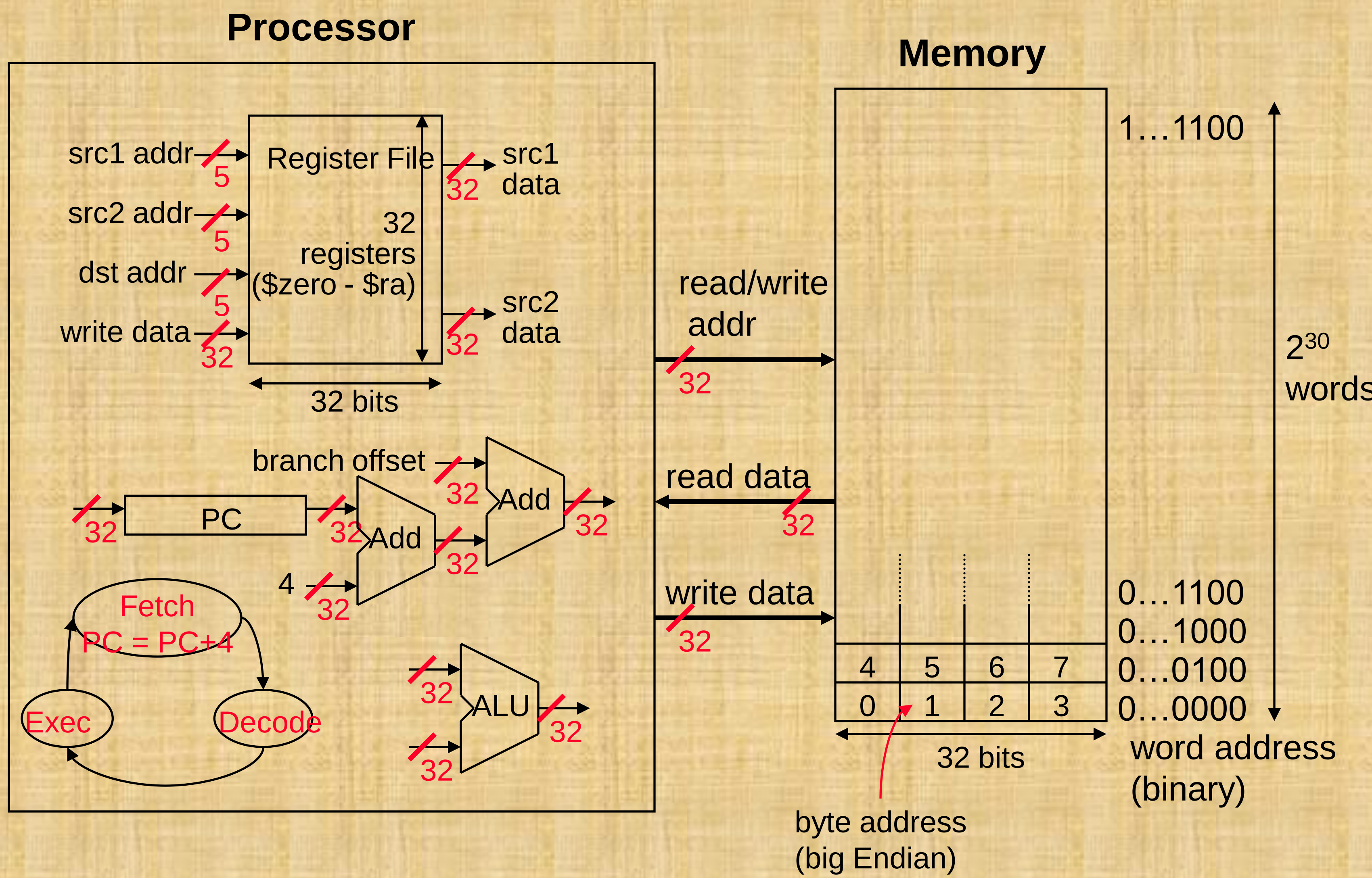
Chapter 3

Computer Abstractions and Technology

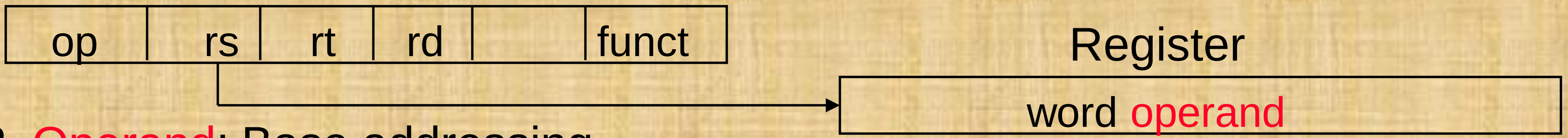
Lesson 6: Arithmetic for Computers

Review: MIPS Organization

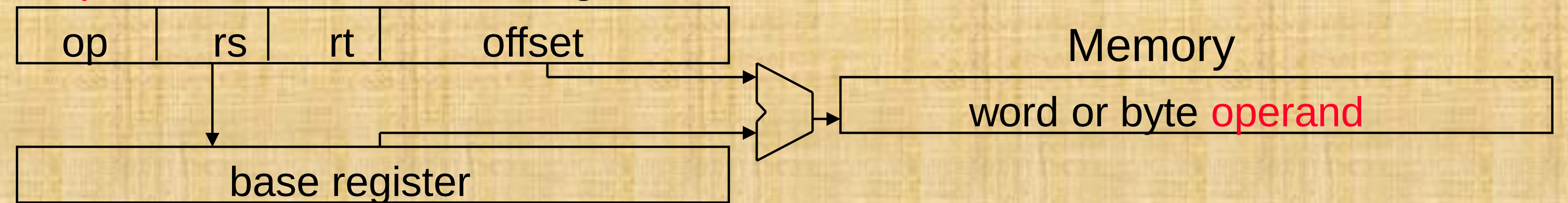


Review: MIPS Addressing Modes

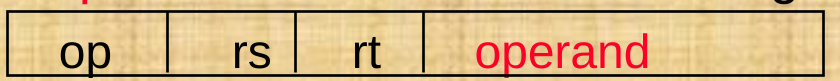
1. Operand: Register addressing



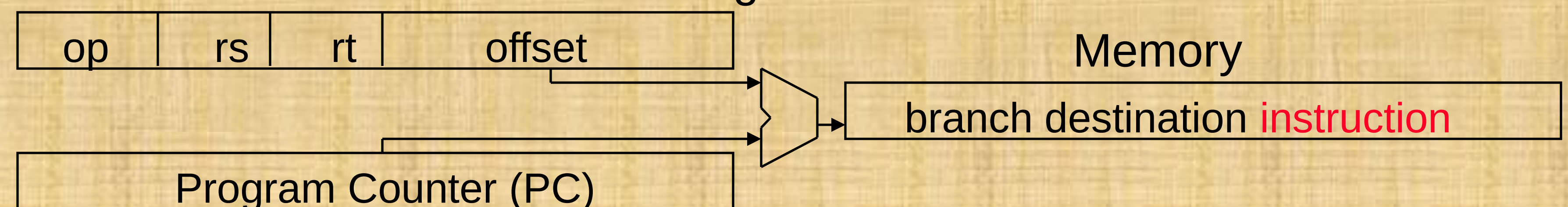
2. Operand: Base addressing



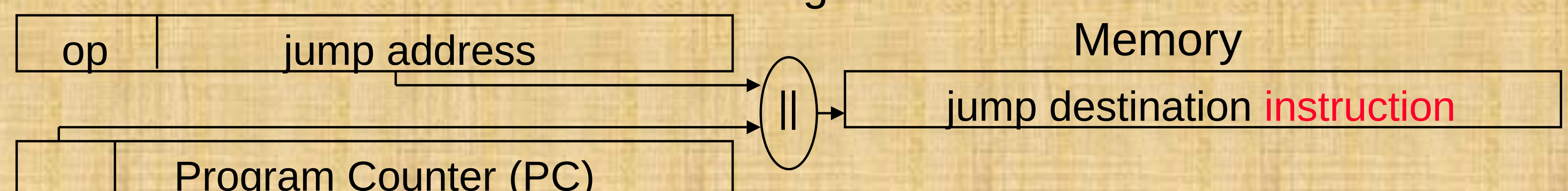
3. Operand: Immediate addressing



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_{two} = 0_{ten}
0000 0000 0000 0000 0000 0000 0000 0001_{two} = + 1_{ten}
...

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

1111 1111 1111 1111 1111 1111 1111 1110_{two} = – 2_{ten}
1111 1111 1111 1111 1111 1111 1111 1111_{two} = – 1_{ten}

maxint

minint

MSB

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'sc binary	decimal
-2^3 =	1000	-8
$-(2^3 - 1)$ =	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
$2^3 - 1$ =	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`

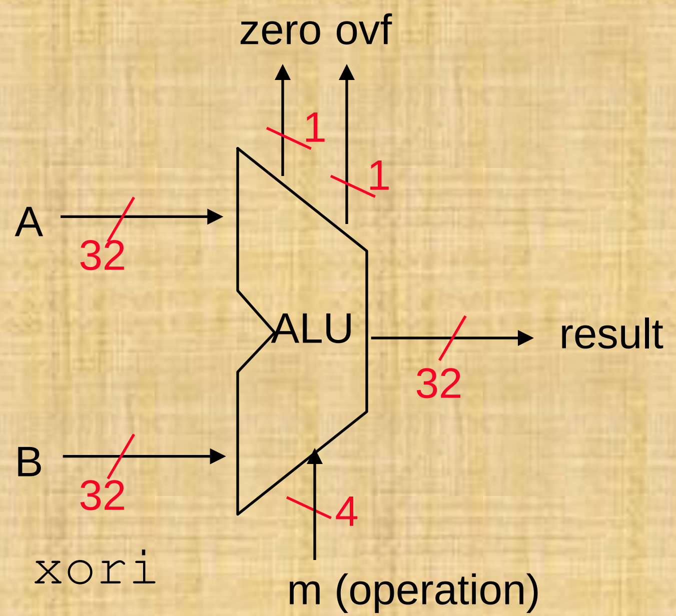
  `sub, subu, neg`

  `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 Addition

- Example: 7 + 6

	(0)	*(0)*	*(1)*	*(1)*	*(0)*	*(Carries)*
. . .	0	0	0	1	1	1
. . .	0	0	0	1	1	0
. . .	(0) 0	(0) 0	(0) 1	(1) 1	(1) 0	(0) 1

- Overflow if result out of range
  - Adding +ve and –ve operands, no overflow
  - Adding two +ve operands
    - Overflow if result sign is 1
  - Adding two –ve operands
    - Overflow if result sign is 0

Integer Subtraction

- Add negation of second operand
- Example: 7 – 6 = 7 + (–6)

+7:	0000 0000 … 0000 0111
–6:	1111 1111 … 1111 1010
+1:	0000 0000 … 0000 0001

- 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

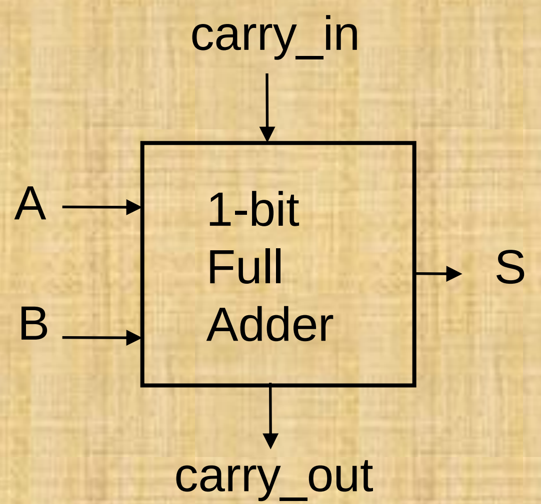
Dealing with Overflow

- 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

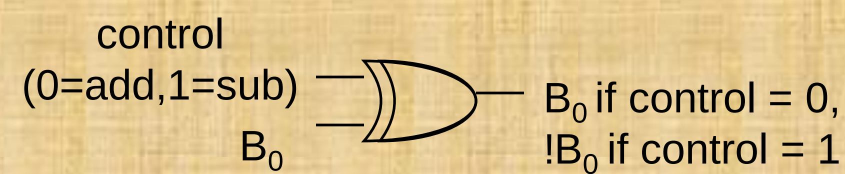
S = A $\oplus$ B $\oplus$ carry_in (odd parity function)

carry_out = A&B | A&carry_in | B&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?

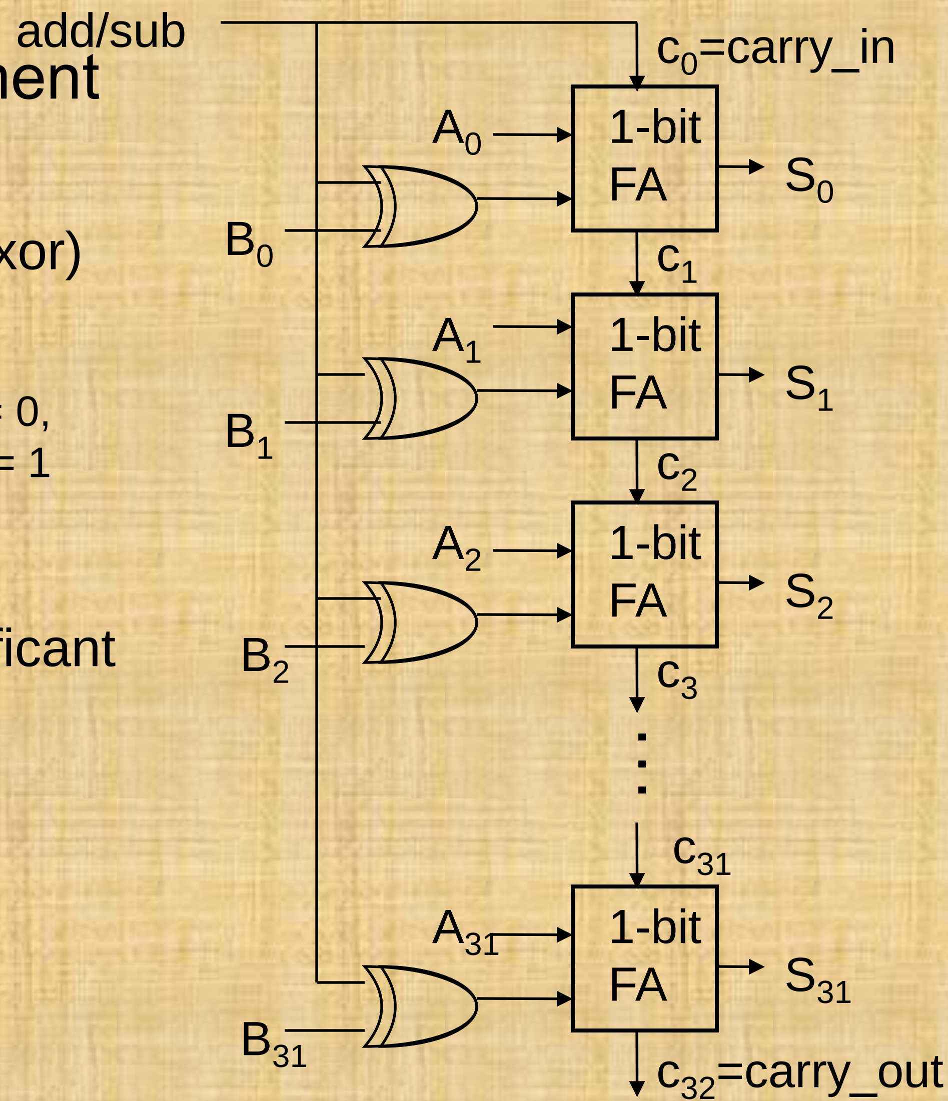
A 32-bit Ripple Carry Adder/Subtractor

- Remember 2's complement is just
  - complement all the bits (xor)



  - add a 1 in the least significant bit

```
A     0111   →     0111
B   - 0110   →   + 1001
    ------            1
      0001        -----
                 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

0	1	1	1		
	0	1	1	1	7
+	0	0	1	1	3
	1	0	1	0	– 6

1	0				
	1	1	0	0	–4
+	1	0	1	1	– 5
	0	1	1	1	7

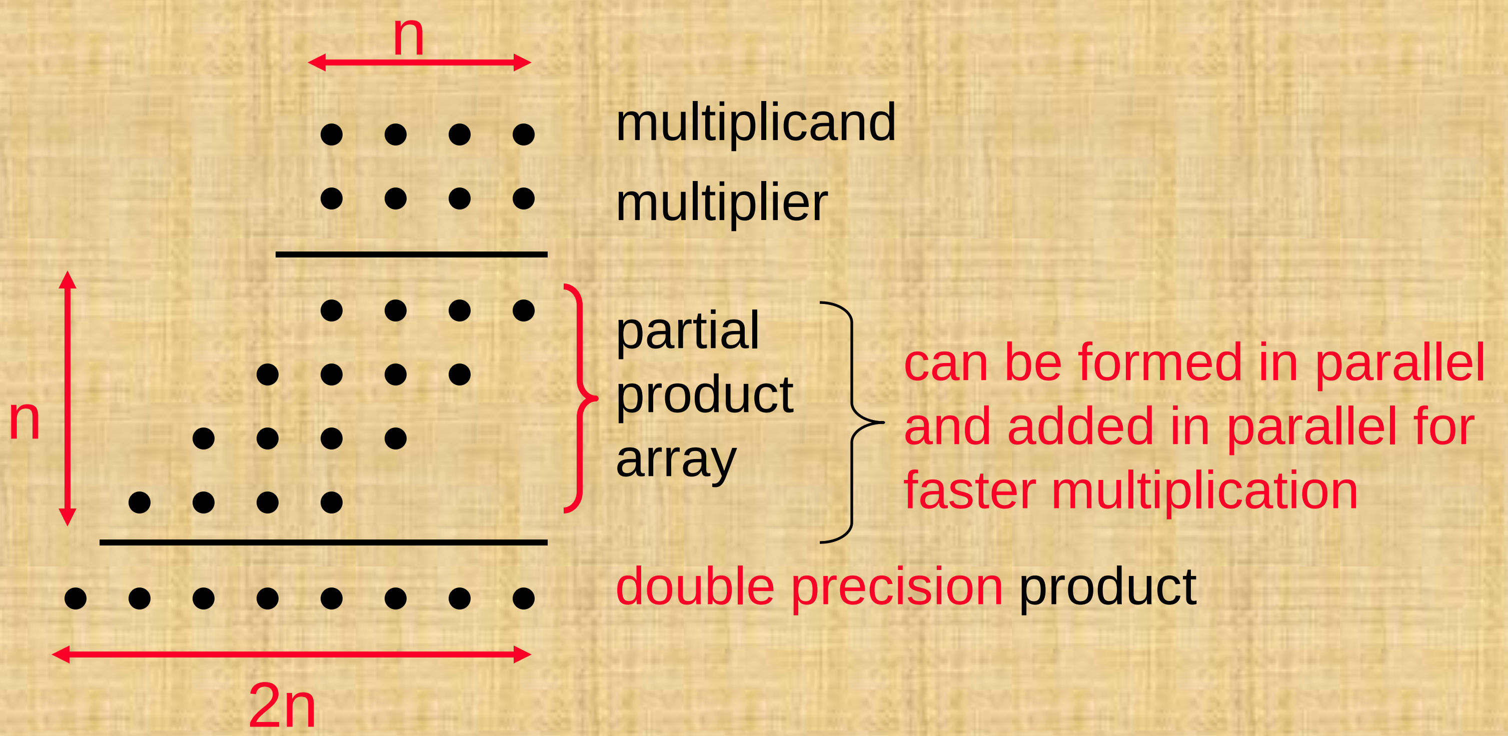
MIPS Overflow detection

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

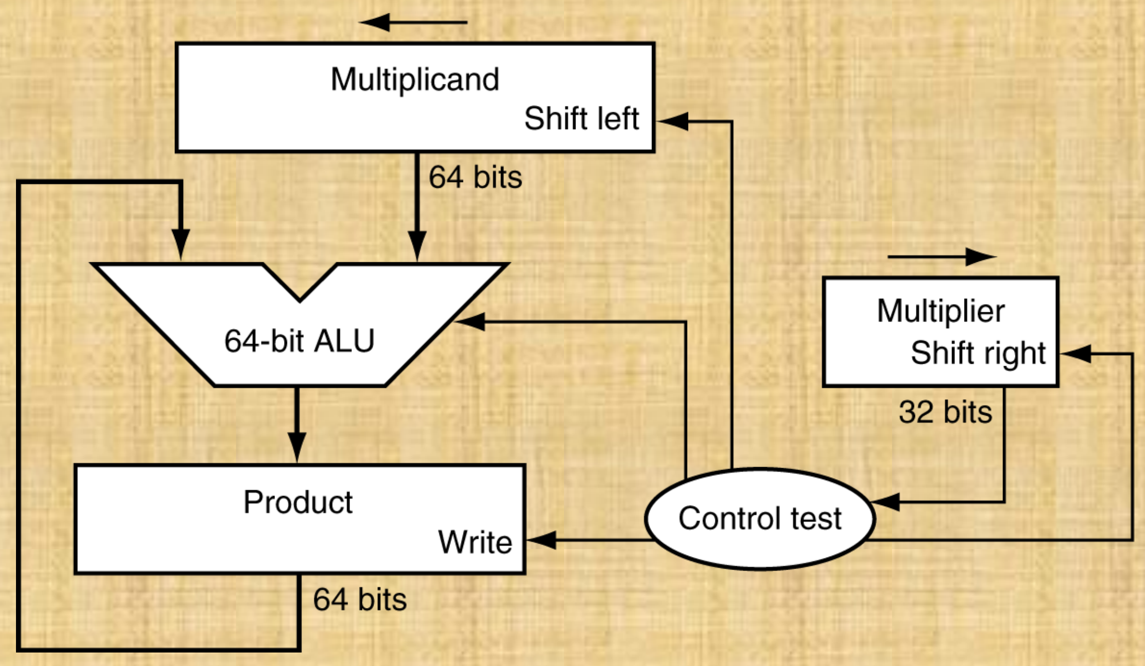
- Start with long-multiplication approach

multiplicand → 1000

multiplier → × 1001

```
      1000
   ×  1001
   -------
      1000
     0000
    0000
   1000
   -------
   1001000
```

product → 1001000



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