

## 23 September

- Chapter 4 – Arithmetic and its implementation
- Assignments 5,6 and 7 posted to the class web page

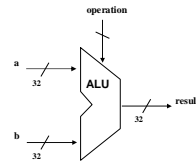
9/23/2004

Comp 120 Fall 2004

1

## Arithmetic

- Where we've been:
  - Performance (seconds, cycles, instructions)
  - Abstractions:
    - Instruction Set Architecture
    - Assembly Language and Machine Language
- What's ahead:
  - Implementing the Architecture



9/23/2004

Comp 120 Fall 2004

2

## Numbers

- Bits are just bits (no inherent meaning)
  - conventions define relationship between bits and numbers
- Binary numbers (base 2)
  - 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001...
  - decimal:  $0 \dots 2^n - 1$
- Of course it gets more complicated:
  - numbers are finite (overflow)
  - fractions and real numbers
  - negative numbers
  - e.g., no MIPS subi; addi can add a negative number)
- How do we represent negative numbers?
  - i.e., which bit patterns will represent which numbers?

9/23/2004

Comp 120 Fall 2004

3

## Possible Representations

- | Sign Magnitude: | One's Complement | Two's Complement |
|-----------------|------------------|------------------|
| 000 = +0        | 000 = +0         | 000 = +0         |
| 001 = +1        | 001 = +1         | 001 = +1         |
| 010 = +2        | 010 = +2         | 010 = +2         |
| 011 = +3        | 011 = +3         | 011 = +3         |
| 100 = -0        | 100 = -3         | 100 = -4         |
| 101 = -1        | 101 = -2         | 101 = -3         |
| 110 = -2        | 110 = -1         | 110 = -2         |
| 111 = -3        | 111 = -0         | 111 = -1         |
- Issues: balance, number of zeros, ease of operations
  - Which one is best? Why?

9/23/2004

Comp 120 Fall 2004

4

## MIPS uses 2's Complement

•32 bit signed numbers:

```

0000 0000 0000 0000 0000 0000 0000 0000two = 0ten
0000 0000 0000 0000 0000 0000 0000 0001two = + 1ten
0000 0000 0000 0000 0000 0000 0000 0010two = + 2ten
...
0111 1111 1111 1111 1111 1111 1111 1110two = + 2,147,483,646ten maxint
0111 1111 1111 1111 1111 1111 1111 1111two = + 2,147,483,647ten
1000 0000 0000 0000 0000 0000 0000 0000two = - 2,147,483,648ten minint
1000 0000 0000 0000 0000 0000 0000 0001two = - 2,147,483,647ten
1000 0000 0000 0000 0000 0000 0000 0010two = - 2,147,483,646ten
...
1111 1111 1111 1111 1111 1111 1111 1101two = - 3ten
1111 1111 1111 1111 1111 1111 1111 1110two = - 2ten
1111 1111 1111 1111 1111 1111 1111 1111two = - 1ten
    
```

9/23/2004

Comp 120 Fall 2004

5

## Two's Complement Operations

- Negating a two's complement number: invert all bits and add 1
  - remember: "negate" and "invert" are quite different!
  - $-3 == -(0011) == -0011 + 1 == 1100 + 1 == 1101$
- Converting n bit numbers into numbers with more than n bits:
  - MIPS 16 bit immediate gets converted to 32 bits for arithmetic
  - copy the most significant bit (the sign bit) into the other bits
    - 0010 -> 0000 0010
    - 1010 -> 1111 1010
  - "sign extension" (lbu vs. lb)

9/23/2004

Comp 120 Fall 2004

6

## Addition & Subtraction

- Just like in grade school (carry/borrow 1s)
 

00111	00111	00110
+ 00110	- 00110	- 00101
01101	00001	00001
- Two's complement operations easy
  - subtraction using addition of negative numbers
 

00000111	00000111
- 00000110	+ 11111010
00000001	
- Overflow (result too large for finite computer word):
  - e.g., adding two n-bit numbers does not yield an n-bit number
 

01110000	
+ 00010000	<i>note that overflow term is somewhat misleading,</i>
10000000	<i>it does not mean a carry "overflowed"</i>

9/23/2004

Comp 120 Fall 2004

7

## Detecting Overflow

- No overflow when adding a positive and a negative number
- No overflow when signs are the same for subtraction
- Overflow occurs when the value affects the sign:
  - overflow when adding two positives yields a negative
  - or, adding two negatives gives a positive
  - or, subtract a negative from a positive and get a negative
  - or, subtract a positive from a negative and get a positive
- Consider the operations  $A + B$ , and  $A - B$ 
  - Can overflow occur if  $B$  is 0?
  - Can overflow occur if  $A$  is 0?

9/23/2004

Comp 120 Fall 2004

8

## Effects of Overflow

- An exception (interrupt) occurs
  - Control jumps to predefined address for exception
  - Interrupted address is saved for possible resumption
- Details based on software system / language
  - example: flight control vs. homework assignment
- Don't always want to detect overflow
  - new MIPS instructions: `addu`, `addiu`, `subu`

*note: addiu still sign-extends the operand*  
*note: sltu, sltiu for unsigned comparisons*

9/23/2004

Comp 120 Fall 2004

9

## 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

9/23/2004

Comp 120 Fall 2004

10

## 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

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

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

$$D=A|B|C \quad E=(\neg A \& B \& C) | (A \& \neg B \& C) | (A \& B \& \neg C) \quad F=A \& B \& C$$

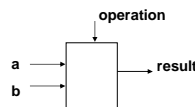
9/23/2004

Comp 120 Fall 2004

11

## 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



- Possible Implementation (sum-of-products):

$$\sim op \& (a|b) | op \& a \& b == \sim op \& a | \sim op \& b | op \& a \& b$$

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

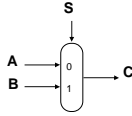
9/23/2004

Comp 120 Fall 2004

12

## 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:

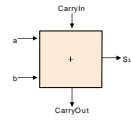
9/23/2004

Comp 120 Fall 2004

13

## 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
  - Don't 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:



$$c_{out} = a \& b \mid a \& c_{in} \mid b \& c_{in}$$

$$sum = a \oplus b \oplus c_{in} \mid \sim a \& b \oplus \sim c_{in} \mid \sim a \& \sim b \& c_{in} \mid a \& b \& c_{in}$$

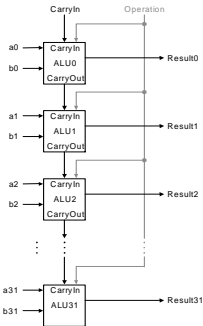
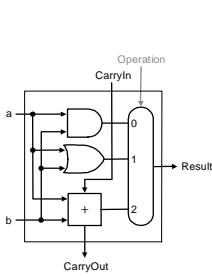
a	b	cin	sum	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

9/23/2004

Comp 120 Fall 2004

14

## Building a 32 bit ALU



9/23/2004

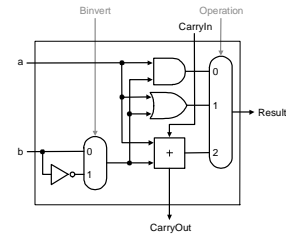
Comp 120 Fall 2004

15

## What about subtraction (a - b) ?

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

- A very clever solution:



9/23/2004

Comp 120 Fall 2004

16

## Tailoring the ALU to the MIPS

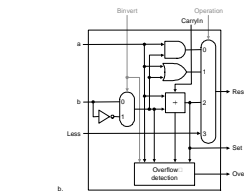
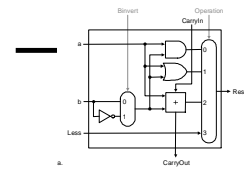
- Need to support the set-on-less-than instruction (slt)
  - remember: slt is an arithmetic instruction
  - produces a 1 if  $rs < rt$  and 0 otherwise
  - use subtraction:  $(a-b) < 0$  implies  $a < b$
- Need to support test for equality (beq \$t5, \$t6, \$t7)
  - use subtraction:  $(a-b) = 0$  implies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

9/23/2004

Comp 120 Fall 2004

17

## Implement SLT



9/23/2004

Comp 120 Fall 2004

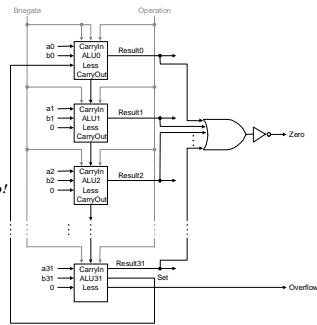
18

## 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!*



9/23/2004

Comp 120 Fall 2004

19

## 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)

9/23/2004

Comp 120 Fall 2004

20