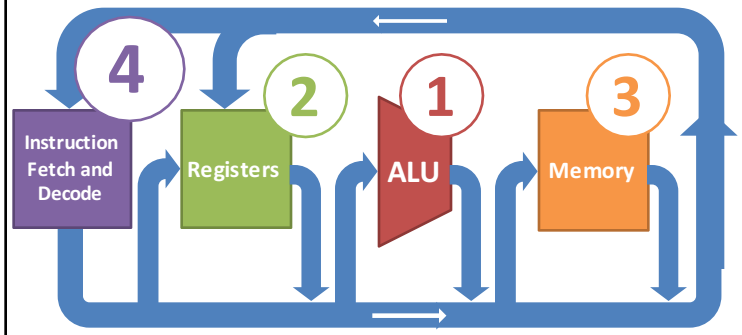
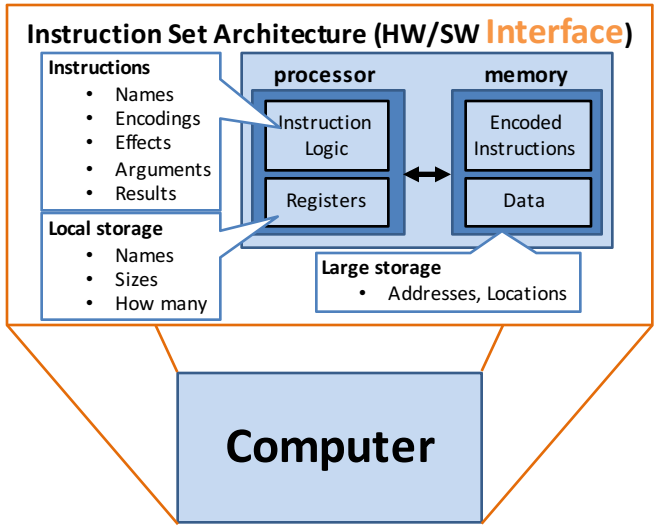
A tiny ISA and data path



Instruction Set Architecture (HW/SW Interface)



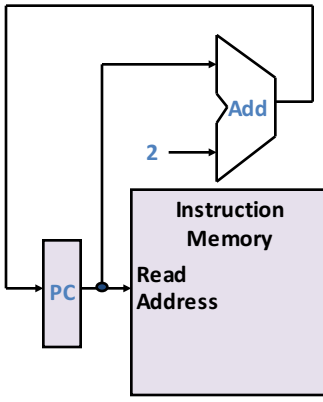
A tiny ISA and data path

- **Word size = 16 bits, data bus = 16 bits.**
  - Register size = 16 bits.
  - ALU and memory handle 16-bit values.
- **Memory is byte-addressable.**
- **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*.**
- Each instruction executes in a single clock cycle.
- Special **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
…	…
255	256th instruction, high-order byte

3

Instruction Fetch



Fetch instruction from memory.

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

4

Arithmetic Instructions and Encodings

ADD R3, R6, R8

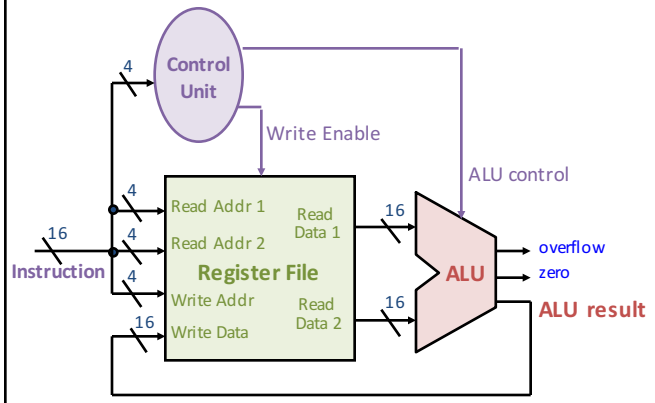
Op	Rs	Rt	Rd
0010	0011	0110	1000

16-bit Encoding

Instruction	Meaning	Opcode	Rs	Rt	Rd
ADD *Rs, Rt, Rd*	*Rd := Rs + Rt*	0010	0-15	0-15	0-15
SUB *Rs, Rt, Rd*	*Rd := Rs - Rt*	0011	0-15	0-15	0-15
AND *Rs, Rt, Rd*	*Rd := Rs & Rt*	0100	0-15	0-15	0-15
OR *Rs, Rt, Rd*	*Rd := Rs | Rt*	0101	0-15	0-15	0-15
...					

5

Instruction Decode, Register Access, ALU



6

Memory Instructions and Encodings

SW R6, 8(R3)

Op	Rs	Rt	Rd
0001	0011	0110	1000

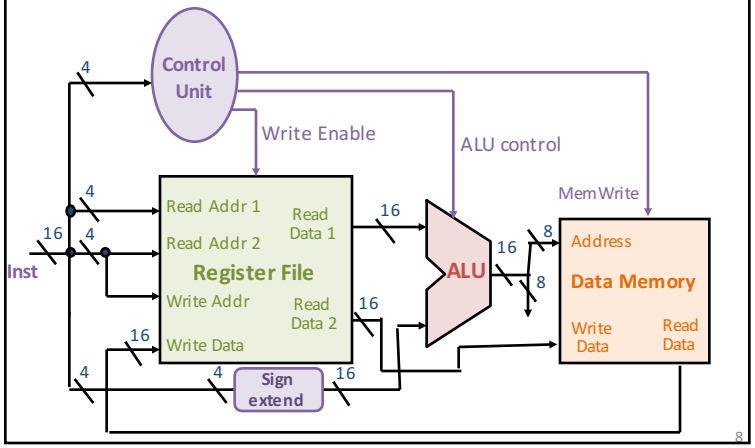
16-bit Encoding

Instruction	Meaning	Opcode	Rs	Rt	Rd
LW *Rt, offset(Rs)*	*Rt := Memory[Rs + offset]*	0000	0-15	0-15	offset
SW *Rt, offset(Rs)*	*Memory[Rs + offset] := Rt*	0001	0-15	0-15	offset
...					

7

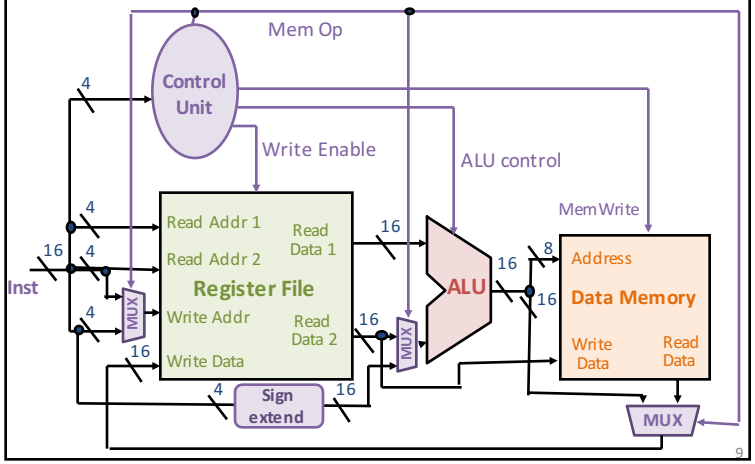
Memory access

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



8

MUXes to the rescue!



9

More questions:

Next time:

What's inside the Control Unit?

How do we support programs with *conditionals* and *loops*?

Next few weeks:

How do these instructions relate to the programs I write?

10

Control Instructions and Encodings

BEQ R1, R2, 28

Op	Rs	Rt	Rd
0111	0001	0010	1110

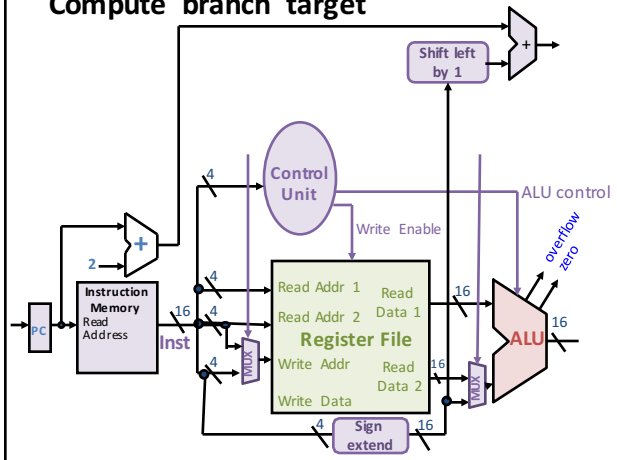
16-bit Encoding

Instruction	Meaning	Opcode	Rs	Rt	Rd
BEQ *Rs, Rt, offset*	*If Rs == Rt then* *PC := PC + 2 + offset*2* *Else* *PC := PC + 2 (normal)*	0111	0-15	0-15	offset
JMP *offset*	*PC := offset*2*	1000	O	f f	s e t
…					

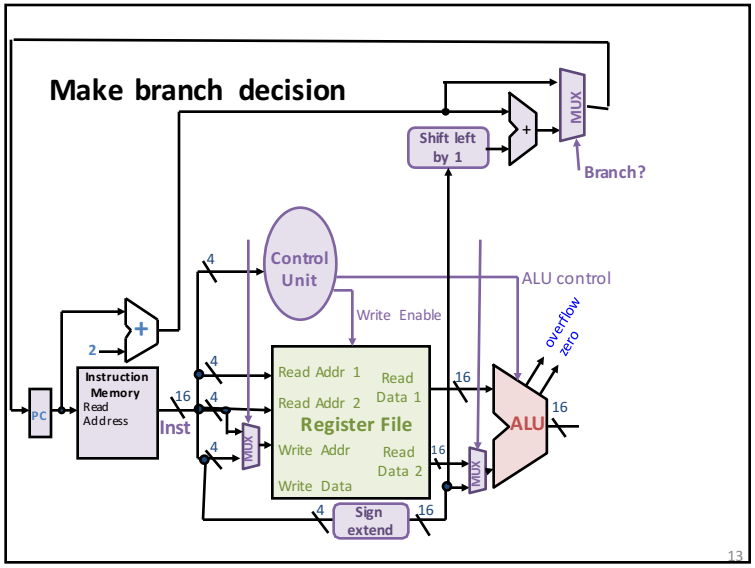
Use these to implement: if-else, loops, etc.

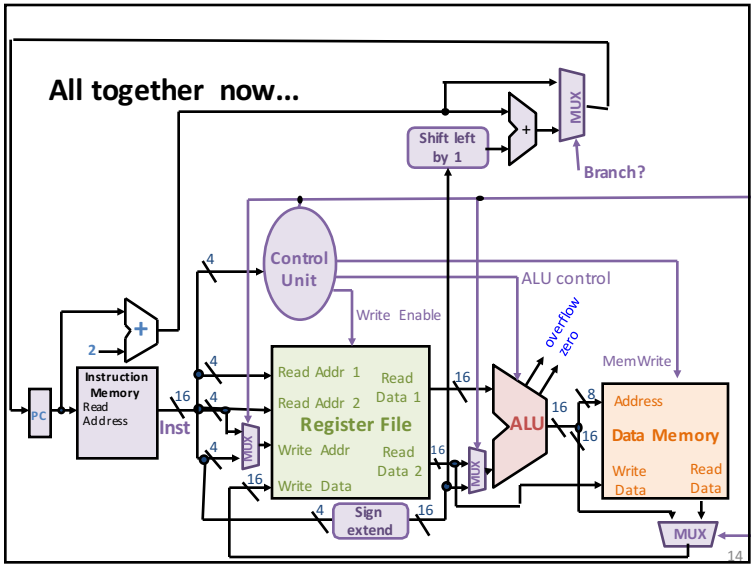
11

Compute branch target



12


Make branch decision
Shift left by 1
MUX
Branch?
Control Unit
ALU control
Write Enable
overflow
zero
Instruction Memory
Read Address
PC
Inst
Register File
Read Addr 1
Read Addr 2
Write Addr
Write Data
Read Data 1
Read Data 2
ALU
Sign extend
13


All together now…
Shift left by 1
MUX
Branch?
Control Unit
ALU control
Write Enable
overflow
zero
MemWrite
Instruction Memory
Read Address
PC
Inst
Register File
Read Addr 1
Read Addr 2
Write Addr
Write Data
Read Data 1
Read Data 2
ALU
Address
Data Memory
Write Data
Read Data
Sign extend
MUX
14