Integer Multiplication and Division

COE 301

Computer Organization

Prof. Muhamed Mudawar

College of Computer Sciences and Engineering

King Fahd University of Petroleum and Minerals

Presentation Outline

- Unsigned Integer Multiplication
- Signed Integer Multiplication
- Faster Integer Multiplication
- Integer Division
- Integer Multiplication and Division in MIPS

Unsigned Integer Multiplication

- Paper and Pencil Example:

Multiplicand		$1100_2 = 12$
Multiplier	×	$1101_2 = 13$
		1100
		0000
		1100
		1100
Product		$10011100_2 = 156$

Binary multiplication is easy
0 × multiplicand = 0
1 × multiplicand = multiplicand

- m-bit multiplicand × n-bit multiplier = (m+n)-bit product
- Accomplished via shifting and addition
- Consumes more time and more chip area than addition

Sequential Unsigned Multiplication

- Initialize Product = 0
- Check each bit of the Multiplier
- If Multiplier bit = 1 then Product = Product + Multiplicand
- Rather than shifting the multiplicand to the left

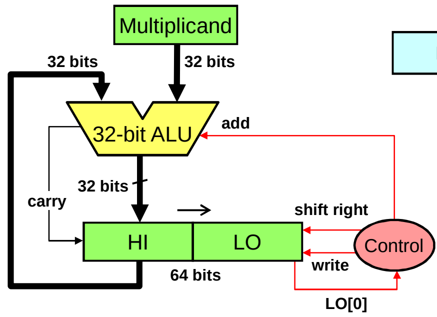
  Instead, Shift the Product to the Right

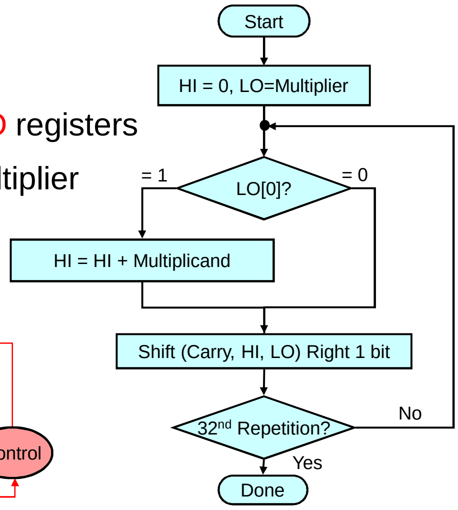
  Has the same net effect and produces the same result

  Minimizes the hardware resources
- One cycle per iteration (for each bit of the Multiplier)
  - Addition and shifting can be done simultaneously

Sequential Multiplication Hardware

- ❖ Initialize HI = 0
- ❖ Initialize LO = Multiplier
- ❖ Final Product = HI and LO registers
- ❖ Repeat for each bit of Multiplier





Sequential Multiplier Example

- ❖ 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 (HI = 0, LO = Multiplier)	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 (Carry, HI, LO) by 1 bit	1 1 0 0		0 1 1 0 0 1 1 0
2	LO[0] = 0 => Do Nothing			
	Shift Right (Carry, HI, LO) by 1 bit	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 (Carry, HI, LO) by 1 bit	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 (Carry, HI, LO) by 1 bit	1 1 0 0		1 0 0 1 1 1 0 0

Next . . .

- ❖ Unsigned Integer Multiplication
- ❖ Signed Integer Multiplication
- ❖ Faster Integer Multiplication
- ❖ Integer Division
- ❖ Integer Multiplication and Division in MIPS

Signed Integer Multiplication

- ❖ So far, we have dealt with unsigned integer multiplication
- ❖ First Attempt:
  - ✧ 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
- ❖ Better Version:
  - ✧ Use 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           1100₂ = -4
Multiplier          ×  0101₂ = +5
                    -------------
Sign-extension      11111100
                    111100
                    -------------
Product             11101100₂ = -20
```

❖ Case 2: Negative Multiplier

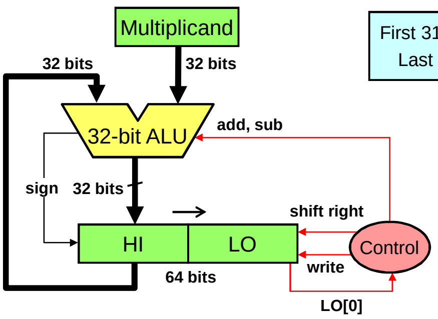
```
Multiplicand           1100₂ = -4
Multiplier          ×  1101₂ = -3
                    -------------
Sign-extension      11111100
                    111100
                    00100     (2's complement of 1100)
                    -------------
Product             00001100₂ = +12
```

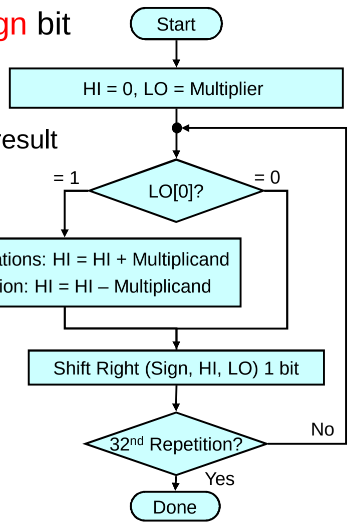

Sequential Signed Multiplier

❖ ALU produces 32-bit result + Sign bit

❖ Check for overflow

- ✧ No overflow ➔ Extend sign-bit of result
- ✧ Overflow ➔ Invert sign bit





Signed Multiplication Example

- Consider: 1100_2 (-4) × 1101_2 (-3), Product = 00001100_2
- Check for overflow: No overflow ➔ Extend sign bit
- Last iteration: add 2's complement of Multiplicand

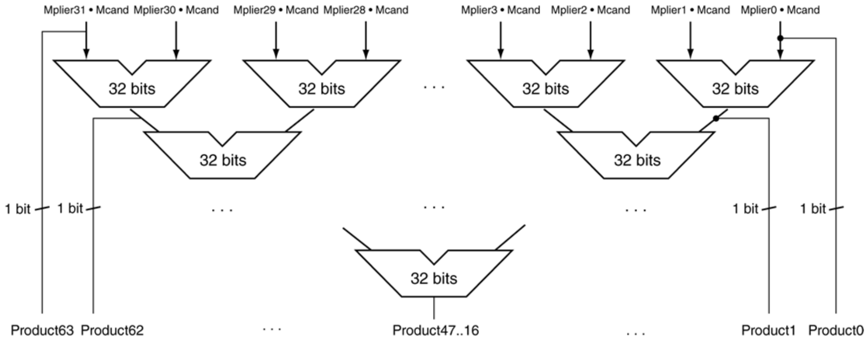
Iteration		Multiplicand	Sign	Product = HI, LO
0	Initialize (HI = 0, LO = Multiplier)	1 1 0 0		0 0 0 0 1 1 0 1
1	LO[0] = 1 => ADD		1	1 1 0 0 1 1 0 1
	Shift (Sign, HI, LO) right 1 bit	1 1 0 0		1 1 1 0 0 1 1 0
2	LO[0] = 0 => Do Nothing			
	Shift (Sign, HI, LO) right 1 bit	1 1 0 0		1 1 1 1 0 0 1 1
3	LO[0]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

Next . . .

- Unsigned Integer Multiplication
- Signed Integer Multiplication
- Faster Integer Multiplication
- Integer Division
- Integer Multiplication and Division in MIPS

Faster Integer Multiplier

❖ Uses Multiple Adders (Cost vs. Performance)



- Can be pipelined

Using Multiple Adders

❖ 32-bit adder for each bit of the 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

❖ Array multiplier can be optimized

- ✧ Additions can be done in parallel
- ✧ Multiple-level tree reduction to produce final product
- ✧ Carry save adders reduce delays

Carry Save Adders

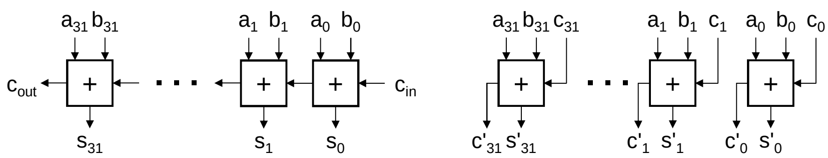
❖ Used when adding multiple numbers (as in multipliers)

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

- ✧ The carry does not propagate as in a carry-propagate adder
- ✧ This is why a carry-save is faster than a carry-propagate adder

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

- ✧ It adds 3 numbers and produces partial sum and carry bits



Carry-Propagate Adder   Carry-Save Adder

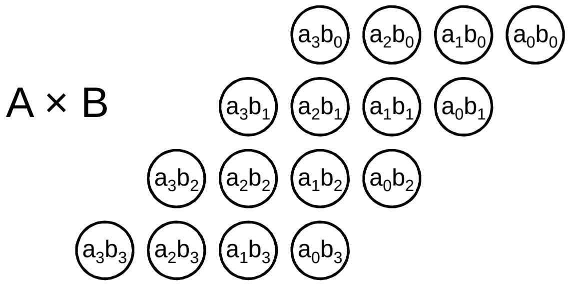
Tree Multiplier - 1 of 2

❖ Suppose we want to multiply two numbers A and B

- ✧ Example on 4-bit numbers: $A = a_3\ a_2\ a_1\ a_0$ and $B = b_3\ b_2\ b_1\ b_0$

❖ Step 1: AND (multiply) each bit of A with each bit of B

- ✧ Requires n^2 AND gates and produces n^2 product bits
- ✧ Position of $a_ib_j = (i+j)$. For example, Position of $a_2b_3 = 2+3 = 5$

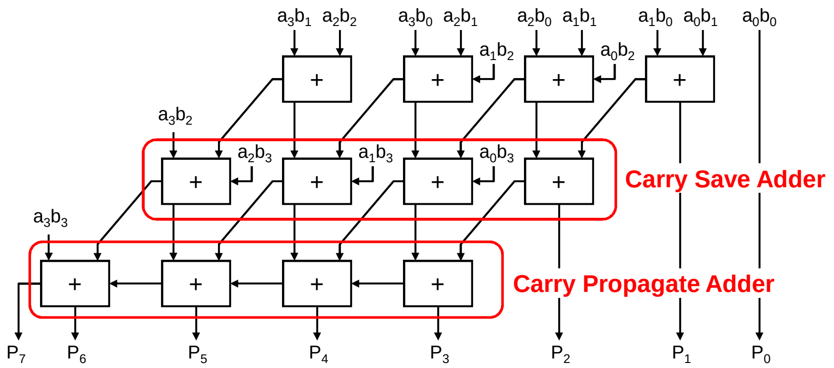


Tree Multiplier - 2 of 2

Step 2: Use carry save adders to add the partial products

- ✧ Reduce the partial products to just two numbers

Step 3: Add last two numbers using a carry-propagate adder



Next . . .

- ❖ Unsigned Integer Multiplication
- ❖ Signed Integer Multiplication
- ❖ Faster Integer Multiplication
- ❖ Integer Division
- ❖ Integer Multiplication and Division in MIPS

Unsigned Division (Paper & Pencil)

```
                     10011₂ = 19        Quotient
Divisor   1011₂ ) 11011001₂ = 217       Dividend
                 -1011
                    10
                    101
                    1010
                    10100
                    -1011
                     1001
                     10011
                     -1011
                      1000₂ = 8         Remainder
```

Dividend =
Quotient × Divisor
+ Remainder
217 = 19 × 11 + 8

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

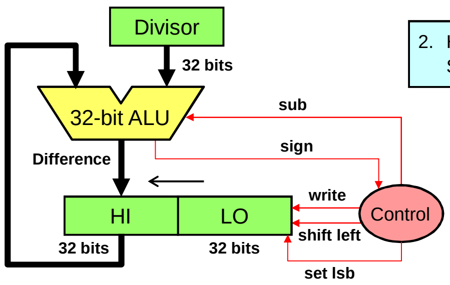
Binary division is accomplished via shifting and subtraction

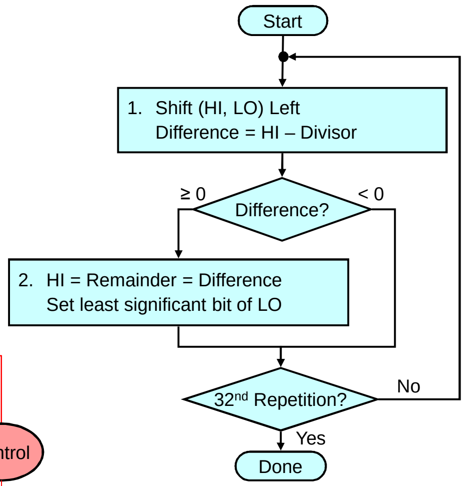
Sequential Division

- ❖ Uses two registers: HI and LO
- ❖ Initialize: HI = Remainder = 0 and LO = Dividend
- ❖ Shift (HI, LO) LEFT by 1 bit (also Shift Quotient LEFT)
  - ✧ Shift the remainder and dividend registers together LEFT
  - ✧ Has the same net effect of shifting the divisor RIGHT
- ❖ Compute: Difference = Remainder – Divisor
- ❖ If (Difference ≥ 0) then
  - ✧ Remainder = Difference
  - ✧ Set Least significant Bit of Quotient
- ❖ Observation to Reduce Hardware:
  - ✧ LO register can be also used to store the computed Quotient

Sequential Division Hardware

- Initialize:
  - HI = 0, LO = Dividend
- Results:
  - HI = Remainder
  - LO = Quotient





Unsigned Integer Division Example

- Example: 1110_2 / 0011_2 (4-bit dividend & divisor)
- Result Quotient = 0100_2 and Remainder = 0010_2
- 4-bit registers for Remainder and Divisor (4-bit ALU)

Iteration		HI	LO	Divisor	Difference
0	Initialize	0 0 0 0	1 1 1 0	0 0 1 1	
1	1: Shift Left, Diff = HI - Divisor	0 0 0 1 ←	1 1 0 0	0 0 1 1	1 1 1 0
	2: Diff < 0 => Do Nothing				
2	1: Shift Left, Diff = HI - Divisor	0 0 1 1 ←	1 0 0 0	0 0 1 1	0 0 0 0
	2: Rem = Diff, set **lsb** of LO	0 0 0 0	1 0 0 1		
3	1: Shift Left, Diff = HI - Divisor	0 0 0 1 ←	0 0 1 0	0 0 1 1	1 1 1 0
	2: Diff < 0 => Do Nothing				
4	1: Shift Left, Diff = HI - Divisor	0 0 1 0 ←	0 1 0 0	0 0 1 1	1 1 1 1
	2: Diff < 0 => Do Nothing				

Signed Integer 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 Integer 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:

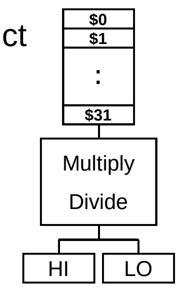
Dividend = Quotient × Divisor + Remainder

Next . . .

- ❖ Unsigned Integer Multiplication
- ❖ Signed Integer Multiplication
- ❖ Faster Multiplication
- ❖ Integer Division
- ❖ Integer Multiplication and Division in MIPS

Integer Multiplication in MIPS

- ❖ 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 of product**
  - ✧ **LO = low-order 32-bit of product**
- ❖ MIPS also has a special `mul` instruction
  - ✧ `mul   $s0,$s1,$s2` **\$s0 = \$s1 × \$s2**
  - ✧ **Put low-order 32 bits into destination register**
  - ✧ **HI & LO are undefined**



Integer Division in MIPS

❖ Divide instructions

- ✧ **`div   $s1,$s2`** **Signed division**
- ✧ **`divu $s1,$s2`** **Unsigned division**

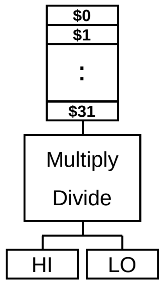
❖ Division produces quotient and remainder

❖ Separate pair of 32-bit registers

- ✧ **HI = 32-bit remainder**
- ✧ **LO = 32-bit quotient**
- ✧ If divisor is 0 then result is **unpredictable**

❖ Moving data from HI/LO to MIPS registers

- ✧ **`mfhi  Rd`** (move from HI to Rd)
- ✧ **`mflo  Rd`** (move from LO to Rd)



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
mul Rd, Rs, Rt	Rd = Rs × Rt	0x1c	Rs^5	Rt^5	Rd^5	0	0x02
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

❖ 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
# Input:    $a0 = unsigned integer
# In/Out:   $a1 = address of string buffer (12 bytes)
#--------------------------------------------------------------
int2str:
   move   $t0, $a0         # $t0 = dividend = unsigned integer
   li     $t1, 10          # $t1 = divisor = 10
   addiu  $a1, $a1, 11     # start at end of string buffer
   sb     $zero, 0($a1)    # store a NULL byte
convert:
   divu   $t0, $t1         # LO  = quotient, HI = remainder
   mflo   $t0              # $t0 = quotient
   mfhi   $t2              # $t2 = remainder
   addiu  $t2, $t2, 0x30   # convert digit to a character
   addiu  $a1, $a1, -1     # point to previous byte
   sb     $t2, 0($a1)      # store digit character
   bnez   $t0, convert     # loop if quotient is not 0
   jr     $ra              # return to caller
```