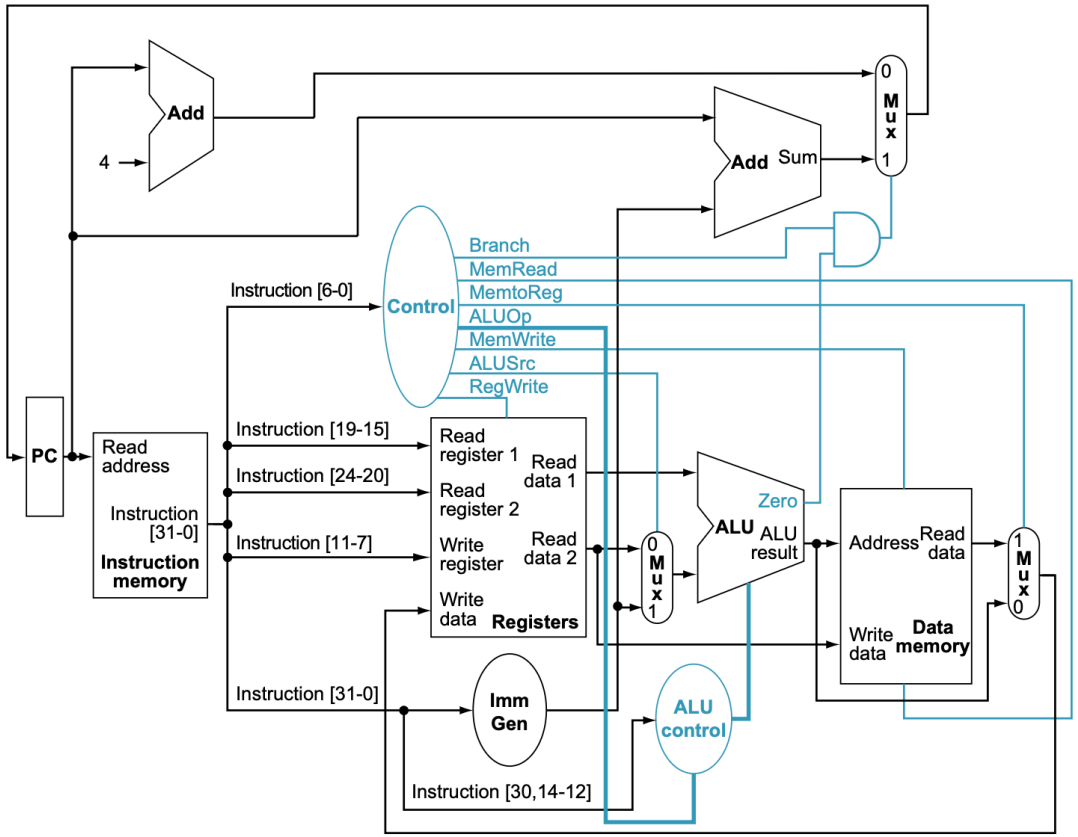
- PCSrc =
  Branch (from Control) & Zero (from ALU)
- True:
  PC = SignExt(imm[12:1]) << 1 + PC

```
PC=PC+{imm,1'b0}
```

Why?

- False:
  PC = PC + 4

Instruction	Branch
R-format	0
lw	0
sw	0
beq	1

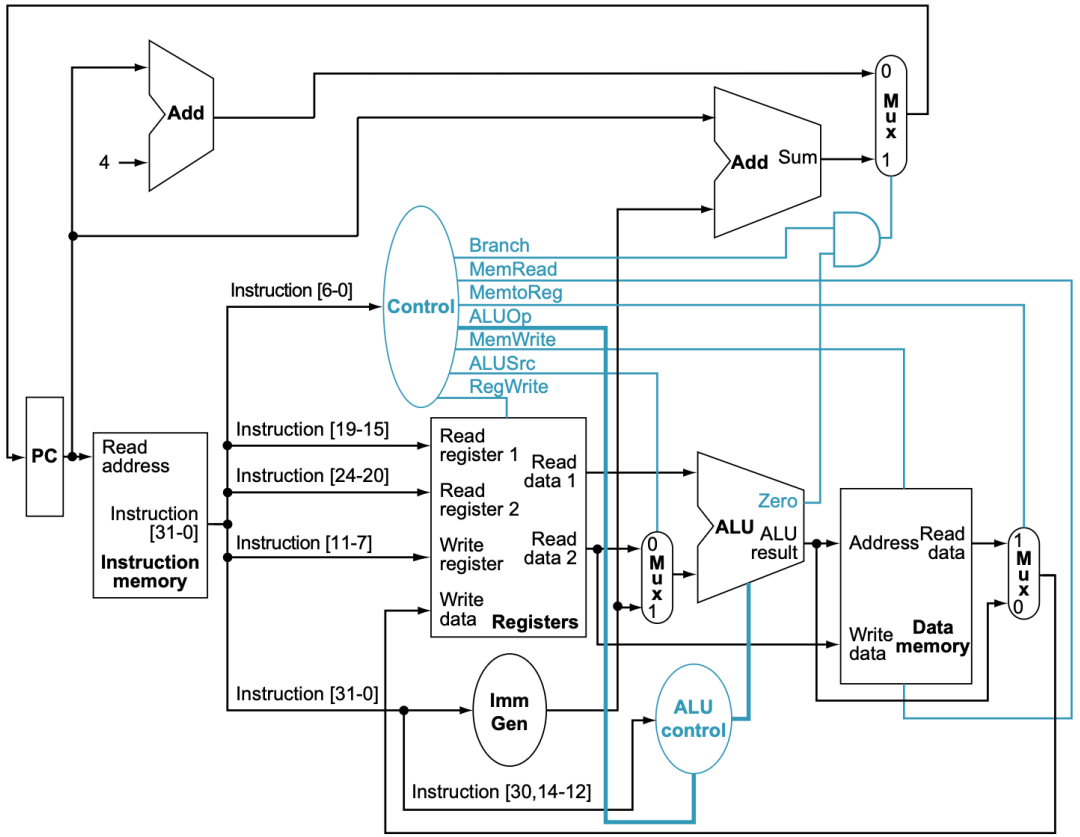


Control Signals: ALUSrc

- True:
  ImmGen (lw/I-Format and sw/S-Format)
- False:
  ReadData2 (R-Format, beq/SB-Format)

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode

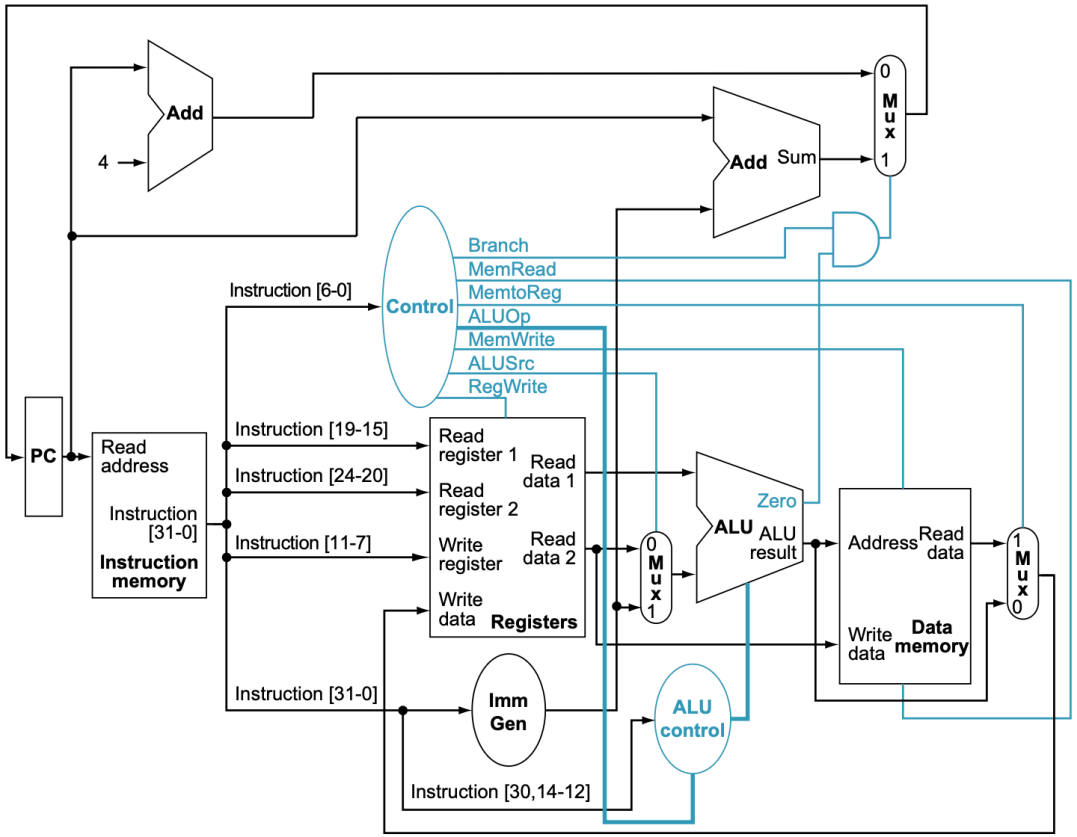
Instruction	ALUSrc
R-format	0
lw	1
sw	1
beq	0



Control Signals: MemToReg

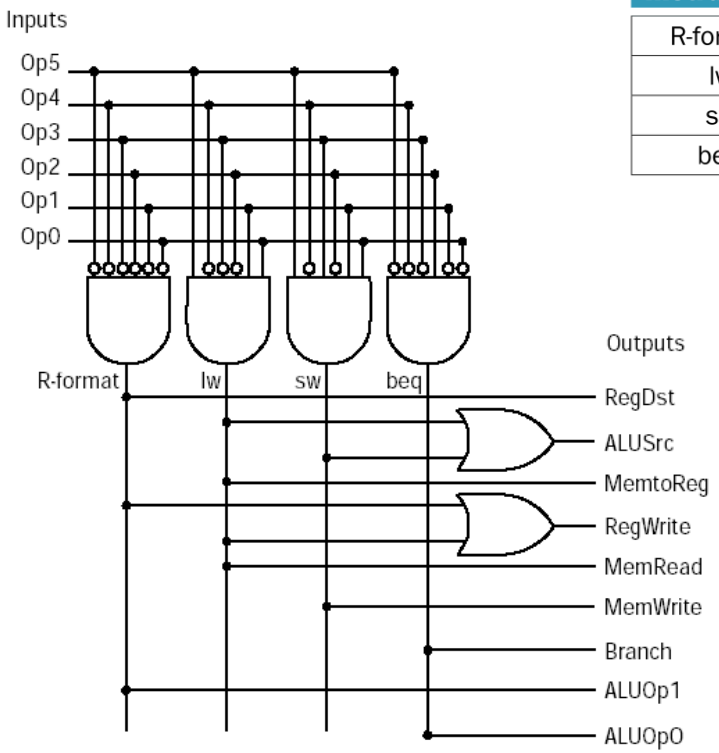
- True:
  RegisterWriteData = MemReadData
- False:
  RegisterWriteData = ALUResult

Instruction	Memto-Reg
R-format	0
lw	1
sw	X
beq	X



Control Design

Instruction	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
lw	1	1	1	1	0	0	0	0
sw	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

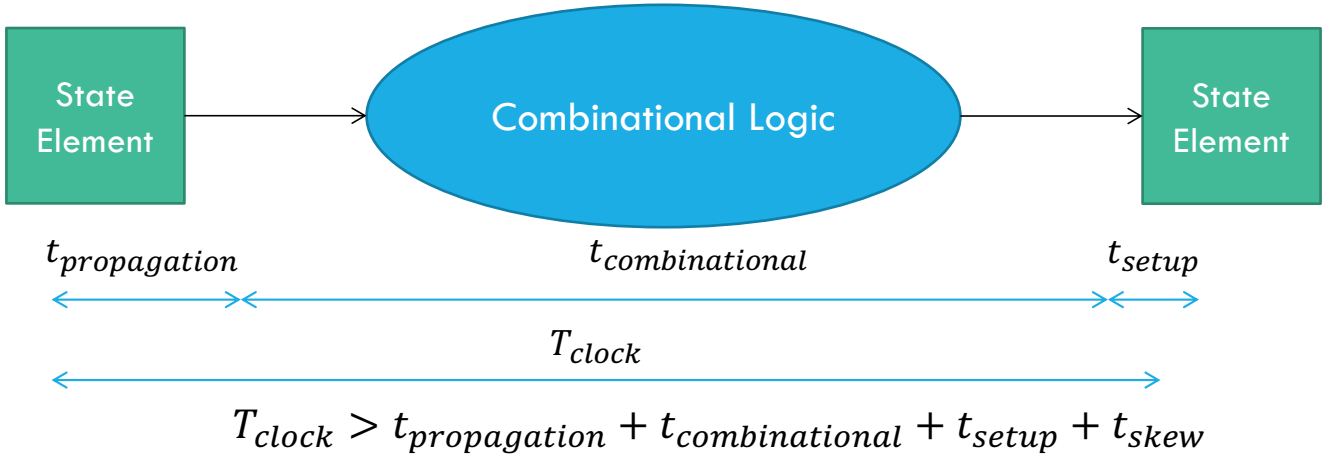


Memory Reference: `lw, sw`
Arithmetic/Logical: `add, sub, and, or, slt`
Control flow: `beq`

Truth Table (above) -> SoP -> Circuit

From MIPS, how can you tell? Works with RISC-V!

Single-Cycle Implementation



Cycle time determined by the length of the longest path.

What is longest state element to state element path?

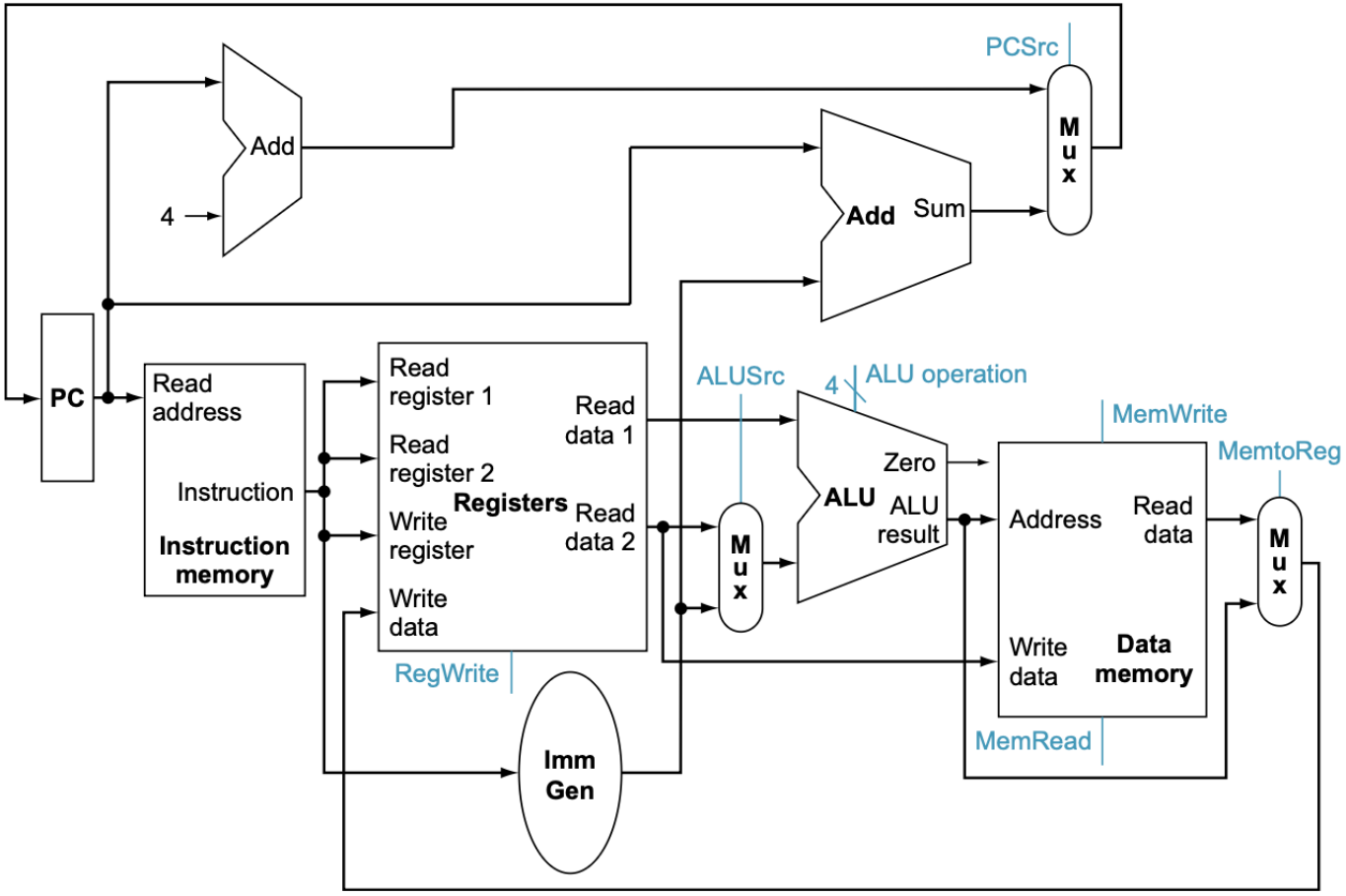
Single-Cycle Implementation

Calculate cycle time assuming negligible delays except:

memory (2ns)

ALU/adders (2ns)

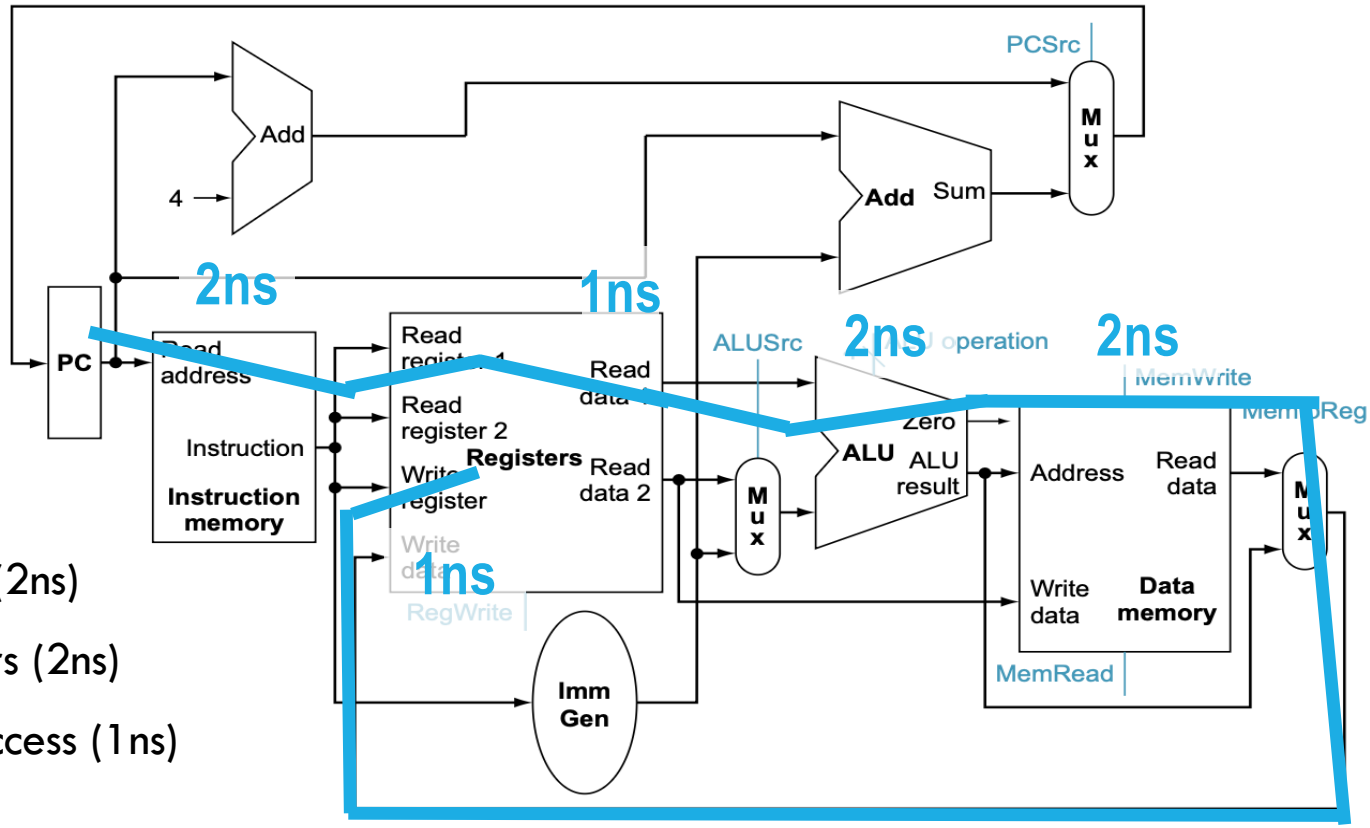
register file access (1ns)



memory (2ns)

ALU/adders (2ns)

register file access (1ns)



Load Word is longest running instruction
(prove by computing time for sw, beq, R-type)

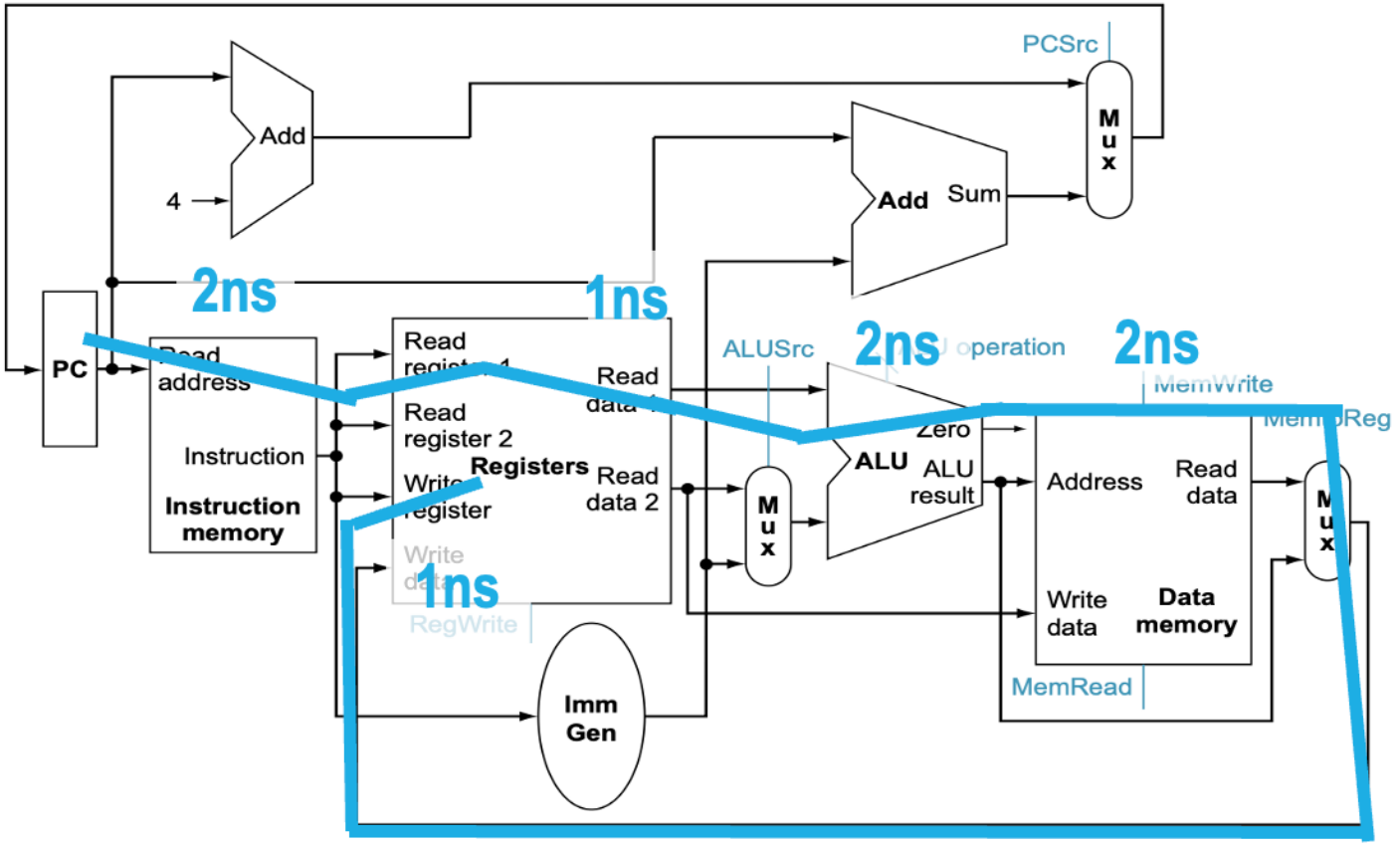
2ns + 1ns + 2ns + 2ns + 1ns = **8ns**

Magic Moment!

At this point, you can design a processor!

1. Analyze instruction set for datapath requirements
2. Select set of datapath components
3. Assemble datapath meeting the requirements
4. Analyze implementation of each instruction to determine setting of control signals
5. Assemble the control logic
6. Set clock rate based on longest path

What's wrong with the single-cycle implementation?

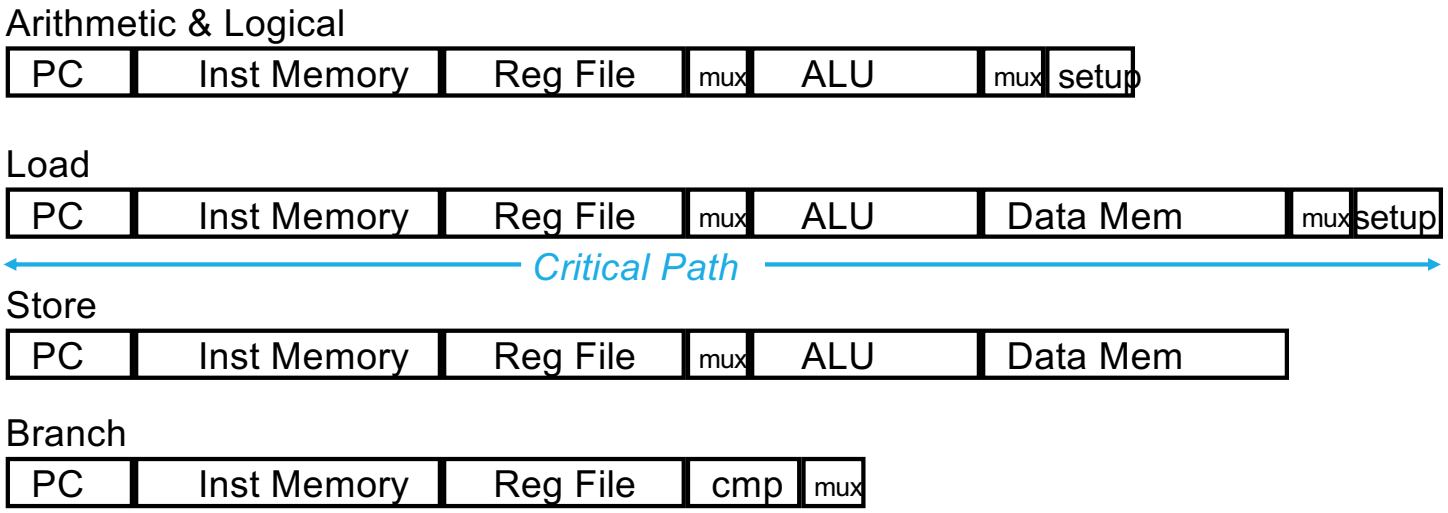


What's wrong with the single-cycle implementation?

memory (2ns), ALU/adders (2ns), register file access (1ns)

Inst.	Inst. Mem	Reg. Read	ALU	Data Mem	Reg. Write	Total
ALU	2	1	2		1	6
lw	2	1	2	2	1	8
sw	2	1	2	2		7
beq	2	1	2			5

What's wrong with the single-cycle implementation?



- Long cycle time, much of it wasted
- All instructions take as much time as the slowest
- Real memory is not so nice as our idealized memory
  - (cannot always get the job done in fixed amount of time)

How bad is it?

Assume: 100 instructions executed

- 25% of instructions are loads (8ns),
- 10% of instructions are stores (7ns),
- 45% of instructions are adds (6ns), and
- 20% of instructions are branches (5ns).

Single-cycle execution:

$$100 * 8ns = 800\ ns$$

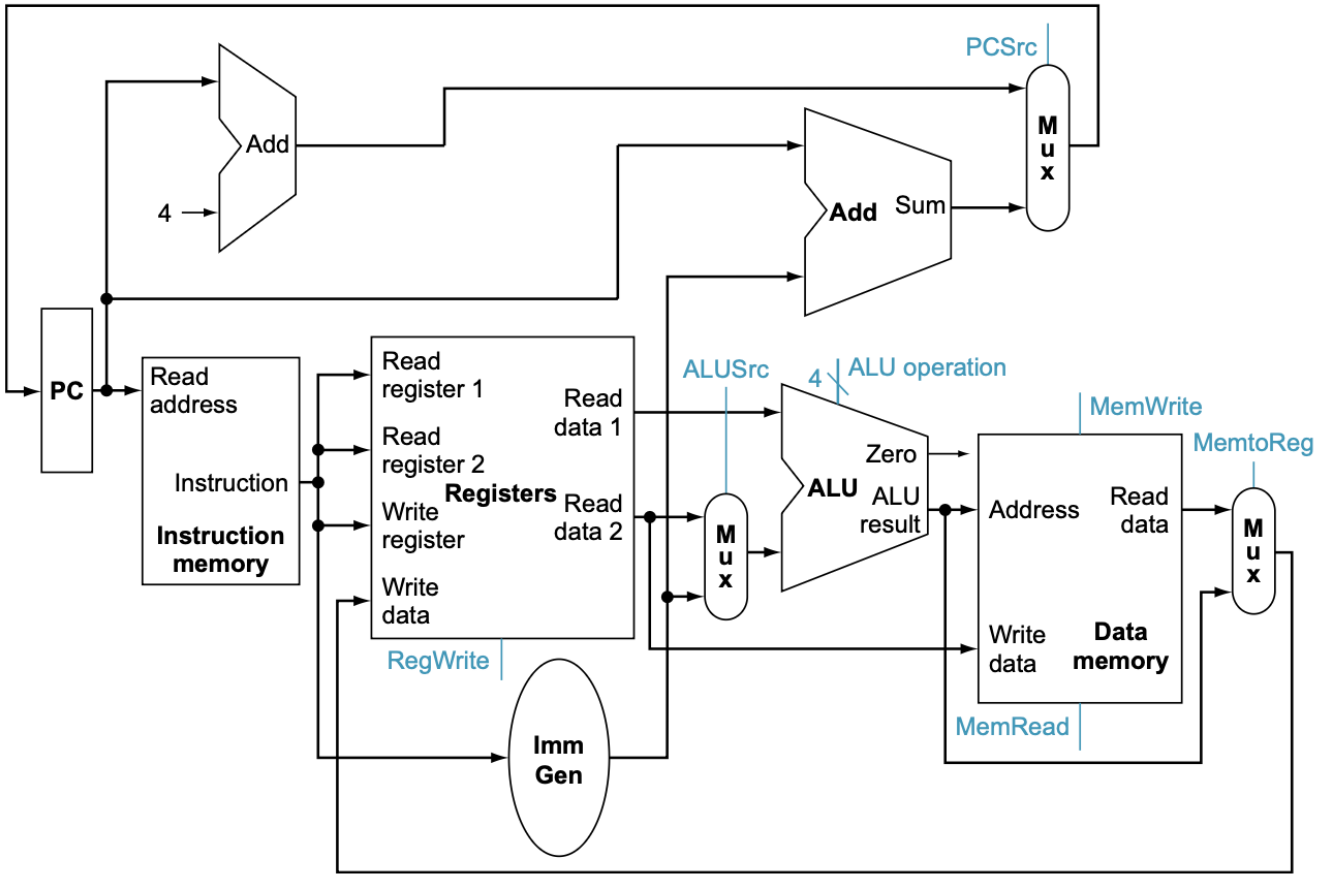
Optimal execution:

$$25*8ns + 10*7ns + 45*6ns + 20*5ns = 640\ ns$$

$$\text{Speedup} = 800/640 = 1.25$$

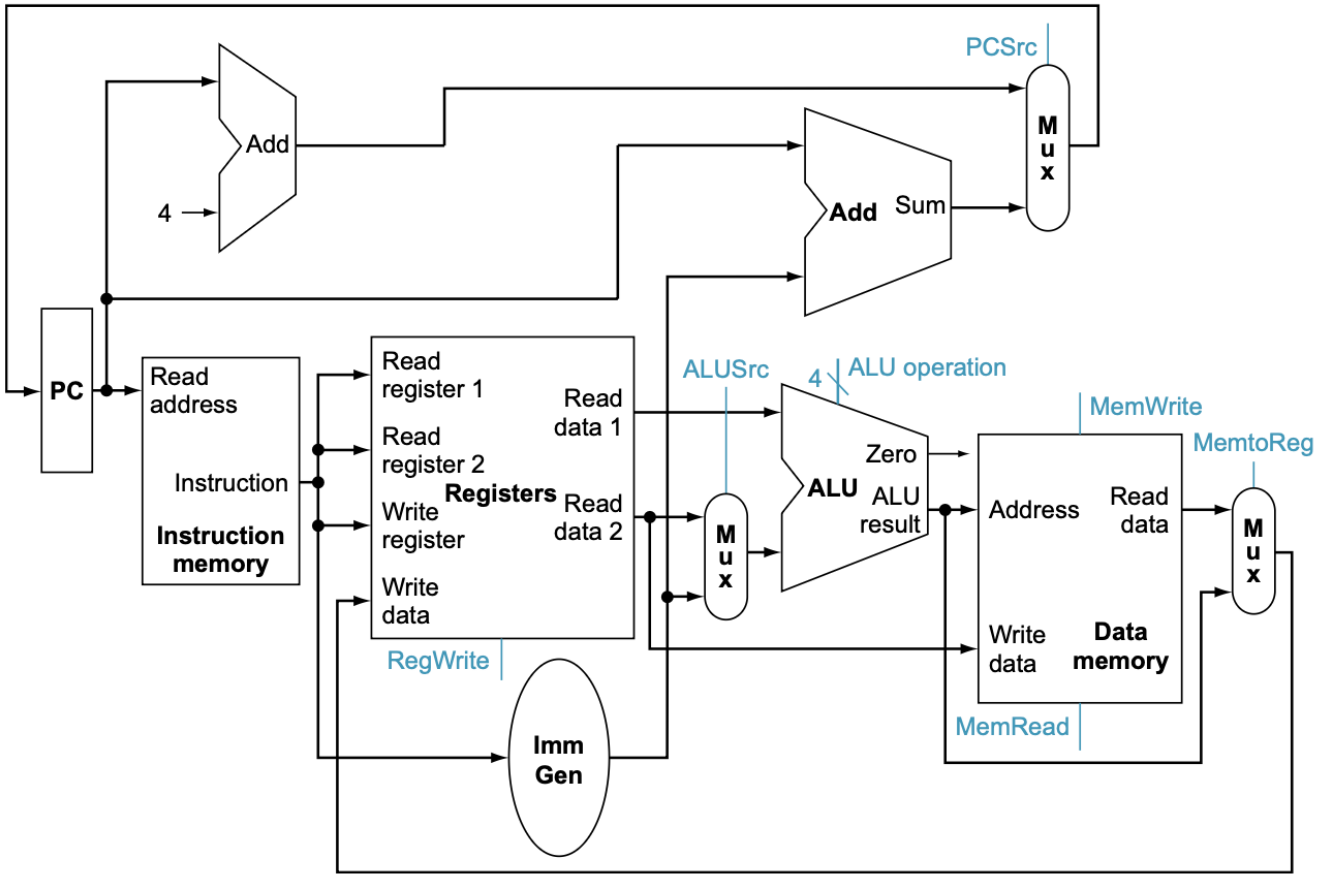
Other Issues

- Instruction and Data Memory are the same
- Some units hold value long after job is complete
  - Could reuse ALU for example
  - Underutilized resources (wasteful of area/power)

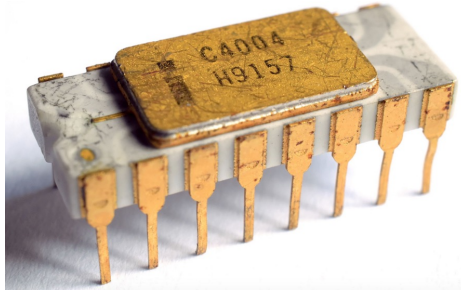


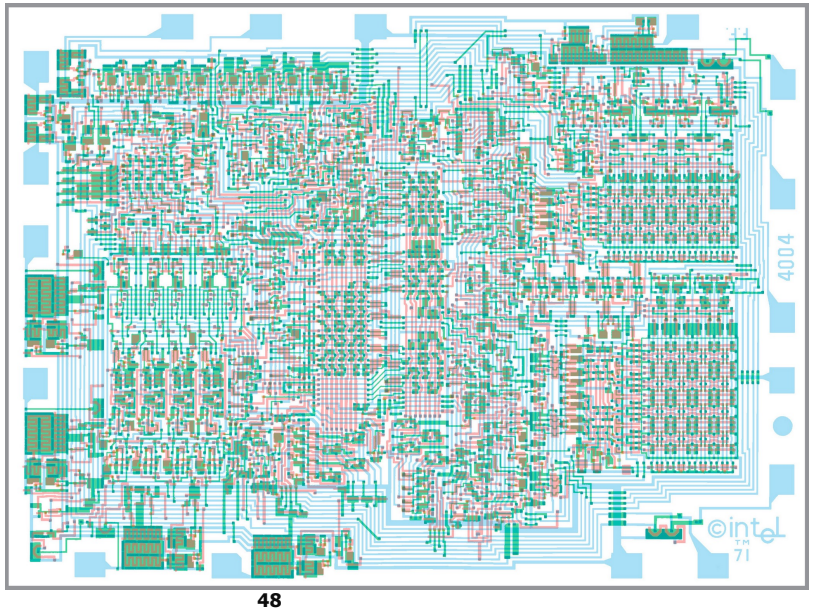
Other Issues

- Floating point and other VERY LONG instructions



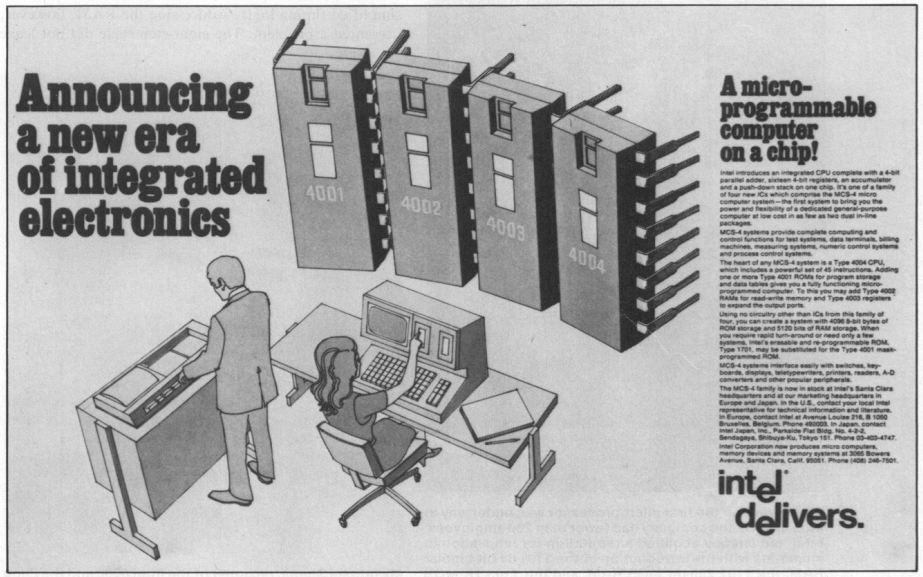
Welcome to 1971!!!
1949 was nice, but…





The Intel 4004, 1971:

- **First microprocessor (stored-program, electronic, general-purpose)**
- 4-bit CPU with 2,250 transistors, 2 designers
- 10.8 µSecond instruction cycle
- http://www.intel4004.com/ http://e4004.szyc.org/
- Full design recently released by Intel



The first advertisement for a microprocessor in Electronic News in November 1971. [1]

FIN

Please fill out ICQ Form here!

https://tinyurl.com/Fa25ICQ.