Integer Multiplication and Division

ICS 233

Computer Architecture and Assembly Language

Dr. Aiman El-Maleh

College of Computer Sciences and Engineering

King Fahd University of Petroleum and Minerals

Outline

❖ **Unsigned Multiplication**

❖ Signed Multiplication

❖ Faster Multiplication

❖ Unsigned Division

❖ Signed Division

❖ Multiplication and Division in MIPS

Unsigned Multiplication

❖ Paper and Pencil Example:

$$
\begin{array}{r}
\text{Multiplicand} \qquad 1100_2 = 12 \\
\text{Multiplier} \qquad \times \quad 1101_2 = 13 \\
\hline
1100 \\
0000 \\
1100 \\
1100 \\
\hline
\text{Product} \qquad 10011100_2 = 156
\end{array}
$$

Binary multiplication is easy

$0 \times$ multiplicand = 0

$1 \times$ multiplicand = multiplicand

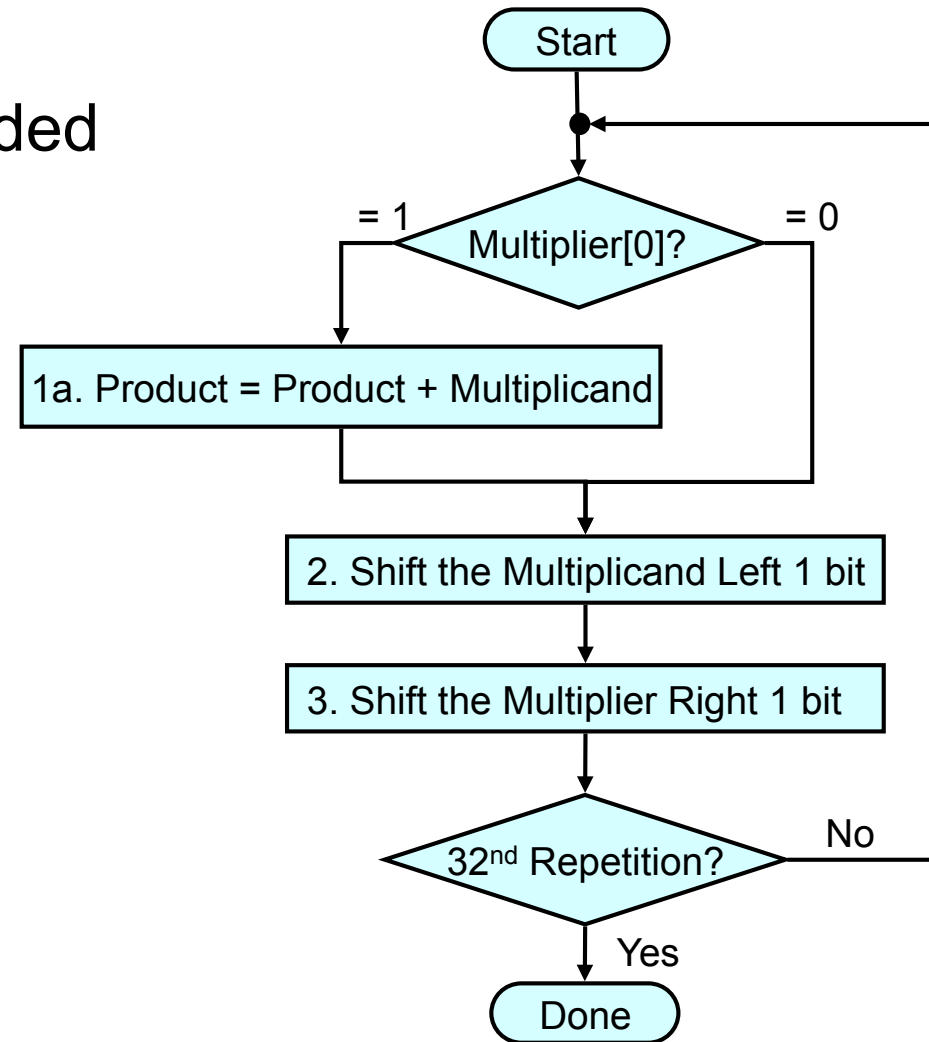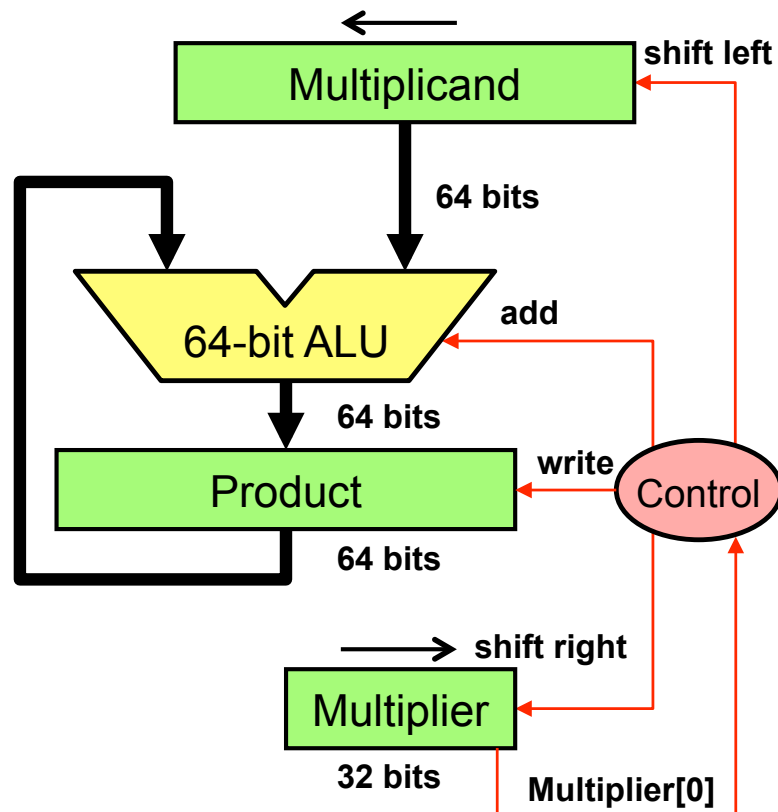❖ m-bit multiplicand × n-bit multiplier = (m+n)-bit product

❖ Accomplished via shifting and addition

Version 1 of Multiplication Hardware

❖ Initialize Product = 0

❖ Multiplicand is zero extended

shift left

Multiplicand

64 bits

64-bit ALU   **add**

64 bits

Product   **write**   Control

64 bits

shift right

Multiplier

32 bits   **Multiplier[0]**

Start

Multiplier[0]?   = 1   = 0

1a. Product = Product + Multiplicand

2. Shift the Multiplicand Left 1 bit

3. Shift the Multiplier Right 1 bit

32nd Repetition?   No

Yes

Done

Multiplication Example (Version 1)

❖ Consider: $1100_2 \times 1101_2$, Product = 10011100_2

❖ 4-bit multiplicand and multiplier are used in this example

❖ Multiplicand is zero extended because it is unsigned

Iteration		Multiplicand	Multiplier	Product
0	Initialize	00001100	1101	00000000
1	Multiplier[0] = 1 => ADD			00001100
	SLL Multiplicand and SRL Multiplier	00011000	0110	
2	Multiplier[0] = 0 => Do Nothing			00001100
	SLL Multiplicand and SRL Multiplier	00110000	0011	
3	Multiplier[0] = 1 => ADD			00111100
	SLL Multiplicand and SRL Multiplier	01100000	0001	
4	Multiplier[0] = 1 => ADD			10011100
	SLL Multiplicand and SRL Multiplier	11000000	0000	

Observation on Version 1 of Multiply

❖ Hardware in version 1 can be optimized
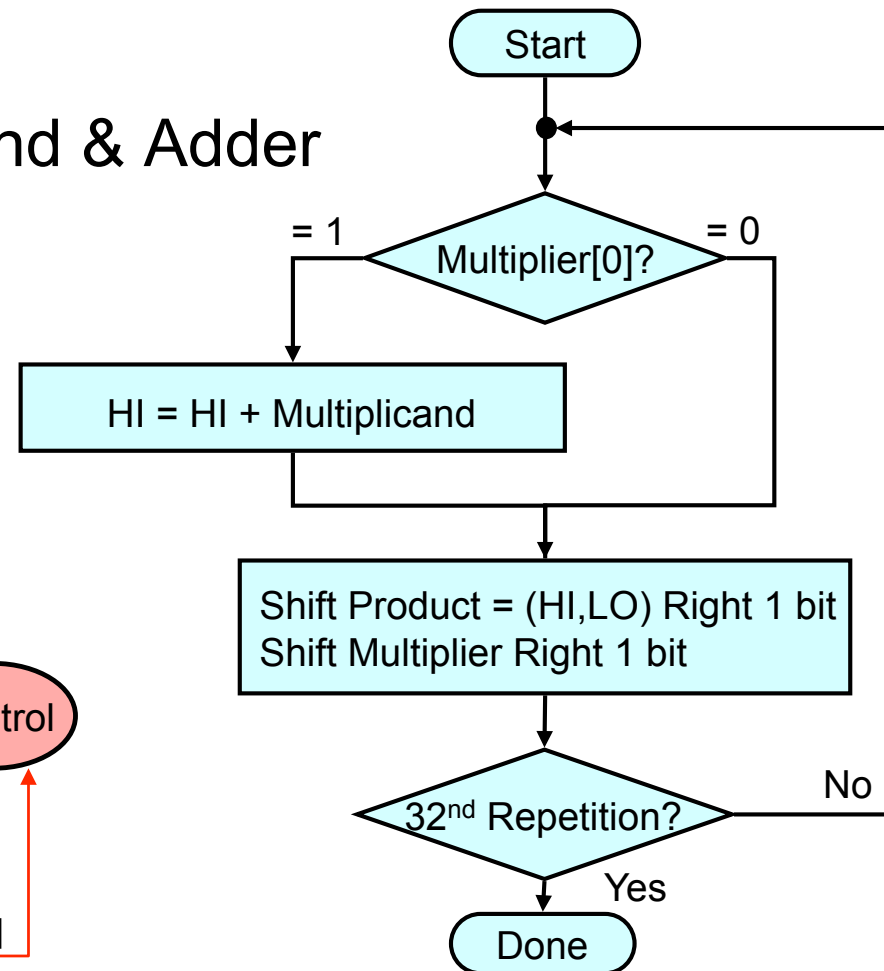
❖ Rather than shifting the multiplicand to the left
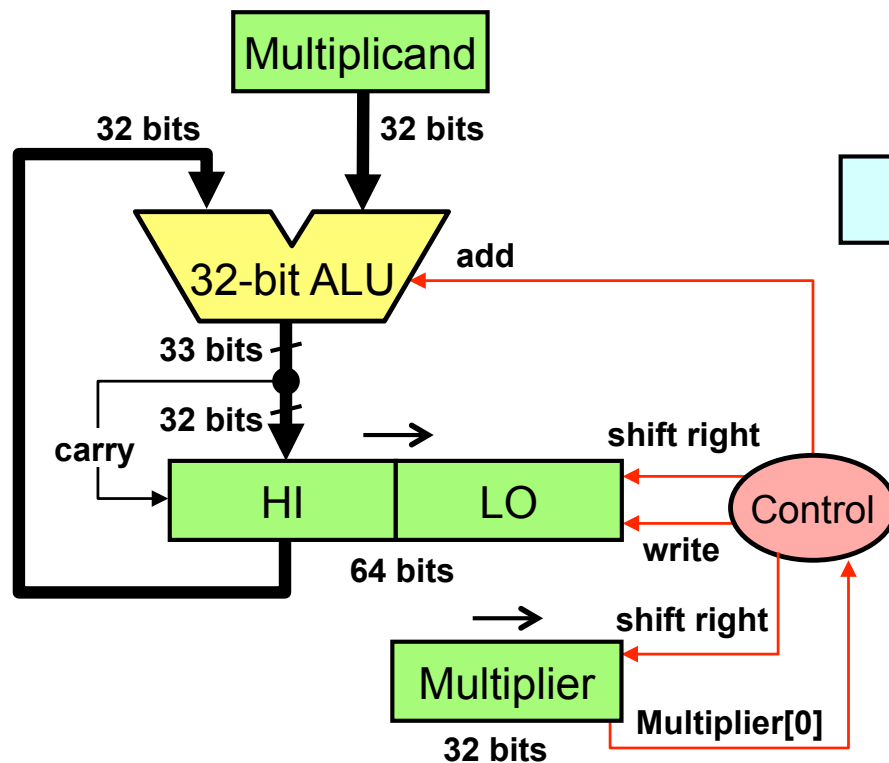
Instead, shift the product to the right

Has the same net effect and produces the same results

❖ Reduce Hardware

◇ Multiplicand register can be reduced to 32 bits only

◇ We can also reduce the adder size to 32 bits

❖ One cycle per iteration
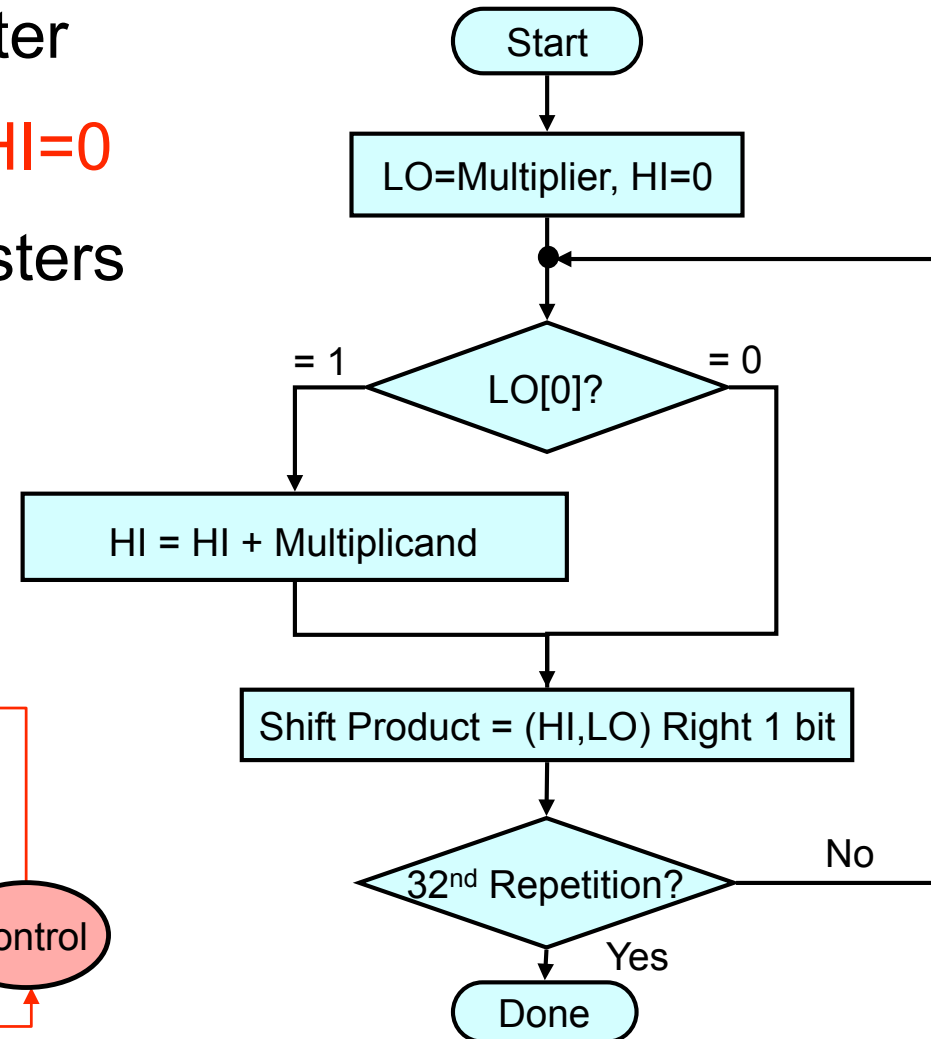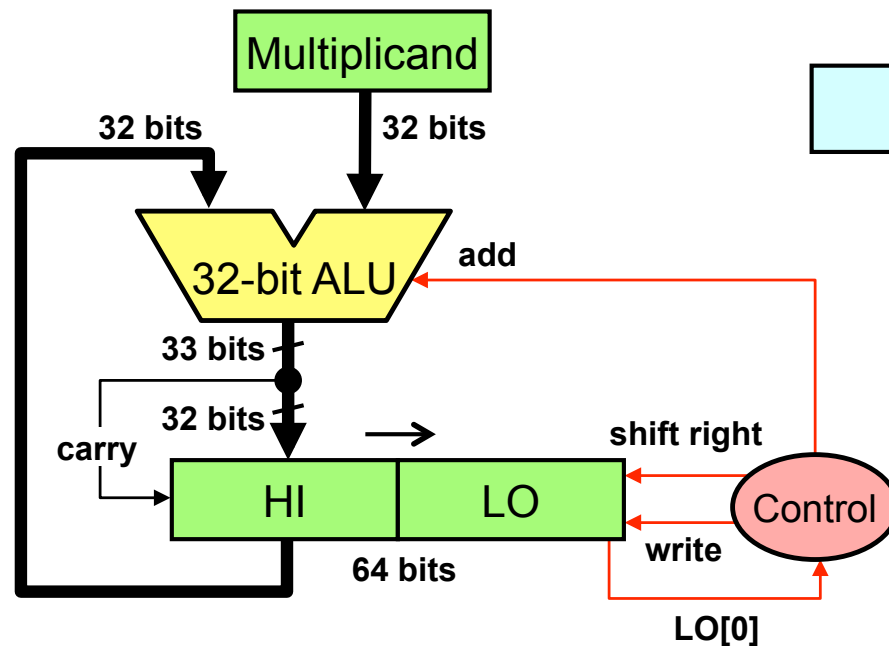
◇ Shifting and addition can be done simultaneously

Version 2 of Multiplication Hardware

❖ Product = HI and LO registers, HI=0

❖ Product is shifted right

❖ Reduced 32-bit Multiplicand & Adder

Multicand

32 bits   32 bits

32-bit ALU   **add**

33 bits

32 bits

carry   **shift right**

HI   LO   Control

64 bits   **write**

shift right

Multiplier   **Multiplier[0]**

32 bits

Start

Multiplier[0]?   = 1   = 0

HI = HI + Multiplicand

Shift Product = (HI,LO) Right 1 bit
Shift Multiplier Right 1 bit

32nd Repetition?   No

Yes

Done

Refined Version of Multiply Hardware

❖ Eliminate Multiplier Register

❖ Initialize LO = Multiplier, HI=0

❖ Product = HI and LO registers

Multiplicand

32 bits | 32 bits

32-bit ALU — add

33 bits

32 bits

carry

shift right

HI | LO

64 bits

write

Control

LO[0]

Start

LO=Multiplier, HI=0

= 1 ← LO[0]? → = 0

HI = HI + Multiplicand

Shift Product = (HI,LO) Right 1 bit

32nd Repetition? → No

Yes

Done

Multiply Example (Refined Version)

❖ Consider: $1100_2 \times 1101_2$, Product = 10011100_2

❖ 4-bit multiplicand and multiplier are used in this example

❖ 4-bit adder produces a 5-bit sum (with carry)

Iteration		Multiplicand	Carry	Product = HI, LO
0	Initialize (LO = Multiplier, HI=0)	1 1 0 0		0 0 0 0 1 1 0 1
1	LO[0] = 1 => ADD		0	1 1 0 0 1 1 0 1
	Shift Right Product = (HI, LO)	1 1 0 0		0 1 1 0 0 1 1 0
2	LO[0] = 0 => Do Nothing			
	Shift Right Product = (HI, LO)	1 1 0 0		0 0 1 1 0 0 1 1
3	LO[0] = 1 => ADD		0	1 1 1 1 0 0 1 1
	Shift Right Product = (HI, LO)	1 1 0 0		0 1 1 1 1 0 0 1
4	LO[0] = 1 => ADD		1	0 0 1 1 1 0 0 1
	Shift Right Product = (HI, LO)	1 1 0 0		1 0 0 1 1 1 0 0

Next . . .

❖ Unsigned Multiplication

❖ Signed Multiplication

❖ Faster Multiplication

❖ Unsigned Division

❖ Signed Division

❖ Multiplication and Division in MIPS

Signed Multiplication

❖ So far, we have dealt with unsigned integer multiplication

❖ Version 1 of Signed Multiplication

  ◈ Convert multiplier and multiplicand into positive numbers

      ▪ If negative then obtain the 2's complement and remember the sign

  ◈ Perform unsigned multiplication

  ◈ Compute the sign of the product

  ◈ If product sign < 0 then obtain the 2's complement of the product

❖ Refined Version:

  ◈ Use the refined version of the unsigned multiplication hardware

  ◈ When shifting right, extend the sign of the product

  ◈ If multiplier is negative, the last step should be a subtract

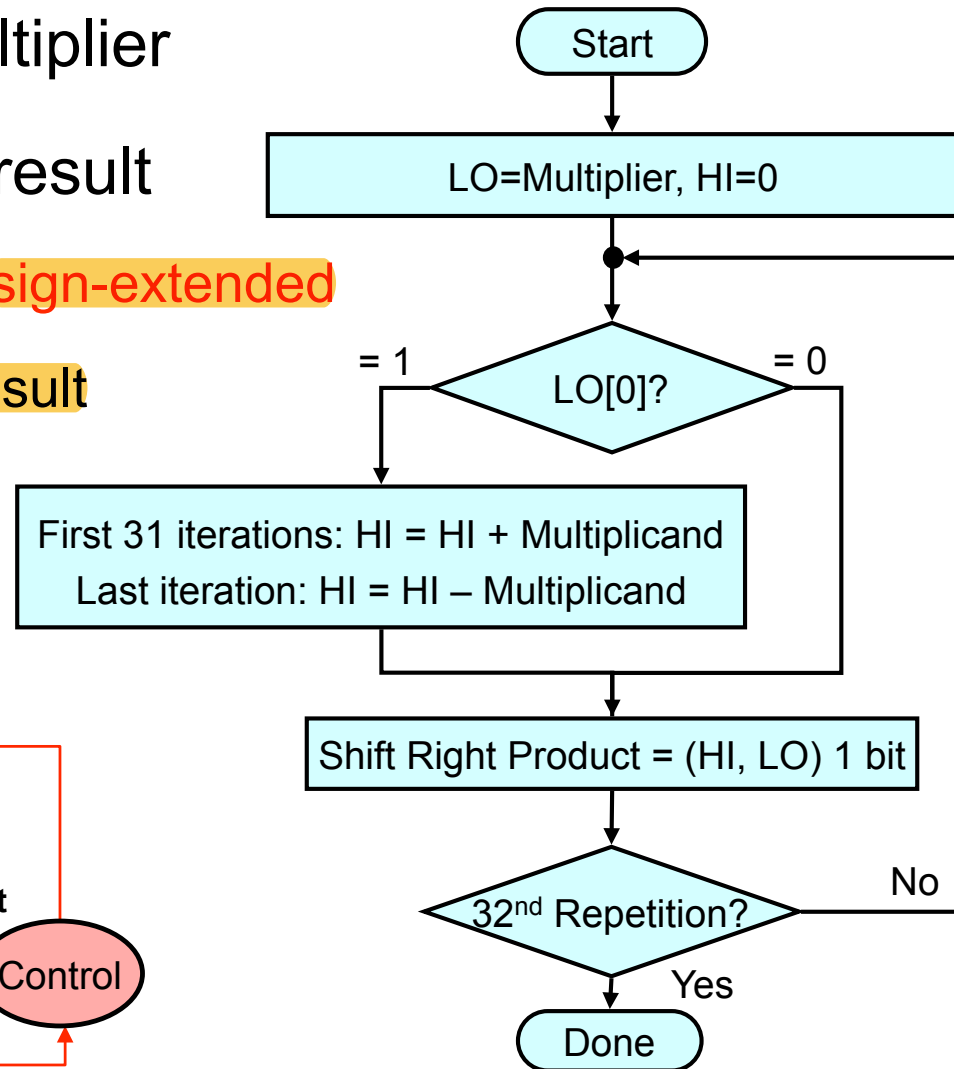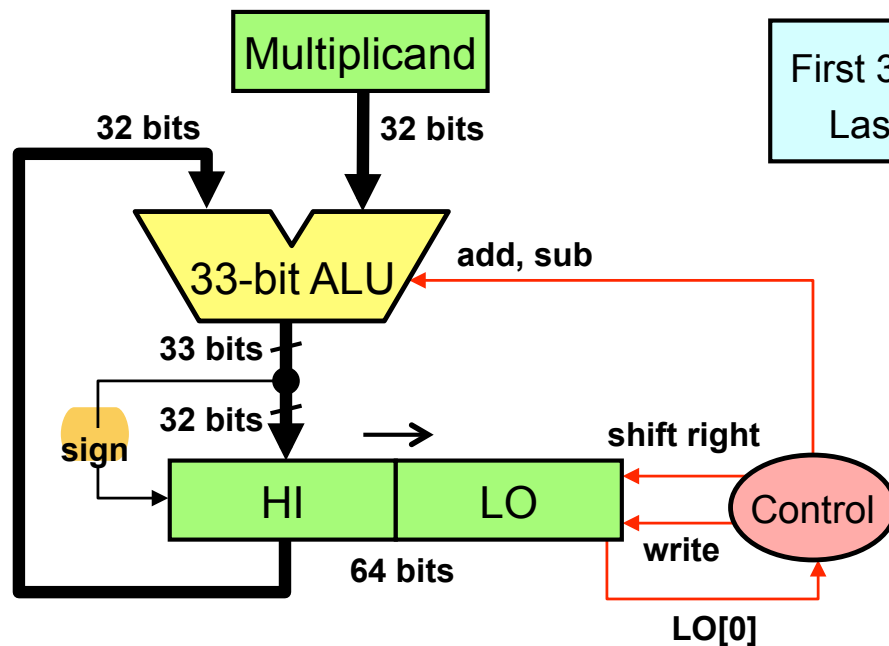# Signed Multiplication (Pencil & Paper)

❖ Case 1: Positive Multiplier

Multiplicand	1100_2	$= -4$
Multiplier	$\times \quad 0101_2$	$= +5$

Sign-extension ⟶ $\boxed{1111}1100$
⟶ $\boxed{11}1100$

Product $\qquad 11101100_2 = -20$

❖ Case 2: Negative Multiplier

Multiplicand	1100_2	$= -4$
Multiplier	$\times \quad 1101_2$	$= -3$

Sign-extension ⟶ $\boxed{1111}1100$
⟶ $\boxed{11}1100$
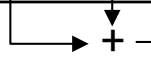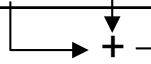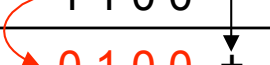00100 **(2's complement of 1100)**

Product $\qquad 00001100_2 = +12$

Signed Multiplication Hardware

❖ Similar to Unsigned Multiplier

❖ ALU produces a 33-bit result

    ✧ Multiplicand and HI are sign-extended

    ✧ Sign is the sign of the result

Start

LO=Multiplier, HI=0

LO[0]?

= 1

= 0

First 31 iterations: HI = HI + Multiplicand

Last iteration: HI = HI – Multiplicand

Shift Right Product = (HI, LO) 1 bit

32nd Repetition?

No

Yes

Done

Multiplicand

32 bits

32 bits

33-bit ALU

add, sub

33 bits

32 bits

sign

shift right

HI   LO

Control

write

64 bits

LO[0]

Signed Multiplication Example

❖ Consider: 1100_2 (-4) × 1101_2 (-3), Product = 00001100_2

❖ Multiplicand and HI are sign-extended before addition

❖ Last iteration: add 2's complement of Multiplicand

Iteration		Multiplicand	Sign	Product = HI, LO
0	Initialize (LO = Multiplier)	1 1 0 0		0 0 0 0 1 1 0 1
1	LO[0] = 1 => ADD		1	1 1 0 0 1 1 0 1
1	Shift Product = (HI, LO) right 1 bit	1 1 0 0		1 1 1 0 0 1 1 0
2	LO[0] = 0 => Do Nothing			
2	Shift Product = (HI, LO) right 1 bit	1 1 0 0		1 1 1 1 0 0 1 1
3	LO[0] = 1 => ADD		1	1 0 1 1 0 0 1 1
3	Shift Product = (HI, LO) right 1 bit	1 1 0 0		1 1 0 1 1 0 0 1
4	LO[0] = 1 => SUB (ADD 2's compl)	0 1 0 0	0	0 0 0 1 1 0 0 1
4	Shift Product = (HI, LO) right 1 bit			0 0 0 0 1 1 0 0

Next . . .

❖ Unsigned Multiplication

❖ Signed Multiplication

❖ Faster Multiplication

❖ Unsigned Division

❖ Signed Division

❖ Multiplication and Division in MIPS

Faster Multiplication Hardware

❖ **32-bit adder for each bit of the multiplier**

  ✧ 31 adders are needed for a 32-bit multiplier

  ✧ AND multiplicand with each bit of multiplier

  ✧ Product = accumulated shifted sum

❖ **Each adder produces a 33-bit output**

  ✧ Most significant bit is a carry bit

  ✧ Least significant bit is a product bit

  ✧ Upper 32 bits go to next adder

❖ **Array multiplier can be optimized**

  ✧ Carry save adders reduce delays

  ✧ Pipelining further improves the speed
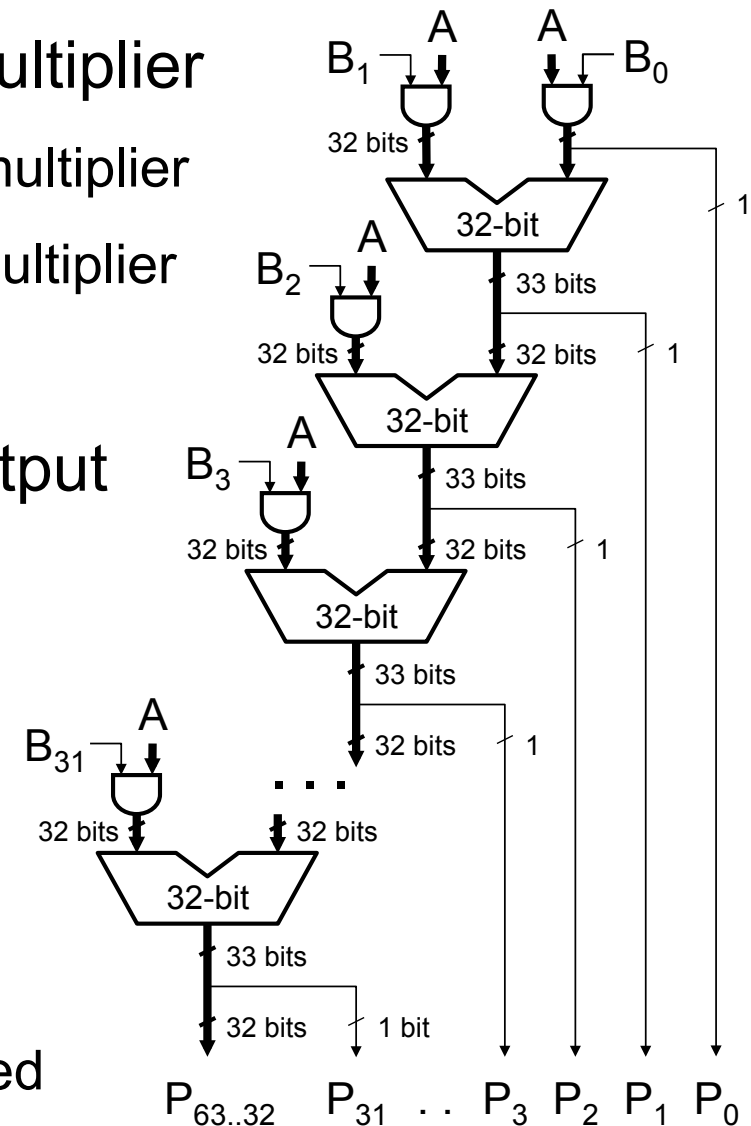
Carry Save Adders

❖ Used when adding multiple numbers (as in multipliers)

❖ All the bits of a carry save adder work in parallel

  ✧ The carry does not propagate as in a ripple-carry adder

  ✧ This is why the carry save adder is much faster than ripple-carry

❖ A carry save adder has 3 inputs and produces two outputs

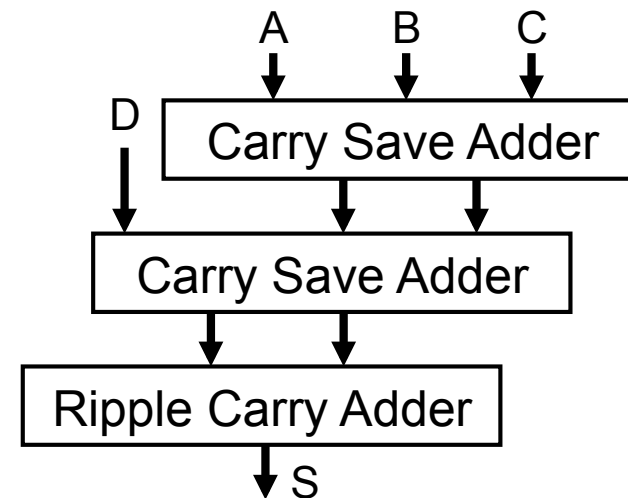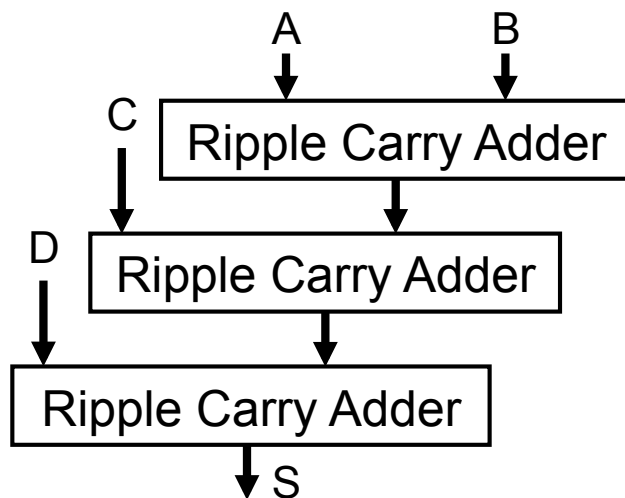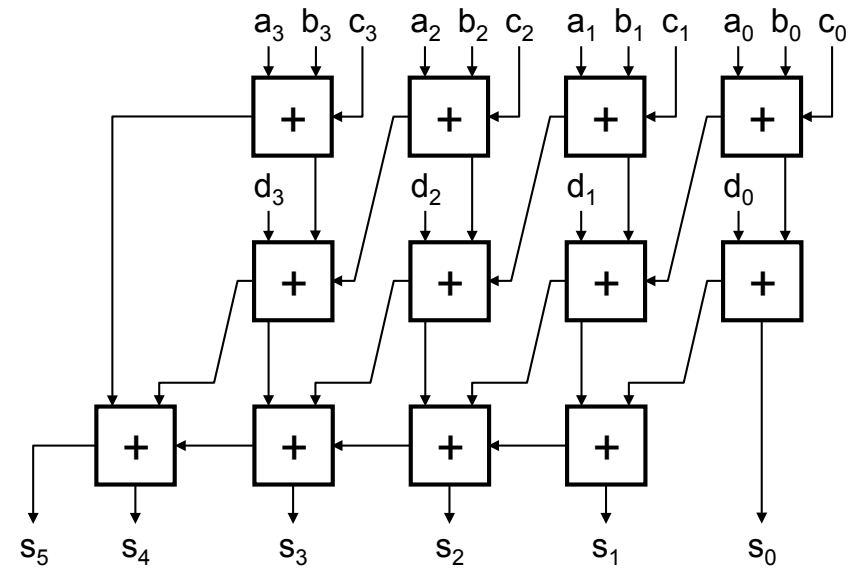 ✧ It adds 3 numbers and produces partial sum and carry bits
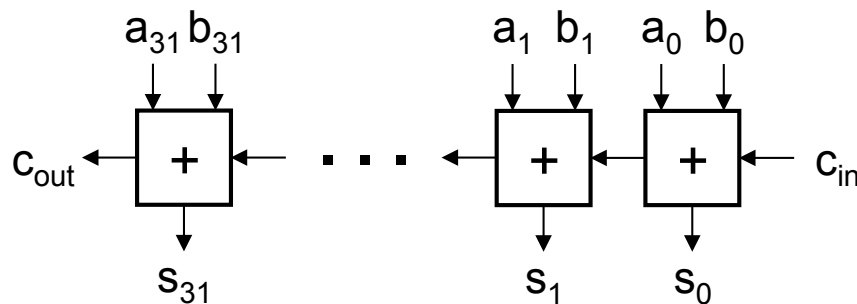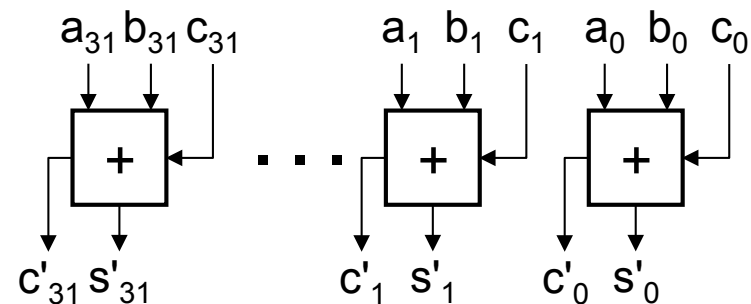
Ripple Carry Adder   Carry Save Adder

Consider Adding: S = A + B + C + D
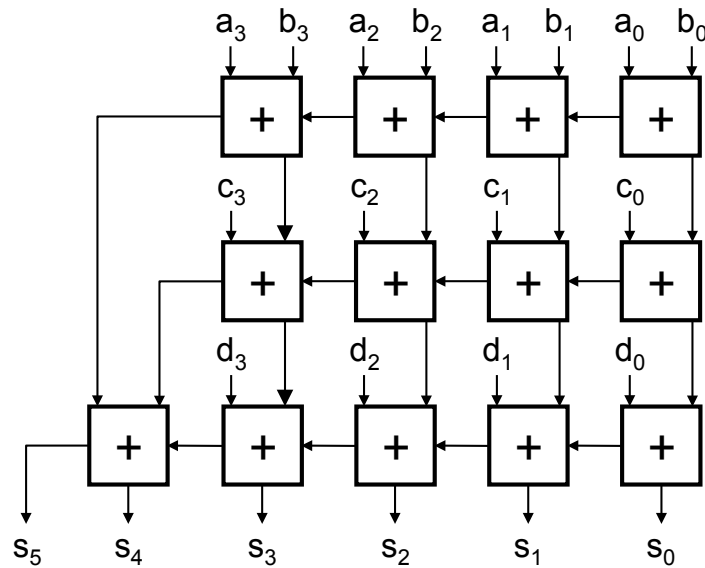
Next . . .

❖ Unsigned Multiplication

❖ Signed Multiplication

❖ Faster Multiplication

❖ Unsigned Division

❖ Signed Division

❖ Multiplication and Division in MIPS

Unsigned Division (Paper & Pencil)

$$10011_2 = 19 \quad \text{Quotient}$$

Divisor $\quad 1011_2 \quad | \quad 11011001_2 = 217 \quad$ Dividend

$$-1011$$

$$10$$
$$101$$
$$1010$$
$$10100$$
$$-1011$$

$$1001$$
$$10011$$
$$-1011$$

$$1000_2 = 8 \quad \text{Remainder}$$

Try to see how big a number can be subtracted, creating a digit of the quotient on each attempt

Binary division is accomplished via shifting and subtraction

Dividend =

Quotient × Divisor

+ Remainder

217 = 19 × 11 + 8

First Division Algorithm & Hardware

❖ Initialize:

  ◇ Remainder = Dividend (0-extended)

  ◇ Load Upper 32 bits of Divisor

  ◇ Quotient = 0

shift right

Divisor

64 bits

64-bit ALU

sub

Difference

sign

Remainder

write

Control

64 bits

shift left

Quotient

set lsb

32 bits

Start

1. Shift the Divisor Right 1 bit
   Shift the Quotient Left 1 bit
   Difference = Remainder – Divisor

≥ 0   Difference?   < 0

2. Remainder = Difference
   Set least significant bit of Quotient

32nd Repetition?   No

Yes

Done
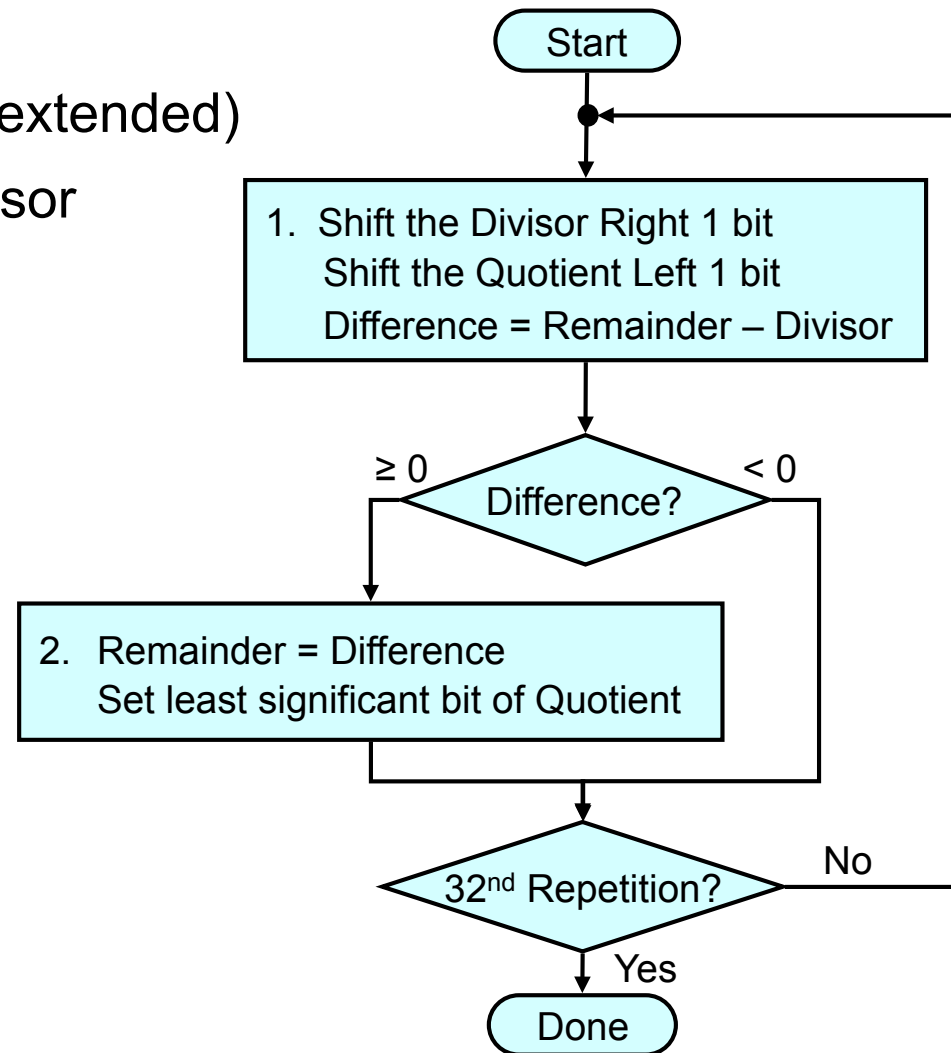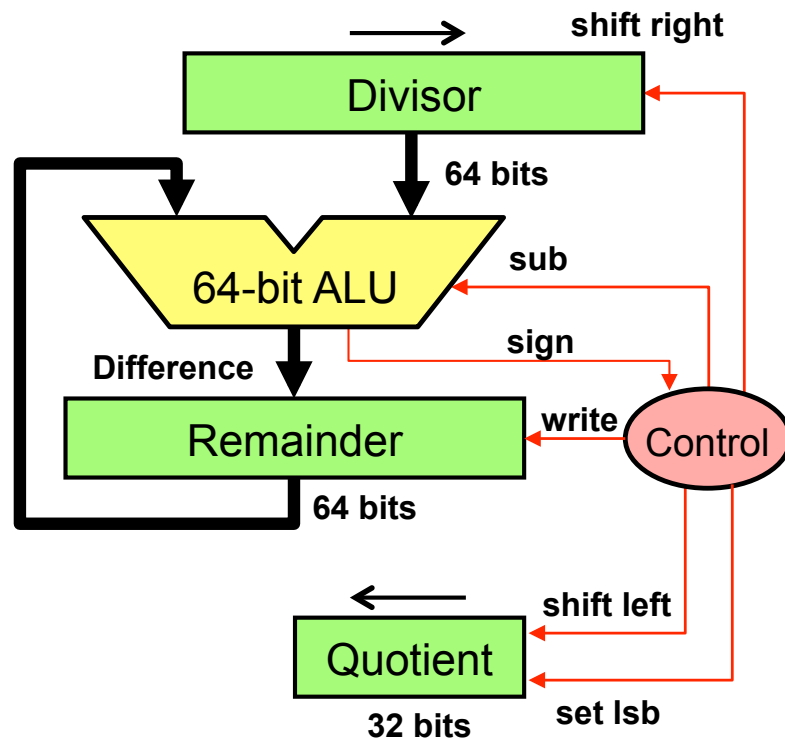
Division Example (Version 1)

❖ Consider: 1110_2 / 0011_2 (4-bit dividend & divisor)

❖ Quotient = 0100_2 and Remainder = 0010_2

❖ 8-bit registers for Remainder and Divisor (8-bit ALU)

Iteration		Remainder	Divisor	Difference	Quotient
0	Initialize	0000**1110**	**0011**0000		0000
1	1: SRL, SLL, Difference	00001110	00011000	11110110	0000
	2: Diff < 0 => Do Nothing				
2	1: SRL, SLL, Difference	00001110	00001100	00000010	0000
	2: Rem = Diff, set lsb Quotient	00000010			000**1**
3	1: SRL, SLL, Difference	00000010	00000110	11111100	0010
	2: Diff < 0 => Do Nothing				
4	1: SRL, SLL, Difference	00000010	00000011	11111111	0100
	2: Diff < 0 => Do Nothing				

Observations on Version 1 of Divide

❖ Version 1 of Division hardware can be optimized

❖ Instead of shifting divisor right,

   Shift the remainder register left

   Has the same net effect and produces the same results

❖ Reduce Hardware:

   ✧ Divisor register can be reduced to 32 bits (instead of 64 bits)

   ✧ ALU can be reduced to 32 bits (instead of 64 bits)

   ✧ Remainder and Quotient registers can be combined

Refined Division Hardware

❖ Observation:

  ✧ Shifting remainder left does the same as shifting the divisor right

❖ Initialize:

  ✧ Quotient = Dividend, Remainder = 0

Start

1. Shift (Remainder, Quotient) Left
   Difference = Remainder – Divisor

≥ 0   Difference?   < 0

2. Remainder = Difference
   Set least significant bit of Quotient

32nd Repetition?   No

Yes

Done

Divisor

32 bits

32-bit ALU

sub

sign

Difference

write

Remainder   Quotient   Control

shift left

32 bits   32 bits

set lsb
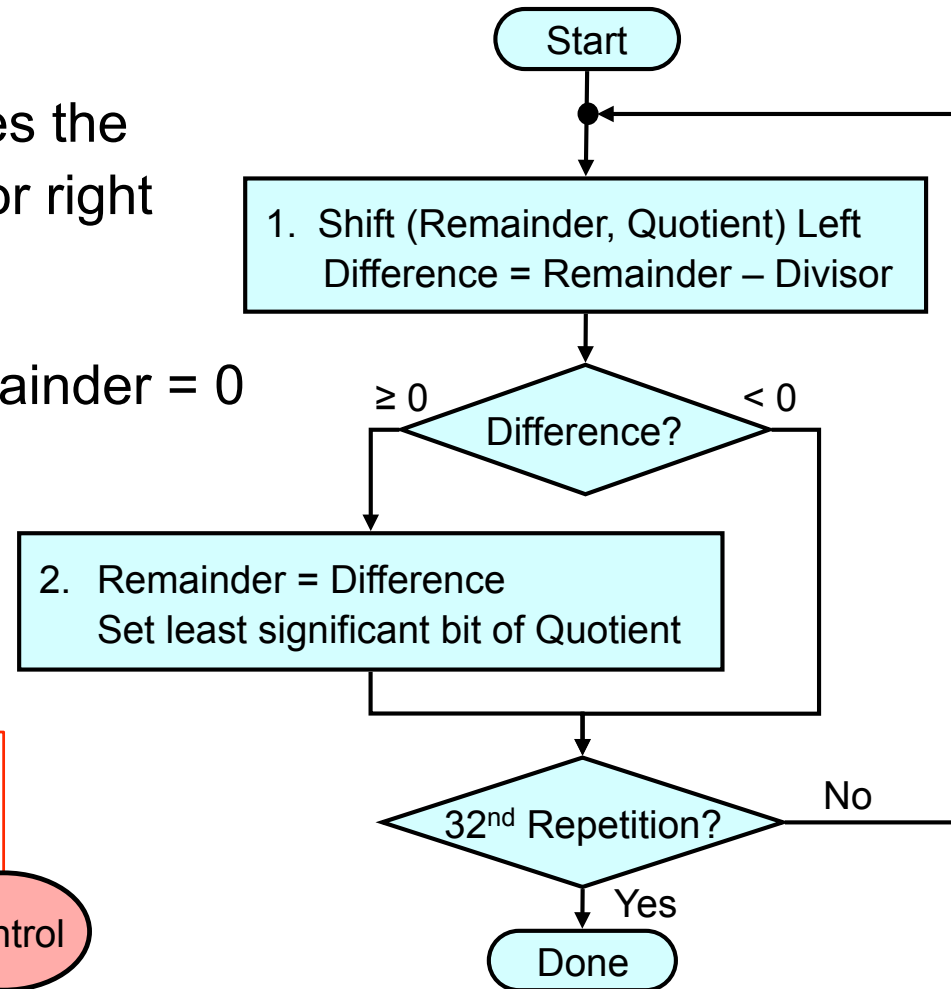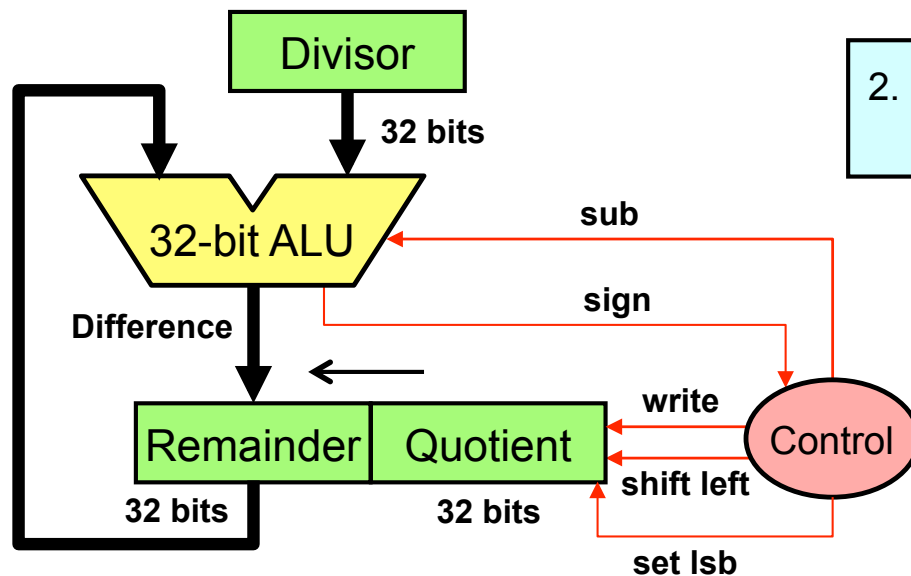
Division Example (Refined Version)

❖ Same Example: $1110_2 / 0011_2$ (4-bit dividend & divisor)

❖ Quotient = 0100_2 and Remainder = 0010_2

❖ 4-bit registers for Remainder and Divisor (4-bit ALU)

Iteration		Remainder	Quotient	Divisor	Difference
0	Initialize	0 0 0 0	1 1 1 0	0 0 1 1	
1	1: SLL, Difference	0 0 0 1 ←	1 1 0 0	0 0 1 1	1 1 1 0
	2: Diff < 0 => Do Nothing				
2	1: SLL, Difference	0 0 1 1 ←	1 0 0 0	0 0 1 1	0 0 0 0
	2: Rem = Diff, set lsb Quotient	0 0 0 0	1 0 0 1		
3	1: SLL, Difference	0 0 0 1 ←	0 0 1 0	0 0 1 1	1 1 1 0
	2: Diff < 0 => Do Nothing				
4	1: SLL, Difference	0 0 1 0 ←	0 1 0 0	0 0 1 1	1 1 1 1
	2: Diff < 0 => Do Nothing				

Next . . .

❖ Unsigned Multiplication

❖ Signed Multiplication

❖ Faster Multiplication

❖ Unsigned Division

❖ Signed Division

❖ Multiplication and Division in MIPS

Signed Division

❖ Simplest way is to remember the signs

❖ Convert the dividend and divisor to positive

   ◇ Obtain the 2's complement if they are negative

❖ Do the unsigned division

❖ Compute the signs of the quotient and remainder

   ◇ Quotient sign = Dividend sign XOR Divisor sign

   ◇ Remainder sign = Dividend sign

❖ Negate the quotient and remainder if their sign is negative

   ◇ Obtain the 2's complement to convert them to negative

Signed Division Examples

1. **Positive** Dividend and **Positive** Divisor

    ✧ Example: +17 / +3   Quotient = +5   Remainder = +2

2. **Positive** Dividend and **Negative** Divisor

    ✧ Example: +17 / –3   Quotient = –5   Remainder = +2

3. **Negative** Dividend and **Positive** Divisor

    ✧ Example: –17 / +3   Quotient = –5   Remainder = –2

4. **Negative** Dividend and **Negative** Divisor

    ✧ Example: –17 / –3   Quotient = +5   Remainder = –2

The following equation must always hold:

Dividend = Quotient × Divisor + Remainder

Next . . .

❖ Unsigned Multiplication

❖ Signed Multiplication

❖ Faster Multiplication

❖ Unsigned Division

❖ Signed Division

❖ Multiplication and Division in MIPS

Multiplication in MIPS

❖ Two Multiply instructions

✦ `mult  $s1,$s2`   **Signed** **multiplication**

✦ `multu $s1,$s2`   **Unsigned multiplication**

❖ 32-bit multiplication produces a 64-bit Product

❖ Separate pair of 32-bit registers

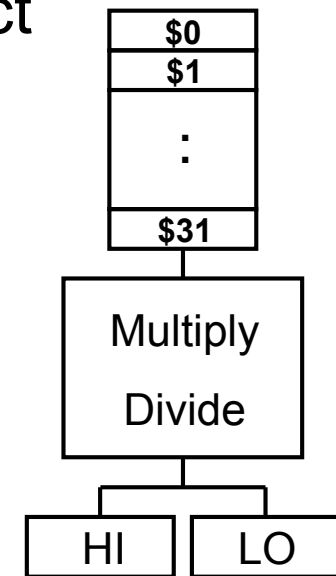✦ **HI = high-order 32-bit**

✦ **LO = low-order 32-bit**

✦ Result of multiplication is always in HI & LO

❖ Moving data from HI/LO to MIPS registers

✦ `mfhi Rd` (move from HI to Rd)

✦ `mflo Rd` (move from LO to Rd)

| $0 |
| $1 |
| : |
| $31 |

Multiply

Divide

| HI | LO |

Division in MIPS

❖ **Two Divide instructions**

  ⬦ `div  $s1,$s2`   **Signed division**

  ⬦ `divu $s1,$s2`   **Unsigned division**

❖ **Division produces quotient and remainder**

❖ **Separate pair of 32-bit registers**
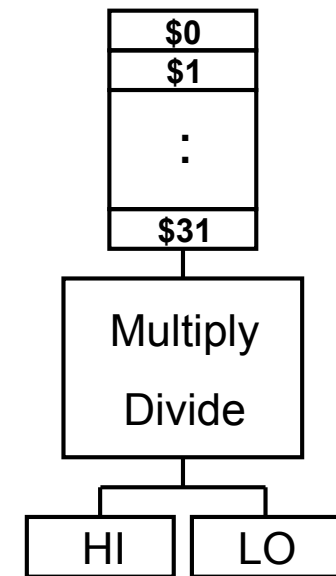
  ⬦ **HI = 32-bit remainder**

  ⬦ **LO = 32-bit quotient**

  ⬦ If divisor is 0 then result is **unpredictable**

❖ **Moving data to HI/LO from MIPS registers**

  ⬦ `mthi Rs` (move to HI from Rs)

  ⬦ `mtlo Rs` (move to LO from Rs)

Integer Multiply/Divide Instructions

Instruction	Meaning	Format					
mult Rs, Rt	Hi, Lo = Rs × Rt	$op^6 = 0$	Rs^5	Rt^5	0	0	0x18
multu Rs, Rt	Hi, Lo = Rs × Rt	$op^6 = 0$	Rs^5	Rt^5	0	0	0x19
div Rs, Rt	Hi, Lo = Rs / Rt	$op^6 = 0$	Rs^5	Rt^5	0	0	0x1a
divu Rs, Rt	Hi, Lo = Rs / Rt	$op^6 = 0$	Rs^5	Rt^5	0	0	0x1b
mfhi Rd	Rd = Hi	$op^6 = 0$	0	0	Rd^5	0	0x10
mflo Rd	Rd = Lo	$op^6 = 0$	0	0	Rd^5	0	0x12
mthi Rs	Hi = Rs	$op^6 = 0$	Rs^5	0	0	0	0x11
mtlo Rs	Lo = Rs	$op^6 = 0$	Rs^5	0	0	0	0x13

❖ Signed arithmetic: mult, div (Rs and Rt are signed)

✧ LO = 32-bit low-order and HI = 32-bit high-order of multiplication

✧ LO = 32-bit quotient and HI = 32-bit remainder of division

❖ Unsigned arithmetic: multu, divu (Rs and Rt are unsigned)

❖ NO arithmetic exception can occur

Integer to String Conversion

❖ Objective: convert an unsigned 32-bit integer to a string

❖ How to obtain the decimal digits of the number?

  ◇ Divide the number by 10, Remainder = decimal digit (0 to 9)

  ◇ Convert decimal digit into its ASCII representation ('0' to '9')

  ◇ Repeat the division until the quotient becomes zero

  ◇ Digits are computed backwards from least to most significant

❖ Example: convert 2037 to a string

  ◇ Divide 2037/10   quotient = 203   remainder = 7   char = '7'

  ◇ Divide 203/10   quotient = 20   remainder = 3   char = '3'

  ◇ Divide 20/10   quotient = 2   remainder = 0   char = '0'

  ◇ Divide 2/10   quotient = 0   remainder = 2   char = '2'

Integer to String Procedure

```
#----------------------------------------------------
# int2str:    Converts an unsigned integer into a string
# Parameters: $a0 = integer to be converted
#             $a1 = string pointer (can store 10 digits)
#----------------------------------------------------
int2str:
    move   $t0, $a0          # $t0 = dividend = integer value
    li     $t1, 10           # $t1 = divisor = 10
    addiu  $a1, $a1, 10      # start at end of string
    sb     $zero, 0($a1)     # store a NULL byte
convert:
    divu   $t0, $t1          # LO  = quotient, HI = remainder
    mflo   $t0              # $t0 = quotient
    mfhi   $t2              # $t2 = remainder
    ori    $t2, $t2, 0x30    # convert digit to a character
    addiu  $a1, $a1, -1      # point to previous char
    sb     $t2, 0($a1)       # store digit character
    bnez   $t0, convert      # loop if quotient is not 0
    jr     $ra
```