October 8

- Rules for Programming Assignments
- Chapter 4 – Arithmetic and its implementation

Cooperation and Honor Code

1. You are encouraged and expected to discuss the material in class and all assignments among yourselves.
2. You are permitted to discuss all aspects of the assignments with anyone.
3. Keys for nearly all homework and programs will be distributed about a week after the due date. **You are not permitted to use the homework or program keys from previous semesters, nor to share the ones received in this course with Comp 120 students in future semesters.**
4. You are encouraged and permitted to discuss and cooperate on all written assignments, but you are expected to understand all material that you submit.

Cooperation and Honor Code

5. You are encouraged to discuss all programming assignments, but not your solutions. Specifically,

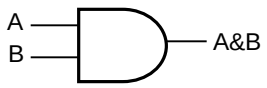
   a. To make learning to use the SPIM simulator easier, you are free to seek help from any source for the first two programming assignments. The programs you hand in must be your own work, but they can be based on unlimited help in all aspects from any sources whatsoever. You should cite in the documentation of your programs anyone who you worked with to complete your assignment.

   b. The rest of the programs are to be done independently by each student. **For details on how the Honor Code applies to these programs, consult the handout 'Honor Code Observation in Computer Science Courses.'**

6. You are prohibited from posting answers to questions from the textbook on the web where the will be accessible to students at other universities.

Logic Functions

A	B	A&B	A|B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1



When we're doing Boolean algebra it is useful to think of AND as multiplication and OR as addition

Boolean Algebra & Gates

- Problem: Consider a logic function with three inputs: A, B, and C.

  Output D is true if at least one input is true
  Output E is true if exactly two inputs are true
  Output F is true only if all three inputs are true

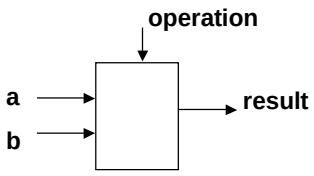
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

- Show the truth table for these three functions.
- Show the Boolean equations for these three functions.

  D=A|B|C E=(~A&B&C)|(A&~B&C)|(A&B&~C) F=A&B&C

An ALU (arithmetic logic unit)

- Let's build an ALU to support the `andi` and `ori` instructions
  - we'll just build a 1 bit ALU, and use 32 of them



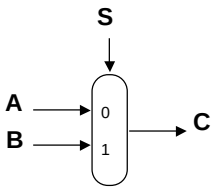
op	a	b	res
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Possible Implementation (sum-of-products):

  ~op & (a|b) | op & a & b == ~op&a | ~op&b | op&a&b

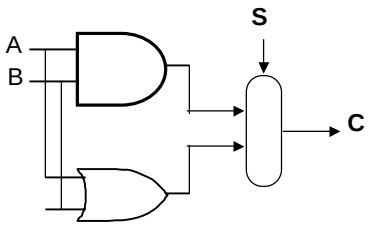
Multiplexor

- Selects one of the inputs to be the output, based on a control input



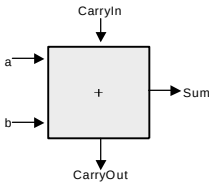
note: we call this a 2-input mux even though it has 3 inputs!

- Lets build our ALU using a MUX:



1-bit ALU for Addition

- Not easy to decide the "best" way to build something
  - Don't want too many inputs to a single gate
  - Dont want to have to go through too many gates
  - for our purposes, ease of comprehension is important
- Let's look at a 1-bit ALU for addition:

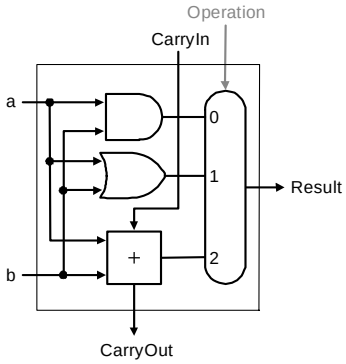


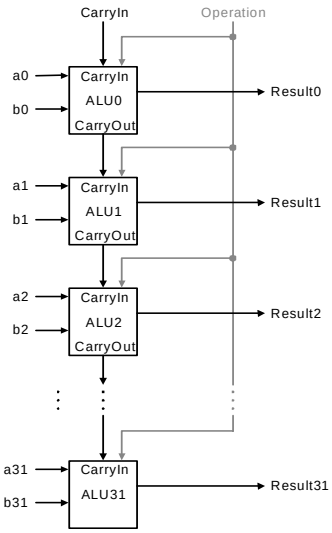
c_{out} = ~a&b&c_{in} | a&~b&c_{in} | a&b&~c_{in} | a&b&c

sum = ~a&~b&c_{in} | ~a&b&~c_{in} | a&~b&~c_{in} | a&b&c_{in}

a	b	cin	sum	cout
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

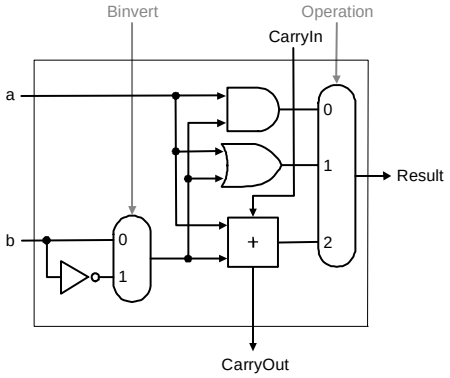
Building a 32 bit ALU





What about subtraction (a – b) ?

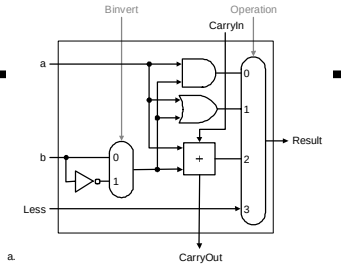
- Two's complement approach: just negate b and add.
- How do we negate?
- A very clever solution:

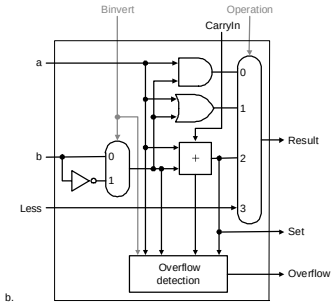


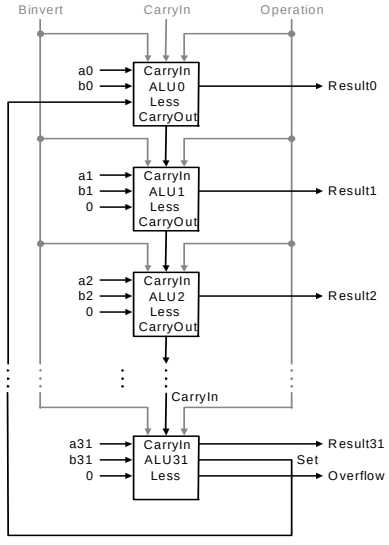
Tailoring the ALU to the MIPS

- Need to support the set-on-less-than instruction (slt)
  - remember: slt is an arithmetic instruction
  - produces a 1 if rs < rt and 0 otherwise
  - use subtraction: (a-b) < 0 implies a < b
- Need to support test for equality (beq $t5, $t6, $t7)
  - use subtraction: (a-b) = 0 implies a = b

Implement SLT





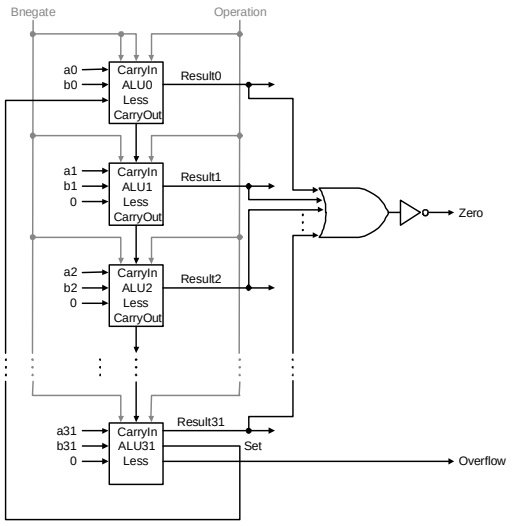


Test for equality

- Notice control lines:

```
000 = and
001 = or
010 = add
110 = subtract
111 = slt
```

•Note: zero is a 1 when the result is zero!



Conclusion

- We can build an ALU to support the MIPS instruction set
  - key idea: use multiplexor to select the output we want
  - we can efficiently perform subtraction using two's complement
  - we can replicate a 1-bit ALU to produce a 32-bit ALU
- Important points about hardware
  - all of the gates are always working
  - the speed of a gate is affected by the number of inputs to the gate
  - the speed of a circuit is affected by the number of gates in series (on the "critical path" or the "deepest level of logic")
- Our primary focus: comprehension, however,
  - Clever changes to organization can improve performance (similar to using better algorithms in software)