



A Simple Processor

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

<https://cs.wellesley.edu/~cs240/>

1

Motivation

Software

```
int x = y * 2;
```

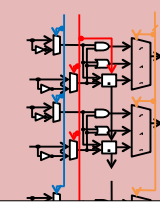
```
int p = q & 0x0000FFFF;
```

```
for (int i = 0; i < 10; i++) {  
    ...  
}
```

How do we connect these?

Hardware

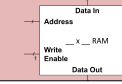
ALU with Adder (compute)



Registers (local data storage)



RAM (larger/longer data storage)



Software

Program, Application

Programming Language

Compiler/Interpreter

Operating System

Instruction Set Architecture

Microarchitecture

Digital Logic

Devices (transistors, etc.)

Solid-State Physics

connection

implementation

3

Instruction Set Architecture (ISA)

Instructions

- Names, Encodings
- Effects
- Arguments, Results
- Abstraction over ALUs

Local storage

- Names, Size
- How many

processor

Instruction Logic

Registers

memory

Encoded Instructions

Data

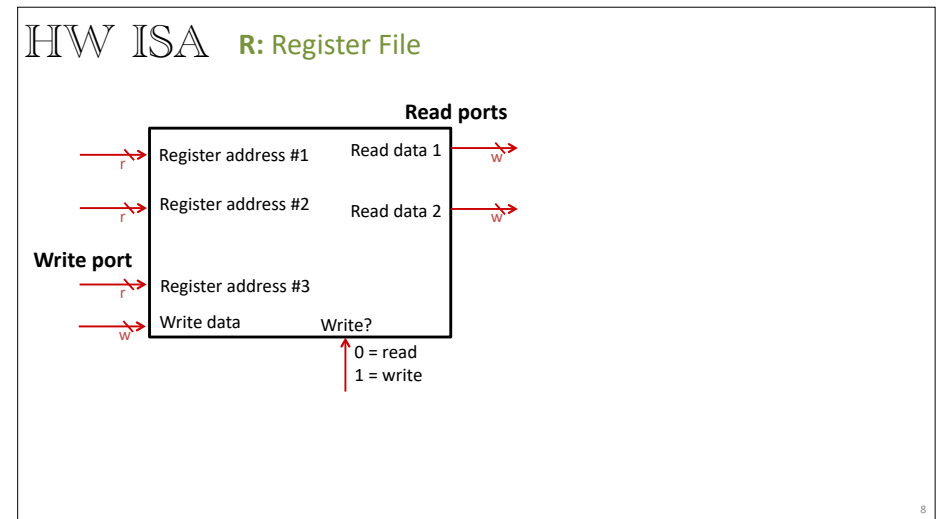
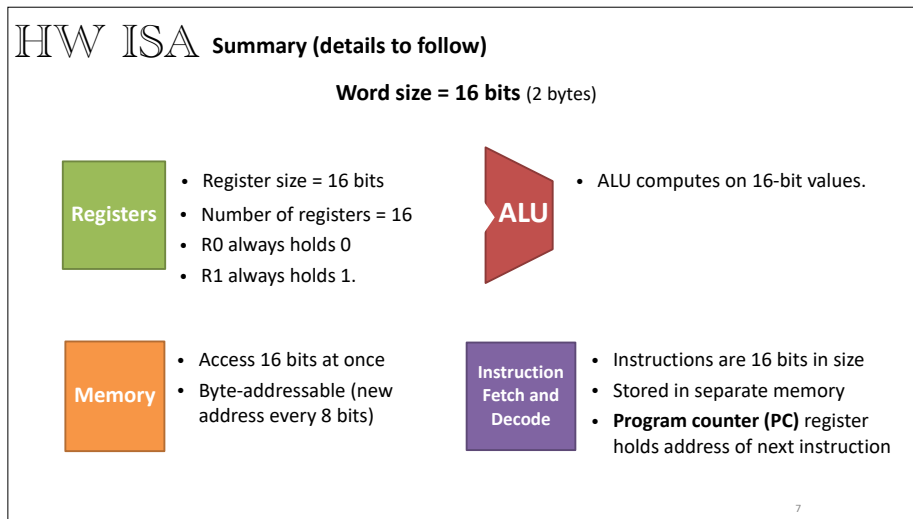
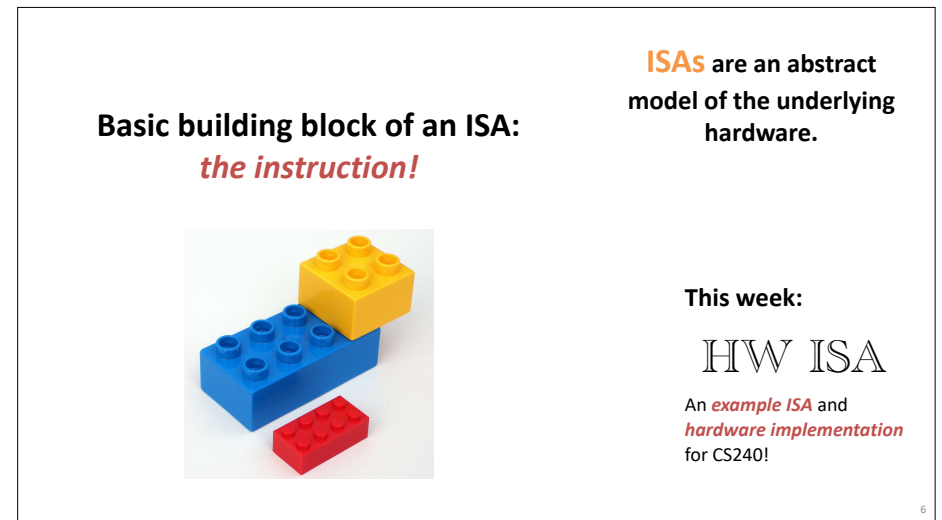
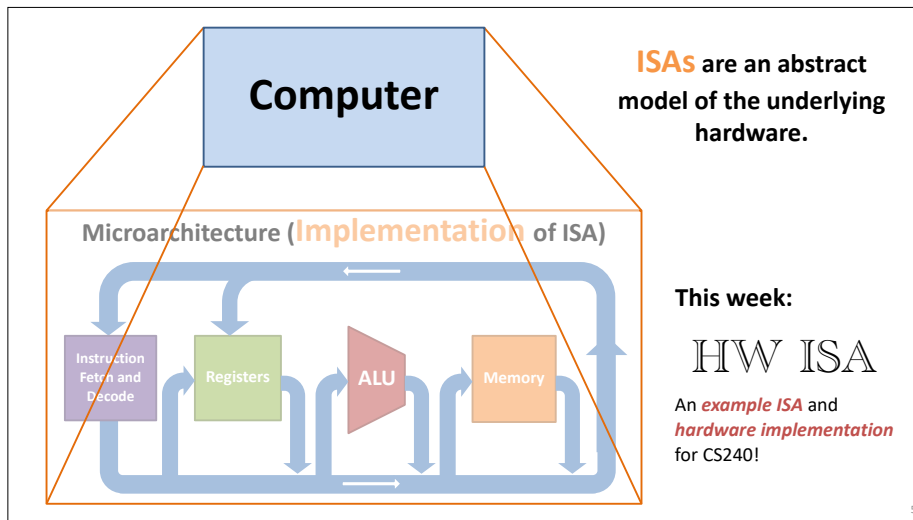
Large storage

- Addresses, Locations

Computer

ISAs define the *interface*
between software and
hardware

4



Using your understanding of powers of 2 needed to make selections, how many bits should be on the labeled busses?

Read ports

Write port

0 = read
1 = write

ex

Word size = 16 bits, # registers = 16

r = ?
w = ?

Options:

- r = 8, w = 8
- r = 16, w = 16
- r = 4, w = 16
- r = 16, w = 4
- None of the above

Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

HW ISA R: Register File

Abstraction!

We'll think of the register file like this:

R0 always holds hardcoded 0
R1 always holds hardcoded 1
R2 – R15: general purpose (instructions can use them to hold anything)

Read ports

Write port

0 = read
1 = write

ex

Word size = 16 bits, # registers = 16

r = ?
w = ?

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

10

HW ISA M: Data Memory

Abstraction!

We'll think of the data memory like this:

Memory is byte-addressable, accesses full words (16 bits)

Memory is "Little Endian": the "little" (low) byte is stored at the lower address.

Example: storing 1 at address 0x0 yields

Address	Contents
0x0 – 0x1	0x01 0x00
0x2 – 0x3	
0x4 – 0x5	
0x6 – 0x7	
0x8 – 0x9	
0xA – 0xB	
0xC – 0xD	
...	

11

What is the full word stored at address 0x2?

0x2345

0x4523

0x2300

0x0023

0x2367

Address	Contents
0x0 – 0x1	0x01 0x00
0x2 – 0x3	0x23 0x45
0x4 – 0x5	0x67 0xab
0x6 – 0x7	
0x8 – 0x9	
0xA – 0xB	
0xC – 0xD	
...	

Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

HW ISA IM: Instruction Memory

Instructions are 1 word in size.

Separate *instruction memory*.

Program Counter (PC) register

- holds address of next instruction to execute.



Program Counter
PC 0x0

Processor Loop

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

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

Abstraction!

We'll think of the instruction memory like this:

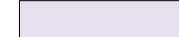
13

HW ISA

Abstraction!

Abstract Machine

PC: Program Counter



Processor Loop

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

M: Data Memory

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

IM: Instruction Memory

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

R: Register File

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

14

HW ISA Instructions

Assembly Syntax	Meaning (R = register file, M = data memory)	16-bit Encoding			
		Opcode	Rs	Rt	Rd
ADD Rs, Rt, Rd	$R[d] \leftarrow R[s] + R[t]$	0010	s	t	d
SUB Rs, Rt, Rd	$R[d] \leftarrow R[s] - R[t]$	0011	s	t	d
AND Rs, Rt, Rd	$R[d] \leftarrow R[s] \& R[t]$	0100	s	t	d
OR Rs, Rt, Rd	$R[d] \leftarrow R[s] R[t]$	0101	s	t	d
LW Rt, offset(Rs)	$R[t] \leftarrow M[R[s] + \text{offset}]$	0000	s	t	offset
SW Rt, offset(Rs)	$M[R[s] + \text{offset}] \leftarrow R[t]$	0001	s	t	offset
BEQ Rs, Rt, offset	If $R[s] == R[t]$ then $PC \leftarrow PC + 2 + \text{offset} * 2$	0111	s	t	offset
JMP offset	$PC \leftarrow \text{offset} * 2$	1000			offset
HALT	Stops program execution	1111			

JMP offset is
unsigned
All other offsets
are signed

15

HW ISA IM: Instruction Memory

Instructions are 1 word in size.

Separate *instruction memory*.

Program Counter (PC) register

- holds address of next instruction to execute.



Program Counter
PC 0x2

Processor Loop

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

Address	Contents
0x0 – 0x1	ADD R0, R1, R2
0x2 – 0x3	SUB R2, R1, R3
0x4 – 0x5	OR R3, R3, R4
0x6 – 0x7	
0x8 – 0x9	
...	

Abstraction!

We'll think of the instruction memory like this:

16

What is the next operation this processor will do?

ADD

SUB

OR

None of the above

Program Counter
PC 0x2

Address Contents
0x0 - 0x1 ADD R0, R1, R2
0x2 - 0x3 SUB R2, R1, R3
0x4 - 0x5 OR R3, R3, R4
0x6 - 0x7
0x8 - 0x9

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

Start the presentation to see live content. For screen share software, share the entire screen. Get help at polllev.com/app

ex Exercise 0

HW ISA

Fill in the rest of the machine state based on this initial state

PC: Program Counter

IM: Instruction Memory

M: Data Memory

R: Register File

Address Contents

0x0 - 0x1	0x0F	0x00
0x2 - 0x3	0x04	0x01
0x4 - 0x5		
0x6 - 0x7		
0x8 - 0x9		
0xA - 0xB		
0xC - 0xD		
...		

Address Contents

0x0 - 0x1	ADD R1, R1, R2
0x2 - 0x3	SW R2, 4(R0)
0x4 - 0x5	HALT
0x6 - 0x7	
0x8 - 0x9	
...	

Reg Contents

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

18

Execution Table for Exercise #0 (shows step-by-step execution)

Solutions

ex

PC	Instr	State Changes
0x0	ADD R1, R1, R2	$R[2] \leftarrow R[1] \& R[1] = 1 + 1 = 0x0002$; $PC \leftarrow PC + 2 = 0 + 2 = 2$
0x2	SW R2, 4(R0)	$M[R[0] + 4] = M[4] \leftarrow R[2] = 0x0002$; $PC \leftarrow PC + 2 = 2 + 2 = 4$
0x4	HALT	Program execution stops

Reminder: the two bytes will be stored in **Little Endian** order when we store them to memory **M**.

That is, the byte 0x02 will be stored in the "little" end of the word—the lower address of the pair of addresses that store the word. 0x00 will be stored at the higher address.

19

ex Exercise 0

Solutions

HW ISA

PC: Program Counter

IM: Instruction Memory

M: Data Memory

R: Register File

Address Contents

0x0 - 0x1	0x0F	0x00
0x2 - 0x3	0x04	0x01
0x4 - 0x5	0x02	0x00
0x6 - 0x7		
0x8 - 0x9		
0xA - 0xB		
0xC - 0xD		
...		

Address Contents

0x0 - 0x1	ADD R1, R1, R2
0x2 - 0x3	SW R2, 4(R0)
0x4 - 0x5	HALT
0x6 - 0x7	
0x8 - 0x9	
...	

Reg Contents

R0	0x0000
R1	0x0001
R2	0x0002
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	

20

ex Exercise 1

HW ISA

Fill in the rest of the machine state based on this initial state

PC: Program Counter

Processor Loop

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

M: Data Memory

Address	Contents
0x0 – 0x1	0x0F 0x00
0x2 – 0x3	0x04 0x01
0x4 – 0x5	
0x6 – 0x7	
0x8 – 0x9	
0xA – 0xB	
0xC – 0xD	
...	

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
...	

R: Register File

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

21

Execution Table for Exercise #1 (shows step-by-step execution)

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

ex

22

ex Exercise 2

HW ISA

Fill in the rest of the machine state based on this initial state

PC: Program Counter

Processor Loop

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

M: Data Memory

Address	Contents
0x0 – 0x1	0x0F 0x00
0x2 – 0x3	0x04 0x01
0x4 – 0x5	
0x6 – 0x7	
0x8 – 0x9	
0xA – 0xB	
0xC – 0xD	
...	

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
...	

R: Register File

Reg	Contents (time: $\rightarrow$)
R0	0x0000
R1	0x0001
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	0x0002
R10	0x0003
R11	
R12	
R13	
R14	
R15	

23

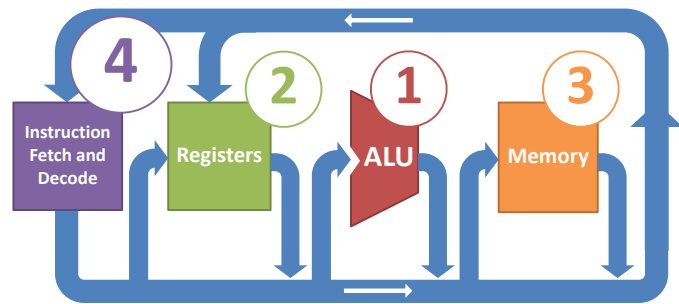
Execution Table for Exercise #2 (shows step-by-step execution)

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

ex

24

HW ARCH microarchitecture



One possible hardware implementation of the HW ISA

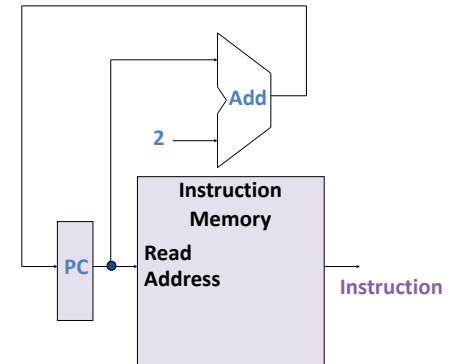
25

Instruction Fetch (default, unless branch or jump)

Fetch instruction from memory.
Increment **program counter (PC)**
to point to the next instruction.

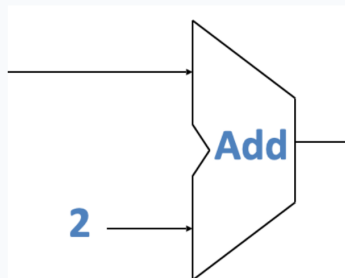
Processor
Loop

```
1. ins ← IM[PC]
2. PC ← PC + 2
3. Do ins
```



26

Which of the following is used inside this unit?



D-flip-flop

Ripple-carry adder

Encoder

A & B

B & C

C & D

Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

Instruction Encoding: 3 formats

All have 4-bit opcode in MSBs

Arithmetic instructions:

- 2 source register IDs (Rs, Rt)
- 1 destination register ID (Rd)

15:12	11:8	7:4	3:0
opcode	Rs	Rt	Rd

Memory/branch instructions:

- address/source register ID (Rs)
- data/source register ID (Rt)
- 4-bit offset

15:12	11:8	7:4	3:0
opcode	Rs	Rt	offset

Jump instruction:

- 12-bit offset

15:12	11:0
opcode	offset

28

Arithmetic Instructions

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
...					

Example encoding:

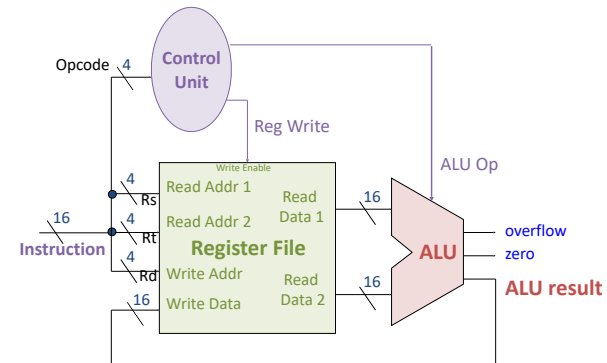
ADD R3, R6, R8

Opcode	Rs	Rt	Rd
0010	0011	0110	1000

29

Arithmetic Instructions:

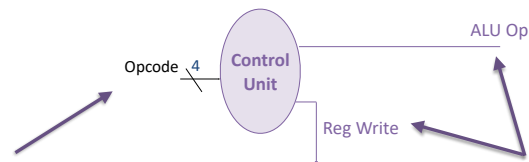
Instruction Decode, Register Access, ALU



30

The control unit

A large instantiation of a truth table that controls parts of the microarchitecture



Input: the opcode from the instructions

Output: many wires controlling decisions

You will implement the control unit on the **Arch** Assignment!

31

Memory Instructions

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

Example encoding:

SW R6, -8(R3)

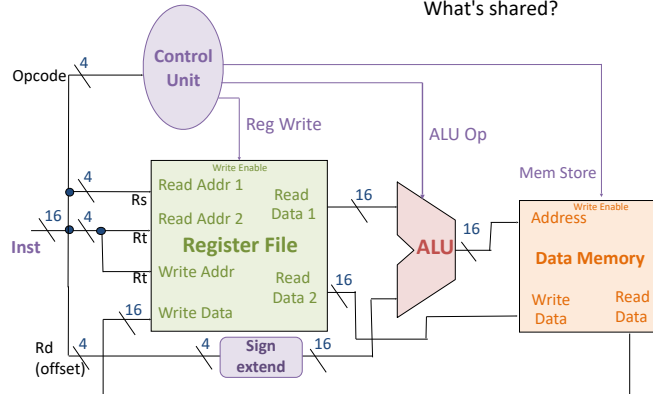
Opcode	Rs	Rt	Rd
0001	0011	0110	1000

32

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

How can we support arithmetic
and memory instructions?

What's shared?

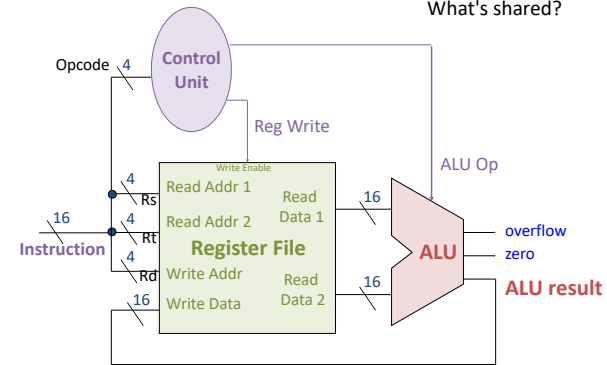


33

Arithmetic Instructions: Instruction Decode, Register Access, ALU

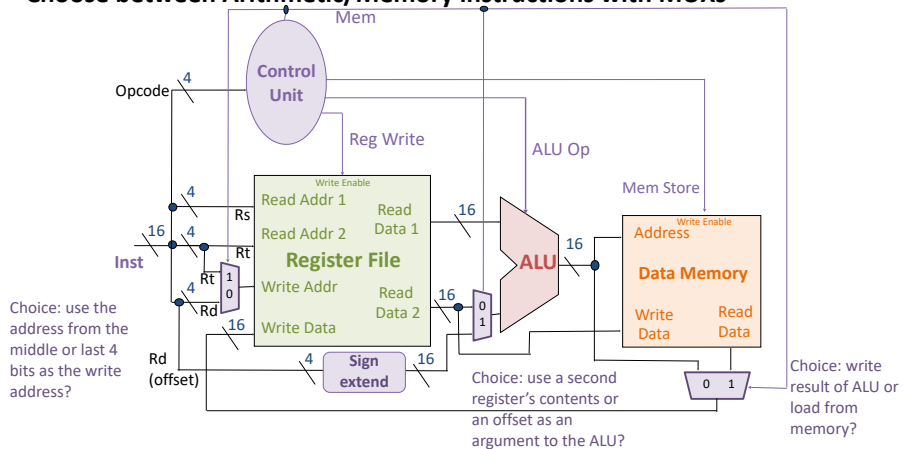
How can we support arithmetic
and memory instructions?

What's shared?



34

Choose between Arithmetic/Memory instructions with MUXs



35

Control-flow Instructions

16-bit Encoding					
Instruction	Meaning	Op	Rs	Rt	Rd
BEQ Rs, Rt, offset	If $R[s] == R[t]$ then $PC \leftarrow PC + 2 + offset * 2$	0111	0-15	0-15	offset
...					

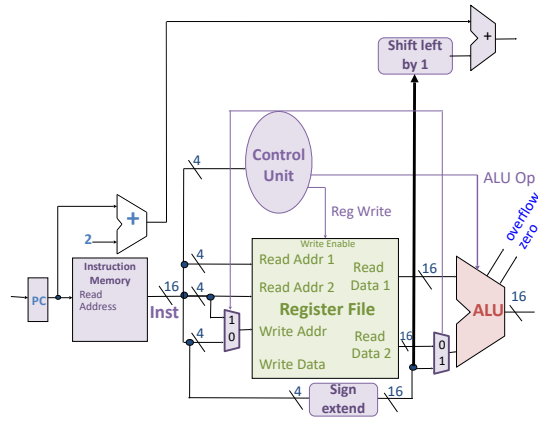
Example encoding:

BEQ R1, R2, -2

Op	Rs	Rt	Rd
0111	0001	0010	1110

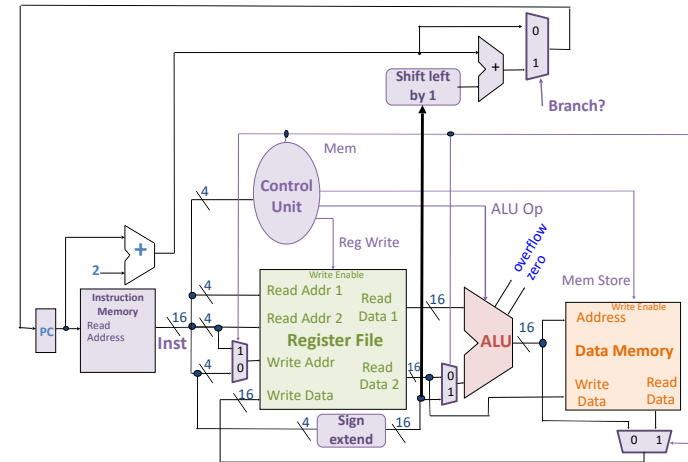
36

Compute branch target for BEQ



37

Make branch decision



38

What's missing from what we covered in lecture?

- Details of Control Unit
 - ALU op is **not** instruction opcode; some translation needed
 - 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 **Arch** Assignment!

39

HW ARCH not the only implementation

Single-cycle architecture

- Relatively simple, (barely!) fits on a slide (and in our heads).
- Every instruction takes one clock cycle each.
- Slowest instruction determines minimum clock cycle.
- Inefficient.

Could it be better?

- Performance, energy, debugging, security, reconfigurability, ...
- Pipelining
- OoO: Out-of-order execution
- Caching
- ... enormous, interesting design space of **Computer Architecture**

10

Conclusion of unit: Computational Building Blocks (HW)

Lectures

Digital Logic
Data as Bits
Integer Representation
Combinational Logic
Arithmetic Logic
Sequential Logic
A Simple Processor

Labs

1: Transistors to Gates
2: Data as Bits
3: Combinational Logic & Arithmetic
4: ALU & Sequential Logic
5: Processor Datapath

Topics

Transistors, digital logic gates
Data representation with bits, bit-level computation
Number representations, arithmetic
Combinational and arithmetic logic
Sequential (stateful) logic
Computer processor architecture overview

Assignments

Gates
Zero
Bits
Arch (out Thursday)

Mid-semester Exam 1: HW
October 16