inst.eecs.berkeley.edu/~cs61c
CS61C : Machine Structures

Lecture 17
Combinational Logic Blocks and Intro to CPU Design

2010-07-20

Instructor Paul Pearce

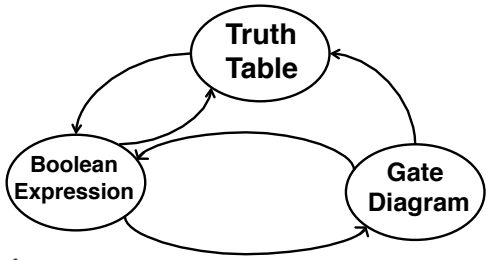
Passwords that are simple, AND safe⇒

Researchers at Harvard and Microsoft have developed a new scheme for password requirements that simply mandates that no single password may be "too popular." Their findings will be presented at Hot Topics in Security next month.

http://www.technologyreview.com/computing/25826/

Review

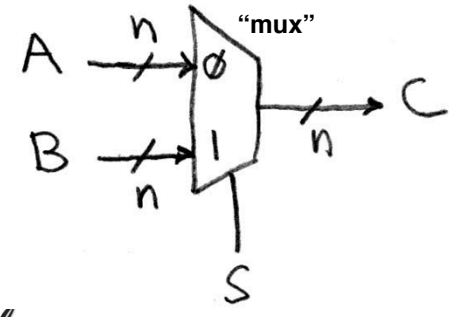
- **Use this table and techniques we learned to transform from 1 to another**



Today

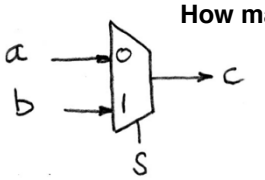
- **Data Multiplexors**
- **Arithmetic and Logic Unit**
- **Adder/Subtractor**
- **Datapath design**

Data Multiplexor (here 2-to-1, n-bit-wide)



N instances of 1-bit-wide mux

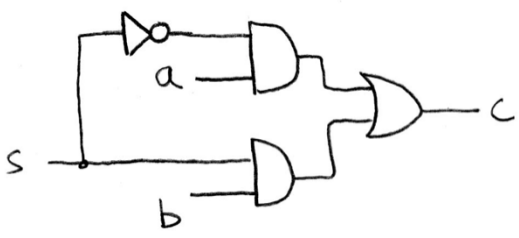
How many rows in TT?



$$
\begin{aligned}
c &= \overline{s}a\overline{b} + \overline{s}ab + s\overline{a}b + sab \\
&= \overline{s}(a\overline{b} + ab) + s(\overline{a}b + ab) \\
&= \overline{s}(a(\overline{b} + b)) + s((\overline{a} + a)b) \\
&= \overline{s}(a(1) + s((1)b) \\
&= \overline{s}a + sb
\end{aligned}
$$

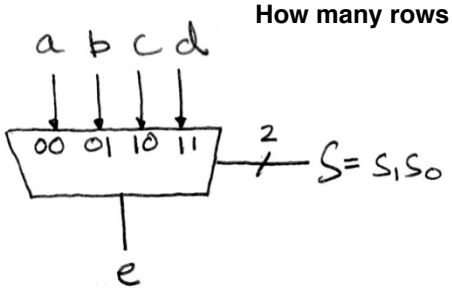
How do we build a 1-bit-wide mux?

$$\overline{s}a + sb$$



4-to-1 Multiplexor?

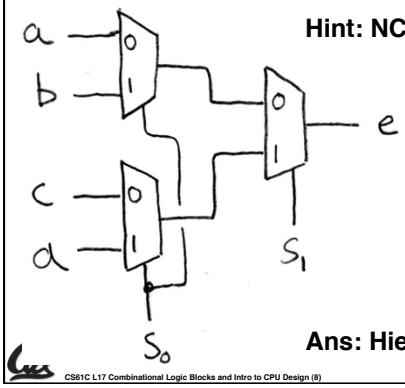
How many rows in TT?



$$e = \overline{s_1}\overline{s_0}a + \overline{s_1}s_0b + s_1\overline{s_0}c + s_1s_0d$$

Is there any other way to do it?

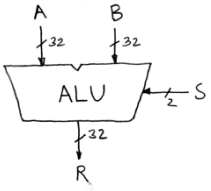
Hint: NCAA tourney!



Ans: Hierarchically!

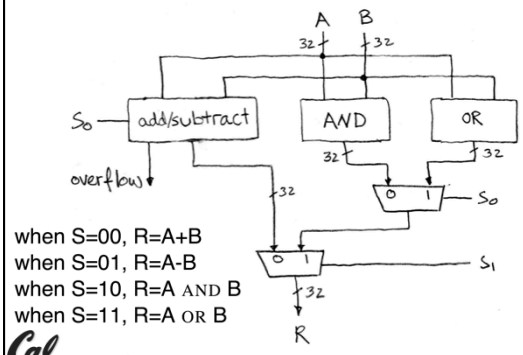
Arithmetic and Logic Unit

- Most processors contain a special logic block called "Arithmetic and Logic Unit" (ALU)
- We'll show you an easy one that does ADD, SUB, bitwise AND, bitwise OR



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

Our simple 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

Adder/Subtracter Design -- how?

- Truth-table, then determine canonical form, then minimize and implement as we've seen before
- Look at breaking the problem down into smaller pieces that we can cascade or hierarchically layer

Administrivia

- If you want a regrade for your midterm, staple a paper with your explanation (clearly indicating on what question you want more points) to your exam and turn it in to your TA in section Monday
  - We'll regrade the exam and your score MIGHT go down…
  - We will regrade with respect to the established rubric, so that all tests are graded equally
- Project 2 will be done with 1 partner (groups of 2). Start thinking about who you want to work with.
  - Hint: Your best friend may not be the best choice for a partner. Projects ruin friendships.
- HW6 due tomorrow at midnight.

Adder/Subtracter – One-bit adder LSB…

	a_3	a_2	a_1	a_0
+	b_3	b_2	b_1	b_0
	s_3	s_2	s_1	s_0

a_0	b_0	s_0	c_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$s_0 =$ ☐
$c_1 =$

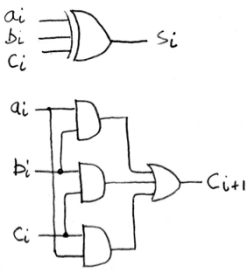
Adder/Subtracter – One-bit adder (1/2)…

	a_3	a_2	a_1	a_0
+	b_3	b_2	b_1	b_0
	s_3	s_2	s_1	s_0

a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

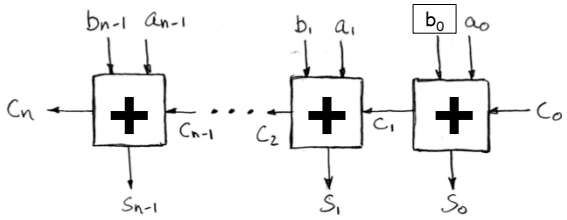
$s_i =$ ☐
$c_{i+1} =$

Adder/Subtracter – One-bit adder (2/2)…



$$s_i = \text{XOR}(a_i, b_i, c_i)$$
$$c_{i+1} = \text{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i$$

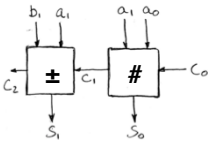
N 1-bit adders ⇒ 1 N-bit adder



What about overflow?
Overflow = c_n?

What about overflow?

- **Consider a 2-bit signed # & overflow:**
  - `10 = -2 + -2 or -1`
  - `11 = -1 + -2 only`
  - `00 = 0 NOTHING!`
  - `01 = 1 + 1 only`
- **Highest adder**
  - **C_1 = Carry-in = C_{in}, C_2 = Carry-out = C_{out}**
  - **No C_{out} or C_{in} ⇒ NO overflow!**
  - **C_{in}, and C_{out} ⇒ NO overflow!**
  - **C_{in}, but no C_{out} ⇒ A,B both > 0, overflow!**
  - **C_{out}, but no C_{in} ⇒ A,B both < 0, overflow!**

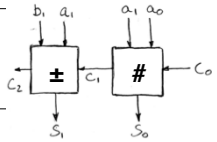


What op?

What about overflow?

- **Consider a 2-bit signed # & overflow:**

```
10 = -2
11 = -1
00 =  0
01 =  1
```

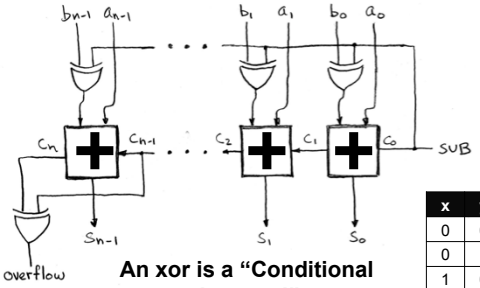


- **Overflows when…**
  - **C_{in}, but no C_{out} ⇒ A,B both > 0, overflow!**
  - **C_{out}, but no C_{in} ⇒ A,B both < 0, overflow!**

$$\text{overflow} = c_n \text{ XOR } c_{n-1}$$

Extremely Clever Subtractor

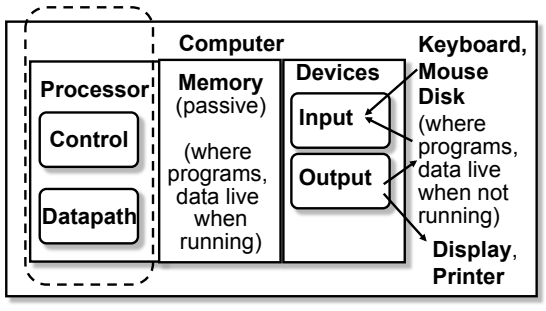
A - B = A + (-B); how do we make "-B"?



x	y	xor
0	0	0
0	1	1
1	0	1
1	1	0

An xor is a "Conditional Inverter!"

Five Components of a Computer



The CPU

- **Processor (CPU): the active part of the computer, which does all the work (data manipulation and decision-making)**
- **Datapath: portion of the processor which contains hardware necessary to perform operations required by the processor (the brawn)**
- **Control: portion of the processor (also in hardware) which tells the datapath what needs to be done (the brain)**

Stages of the Datapath : Overview

- **Problem: a single, atomic block which "executes an instruction" (performs all necessary operations beginning with fetching the instruction) would be too bulky and inefficient**
- **Solution: break up the process of "executing an instruction" into stages, and then connect the stages to create the whole datapath**
  - **smaller stages are easier to design**
  - **easy to optimize (change) one stage without touching the others**

Stages of the Datapath (1/5)

- **There is a wide variety of MIPS instructions: so what general steps do they have in common?**
- **Stage 1: Instruction Fetch**
  - **no matter what the instruction, the 32-bit instruction word must first be fetched from memory (the cache-memory hierarchy)**
  - **also, this is where we Increment PC (that is, PC = PC + 4, to point to the next instruction: byte addressing so + 4)**

Stages of the Datapath (2/5)

- **Stage 2: Instruction Decode**
  - **upon fetching the instruction, we next gather data from the fields (decode all necessary instruction data)**
  - **first, read the `opcode` to determine instruction type and field lengths**
  - **second, read in data from all necessary registers**
    - **for `add`, read two registers**
    - **for `addi`, read one register**
    - **for `jal`, no reads necessary**

Stages of the Datapath (3/5)

- **Stage 3: ALU (Arithmetic-Logic Unit)**
  - the real work of most instructions is done here: arithmetic (+, -, *, /), shifting, logic (&, |), comparisons (`slt`)
  - what about loads and stores?
    - `lw $t0, 40($t1)`
    - the address we are accessing in memory = the value in `$t1` PLUS the value 40
    - so we do this addition in this stage

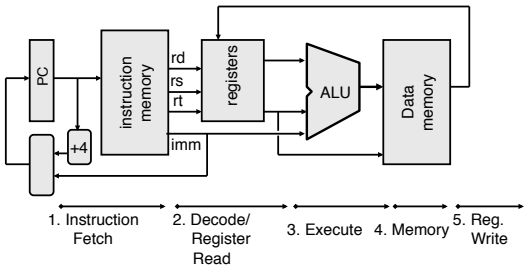
Stages of the Datapath (4/5)

- **Stage 4: Memory Access**
  - actually only the load and store instructions do anything during this stage; the others remain idle during this stage or skip it all together
  - since these instructions have a unique step, we need this extra stage to account for them
  - as a result of the cache system, this stage is expected to be fast

Stages of the Datapath (5/5)

- **Stage 5: Register Write**
  - most instructions write the result of some computation into a register
  - examples: arithmetic, logical, shifts, loads, slt
  - what about stores, branches, jumps?
    - don't write anything into a register at the end
    - these remain idle during this fifth stage or skip it all together

Generic Steps of Datapath



Peer Instruction

1) Truth table for mux with 4-bits of select is 2^4 rows long

2) We could cascade N 1-bit shifters to make 1 N-bit shifter for sll, srl

	12
a)	FF
b)	FT
c)	TF
d)	TT

"And In conclusion…"

- Use muxes to select among input
  - S input bits selects 2^S inputs
  - Each input can be n-bits wide, independent of S
- Can implement muxes hierarchically
- ALU can be implemented using a mux
  - Coupled with basic block elements
- N-bit adder-subtractor done using N 1-bit adders with XOR gates on input
  - XOR serves as conditional inverter
- CPU design involves Datapath,Control
  - Datapath in MIPS involves 5 CPU stages
    1. Instruction Fetch
    2. Instruction Decode & Register Read
    3. ALU (Execute)
    4. Memory
    5. Register Write