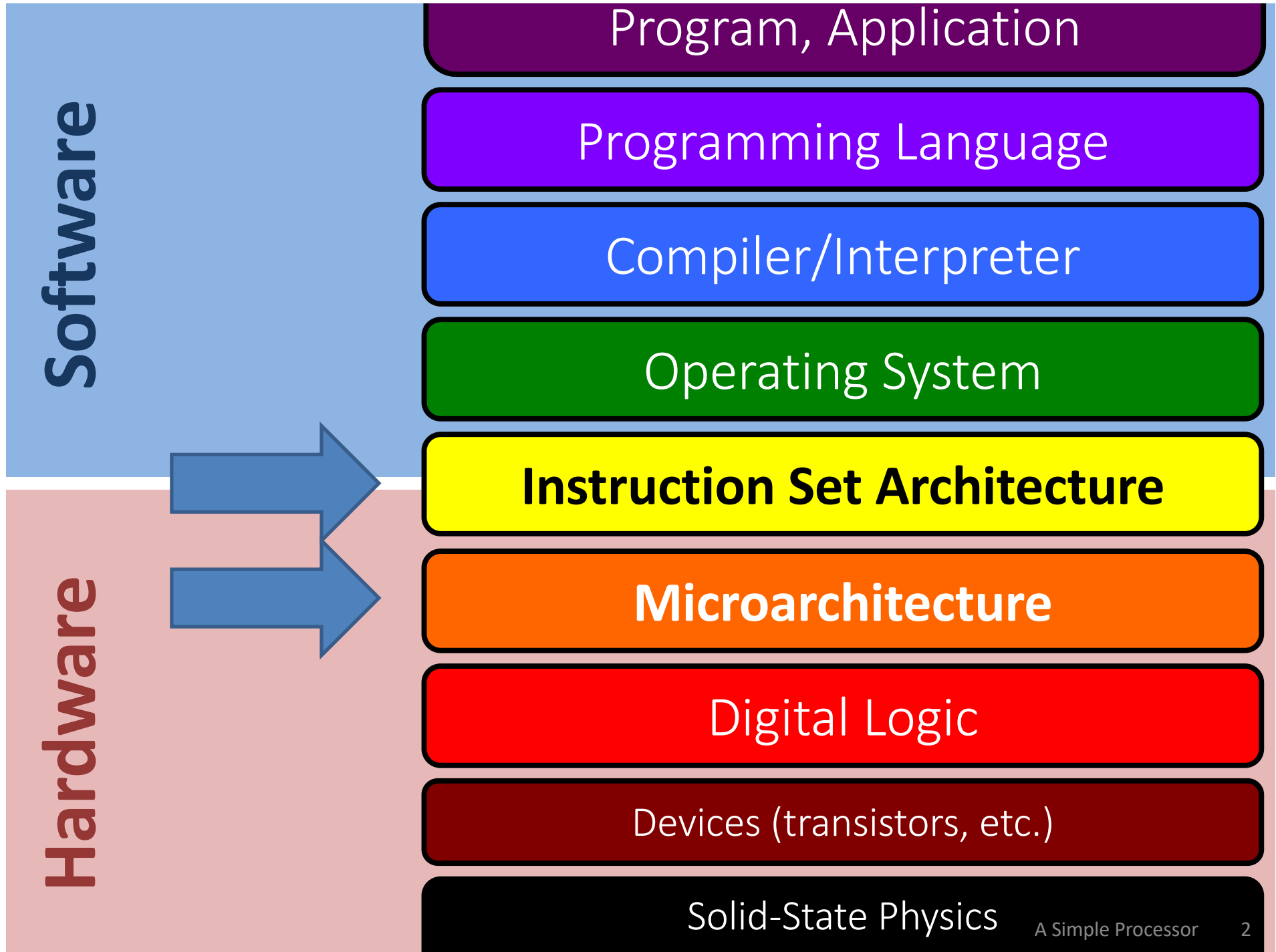


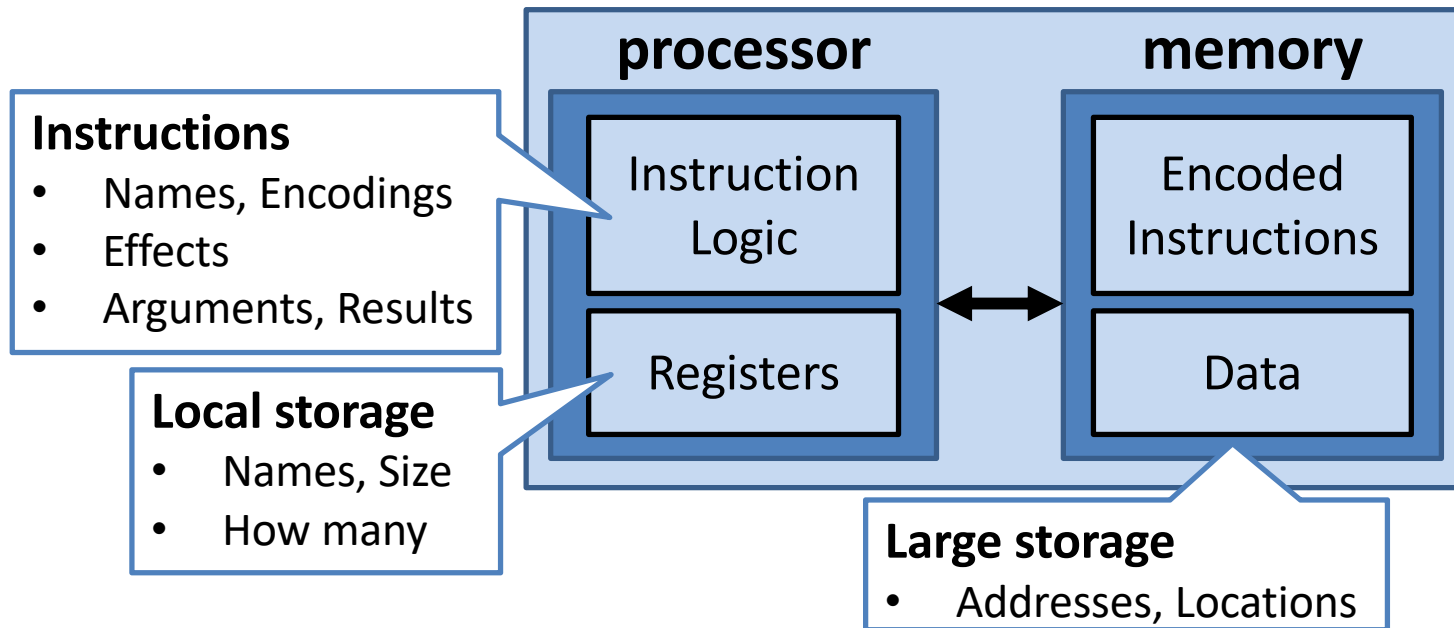


A Simple Processor

1. A simple Instruction Set Architecture
2. A simple microarchitecture (implementation):
Data Path and Control Logic



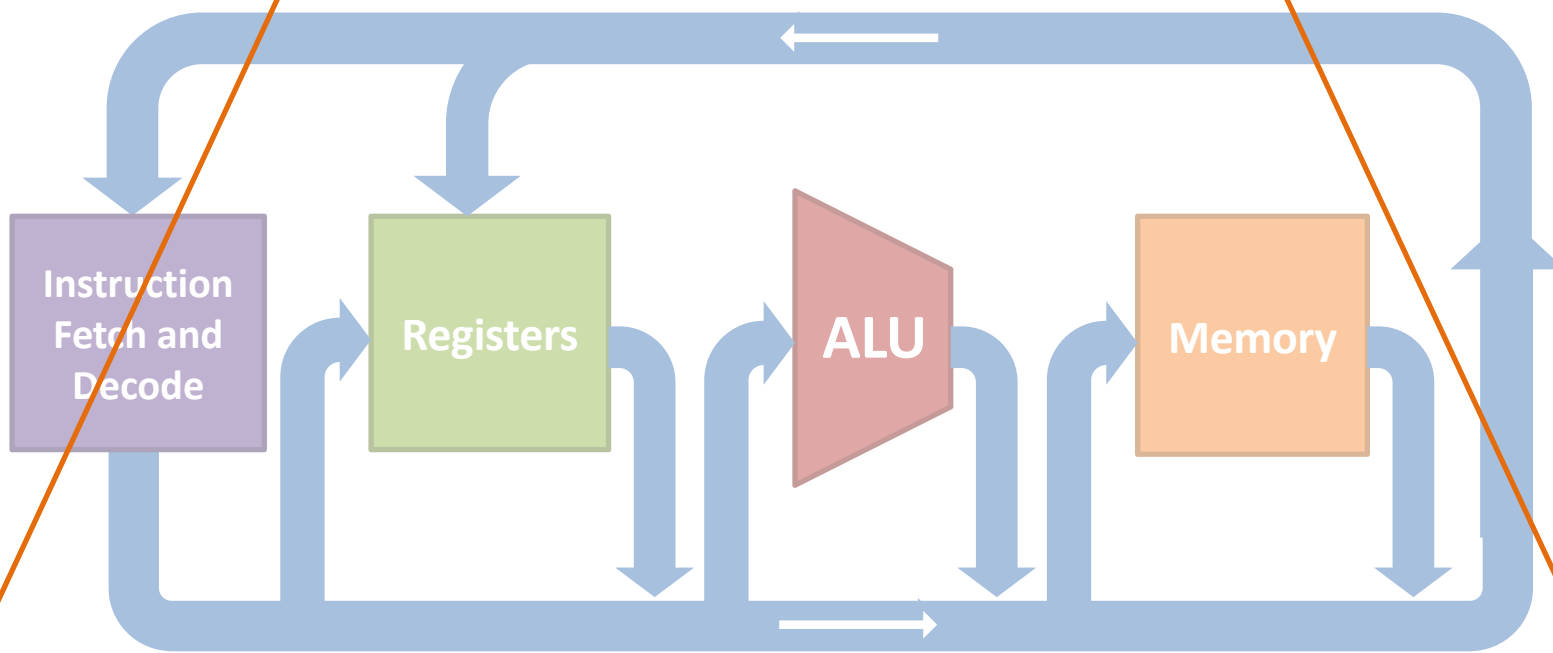
Instruction Set Architecture (HW/SW Interface)



Computer

Computer

Microarchitecture (Implementation of ISA)



HW ISA

An example made-up instruction set architecture

Word size = 16 bits

- Register size = 16 bits.
- ALU computes on 16-bit values.

Memory is byte-addressable, accesses full words (byte pairs).

16 registers: R0 - R15


- R0 always holds hardcoded 0
- R1 always holds hardcoded 1
- R2 – R15: general purpose

Instructions are 1 word in size.

Separate *instruction memory*.

Program Counter (PC) register

- holds address of next instruction to execute.



Address	Contents
0	First instruction, low-order byte
1	First instruction, high-order byte
2	Second instruction, low-order byte
...	...

R: Register File

Reg	Contents	Reg	Contents
R0	0x0000	R8	
R1	0x0001	R9	
R2		R10	
R3		R11	
R4		R12	
R5		R13	
R6		R14	
R7		R15	

M: Data Memory

Address	Contents	
0x0 – 0x1		
0x2 – 0x3		
0x4 – 0x5		
0x6 – 0x7		
0x8 – 0x9		
0xA – 0xB		
0xC – 0xD		
...		

Program Counter

PC

Processor
Loop

1. $ins \leftarrow IM[PC]$
2. $PC \leftarrow PC + 2$
3. Do ins

HW ISA Abstract Machine

IM: Instruction Memory

Address	Contents
0x0 – 0x1	
0x2 – 0x3	
0x4 – 0x5	
0x6 – 0x7	
0x8 – 0x9	
...	

HW ISA Instructions

MSB **16-bit Encoding** LSB

Assembly Syntax	Meaning (R = register file, M = data memory)	Opcode	Rs	Rt	Rd
ADD <i>Rs, Rt, Rd</i>	$R[d] \leftarrow R[s] + R[t]$	0010	<i>s</i>	<i>t</i>	<i>d</i>
SUB <i>Rs, Rt, Rd</i>	$R[d] \leftarrow R[s] - R[t]$	0011	<i>s</i>	<i>t</i>	<i>d</i>
AND <i>Rs, Rt, Rd</i>	$R[d] \leftarrow R[s] \& R[t]$	0100	<i>s</i>	<i>t</i>	<i>d</i>
OR <i>Rs, Rt, Rd</i>	$R[d] \leftarrow R[s] R[t]$	0101	<i>s</i>	<i>t</i>	<i>d</i>
LW <i>Rt, offset(Rs)</i>	$R[t] \leftarrow M[R[s] + offset]$	0000	<i>s</i>	<i>t</i>	<i>offset</i>
SW <i>Rt, offset(Rs)</i>	$M[R[s] + offset] \leftarrow R[t]$	0001	<i>s</i>	<i>t</i>	<i>offset</i>
BEQ <i>Rs, Rt, offset</i>	If $R[s] == R[t]$ then $PC \leftarrow PC + 2 + offset * 2$	0111	<i>s</i>	<i>t</i>	<i>offset</i>
JMP <i>offset</i>	$PC \leftarrow offset * 2$	1000	<i>o</i>	<i>f</i>	<i>f s e t</i>
HALT	Stops program execution	1111	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> JMP offset is unsigned All others are signed </div>		

R: Register File Exercise #1

Reg	Contents	Reg	Contents
R0	0x0000	R8	
R1	0x0001	R9	
R2		R10	
R3		R11	
R4		R12	
R5		R13	
R6		R14	
R7		R15	

M: Data Memory

Address	Contents	
0x0 – 0x1	0xEB	0xCA
0x2 – 0x3	0xBD	0x56
0x4 – 0x5		
0x6 – 0x7		
0x8 – 0x9		
0xA – 0xB		
0xC – 0xD		
...		

Program Counter

PC

Processor
Loop

1. $ins \leftarrow IM[PC]$
2. $PC \leftarrow PC + 2$
3. Do ins

IM: Instruction Memory

Address	Contents
0x0 – 0x1	LW R3, 0(R0)
0x2 – 0x3	LW R4, 2(R0)
0x4 – 0x5	AND R3, R4, R5
0x6 – 0x7	SW R5, 4(R0)
0x8 – 0x9	HALT
0xA – 0xB	

HW ISA Abstract Machine

Execution Table for *Ex. #1* (shows step-by-step execution)

PC	Instr	State Changes
0x0	LW R3 0(R0)	

R: Register File

Exercise #2

M: Data Memory

Reg	Contents	Reg	Contents
R0	0x0000	R8	
R1	0x0001	R9	2
R2		R10	3
R3		R11	
R4		R12	
R5		R13	
R6		R14	
R7		R15	

Address	Contents	
0x0 – 0x1		
0x2 – 0x3		
0x4 – 0x5		
0x6 – 0x7		
0x8 – 0x9		
0xA – 0xB		
0xC – 0xD		
...		

Program Counter

PC

Processor
Loop

1. $ins \leftarrow IM[PC]$
2. $PC \leftarrow PC + 2$
3. Do ins

IM: Instruction Memory

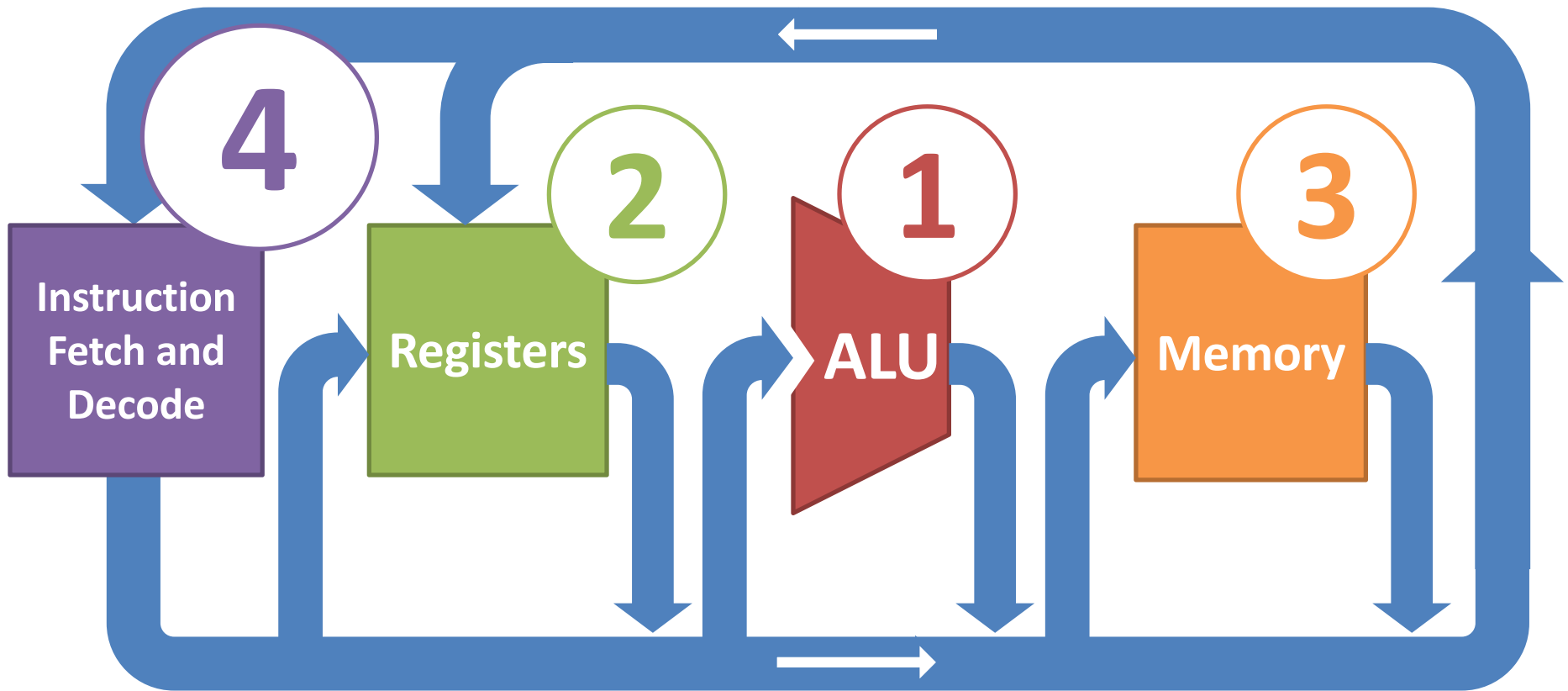
Address	Contents
0x0 – 0x1	SUB R8, R8, R8
0x2 – 0x3	BEQ R9, R0, 3
0x4 – 0x5	ADD R10, R8, R8
0x6 – 0x7	SUB R9, R1, R9
0x8 – 0x9	JMP 1
0xA – 0xB	HALT

HW ISA Abstract Machine

Execution Table for *Ex. #2*

PC	Instr	State Changes
0x0	SUB R8, R8, R8	

HW ARCH microarchitecture

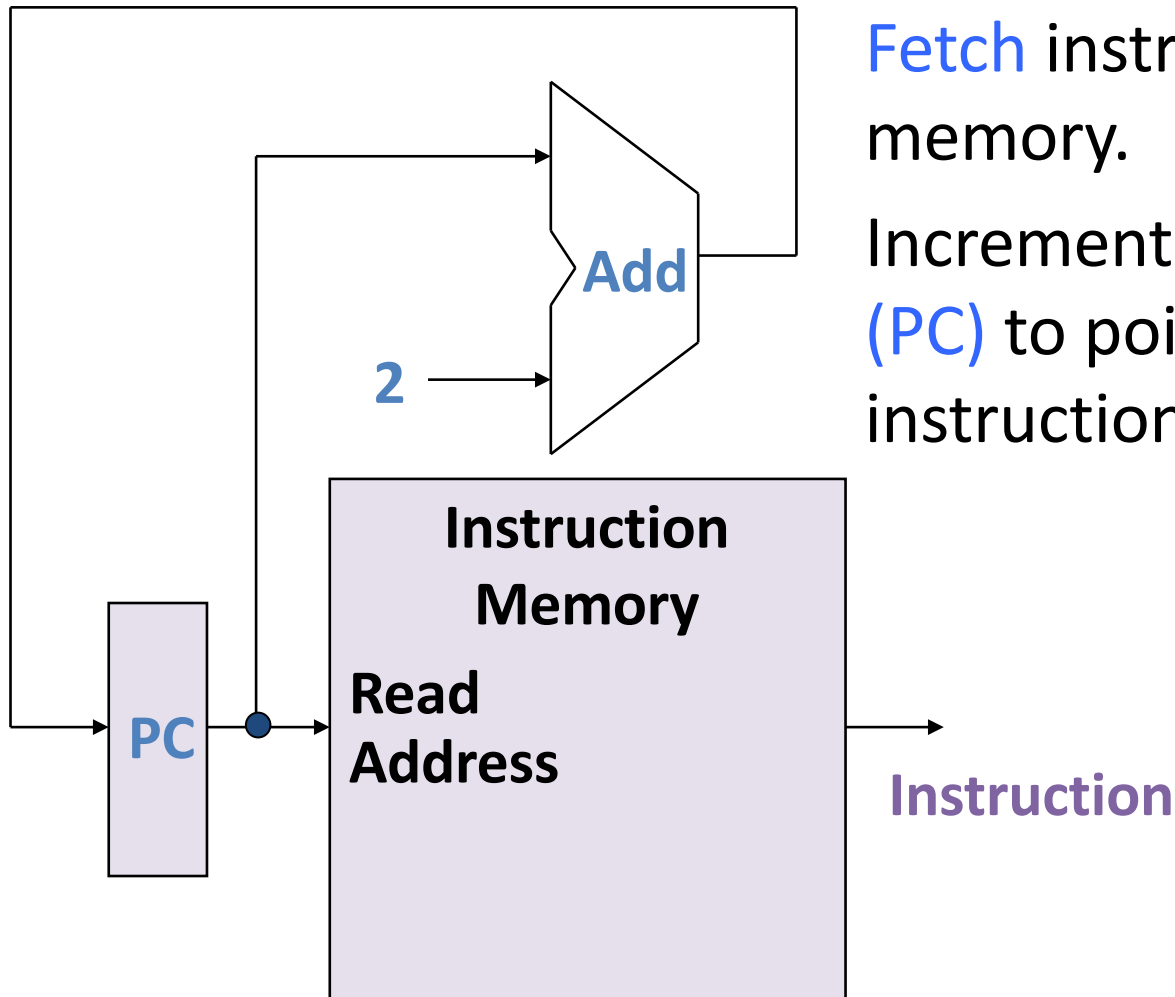


One possible hardware implementation of the HW ISA

Instruction Fetch (default, unless branch or jump)

Processor
Loop

1. $ins \leftarrow IM[PC]$
2. $PC \leftarrow PC + 2$
3. Do *ins*



Fetch instruction from memory.


Increment program counter (PC) to point to the next instruction.

Instruction Encoding: 3 formats

All have 4-bit opcode in MSBs

Arithmetic instructions:

- 2 source register IDs (R_s, R_t)
- 1 destination register ID (R_d)



15:12	11:8	7:4	3:0
<i>opcode</i>	R_s	R_t	R_d

Memory/branch instructions:

- address/source register ID (R_s)
- data/source register ID (R_t)
- 4-bit offset

15:12	11:8	7:4	3:0
<i>opcode</i>	R_s	R_t	<i>offset</i>

Jump instruction:

- 12-bit offset

15:12	11:0
<i>opcode</i>	<i>offset</i>

Arithmetic Instructions

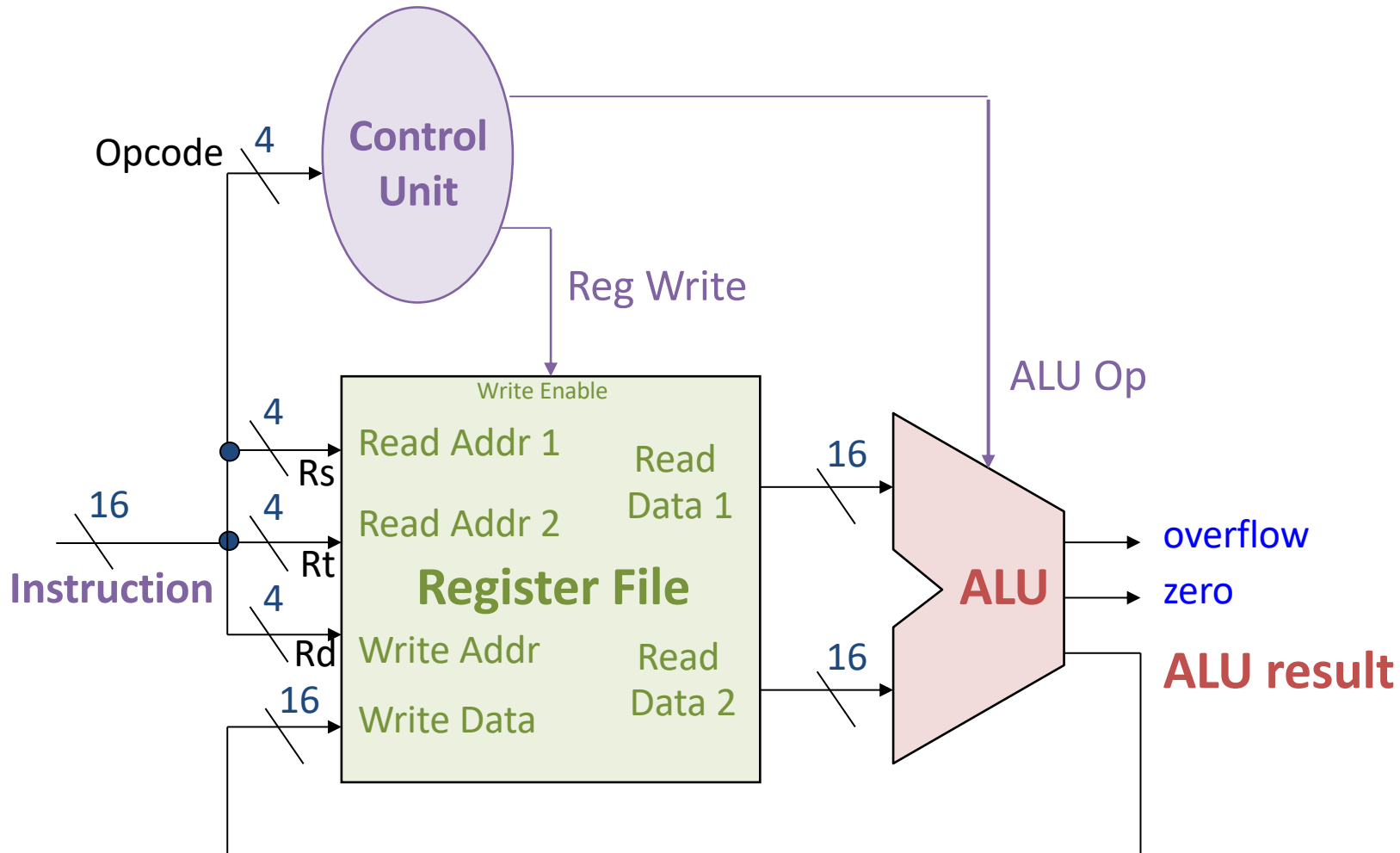
ADD R3, R6, R8

Opcode	Rs	Rt	Rd
0010	0011	0110	1000

16-bit Encoding

Instruction	Meaning	Opcode	Rs	Rt	Rd
ADD R_s, R_t, R_d	$R[d] \leftarrow R[s] + R[t]$	0010	0-15	0-15	0-15
SUB R_s, R_t, R_d	$R[d] \leftarrow R[s] - R[t]$	0011	0-15	0-15	0-15
AND R_s, R_t, R_d	$R[d] \leftarrow R[s] \& R[t]$	0100	0-15	0-15	0-15
OR R_s, R_t, R_d	$R_d \leftarrow R[s] R[t]$	0101	0-15	0-15	0-15
...					

Arithmetic Instructions: Instruction Decode, Register Access, ALU



Memory Instructions

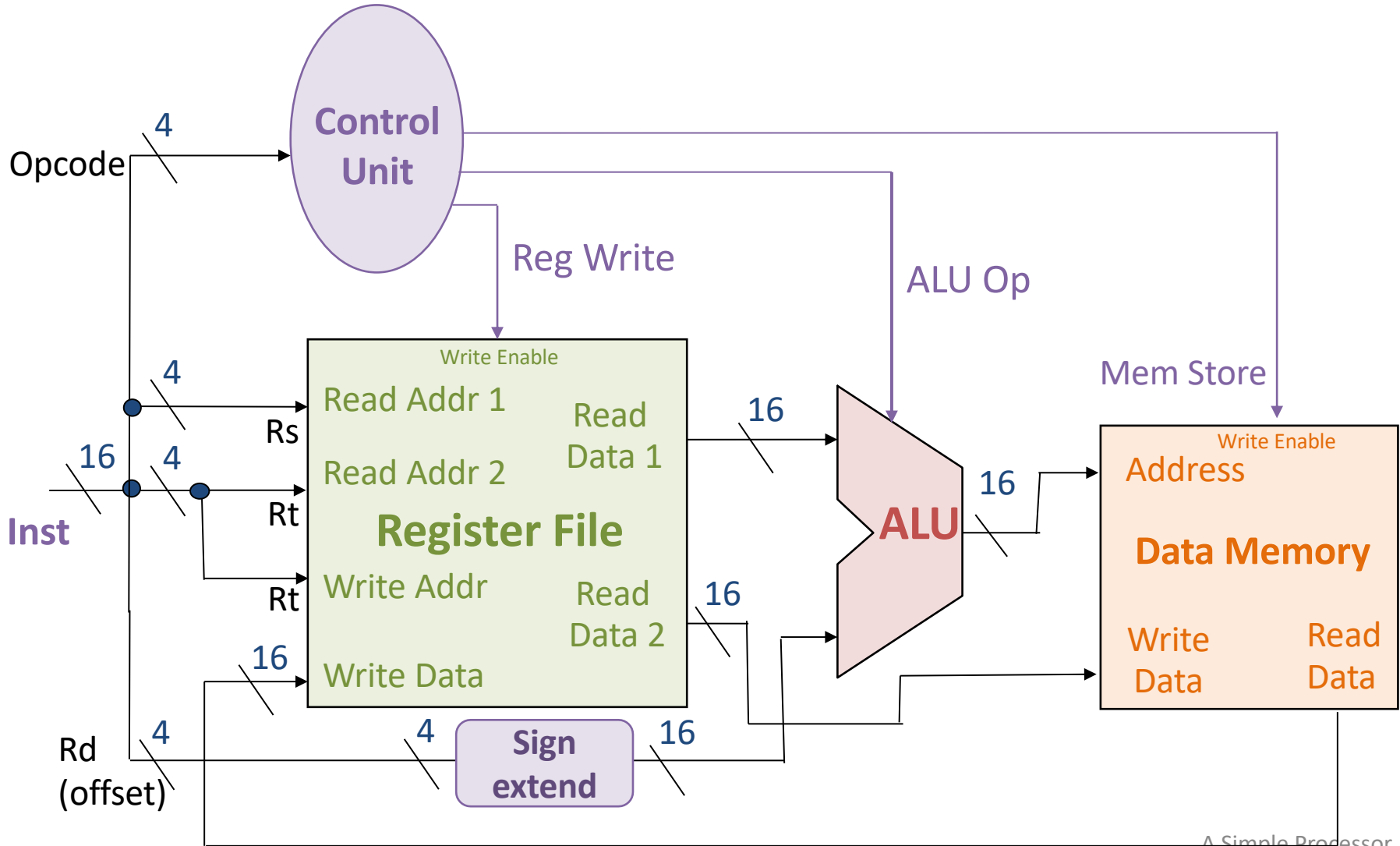
SW R6, -8(R3)

Opcode	Rs	Rt	Rd
0001	0011	0110	1000

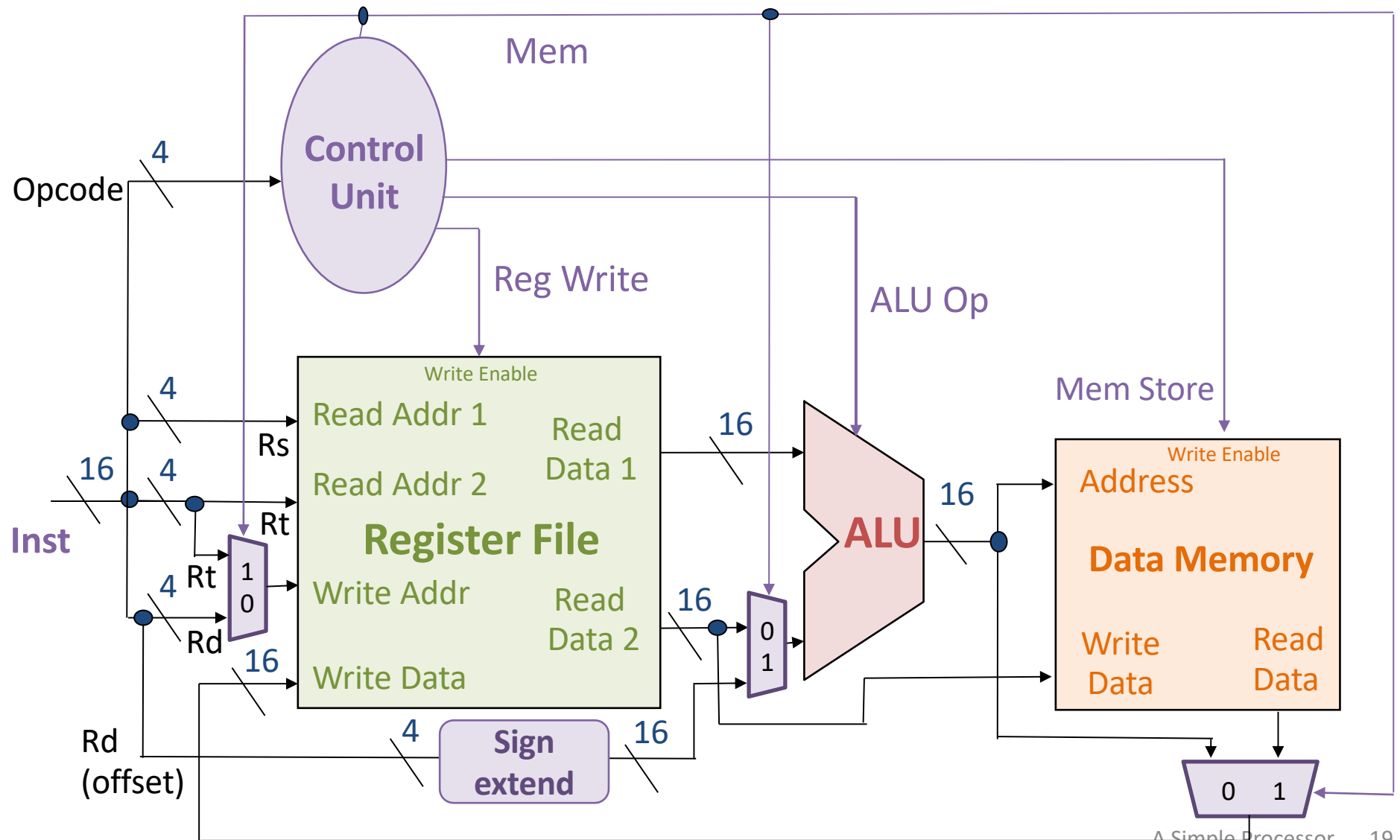
Instruction	Meaning	Op	Rs	Rt	Rd
LW <i>Rt</i> , <i>offset</i> (<i>Rs</i>)	$R[t] \leftarrow \text{Mem}[R[s] + \text{offset}]$	0000	0-15	0-15	<i>offset</i>
SW <i>Rt</i> , <i>offset</i> (<i>Rs</i>)	$\text{Mem}[R[s] + \text{offset}] \leftarrow R[t]$	0001	0-15	0-15	<i>offset</i>
...					

Memory Instructions: Instruction Decode, Register/Memory Access, ALU

How can we support arithmetic
and memory instructions?
What's shared?



Choose between Arithmetic/Memory Instructions with MUXs



Control-flow Instructions

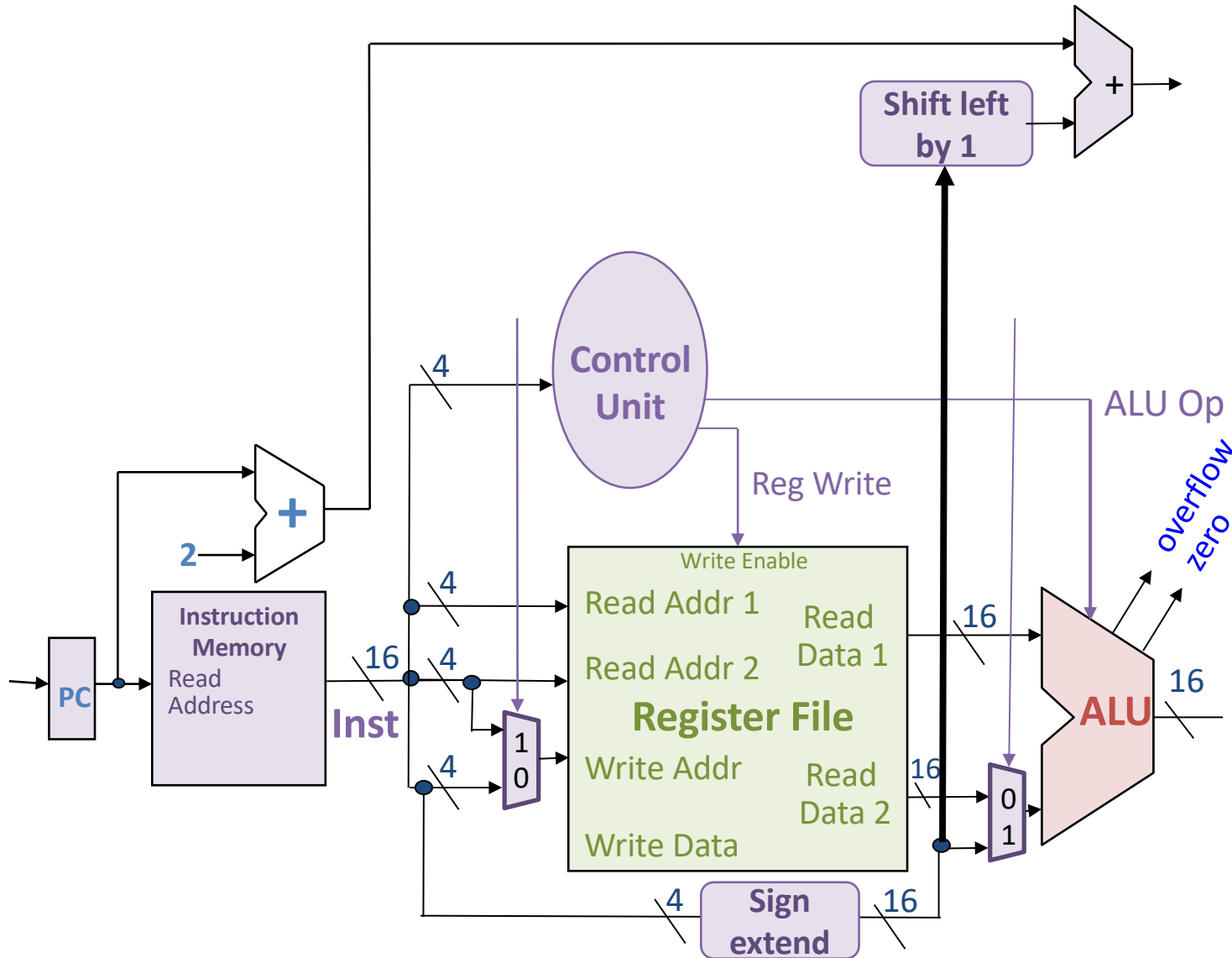
BEQ R1, R2, -2

Op	Rs	Rt	Rd
0111	0001	0010	1110

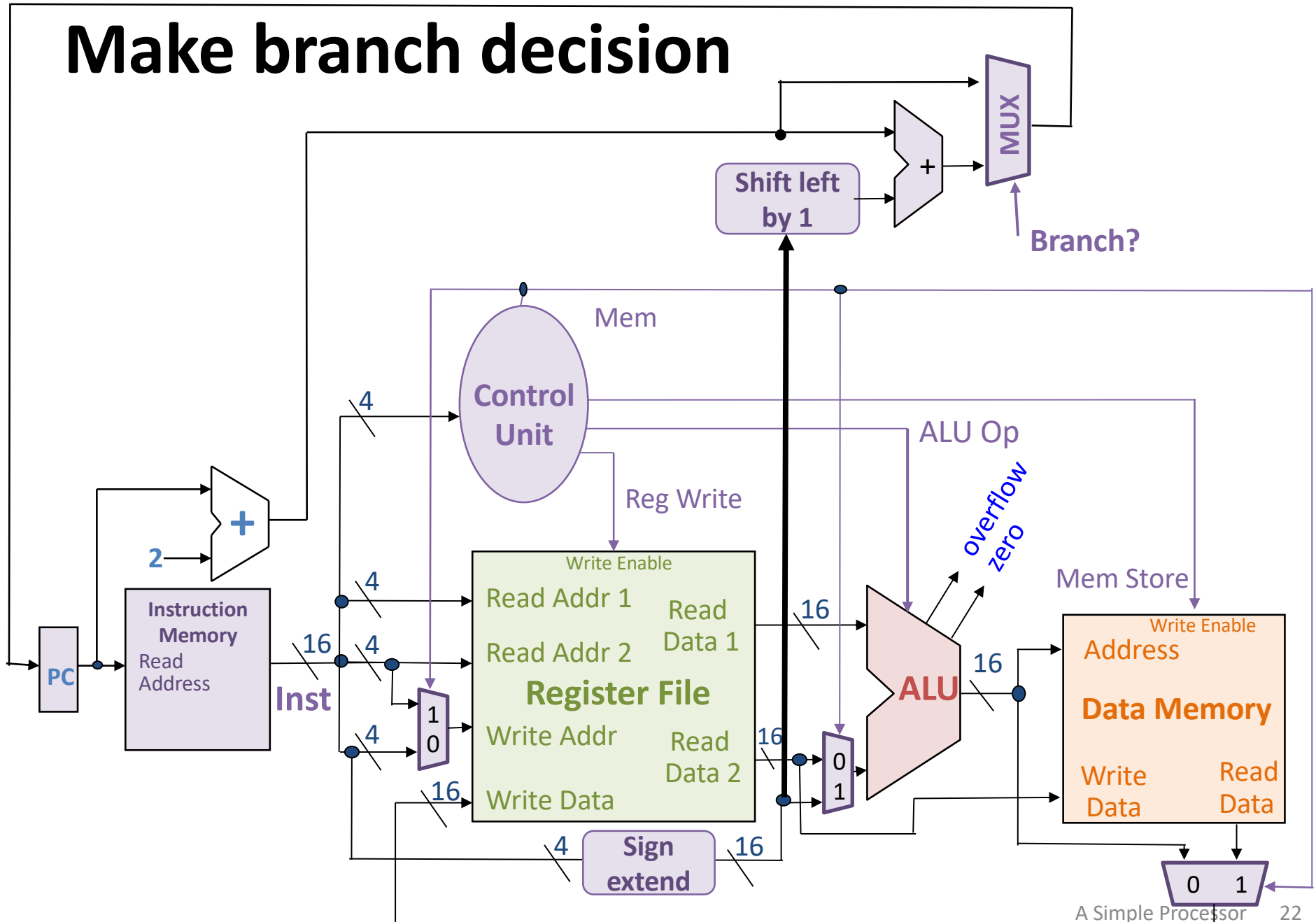
16-bit Encoding

Instruction	Meaning	Op	Rs	Rt	Rd
BEQ <i>Rs, Rt, offset</i>	<i>If R[s] == R[t] then PC ← PC + 2 + offset*2</i>	0111	0-15	0-15	<i>offset</i>
...					

Compute branch target for BEQ



Make branch decision



What's missing?

- Details of Control Unit
 - ALU op is **not** instruction opcode; some translation involved
 - Reg Write bit (for ADD, SUB, AND, OR, LW)
 - Mem Store bit (for SW)
 - Mem bit (arithmetic/memory MUX bit)
 - Branch bit (for BEQ)
- Implementation of JMP
- Implementation of HALT (basically stops the clock running the computer; we won't implement this)

See Lab 5 and Arch Assignment!

HW ARCH: not the only implementation

Single-cycle architecture

- Simple, "easily" fits on a slide (and in your head).
- One instruction takes one clock cycle.
- Slowest instruction determines minimum clock cycle.
- Inefficient.

Could it be better?

- Performance, energy, debugging, security, reconfigurability, ...
- Pipelining
- OoO: out-of-order execution
- SIMD: single instruction multiple data
- Caching
- Microcode vs. direct hardware implementation
- ... enormous, interesting design space of **Computer Architecture**