Intro to Sequential Logic

CS 64: Computer Organization and Design Logic
Lecture #14
Winter 2019

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

Administrative

- Lab #7
  - On Thursday
  - It's a little on the tough side, so partner up if you want to
  - Due next week on Wednesday
  - Paper copy

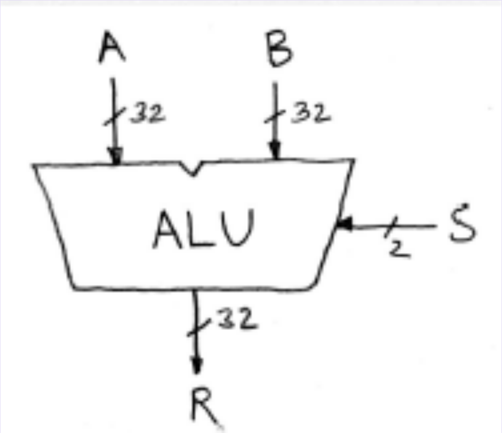
Lecture Outline

- General ALU Design
- Sequential Logic
- S-R Latch
- D-Latch
- Reviewing what's needed for Lab 7

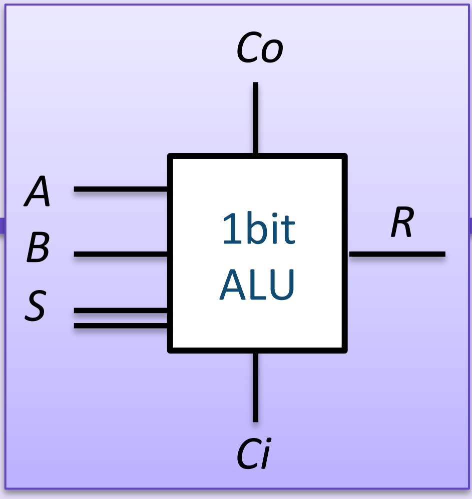
Arithmetic-Logic Unit (ALU)

- Recall: the ALU does all the computations necessary in a CPU

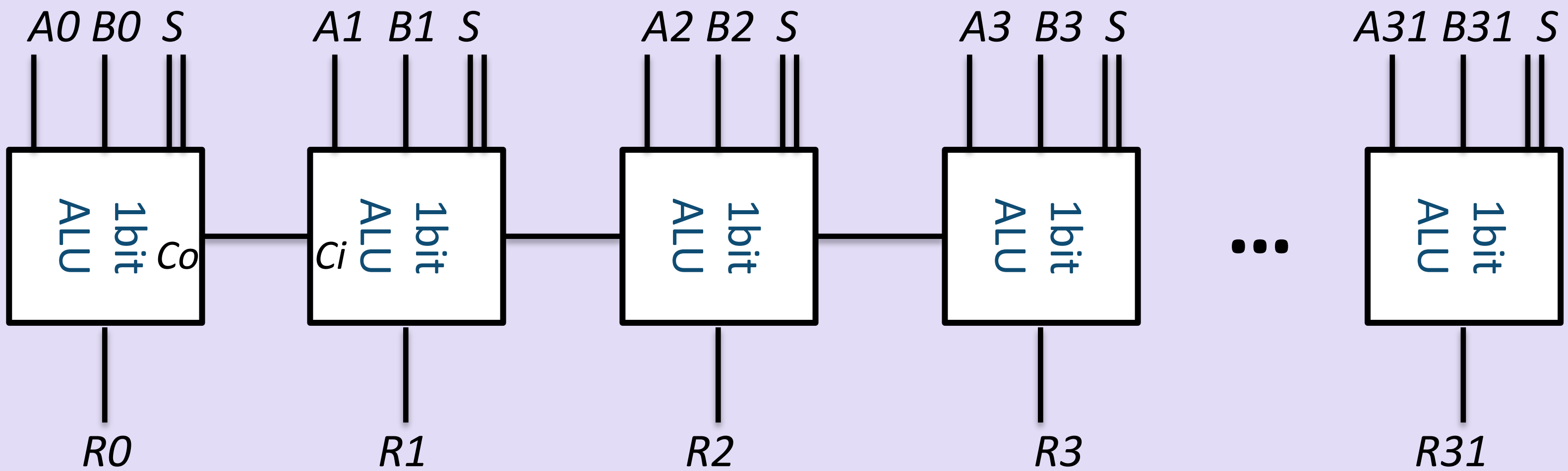
- The previous circuit was a simplified ALU:
  - When S = 00, R = A + B
  - When S = 01, R = A – B
  - When S = 10, R = A AND B
  - When S = 11, R = A OR B



Simplified ALU

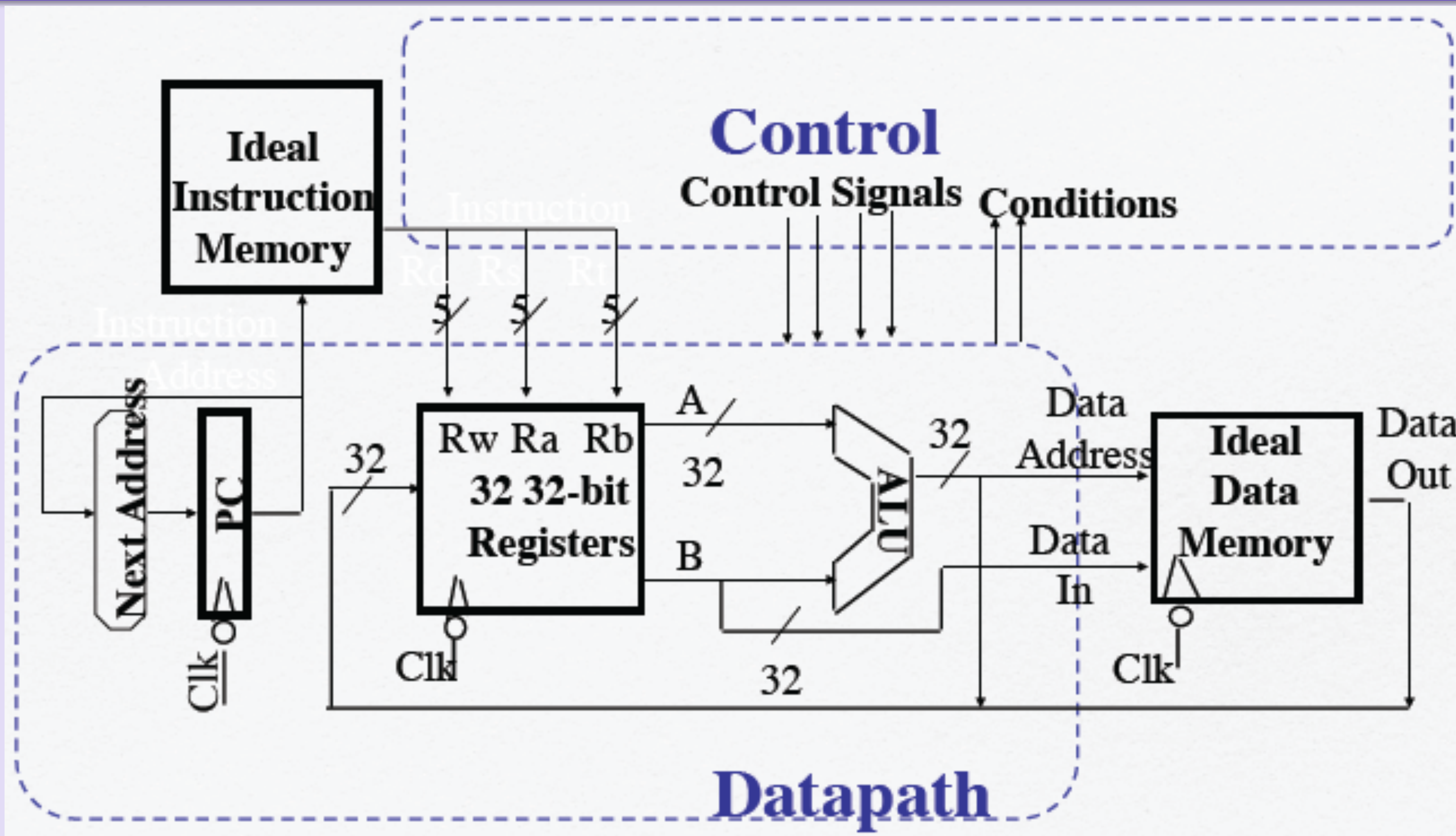


- We can string 1-bit ALUs together to make bigger-bit ALUs (e.g. 32b ALU)



Abstract Schematic of the MIPS CPU

Relevant to a future lab…



Combinatorial vs. Sequential Logic

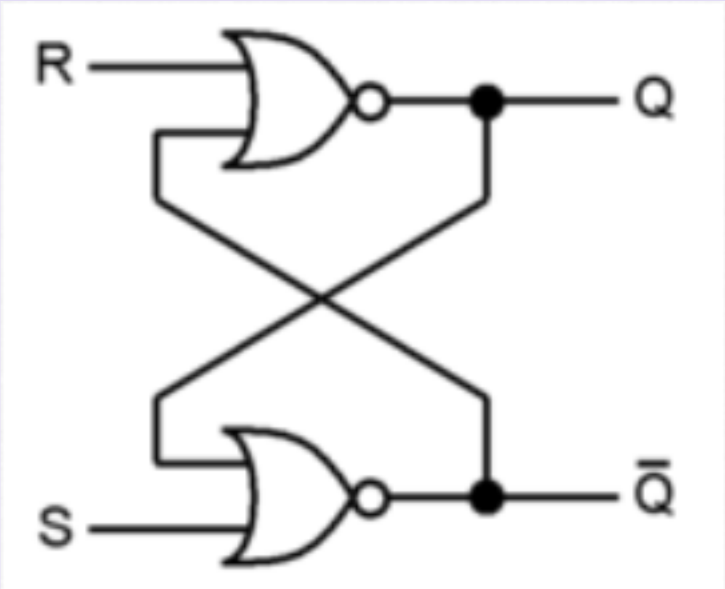
- The CPU schematic shows
  both **combinatorial** *and* **sequential** logic blocks

- **Combinatorial Logic**
  – Combining multiple logic blocks
  – The output is a function **only** of the present inputs
  – There is no memory of past "states"

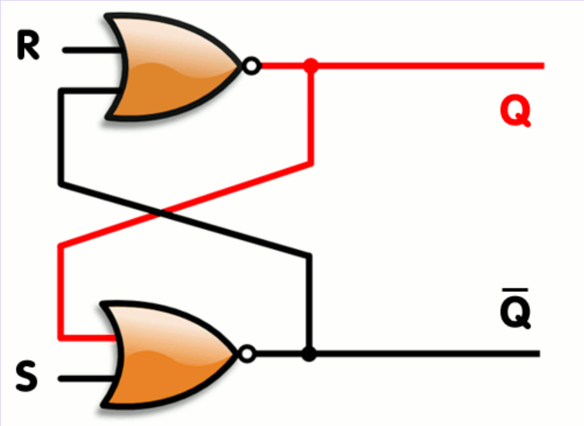
- **Sequential Logic**
  – Combining multiple logic blocks
  – The output is a function of ***both*** the present inputs and ***past*** inputs
  – There exists a memory of past "states"

The S-R Latch

- Only involves 2 NORs
- The outputs are fed-back to the inputs
- The result is that the output state (either a 1 or a 0) is maintained even if the input changes!



How a S-R Latch Works



S	R	Q_0	Comment
0	0	Q*	Hold output
0	1	0	Reset output
1	0	1	Set output
1	1	X	Undetermined

- Note that if one NOR input is **0**, the output becomes the inverse of the other input
- So, if output Q already exists and if S = 0, R = 0, then Q will remain at whatever it was before! (**hold output state**)
- If S = 0, R = 1, then Q becomes 0 (**reset output**)
- If S = 1, R = 0, then Q becomes 1 (**set output**)
- Making S = 1, R = 1 is *not allowed* (**gives an undetermined output**)

Consequences?

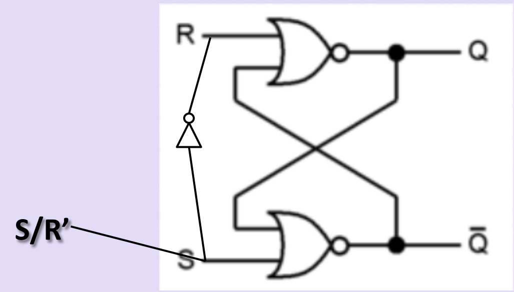
S	R	Q_0	Comment
0	0	Q*	Hold output
0	1	0	Reset output
1	0	1	Set output
1	1	X	Undetermined

- As long as **S = 0 and R = 0**, the circuit output holds memory of its prior value (state)
- To change the output, just make S = 1 (but also R = 0) to make the output 1 (set) ***OR*** S = 0 (but also R = 1) to make the output 0 (reset)
- Just avoid S = 1, R = 1…

About that S = 1, R = 1 Case…

S	R	Q_0	Comment
0	0	Q*	Hold output
0	1	0	Reset output
1	0	1	Set output
1	1	X	Undetermined

- What if we avoided it on purpose by making R = NOT (S)?
  - Where's the problem?

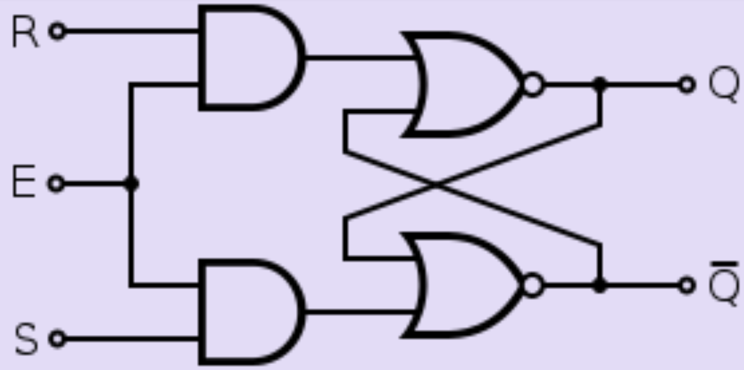


- This, by itself, precludes a case when R = S = 0
  - You'd need that if you want to preserve the previous output state!

- Solution: the ***clocked latch*** and ***the flip-flop***

Adding an "Enable" Input: **The Gated S-R Latch**

- Create a way to "gate" the inputs
  - R/S inputs go through *only if* an **"enable input" (E) is 1**
  - If E is 0, then the S-R latch gets SR = 00 and it hold the state of previous outputs



- So, the truth table would change from a "normal" S-R Latch:

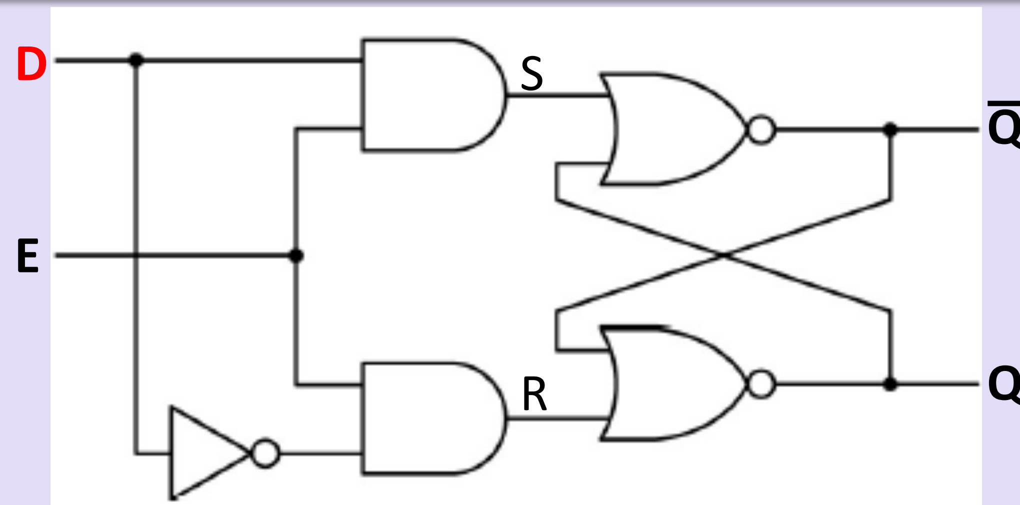
S	R	Q_0	Comment
0	0	Q*	Hold output
0	1	0	Reset output
1	0	1	Set output
1	1	X	Undetermined

S	R	E	Q_0	Comment
X	X	0	Q*	Hold output
0	1	1	0	Reset output
1	0	1	1	Set output

We got rid of the "undetermined" state!!! ☺☺☺

Combining R and S inputs into One: **The Gated D Latch**

- Force S and R inputs to *always be opposite of each other*
  - Make them the same as an input D, where **D = S** and **!D = R**.



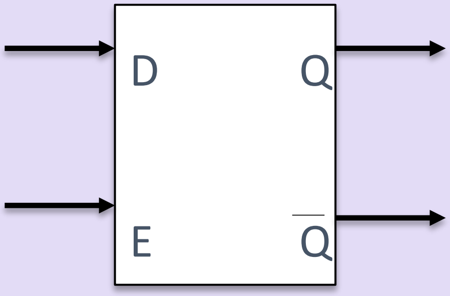
- Create a way to "gate" the D input
  - D input goes through only if an enable input (E) is 1
  - If E is 0, then hold the state of the previous outputs

D	E	Q_0	Comment
X	0	Q*	Hold output
0	1	0	Reset output
1	1	1	Set output

We got rid of an extra input!!! ☺☺☺

The Gated D Latch

- The gated D-Latch is very commonly used in electronic circuits in computer hardware, especially as a register because it's a circuit that holds memory!



Whatever data you present to the input D,

the D-Latch will **hold** that value (*as long as input E is 0*)

You can **present** this value to output Q *as soon as input E is 1*.

There is no synchronicity to this circuit (*meaning what???*)

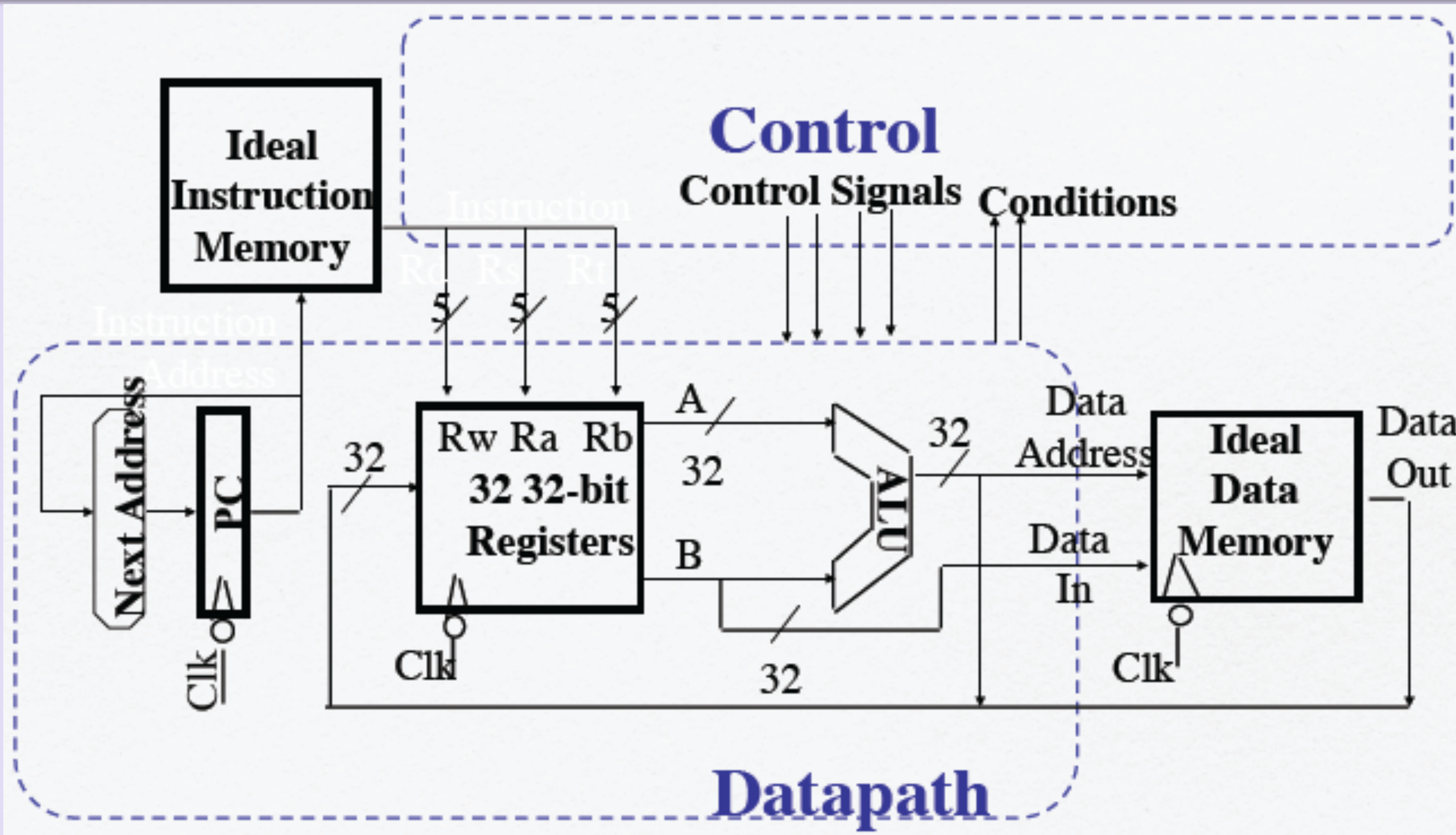
Lab 7

What's Lab 7 About?

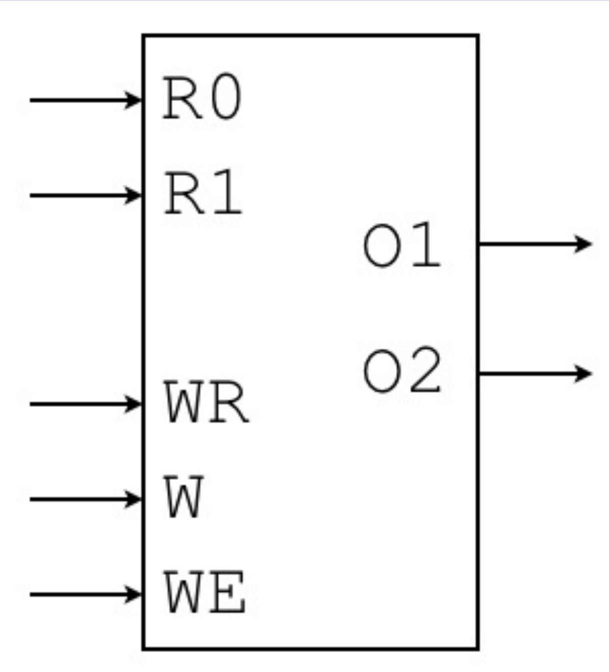
- Task 1: Design a "simple" ALU
- Task 2: Design a "simple" Register Block using D-Latches
- Task 3: You are given a specification for a "simple" CPU that uses:
  - 1 "Simple" Register Block
  - 1 "Simple" ALU
  - 1 Abstract Computer Memory Interface
  - As many ANDs, ORs, NOTs, XORs, Muxes that you need
- Design this CPU! (Task 3)

- You will draw all of these (BE NEAT!)
  - Turn it in (physical copy) at the CS64 BOX in HFH on 2nd floor

Abstract Schematic of the MIPS CPU



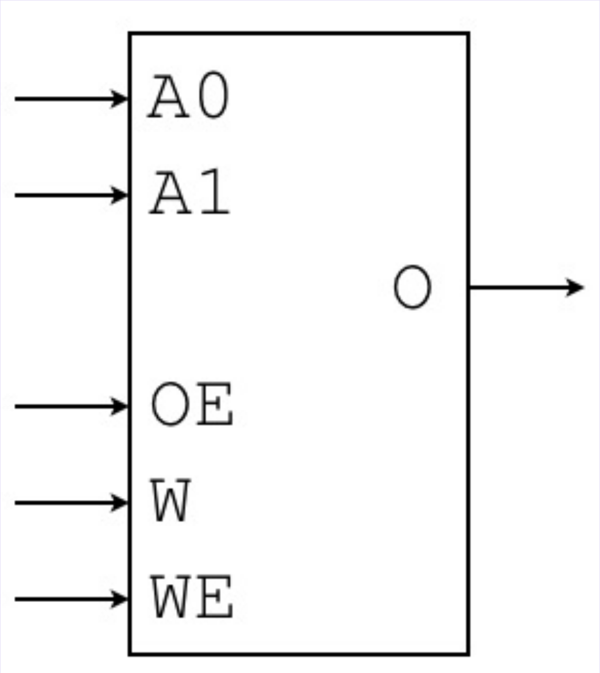
Register Object for Lab 7 (Task 2)



This will be made from D-FFs or D-Latches and Combinatorial Logic

I/O Name	I/O Description
R0	The first register to read, as a single bit. If 0, then reg0 should be read. If 1, then reg1 should be read.
R1	The second register to read, as a single bit. If 0, then reg0 should be read. If 1, then reg1 should be read.
WR	"Write Register". Specifies which register to write to. If 0, then reg0 should be written to. If 1, then reg1 should be written to.
W	The data that should be written to the register specified by WR. This is a single bit.
WE	"Write Enable". If 1, then we will write to a register. If 0, then we will **not** write to a register. Note that if WE = 0, then the inputs to WR and W are effectively ignored.
O1	Value of the first register read. As described previously, this depends on which register was selected to be read, via R0.
O2	Value of the second register read. As described previously, this depends on which register was selected to be read, via R1.

Memory Interface Object for Lab 7



I/O Name	I/O Description
A0	Bit 0 of the address (LSB)
A1	Bit 1 of the address (MSB)
OE	"Output Enable". If 1, then the value at the address specified by A0 and A1 will be read, and sent to the output line O. If 0, then the memory will not be accessed, and the value sent to the output line is unspecified (could be either 0 or 1, in an unpredictable fashion).
W	The value to write to memory.
WE	"Write Enable". If 1, then the value sent into W will be written to memory at the address specified by A0 and A1. If 0, then no memory write occurs (the value sent to W will be ignored).
O	The value read from memory (or unspecified if OE = 0).

Task 3: Build a Mock-CPU!

Actually, just a small instruction decoder and executor…

OP1	OP0	B0	B1	B2	Human-readable Encoding	Description
0	0	0	0	0	xor reg0, reg0, reg0	Compute the XOR of the contents of reg0 with the contents of reg0, storing the result in reg0.
0	1	1	0	1	nor reg1, reg0, reg1	Compute the NOR of the contents of reg0 with the contents of reg1, storing the result in reg1.
1	0	1	0	1	load reg1, 01	Copy the bit stored at address 01 (decimal 1) into register reg1.
1	1	0	1	0	store reg0, 10	Store the contents of reg0 at address 10 (decimal 2)
1	1	1	1	1	store reg1, 11	Store the contents of reg1 at address 11 (decimal 3)

These say something about which **registers** are used

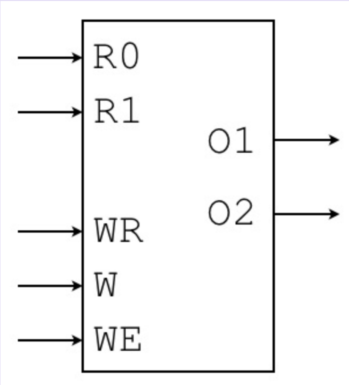
These say something about which **operation** is being done

Hints for Task 3

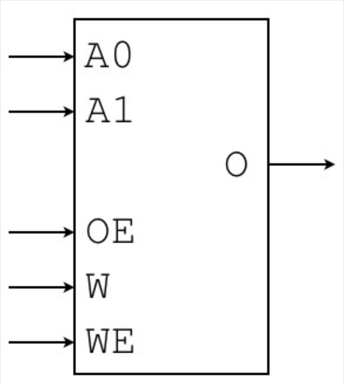
- Design the final circuit in pieces:
  - One piece for each of the 3 types of instruction: load, store, XOR/NOR
- For example, the store task:
  - If an output isn't used, tie it to a permanent "0" (i.e. ground)
  - If an input isn't used, then you can use "X" (don't care) on it

What registers am I using? Which instruction bits go where?

What controls should I use? What values should they be?



How do I best connect the registers to the memory interface? Again, ask yourself if some of the inputs here should be op-code bits.



Is the output even used in this "store" task?

Tying In All The Pieces (Task 3)

- Now see how they can all fit together
  - **You will have 1 register block + 1 memory interface**
  - You *won't* need to use any additional latches here
  - You *will* need to use muxes and regular logic (and the simple ALU you designed earlier – see lab instructions for more details)

- REQUIREMENT:

Use ONLY 1 register block, 1 ALU, and 1 memory interface

YOUR TO-DOs

- Lab 7
  - Start on Thursday
  - Due back on Wednesday
  - Paper copy – not electronic
  - Drop off in the CS64 BOX in HFH 2nd Floor

</LECTURE>