

Software

Program, Application

Programming Language

Compiler/Interpreter

Operating System

**Instruction Set Architecture**

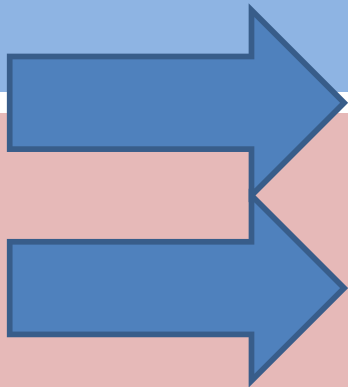
**Microarchitecture**

Digital Logic

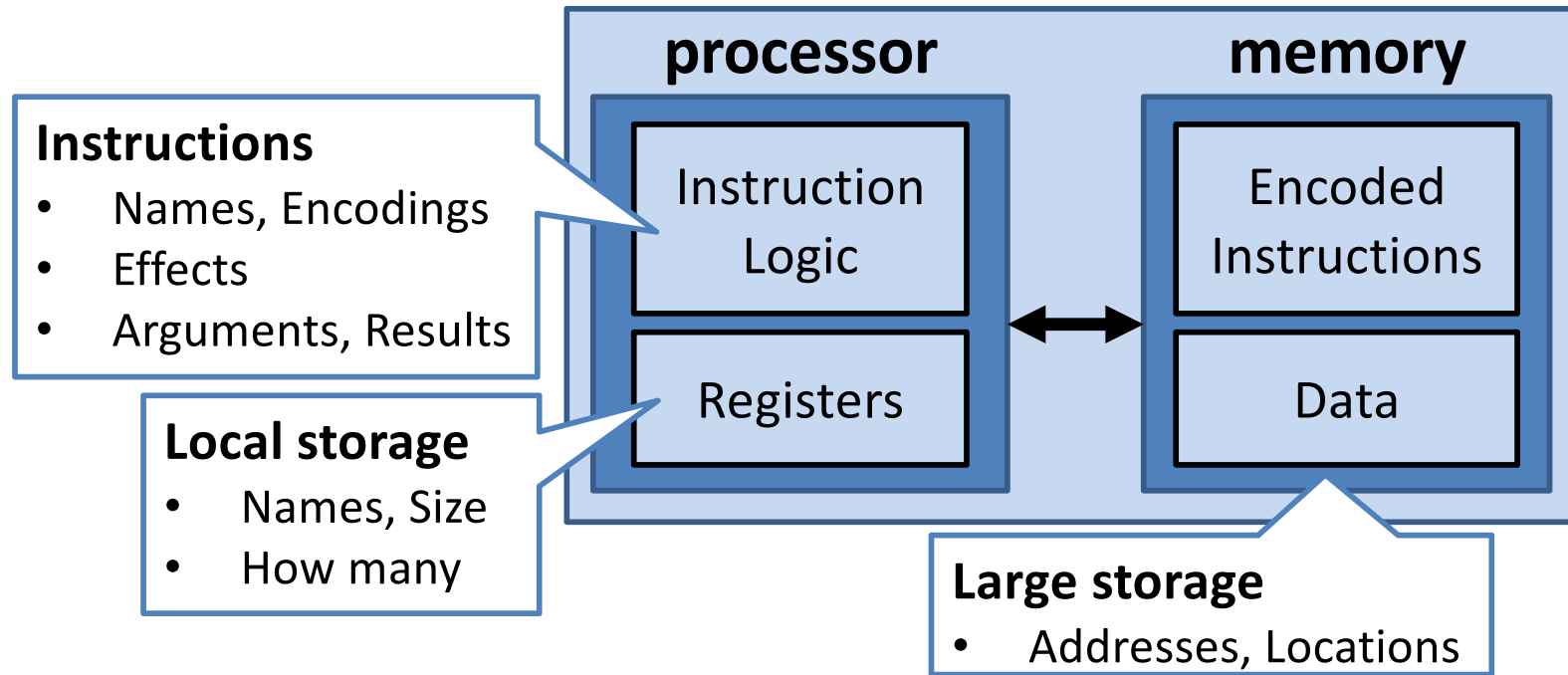
Devices (transistors, etc.)

Solid-State Physics

Hardware



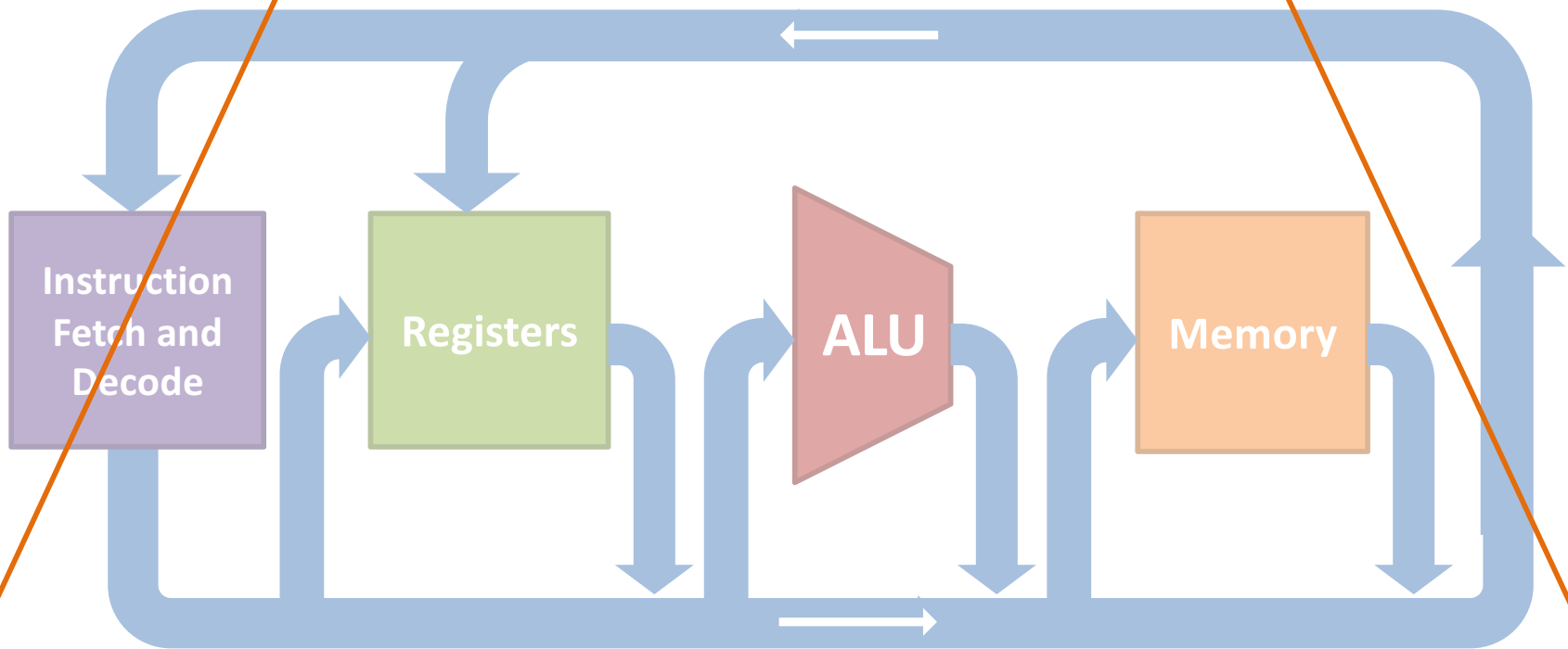
# Instruction Set Architecture (HW/SW **Interface**)



**Computer**

# Computer

## Microarchitecture (Implementation of ISA)



# HW ISA

(HW = Hardware or Hogwarts?)

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

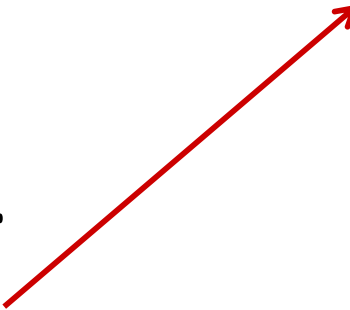
- R0 always holds hardcoded 0
- R1 always holds hardcoded 1
- R2 – R15: general purpose

**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  $ins$

## IM: Instruction Memory

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

HW ISA Abstract  
Machine

# HW ISA Instructions

MSB **16-bit Encoding** LSB

Assembly Syntax	Meaning	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>	$R[t] \leftarrow M[R[s] + \textit{offset}]$	0000	<i>s</i>	<i>t</i>	<i>offset</i>
SW <i>Rt, offset(Rs)</i>	$M[R[s] + \textit{offset}] \leftarrow R[t]$	0001	<i>s</i>	<i>t</i>	<i>offset</i>
BEQ <i>Rs, Rt, offset</i>	If $R[s] == R[t]$ then $PC \leftarrow PC + \textit{offset} * 2$	0111	<i>s</i>	<i>t</i>	<i>offset</i>
JMP <i>offset</i>	$PC \leftarrow \textit{offset} * 2$	1000	<i>o</i> <i>f</i>	<i>f</i> <i>s</i>	<i>e</i> <i>t</i>

(R = register file, M = memory)

## R: Register File

Reg	Contents	Reg	Contents
R0	0x0000	R8	
R1	0x0001	R9	2
R2		R10	3
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

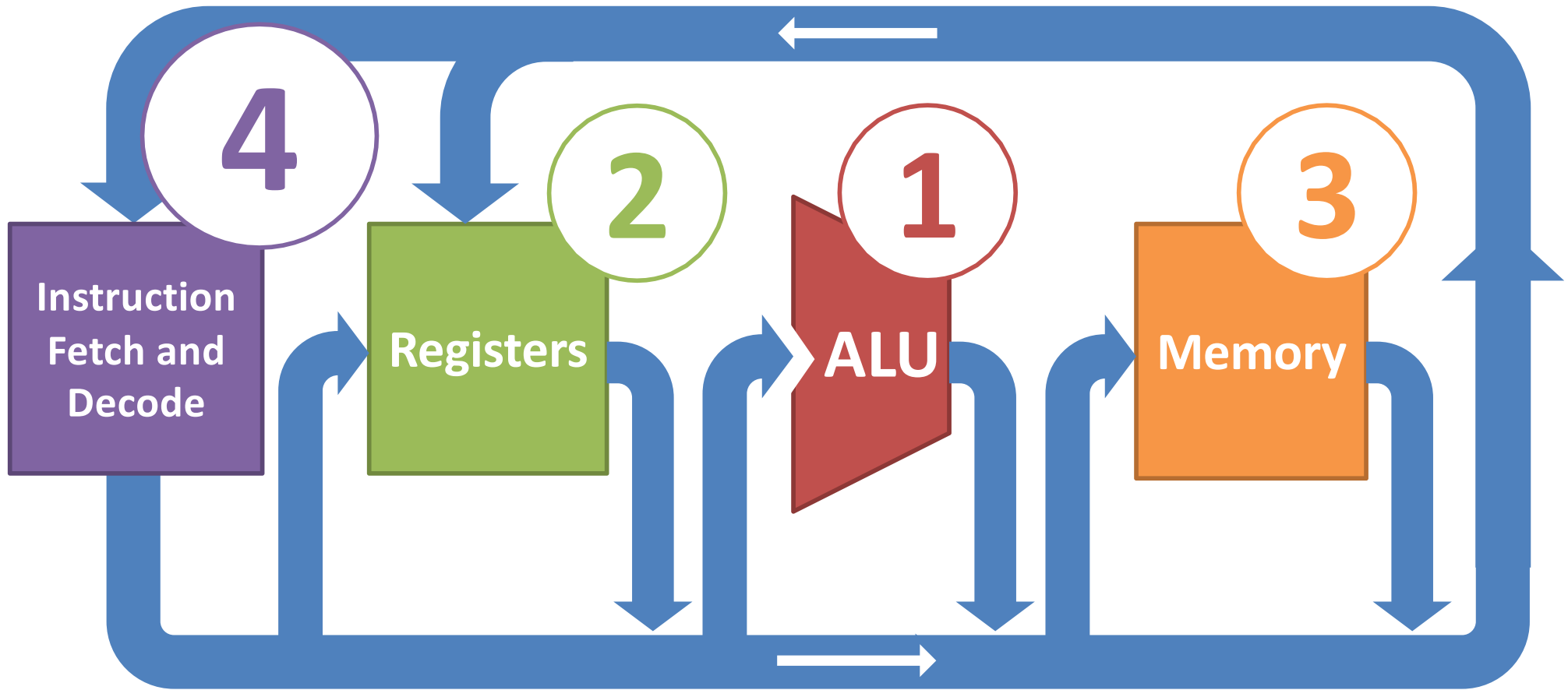
1.  $ins \leftarrow IM[PC]$
2.  $PC \leftarrow PC + 2$
3. Do  $ins$

HW ISA Abstract  
Machine

## IM: Instruction Memory

Address	Contents
0x0 – 0x1	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 microarchitecture



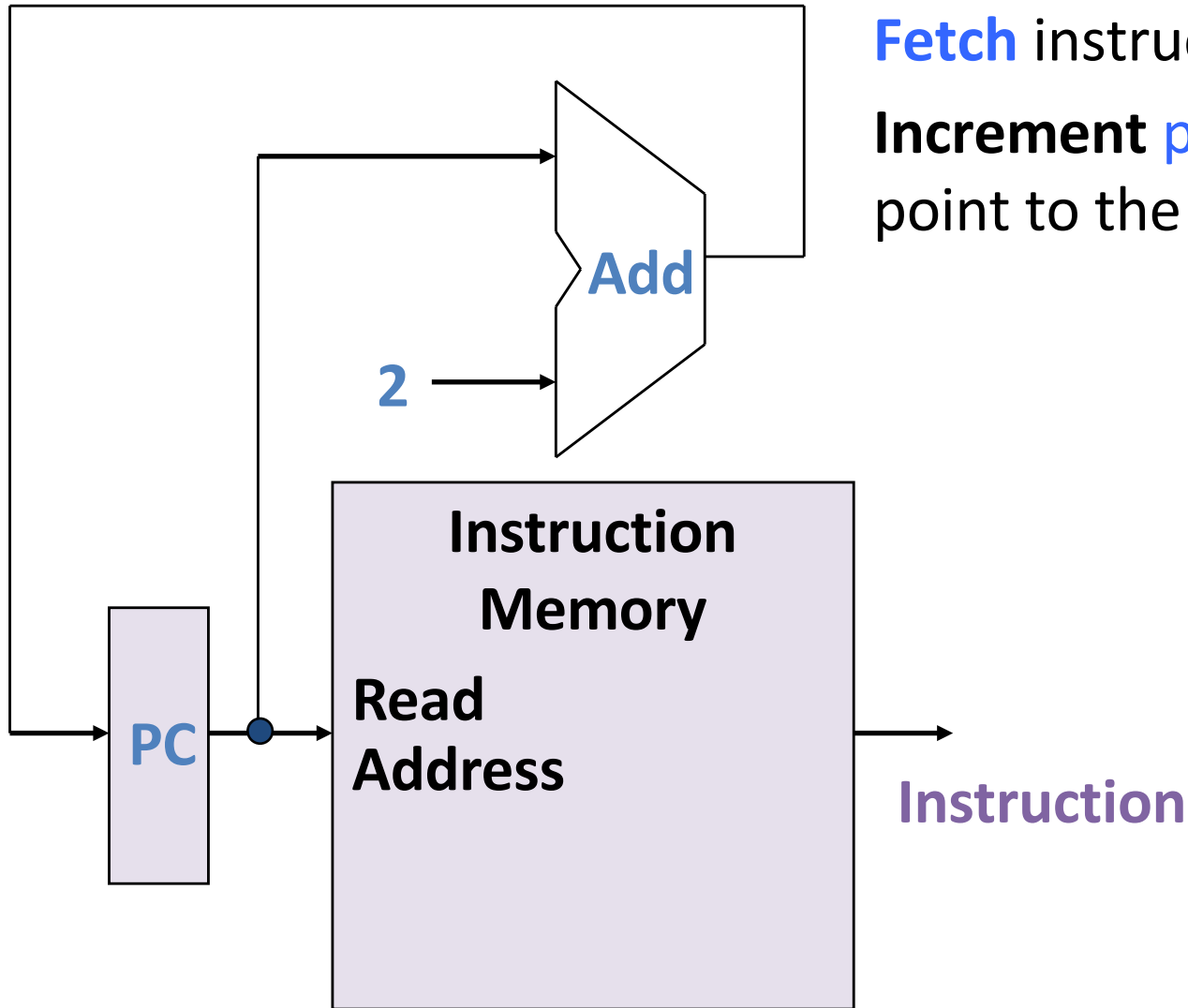
Hardware implementation of the HW ISA



# Instruction Fetch

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.

# Arithmetic Instructions

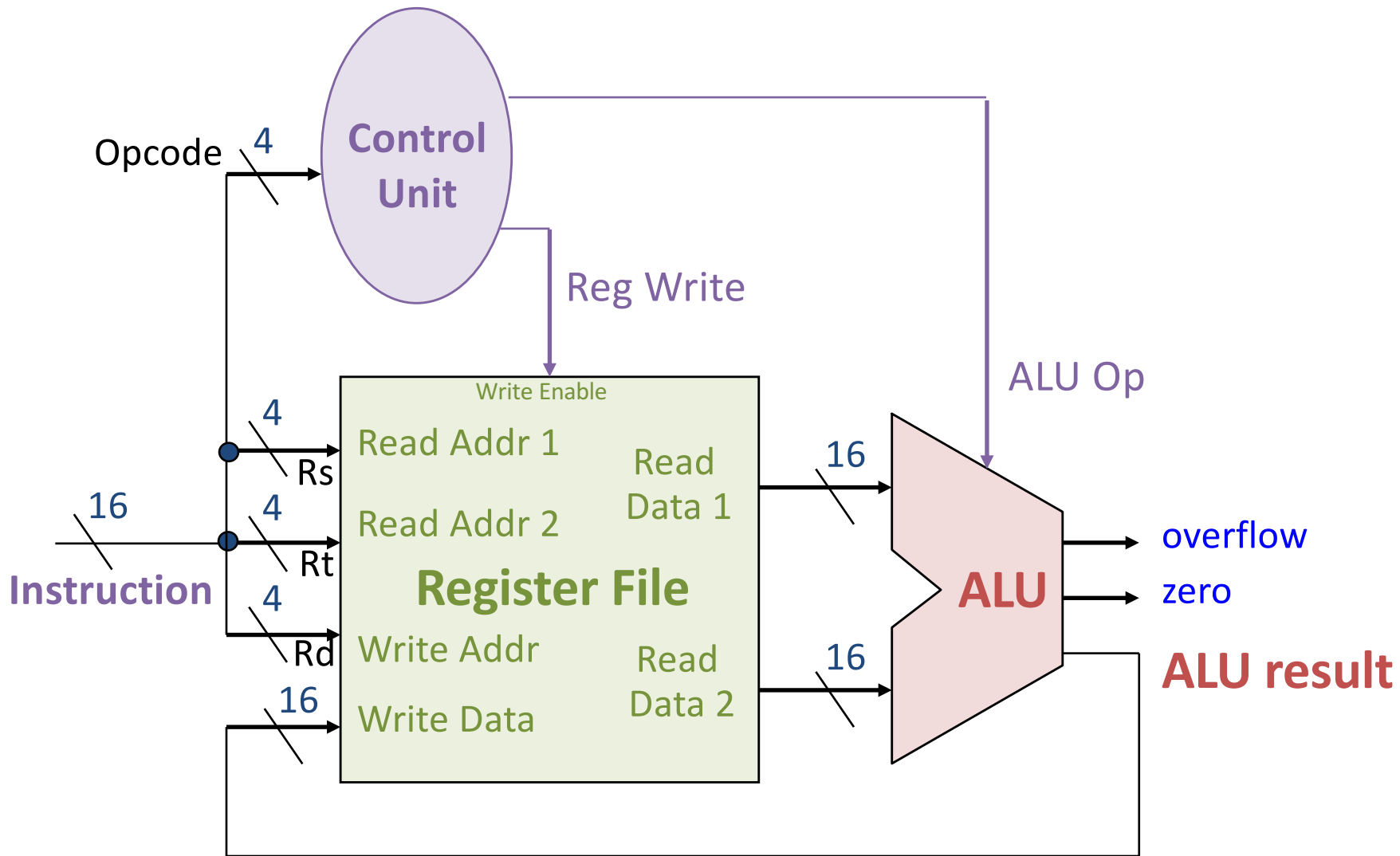
ADD R3, R6, R8

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
...					

# Instruction Decode, Register Access, ALU



# Memory Instructions

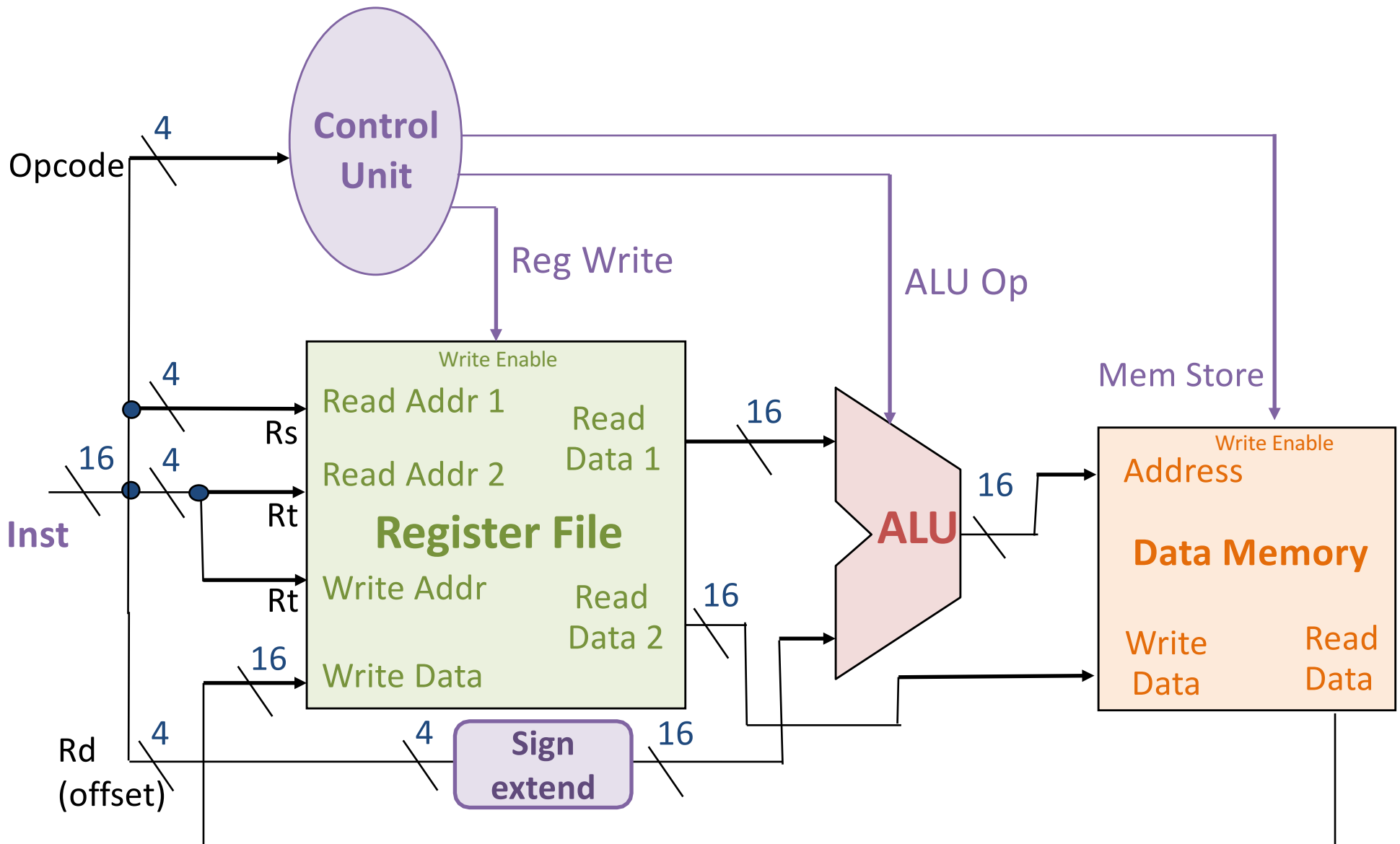
SW R6, -8(R3)

Opcode	Rs	Rt	Rd
0001	0011	0110	1000

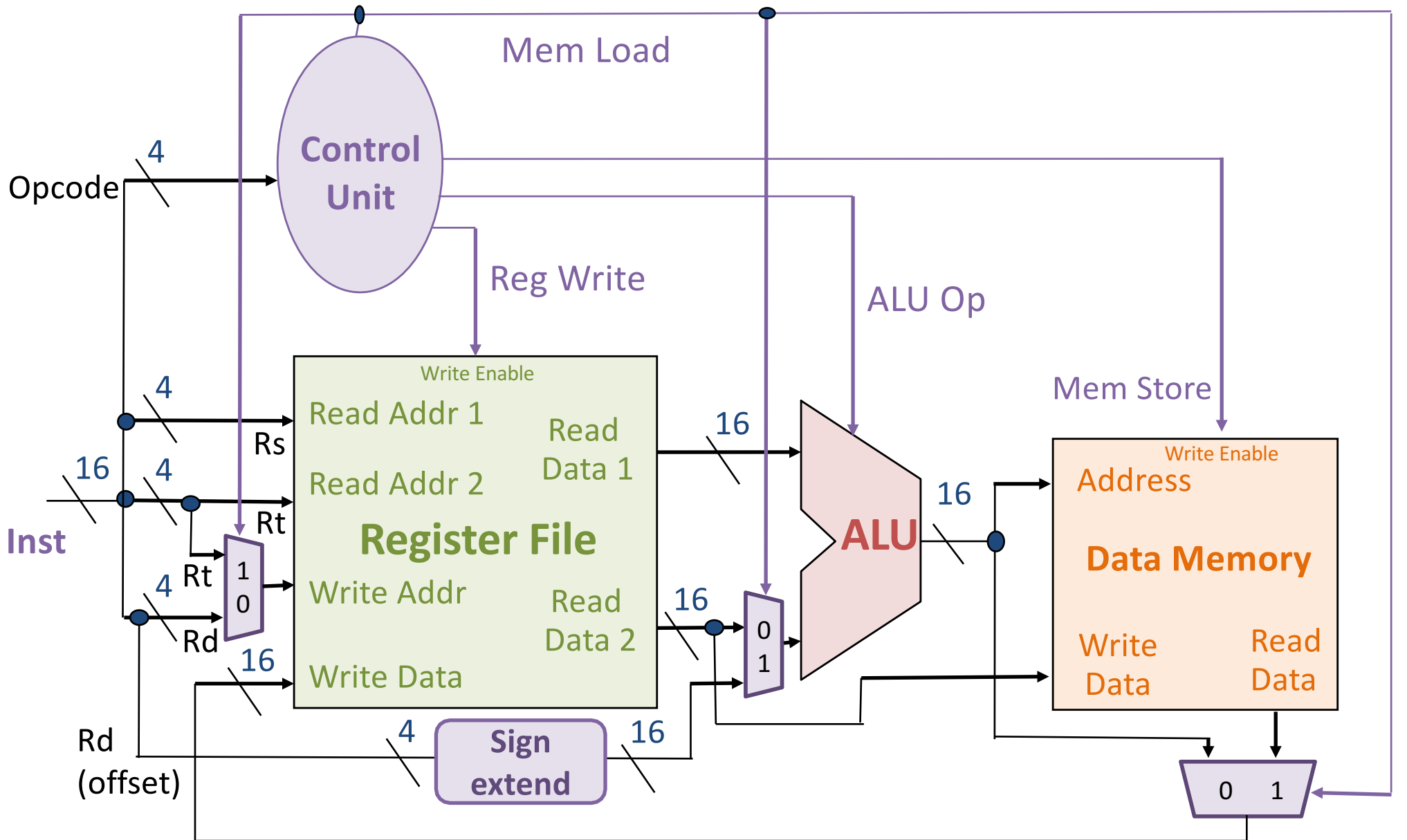
Instruction	Meaning	Op	Rs	Rt	Rd
<i>LW Rt, offset(Rs)</i>	$R[t] \leftarrow Mem[R[s] + offset]$	0000	0-15	0-15	offset
<i>SW Rt, offset(Rs)</i>	$Mem[R[s] + offset] \leftarrow R[t]$	0001	0-15	0-15	offset
...					

# Memory access

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



# Choose with MUXs



# Control-flow Instructions

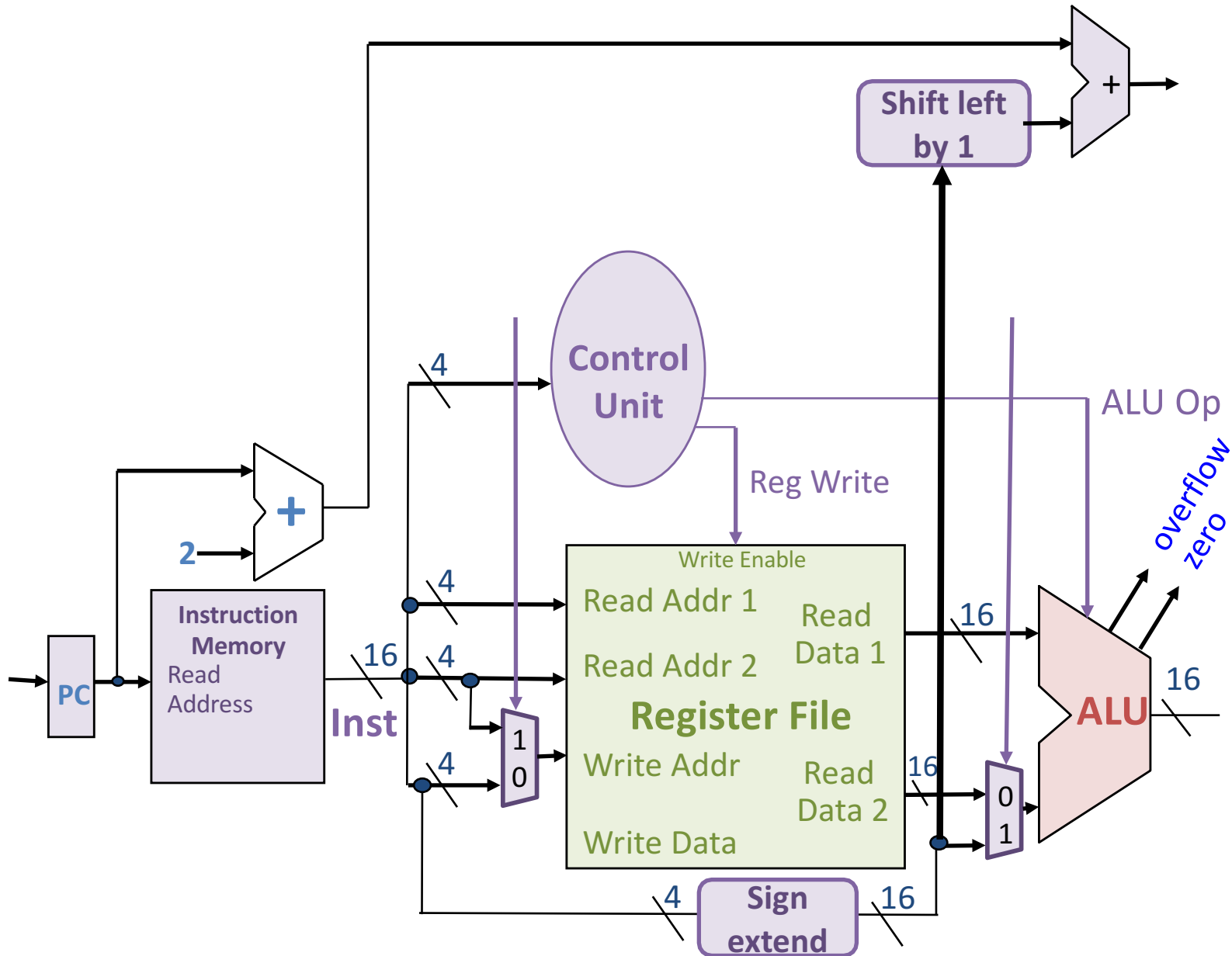
BEQ R1, R2, -4

Op	Rs	Rt	Rd
0111	0001	0010	1110

## 16-bit Encoding

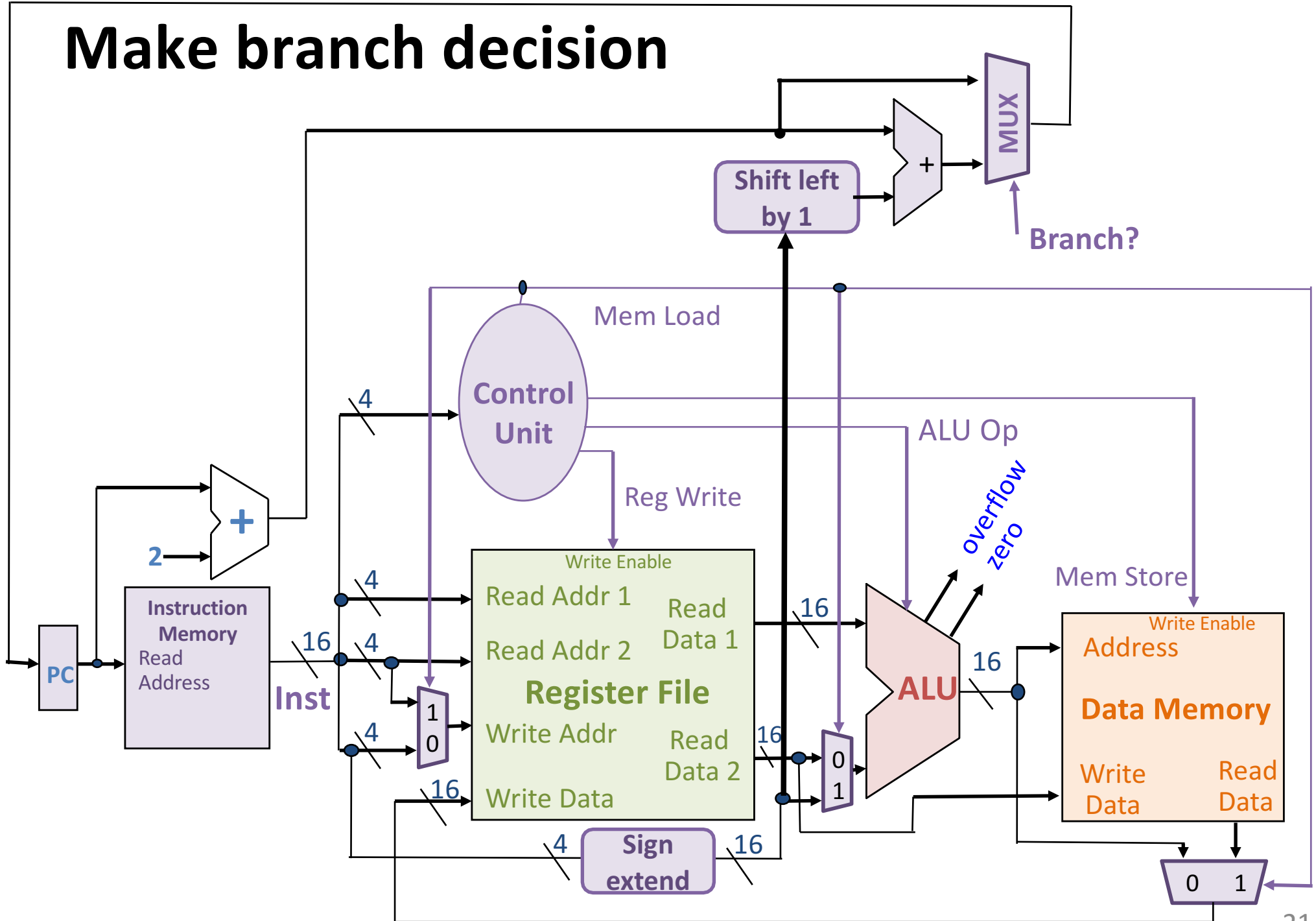
Instruction	Meaning	Op	Rs	Rt	Rd
BEQ <i>Rs, Rt, offset</i>	<p><i>If <math>R[s] == R[t]</math> then</i></p> <p><i><math>PC \leftarrow PC+2 + offset*2</math></i></p> <p><i>Else</i></p> <p><i><math>PC \leftarrow PC+2</math></i></p>	0111	0-15	0-15	offset
...					

# Compute branch target





# Make branch decision



# HW microarchitecture: 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?

- How? 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**