

Introduction

S4.1 Introduction

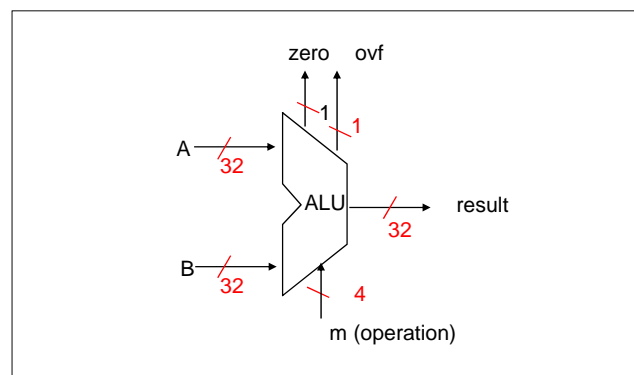
- CPU performance factors
 - Instruction count
 - Determined by ISA and compiler
 - CPI and Cycle time
 - Determined by CPU hardware
- We will examine two MIPS implementations
 - A simplified version
 - A more realistic pipelined version
- Simple subset, shows most aspects
 - Memory reference: l w, s w
 - Arithmetic/logical: add, sub, and, or, s l t
 - Control transfer: beq, j



Chapter 4 — The Processor — 3

Design of a Simple ALU

- 2 inputs, operation and an output



Chapter 4 — The Processor — 4

Possible Representations

Sign Mag.	Two's Comp.	One's Comp.
	1000 = -8	
1111 = -7	1001 = -7	1000 = -7
1110 = -6	1010 = -6	1001 = -6
1101 = -5	1011 = -5	1010 = -5
1100 = -4	1100 = -4	1011 = -4
1011 = -3	1101 = -3	1100 = -3
1010 = -2	1110 = -2	1101 = -2
1001 = -1	1111 = -1	1110 = -1
1000 = -0		1111 = -0
0000 = +0	0000 = 0	0000 = +0
0001 = +1	0001 = +1	0001 = +1
0010 = +2	0010 = +2	0010 = +2
0011 = +3	0011 = +3	0011 = +3
0100 = +4	0100 = +4	0100 = +4
0101 = +5	0101 = +5	0101 = +5
0110 = +6	0110 = +6	0110 = +6
0111 = +7	0111 = +7	0111 = +7

- Issues:
 - balance
 - number of zeros
 - ease of operations
- Which one is best? Why?

MIPS Representations

- 32-bit signed numbers (2's complement):

```

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
0111 1111 1111 1111 1111 1111 1111 1111two = + 2,147,483,647ten
1000 0000 0000 0000 0000 0000 0000 0000two = - 2,147,483,648ten
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
    
```

maxint

minint

- What if the bit string represented addresses?
 - need operations that also deal with only positive (unsigned) integers

Two's Complement Operations

- Negating a two's complement number – complement all the bits and then add a 1
 - remember: “negate” and “invert” are quite different!
 - Starting from LSb, all 0's as is, first 1 as is, then invert
- Converting n-bit numbers into numbers with more than n bits:
 - MIPS 16-bit immediate gets converted to 32 bits for arithmetic
 - sign extend - copy the most significant bit (the sign bit) into the other bits

```
0010  -> 0000 0010
1010  -> 1111 1010
```

- sign extension versus zero extend (lb vs. lbu)



Addition & Subtraction

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

$\begin{array}{r} 0111 \\ + 0110 \\ \hline 1101 \end{array}$	$\begin{array}{r} 0111 \\ - 0110 \\ \hline 0001 \end{array}$	$\begin{array}{r} 0110 \\ - 0101 \\ \hline 0001 \end{array}$
--	--	--

- Two's complement operations are easy
 - do subtraction by negating and then adding

$\begin{array}{r} 0111 \\ - 0110 \\ \hline 0001 \end{array}$	\rightarrow	$\begin{array}{r} 0111 \\ + 1010 \\ \hline 1\ 0001 \end{array}$
--	---------------	---

- Overflow (result too large for **finite** computer word)
 - e.g., adding two n-bit numbers does not yield an n-bit number

$\begin{array}{r} 0111 \\ + 0001 \\ \hline 1000 \end{array}$
--



Building a 1-bit Binary Adder

A	B	carry_in	carry_out	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \text{ xor } B \text{ xor } \text{carry_in}$$

$$\text{carry_out} = A \& B \mid A \& \text{carry_in} \mid B \& \text{carry_in}$$

(majority function)

- How can we use it to build a 32-bit adder?
- How can we modify it easily to build an adder/subtractor?

Building 32-bit Adder

- Just connect the carry-out of the least significant bit FA to the carry-in of the next least significant bit and connect . . .
- Ripple Carry Adder (RCA)
 - advantage: simple logic, so small (low cost)
 - disadvantage: slow and lots of glitching (so lots of energy consumption)

A 32-bit Ripple Carry Adder/Subtractor

- Remember 2's complement is just
 - complement all the bits

control (0=add, 1=sub)
 B_0 B_0 if control = 0, $!B_0$ if control = 1

- add a 1 in the least significant bit

A	0111	→	0111
B	<u>0110</u>	→ +	

A 32-bit Ripple Carry Adder/Subtractor

- Remember 2's complement is just
 - complement all the bits

control (0=add, 1=sub)
 B_0 B_0 if control = 0, $!B_0$ if control = 1

- add a 1 in the least significant bit

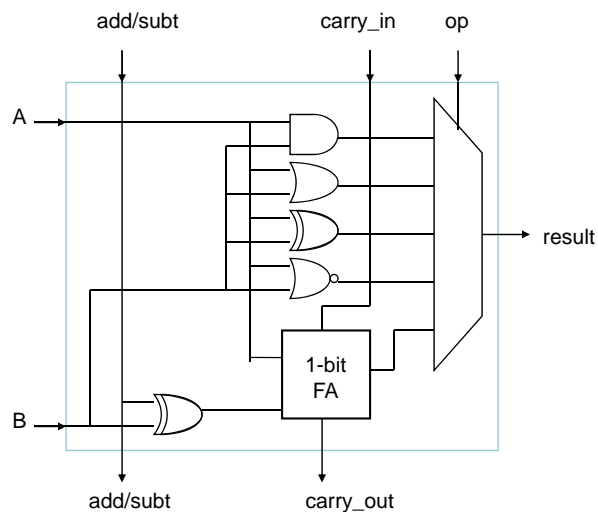
A	0111	→	0111
B	<u>0110</u>	→ +	1001
	0001		<u>1</u>
			1 0001

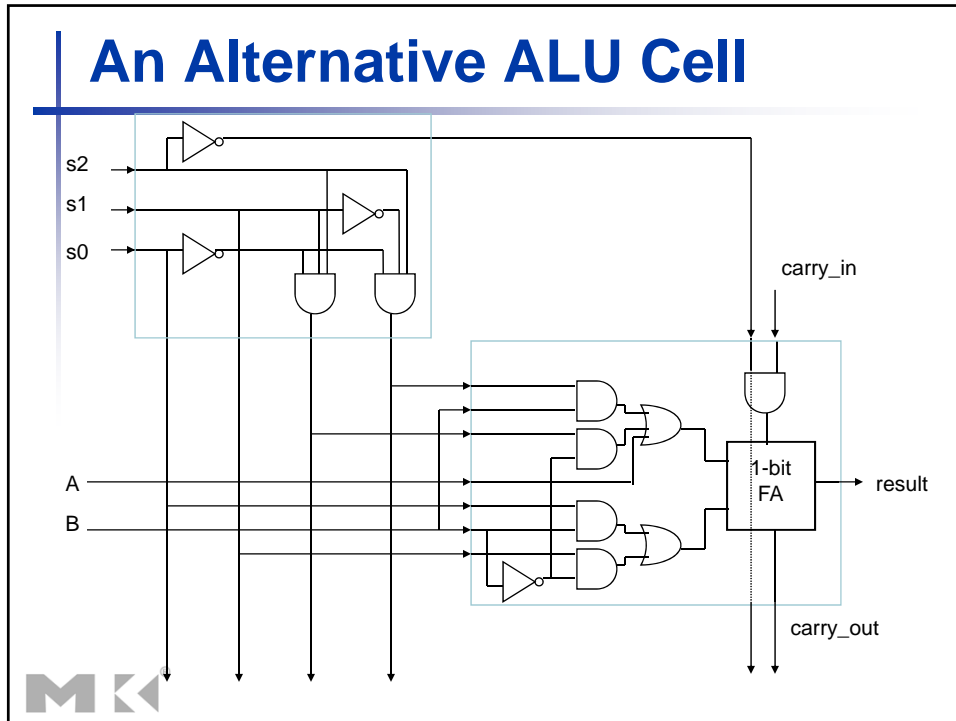
Overflow Detection and Effects

- Overflow: the result is too large to represent in the number of bits allocated
- When adding operands with different signs, overflow cannot occur! Overflow occurs when
 - adding two positives yields a negative
 - or, adding two negatives gives a positive
 - or, subtract a negative from a positive gives a negative
 - or, subtract a positive from a negative gives a positive
- On overflow, an exception (interrupt) occurs
 - Control jumps to predefined address for exception
 - Interrupted address (address of instruction causing the overflow) is saved for possible resumption
- Don't always want to detect (interrupt on) overflow



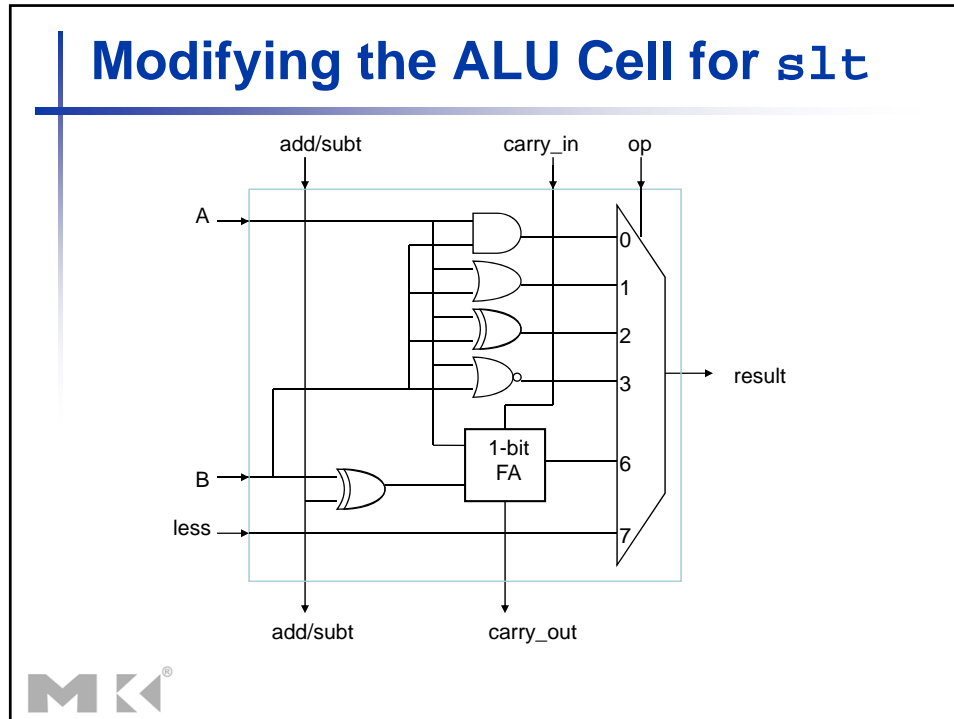
A Simple ALU Cell with Logic Op Support





The Alternative ALU Cell's Control Codes

s2	s1	s0	c_in	result	function
0	0	0	0	A	transfer A
0	0	0	1	A + 1	increment A
0	0	1	0	A + B	add
0	0	1	1	A + B + 1	add with carry
0	1	0	0	A - B - 1	subt with borrow
0	1	0	1	A - B	subtract
0	1	1	0	A - 1	decrement A
0	1	1	1	A	transfer A
1	0	0	x	A or B	or
1	0	1	x	A xor B	xor
1	1	0	x	A and B	and
1	1	1	x	!A	complement A



Overflow Detection

- Overflow occurs when the result is too large to represent in the number of bits allocated
 - adding two positives yields a negative
 - or, adding two negatives gives a positive
 - or, subtract a negative from a positive gives a negative
 - or, subtract a positive from a negative gives a positive
- On your own: **Prove** you can detect overflow by:
 - Carry into MSB xor Carry out of MSB

	0	1	1	1	7
+	0	0	1	1	3
<hr/>					

	1	1	0	0	-4
+	1	0	1	1	-5
<hr/>					

