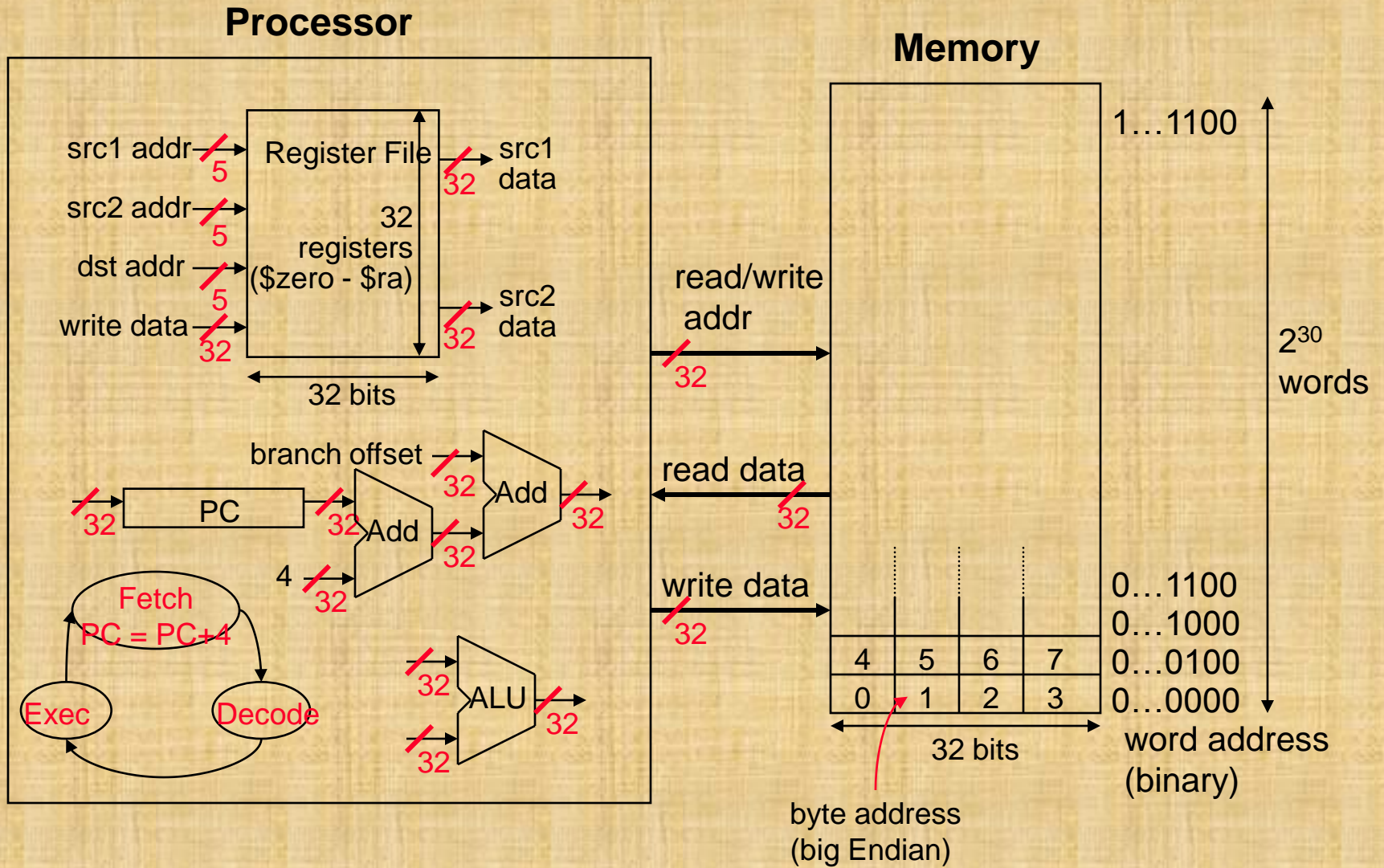


Chapter 3

Computer Abstractions and Technology

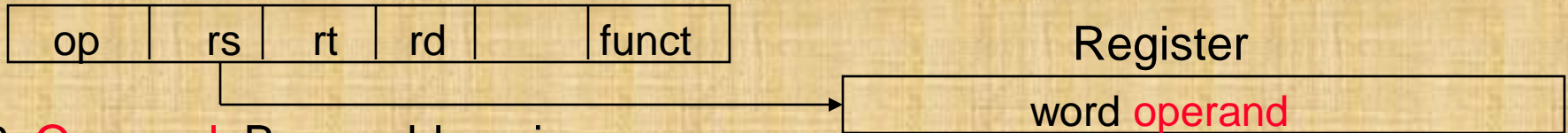
Lesson 6: Arithmetic for Computers

Review: MIPS Organization

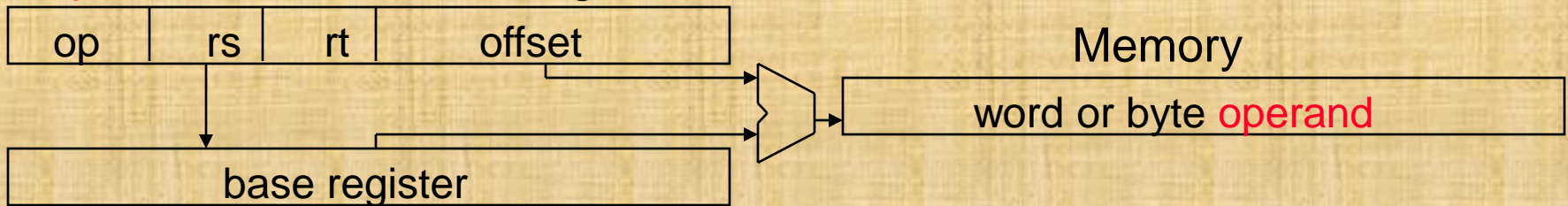


Review: MIPS Addressing Modes

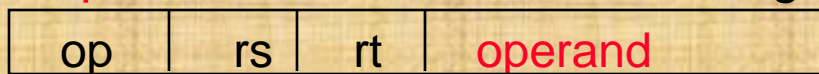
1. **Operand:** Register addressing



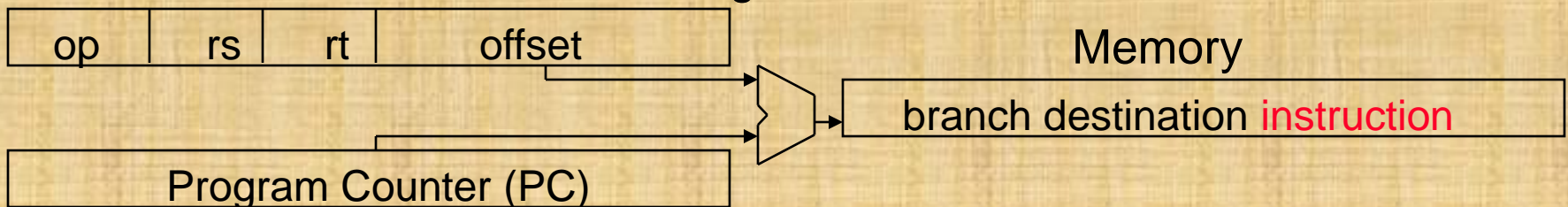
2. **Operand:** Base addressing



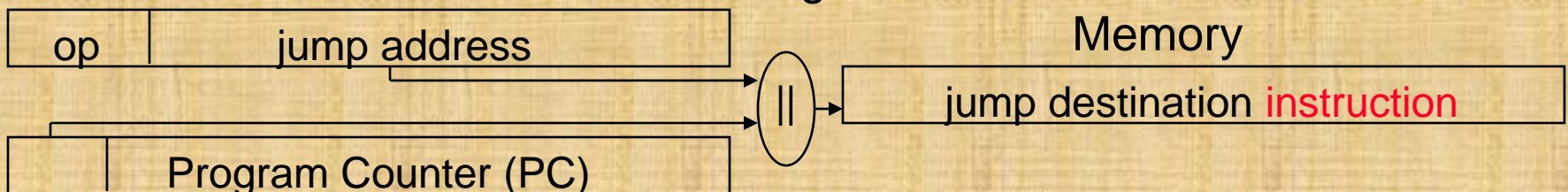
3. **Operand:** Immediate addressing



4. **Instruction:** PC-relative addressing



5. **Instruction:** Pseudo-direct addressing



MIPS Number Representations

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

	0000	0000	0000	0000	0000	0000	0000	0000	$0_{two} = 0_{ten}$
	0000	0000	0000	0000	0000	0000	0000	0001	$1_{two} = +1_{ten}$
	...								
	0111	1111	1111	1111	1111	1111	1111	1110	$= +2,147,483,646_{ten}$
	0111	1111	1111	1111	1111	1111	1111	1111	$= +2,147,483,647_{ten}$
	1000	0000	0000	0000	0000	0000	0000	0000	$= -2,147,483,648_{ten}$
	1000	0000	0000	0000	0000	0000	0000	0001	$= -2,147,483,647_{ten}$
	...								
MSB	1111	1111	1111	1111	1111	1111	1111	1110	$= -2_{ten}$
	1111	1111	1111	1111	1111	1111	1111	1111	$= -1_{ten}$

maxint

minint

LSB

MIPS Sign Extend

- ◆ In byte or half word, the sign extension repeats the most significant bit until the whole word is filled.
- ◆ Converting <32-bit values into 32-bit values
 - copy the most significant bit (the sign bit) into the “empty” bits

0010 -> 0000 0010

1010 -> 1111 1010

◆ **sign extend** versus **zero extend**

▪ i.e. **lb** versus **lbu**

Review: 2's Complement Binary Representation

- ◆ Assume $n=3$

- n is the no. of bits

- ◆ **Negate**

1011

and add a 1

1010

complement all the bits

- ◆ Note: **negate** and **invert** are different!

$$-2^3 =$$
$$-(2^3 - 1) =$$

$$2^3 - 1 =$$

2'sc binary	decimal
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

Arithmetic for Computers

- ◆ Operations on integers
 - Addition and subtraction
 - Multiplication and division
 - Dealing with overflow
- ◆ Floating-point real numbers
 - Representation and operations

MIPS Arithmetic Logic Unit (ALU)

- ◆ Must support the Arithmetic/Logic operations of the ISA

add, addi, addiu, addu

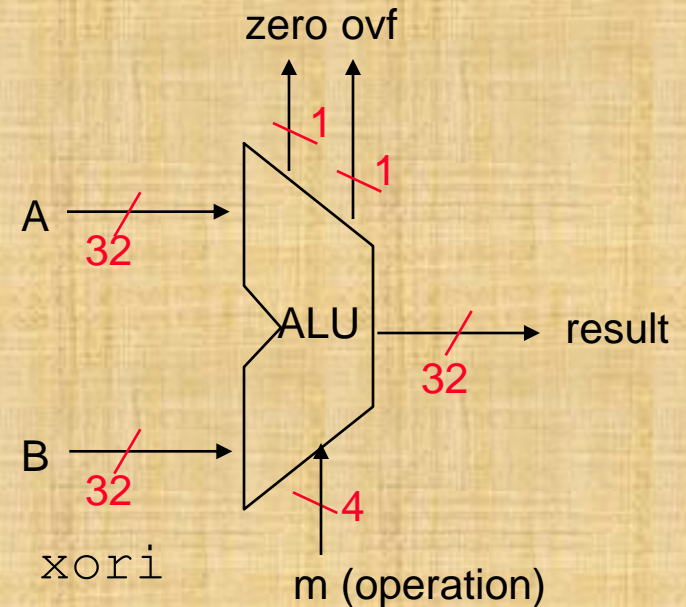
sub, subu, neg

mult, multu, div, divu

sqrt

and, andi, nor, or, ori, xor, xori

beq, bne, slt, slti, sltiu, sltu



- ◆ With special handling for

- ❑ sign extend – addi, addiu andi, ori, xori, slti, sltiu

- ❑ zero extend – lbu, addiu, sltiu

- ❑ no overflow detected – addu, addiu, subu, multu, divu, sltiu, sltu

Integer Subtraction

- ◆ Add negation of second operand

- ◆ Example: $7 - 6 = 7 + (-6)$

$$\begin{array}{r} +7: \quad 0000 \ 0000 \ \dots \ 0000 \ 0111 \\ -6: \quad 1111 \ 1111 \ \dots \ 1111 \ 1010 \\ \hline +1: \quad 0000 \ 0000 \ \dots \ 0000 \ 0001 \end{array}$$

- ◆ Overflow if result out of range
 - ❑ Subtracting two +ve or two -ve operands, no overflow
 - ❑ Subtracting +ve from -ve operand
 - Overflow if result sign is 0
 - ❑ Subtracting -ve from +ve operand
 - Overflow if result sign is 1

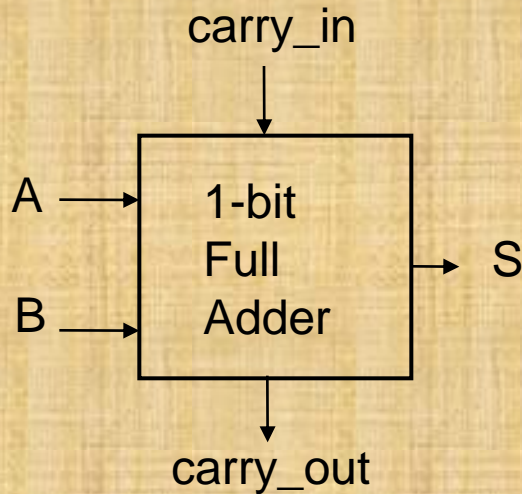
Dealing with Overflow

- ◆ Some languages (e.g., C) ignore overflow
 - Use MIPS `addu`, `addui`, `subu` instructions
- ◆ Other languages (e.g., Ada, Fortran) require raising an exception
 - Use MIPS `add`, `addi`, `sub` instructions
 - On overflow, invoke exception handler
 - Save PC in exception program counter (EPC) register
 - Jump to predefined handler address
 - `mfc0` (move from coprocessor reg) instruction can retrieve EPC value, to return after corrective action

Arithmetic for Multimedia

- ◆ Graphics and media processing operates on vectors of 8-bit and 16-bit data
 - Use 64-bit adder, with partitioned carry chain
 - Operate on 8 8-bit, 4 16-bit, or 2 32-bit vectors
 - SIMD (single-instruction, multiple-data)
- ◆ Saturating operations
 - On overflow, result is largest representable value
 - c.f. 2s-complement modulo arithmetic
 - E.g., clipping in audio, saturation in video

Review: A Full 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

$$S = A \oplus B \oplus \text{carry_in} \quad (\text{odd parity function})$$

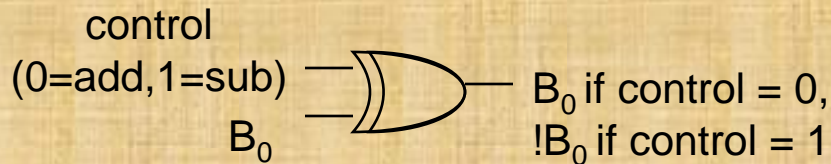
$$\text{carry_out} = A \& B \mid A \& \text{carry_in} \mid B \& \text{carry_in} \\ (\text{majority function})$$

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

A 32-bit Ripple Carry Adder/Subtractor

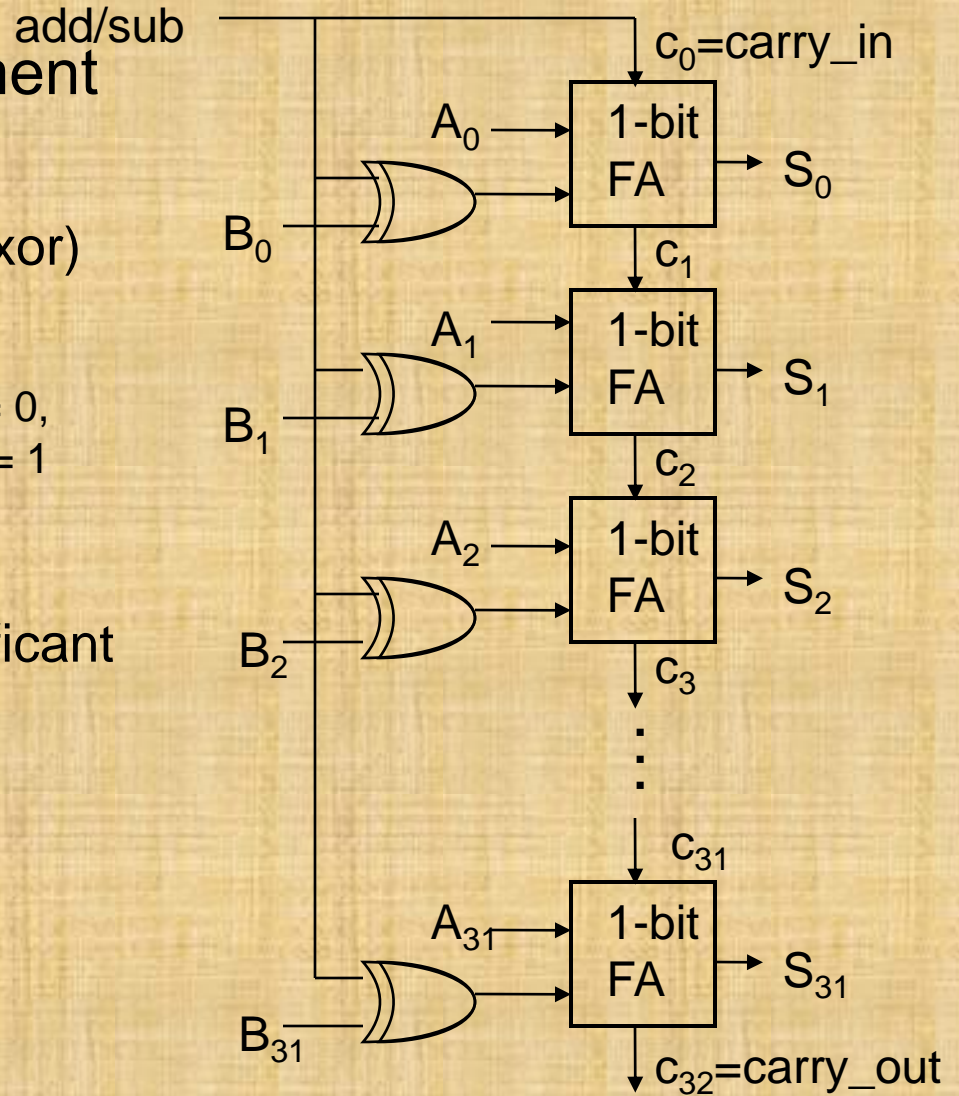
Remember 2's complement is just

- complement all the bits (xor)



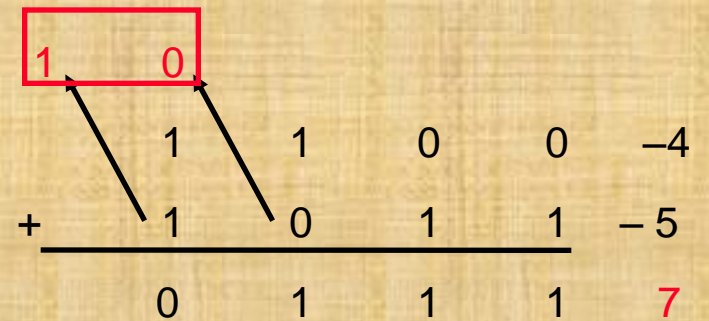
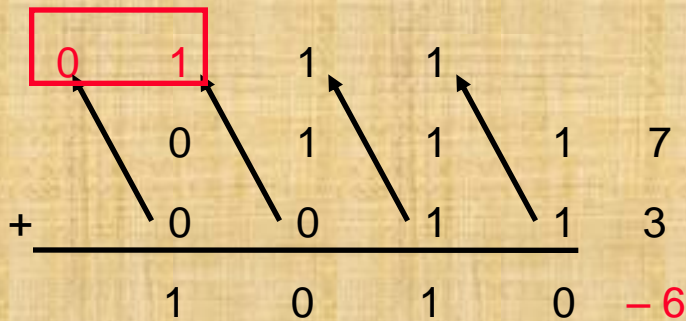
- add a 1 in the least significant bit

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



Overflow Detection

- ◆ Overflow: the result is too large to represent in 32 bits
- ◆ 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 your own: **Prove** you can detect overflow by:
 - ❑ Carry *into* MSB xor Carry *out* of MSB, ex for 4 bit signed numbers



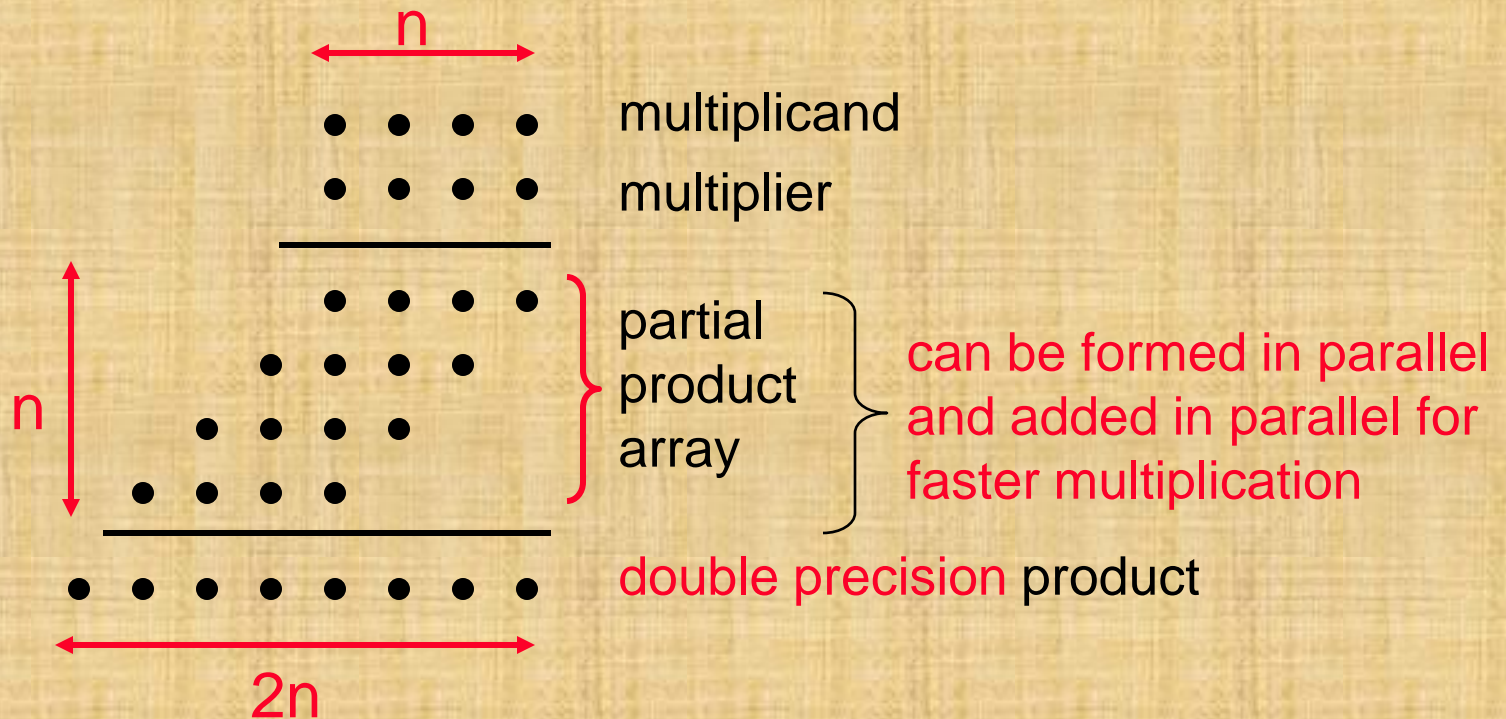
MIPS Overflow detection

Operation	Operand A	Operand B	Result indicating overflow
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

- ◆ no overflow is detected when using
 - `addu, addiu, subu, multu, divu, sltiu, sltu`

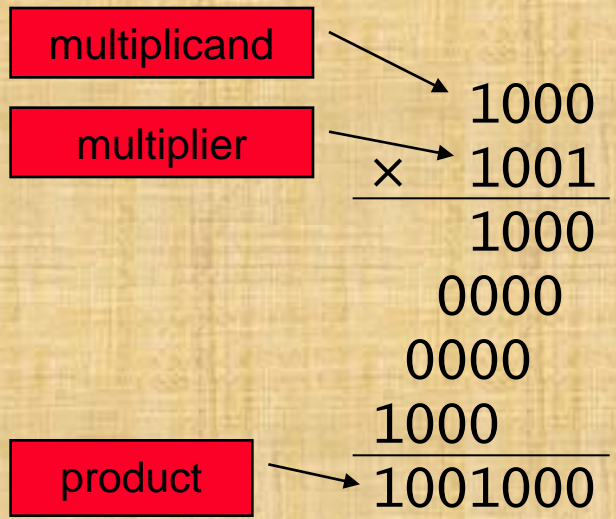
Multiply

- Binary multiplication is just a *bunch* of right shifts and adds

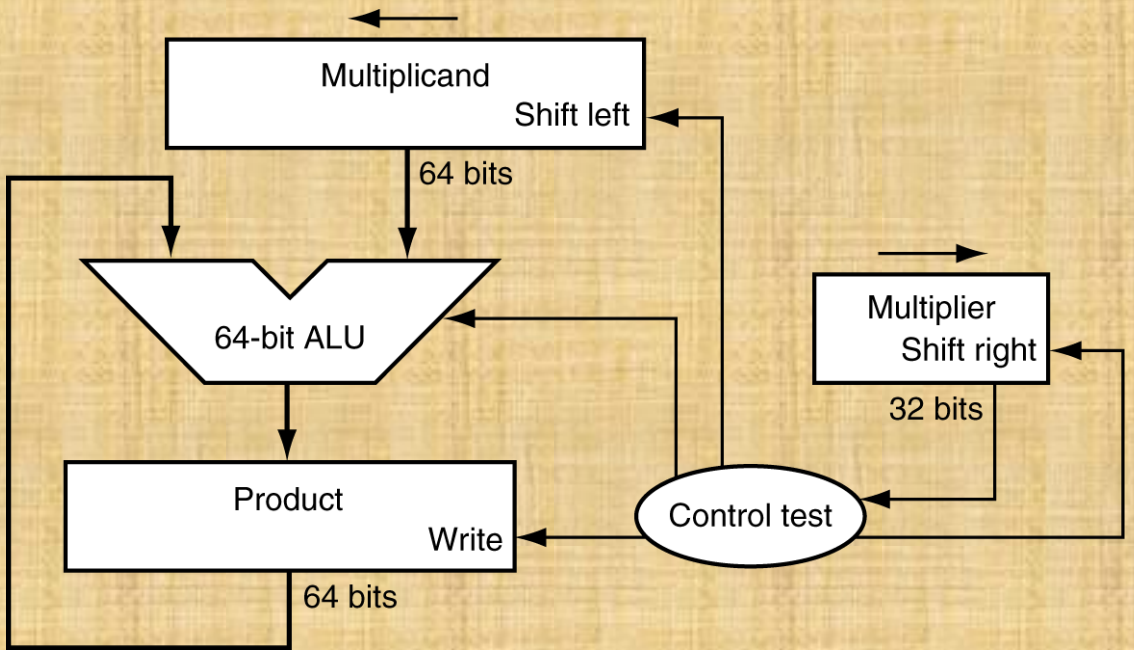


Multiplication

- ◆ Start with long-multiplication approach



Length of product is the sum of operand lengths



Next Lecture and Reminders

◆ Next lecture

- Improving Multiplication
- Addressing Division and Floating Point
 - Reading assignment – PH, Chapter 3, 3.4-3.6 and 3.8

◆ Reminders

- Next class is dedicated to Review the material for the coming Test.
- Test is Oct 1