Systems I

Logic Design I

Topics

- Digital logic
- Logic gates
- Simple combinational logic circuits

A Simple C statement…..

C = A + B;

What pieces of hardware do you think you might need?

- **Storage - for values A, B, C**
- **Computation logic - to compute +**
- **A way to tell the computer to retrieve the values from storage, add them together, and put the result back in storage**
  - **This could be accomplished with a single command (instruction) or with multiple of them.**

Overview of Logic Design

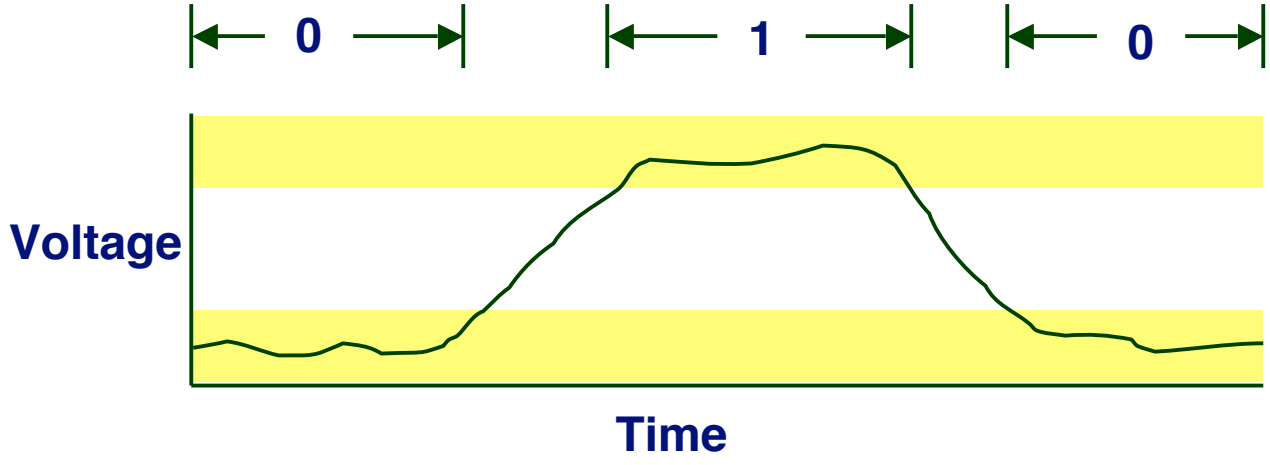
Fundamental Hardware Requirements

- Communication
  - How to get values from one place to another
- Computation
- Storage

Bits are Our Friends

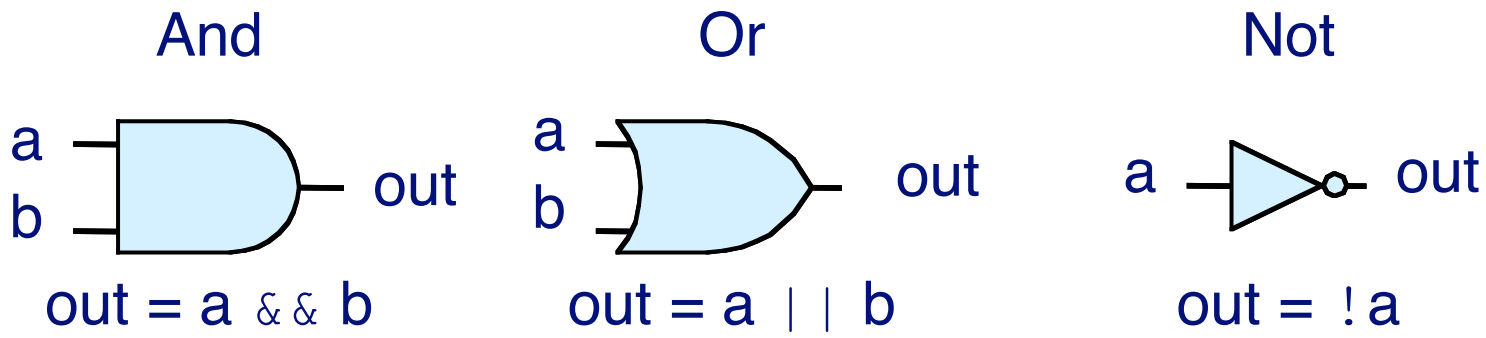
- Everything expressed in terms of values 0 and 1
- Communication
  - Low or high voltage on wire
- Computation
  - Compute Boolean functions
- Storage
  - Store bits of information

Digital Signals

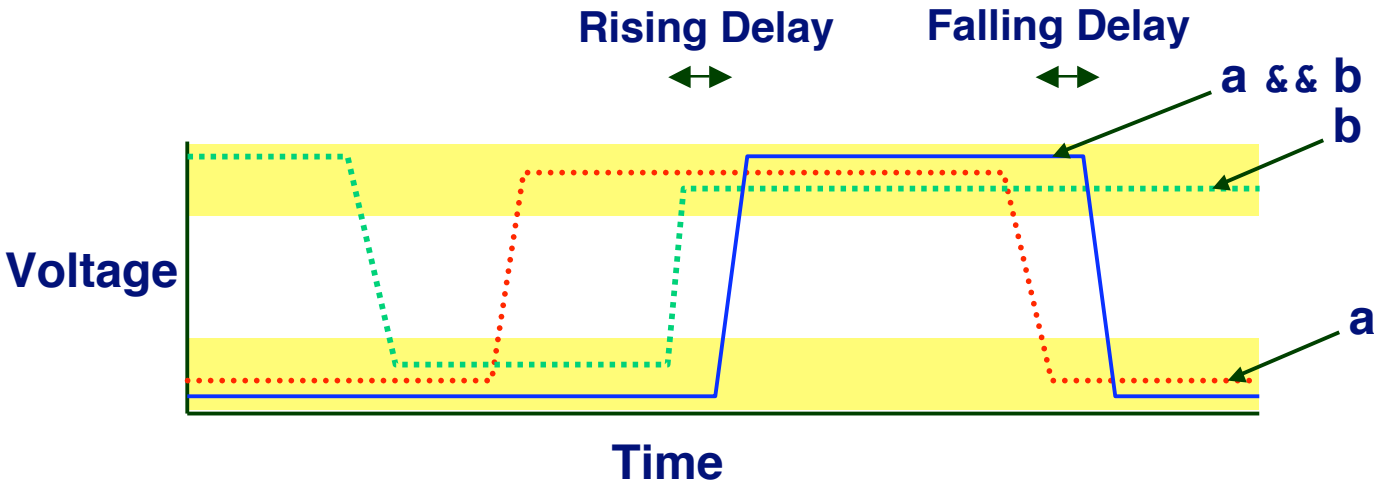


- Use voltage thresholds to extract discrete values from continuous signal
- Simplest version: 1-bit signal
  - Either high range (1) or low range (0)
  - With guard range between them
- Not strongly affected by noise or low quality circuit elements
  - Can make circuits simple, small, and fast

Computing with Logic Gates

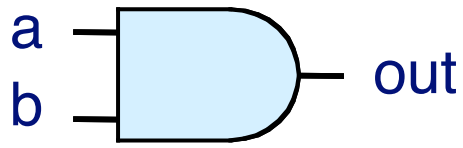


- **Logic gates constructed from transistors**
- **Outputs are Boolean functions of inputs**
- **Respond continuously to changes in inputs**
  - **With some, small delay**



Truth Tables

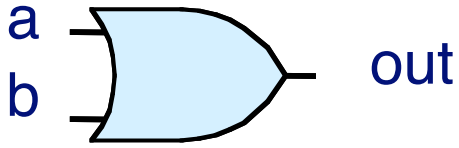
And



out = a && b

a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

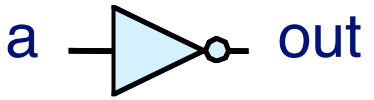
Or



out = a || b

a	b	out
0	0	0
0	1	1
1	0	1
1	1	1

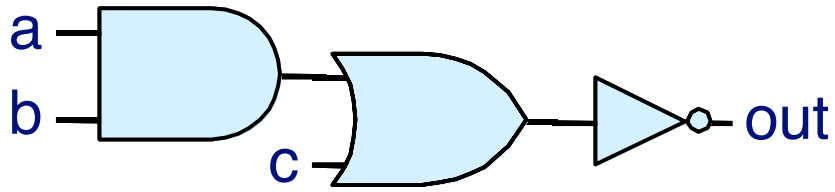
Not



out = !a

a	out
0	1
1	0

What about this?

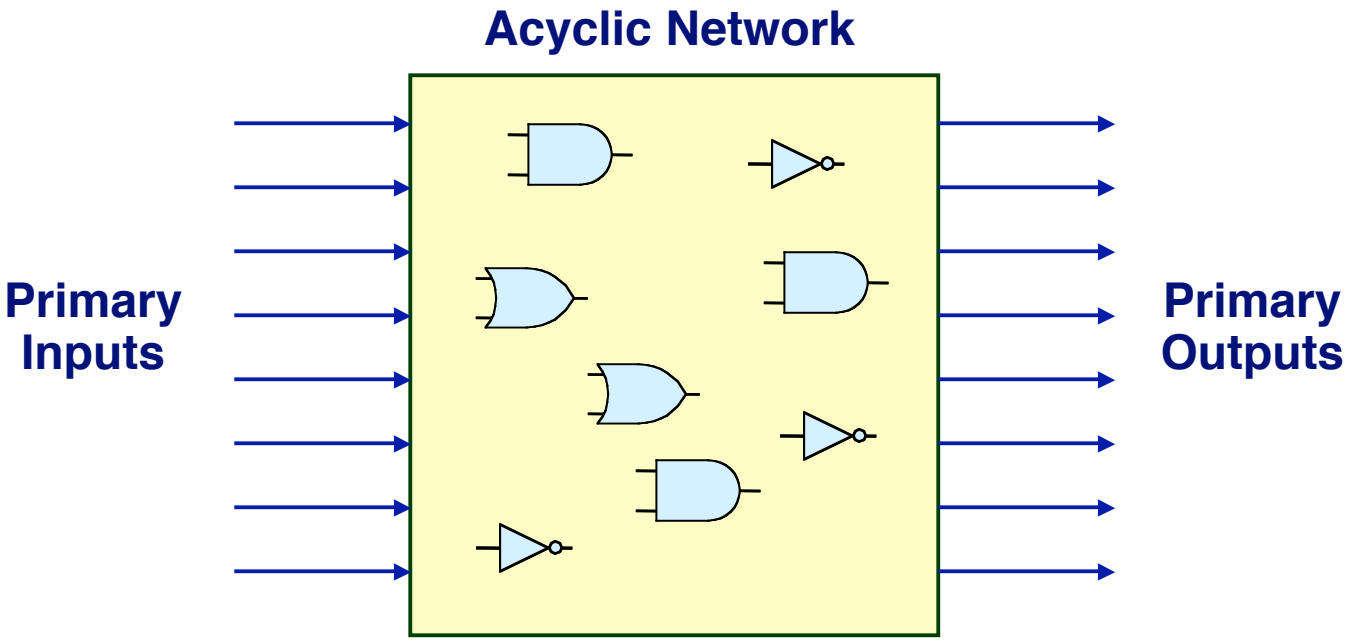


out = !((a && b) ll c)

out = ~(a*b + c)

a	b	c	out
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

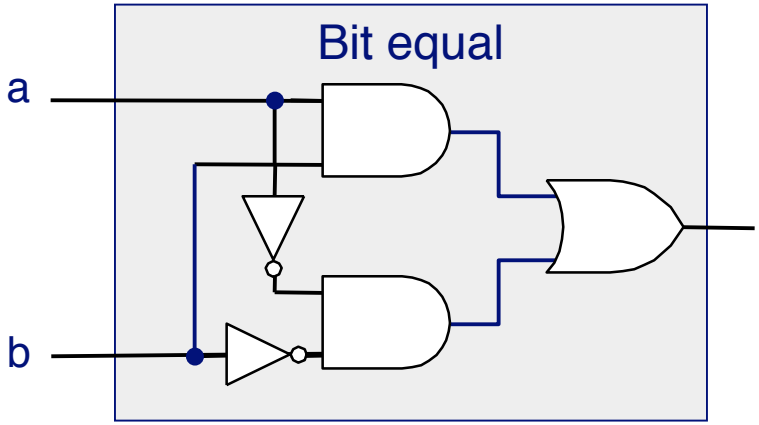
Combinational Circuits



Acyclic Network of Logic Gates

- Continously responds to changes on primary inputs
- Primary outputs become (after some delay) Boolean functions of primary inputs

Bit Equality



HCL Expression

```
bool eq = (a&&b)||(!a&&!b)
```

- Generate 1 if a and b are equal

Hardware Control Language (HCL)

- Very simple hardware description language
  - Boolean operations have syntax similar to C logical operations
- We'll use it to describe control logic for processors

Hardware Control Language

- Very simple hardware description language
- Can only express limited aspects of hardware operation
  - Parts we want to explore and modify

Data Types

- `bool`: Boolean
  - `a`, `b`, `c`, …
- `int`: words
  - `A`, `B`, `C`, …
  - Does not specify word size---bytes, 32-bit words, …

Statements

- `bool a = bool-expr ;`
- `int A = int-expr ;`

HCL Operations

- Classify by type of value returned

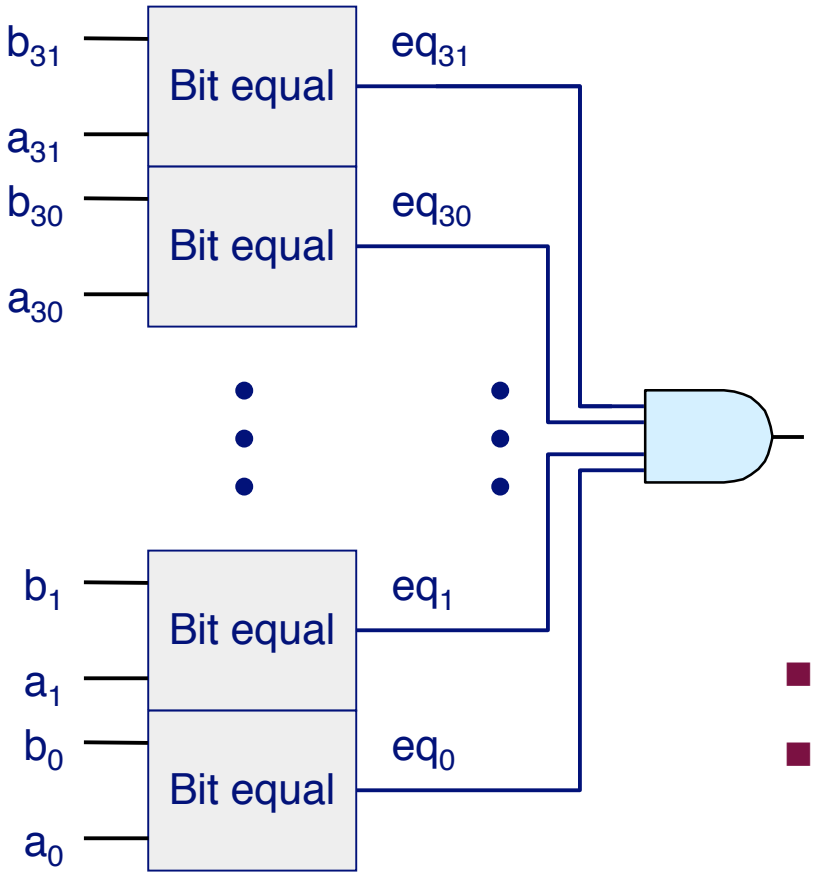
Boolean Expressions

- Logic Operations
  - `a && b, a || b, !a`
- Word Comparisons
  - `A == B, A != B, A < B, A <= B, A >= B, A > B`
- Set Membership
  - `A in { B, C, D }`
    - » Same as `A == B || A == C || A == D`

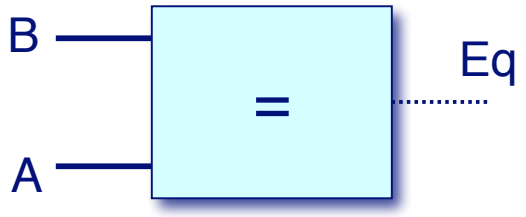
Word Expressions

- Case expressions
  - `[ a : A; b : B; c : C ]`
  - Evaluate test expressions `a`, `b`, `c`, … in sequence
  - Return word expression `A`, `B`, `C`, … for first successful test

Word Equality



Word-Level Representation

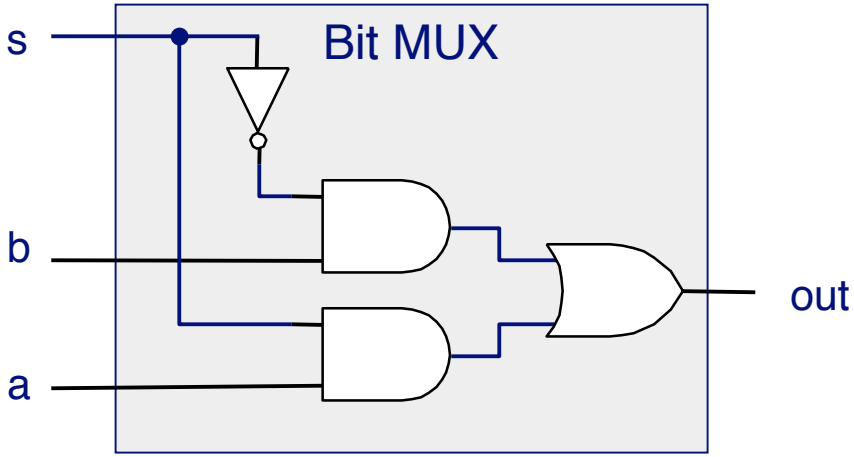


HCL Representation

```
bool Eq = (A == B)
```

- **32-bit word size**
- **HCL representation**
  - **Equality operation**
  - **Generates Boolean value**

Bit-Level Multiplexor

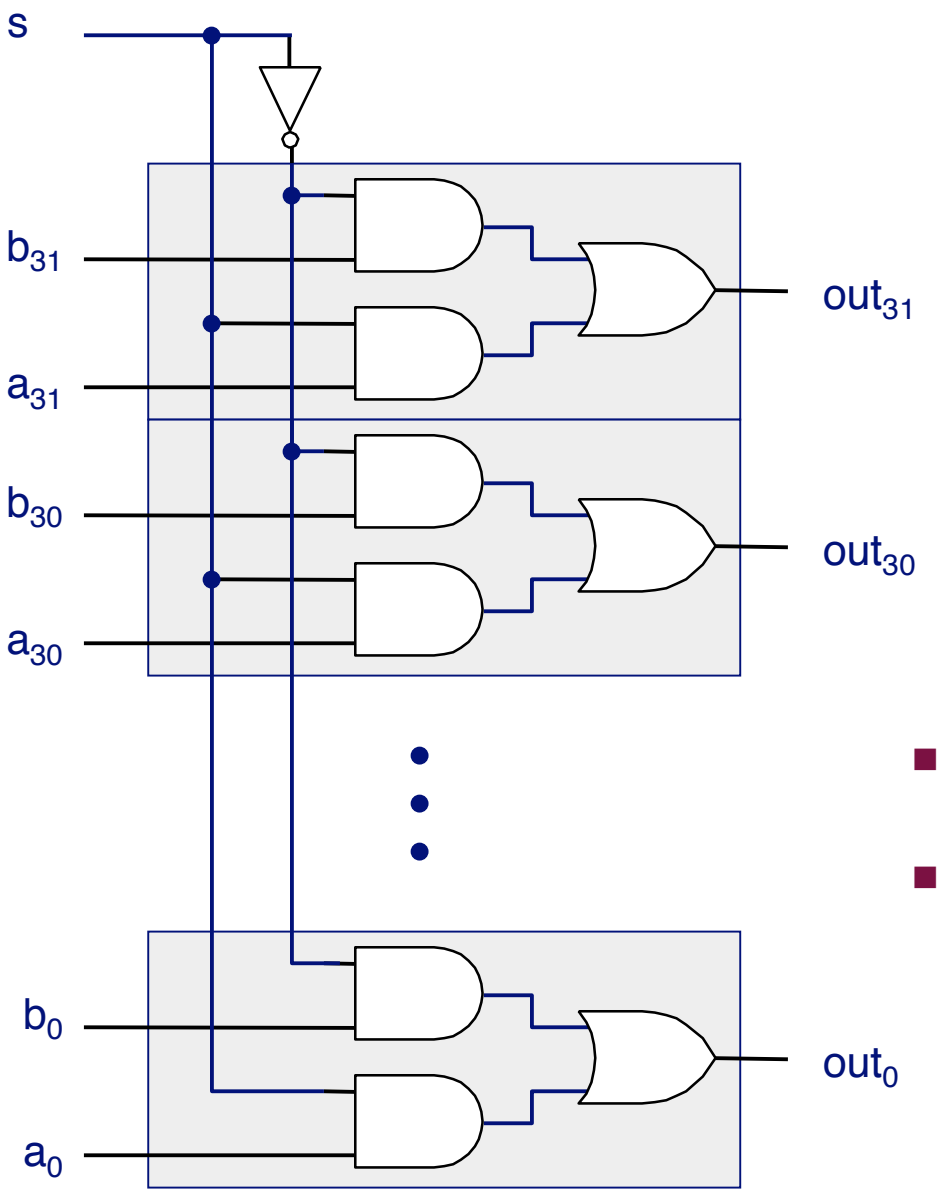


HCL Expression

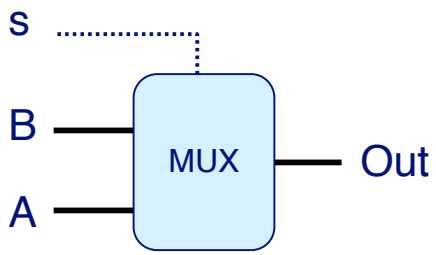
```
bool out = (s&&a)||(!s&&b)
```

- **Control signal s**
- **Data signals a and b**
- **Output a when s=1, b when s=0**

Word Multiplexor



Word-Level Representation



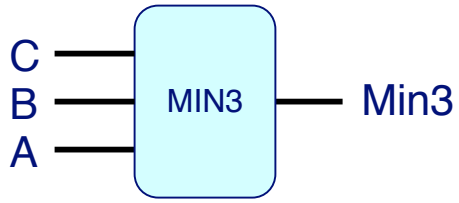
HCL Representation

```
int Out = [
  s : A;
  1 : B;
];
```

- Select input word A or B depending on control signal s
- HCL representation
  - Case expression
  - Series of test : value pairs
  - Output value for first successful test

HCL Word-Level Examples

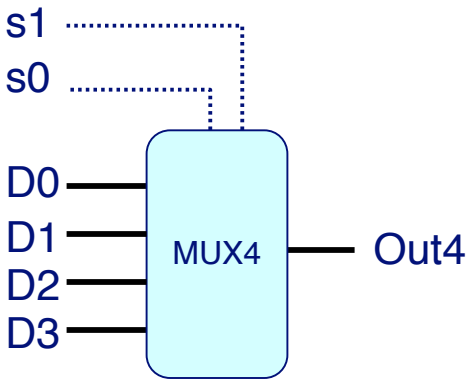
Minimum of 3 Words




```
int Min3 = [
  A < B && A < C : A;
  B < A && B < C : B;
  1              : C;
];
```

- Find minimum of three input words
- HCL case expression
- Final case guarantees match
- How would you build this?

4-Way Multiplexor

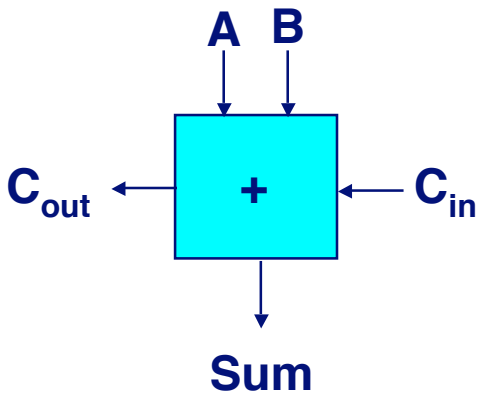



```
int Out4 = [
  !s1&&!s0: D0;
  !s1     : D1;
  !s0     : D2;
  1       : D3;
];
```

- Select one of 4 inputs based on two control bits
- HCL case expression
- Simplify tests by assuming sequential matching

Simple computations are just combinational logic circuits

One Bit Adder



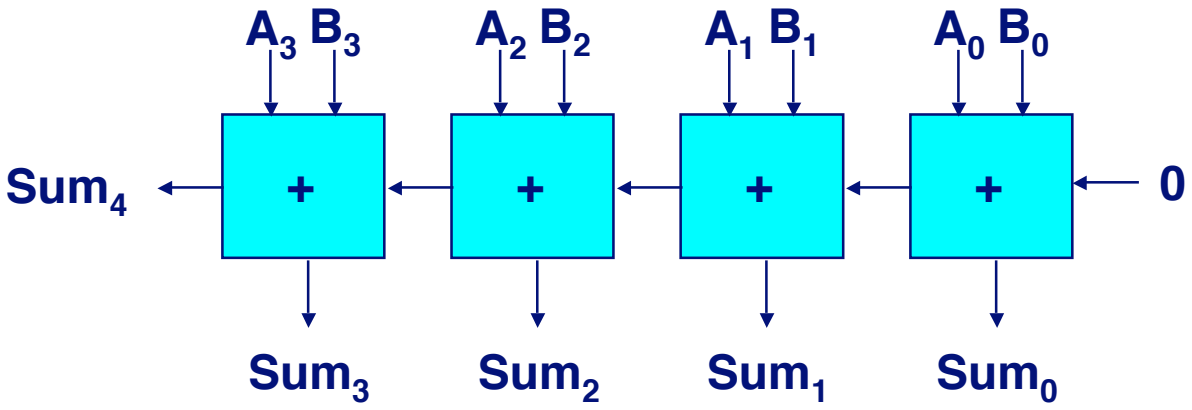
$$Sum = A*B + A*C_{in} + B*C_{in}$$

$$C_{out} = A\text{^}B\text{^}Cin$$
$$= A*B*C_{in} + A*B*C_{in} + A*B*C_{in} + A*B*C_{in}$$

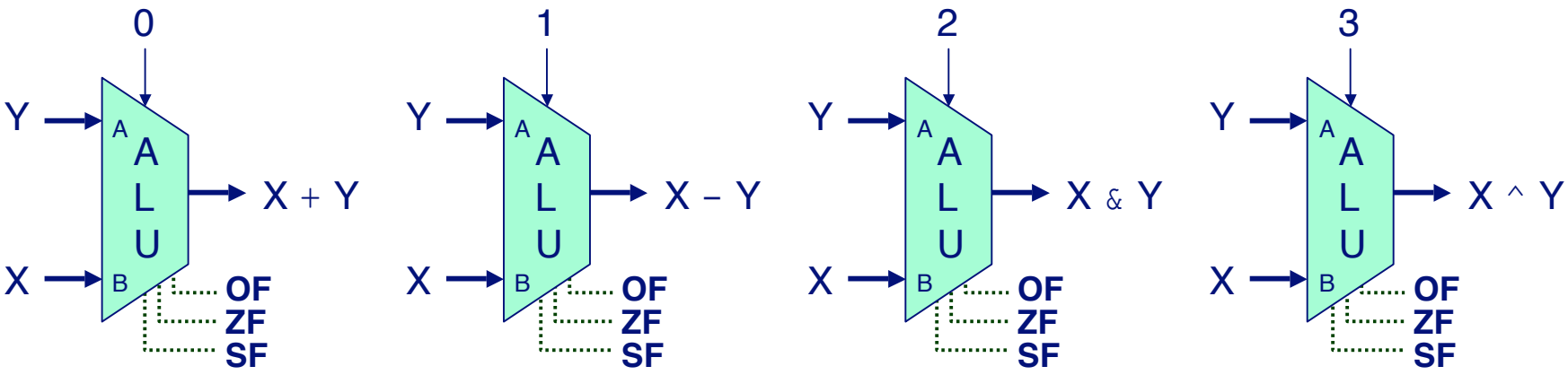
How do you do subtract?

How do you do multiply?

Four Bit Adder

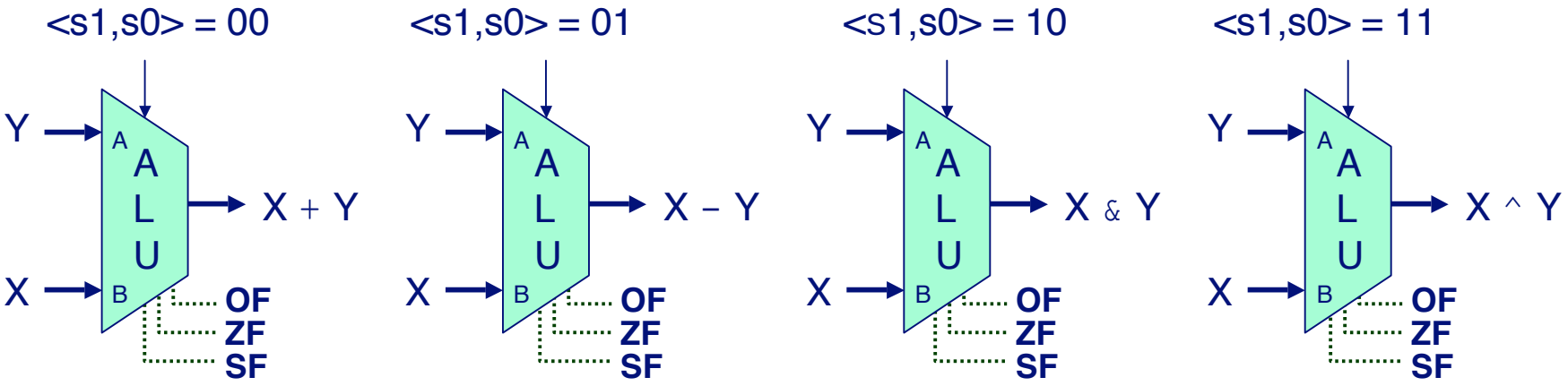


Arithmetic Logic Unit



- Combinational logic
  - Continuously responding to inputs
- Control signal selects function computed
  - Corresponding to 4 arithmetic/logical operations in Y86
- Also computes values for condition codes
  - OF = overflow flag, ZF = Zero Flag, SF = Sign Flag

Arithmetic Logic Unit




```
int Out = [
   !s1&&!s0: X+Y;
   !s1&&s0: Y-Y;
   s1&&!s0: X&Y;
   1       : X^Y;
];
```

Summary

Today

- Basic logic elements
- Combinational logic circuits
  - Truth tables, gates
- Aggregating logic elements
  - Multiplexors, ALUs, etc.

Next Time

- Circuits that remember