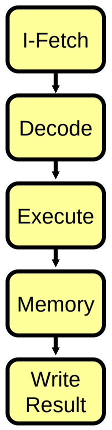
Lecture 8: Datapaths and Pipelining

- **Last Time**
  - **Last of the ISA lectures**

- **Today**
  - **Role of compiler**
  - **Datapath organization**
  - **Datapath control**

Instruction Execution

- **5 basic steps**
  - **fetch instruction (F)**
  - **decode instruction and read registers (R)**
  - **execute (X)**
  - **access memory (M)**
  - **store result (W)**

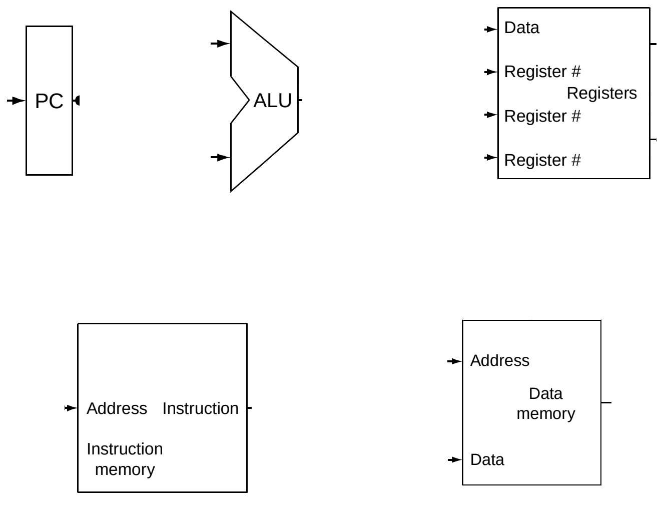


The Processor: Datapath & Control

- We're ready to look at an implementation of the MIPS
- Simplified to contain only:
  - memory-reference instructions: `lw, sw`
  - arithmetic-logical instructions: `add, sub, and, or, slt`
  - control flow instructions: `beq, j`
- Generic Implementation:
  - use the program counter (PC) to supply instruction address
  - get the instruction from memory
  - read registers
  - use the instruction to decide exactly what to do
- All instructions use the ALU after reading the registers

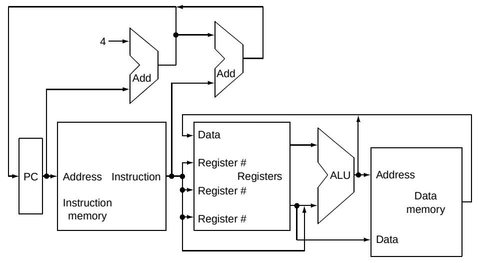
  Why? memory-reference? arithmetic? control flow?

Pieces we'll need



More Implementation Details

- **Abstract / Simplified View:**

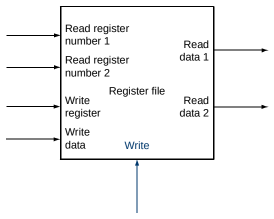


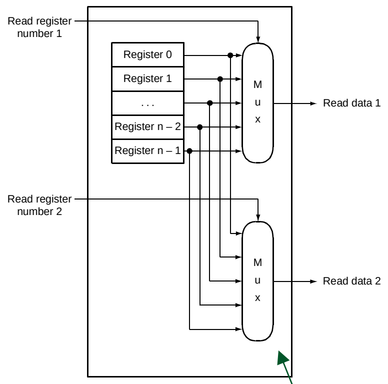
Two types of functional units:

- **elements that operate on data values (combinational)**
- **elements that contain state (sequential)**

Register File

- **Built using D flip-flops**

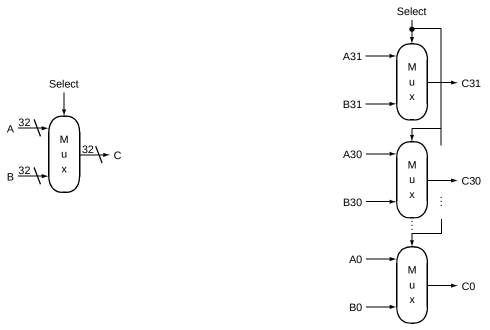




Do you understand? What is the “Mux” above?

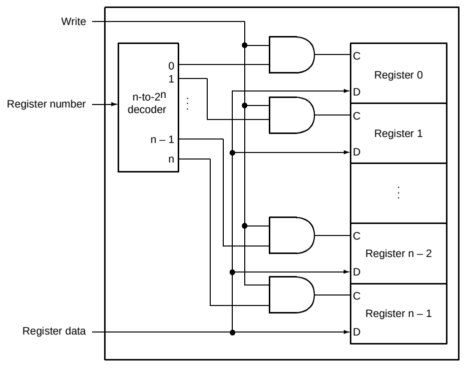
Abstraction

- Make sure you understand the abstractions!
- Sometimes it is easy to think you do, when you don't



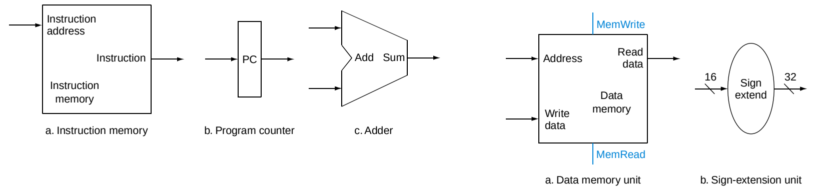
Register File

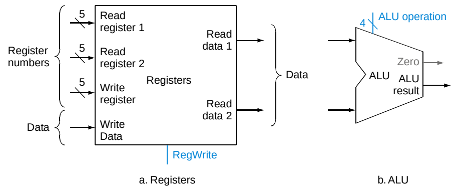
- Note: we still use the real clock to determine when to write



Simple Implementation

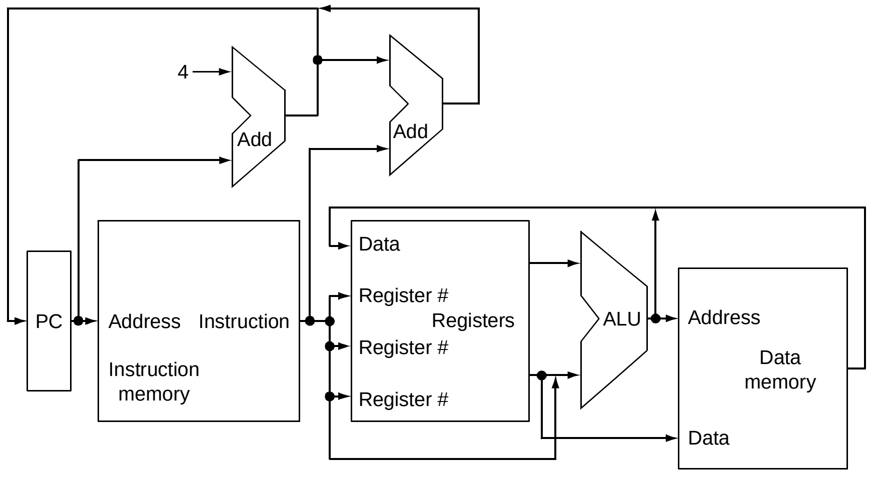
- **Include the functional units we need for each instruction**



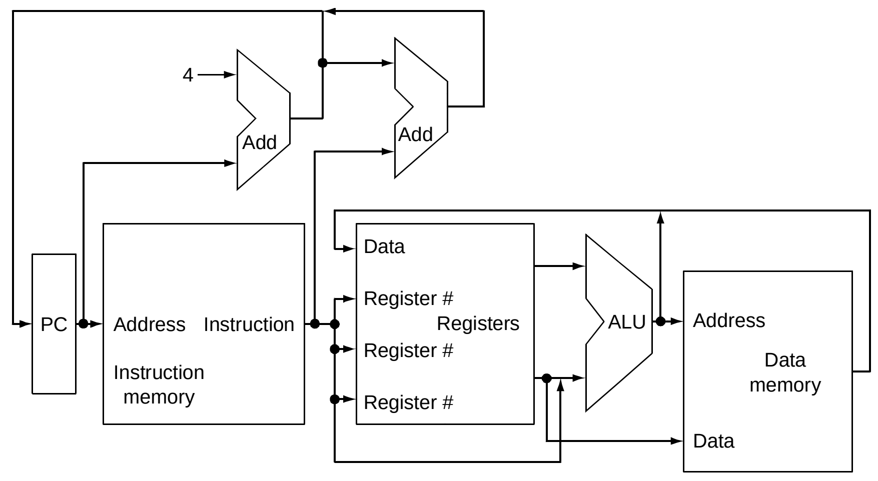


Why do we need this stuff?

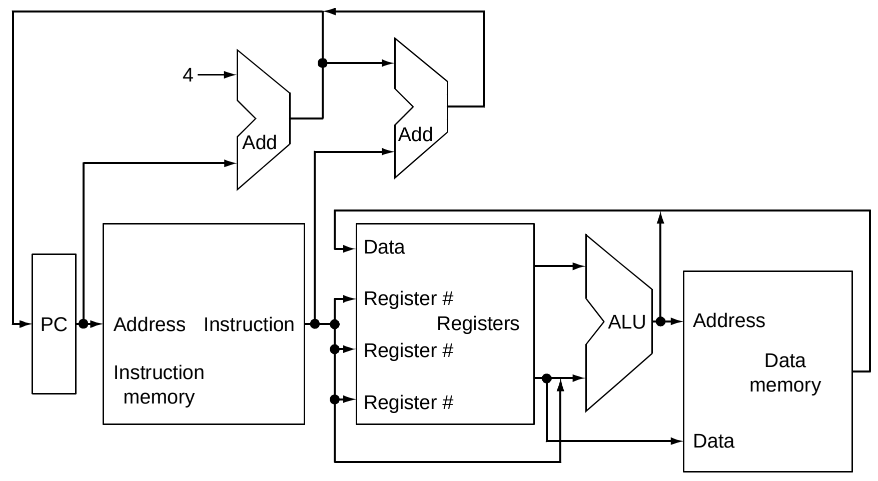
ADD Instruction



LW/SW Instruction

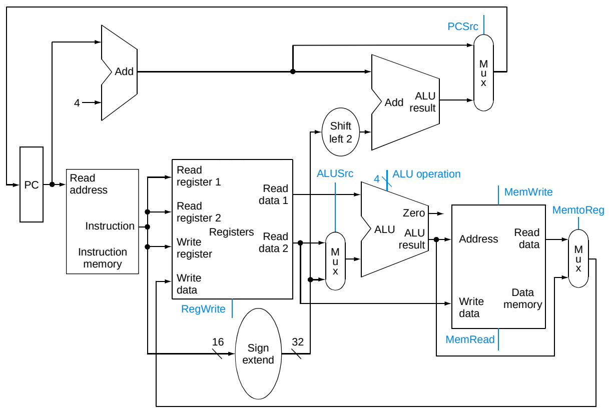


BEQ Instruction



Building the Datapath

- Use multiplexors to stitch them together



Control

- Selecting the operations to perform (ALU, read/write, etc.)
- Controlling the flow of data (multiplexor inputs)
- Information comes from the 32 bits of the instruction
- Example:

add $8, $17, $18 **Instruction Format:**

000000	10001	10010	01000	00000	100000
op	rs	rt	rd	shamt	funct

- ALU's operation based on instruction type and function code

Control

- e.g., what should the ALU do with this instruction
- Example: lw $1, 100($2)

35	2	1	100
op	rs	rt	16 bit offset

- ALU control input

```
0000   AND
0001   OR
0010   add
0110   subtract
0111   set-on-less-than
1100   NOR
```

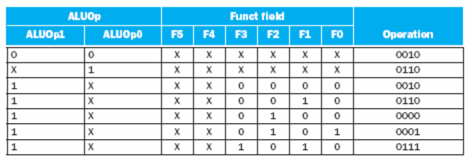
- Why is the code for subtract 0110 and not 0011?

Control

- Must describe hardware to compute 4-bit ALU control input
  - given instruction type
    - 00 = lw, sw
    - 01 = beq,
    - 10 = arithmetic

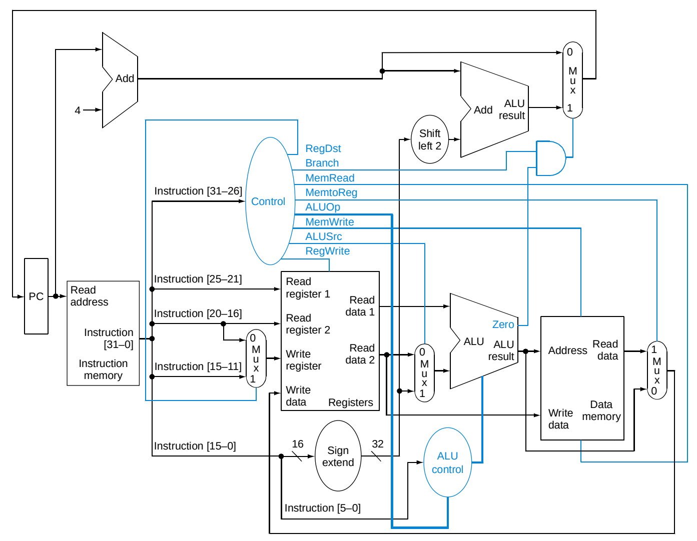
    ALUOp
    computed from instruction type
  - function code for arithmetic

- Describe it using a truth table (can turn into gates):



ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

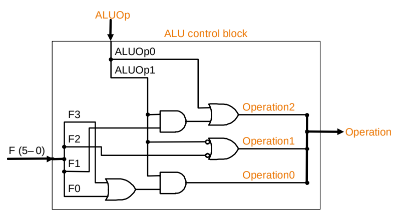
FIGURE 5.13 The truth table for the three ALU control bits (called Operation). The inputs are the ALUOp and function code field. Only the entries for which the ALU control is asserted are shown. Some don't-care entries have been added. For example, the ALUOp does not use the encoding 11, so the truth table can contain entries 1X and X1, rather than 10 and 01. Also, when the function field is used, the first two bits (F5 and F4) of these instructions are always 10, so they are don't-care terms and are replaced with XX in the truth table.

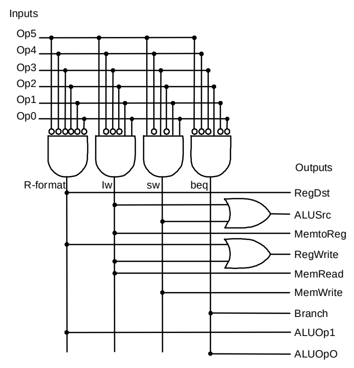


Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

Control

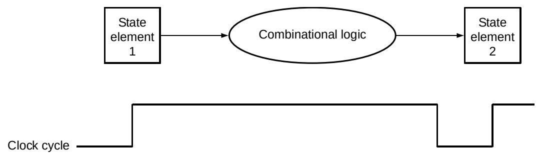
- **Simple combinational logic (truth tables)**





Our Simple Control Structure

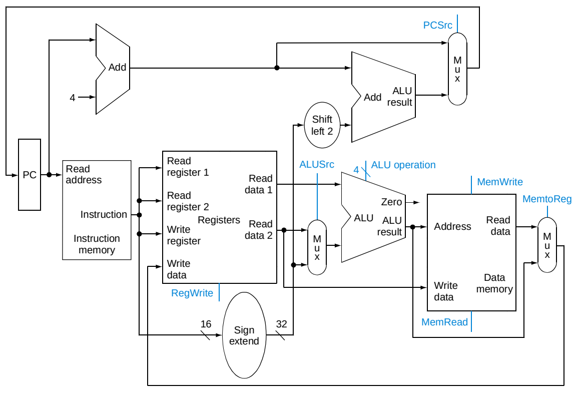
- All of the logic is combinational
- We wait for everything to settle down, and the right thing to be done
  - ALU might not produce “right answer” right away
  - we use write signals along with clock to determine when to write
- Cycle time determined by length of the longest path



We are ignoring some details like setup and hold times

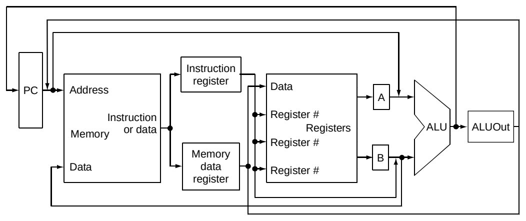
Single Cycle Implementation

- Calculate cycle time assuming negligible delays except:
  - memory (200ps),
    ALU and adders (100ps),
    register file access (50ps)



Where we are headed

- Single Cycle Problems:
  - what if we had a more complicated instruction like floating point?
  - wasteful of area
- One Solution:
  - use a "smaller" cycle time
  - have different instructions take different numbers of cycles
  - a "multicycle" datapath:



Summary

- Datapath organization
- Datapath control

- Next Time
  - Pipelining

- Reading assignment – P&H 5.5, 5.6, 5.9-5.11