Integer Multiplication and Division

COE 233

Logic Design and Computer Organization

Dr. Muhamed Mudawar

King Fahd University of Petroleum and Minerals

Unsigned Integer Multiplication

- Paper and Pencil Example:

```
Multiplicand          1100₂ = 12
Multiplier        ×   1101₂ = 13
                  ----------------
                      1100
                     0000
                    1100
                   1100
                  ----------------
Product           10011100₂ = 156
```

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

- **n-bit multiplicand × n-bit multiplier = (2n)-bit product**
- Accomplished via **shifting** and **addition**
- Consumes more time and more chip area than addition

Signed 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
  - Drawback: additional steps to compute the 2's complement
- Better version:
  - Use the same multiplication hardware
  - Extend the **sign** of the multiplicand in the partial products
  - If multiplier is negative, the **last step** should be a **subtract**

Signed Multiplication (Paper & Pencil)

- 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
```

Unsigned Division (Paper & Pencil)

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

Check how big a number can be subtracted, creating a bit of the quotient on each attempt

Dividend = Quotient × Divisor + Remainder

217 = 19 × 11 + 8

Binary division is done via shifting and subtraction

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

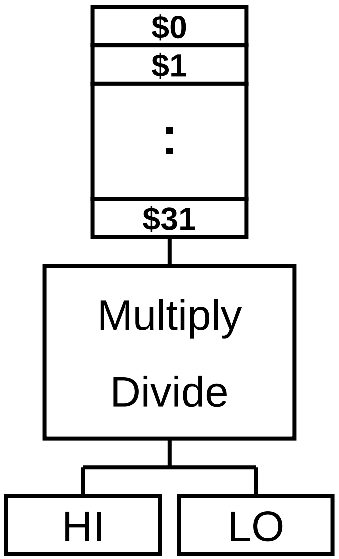
1. **Positive** Dividend and **Positive** Divisor
   - ✧ Example: +17 / +3 Quotient = +5 Remainder = +2
2. **Positive** Dividend and **Negative** Divisor
   - ✧ Example: +17 / –3 Quotient = –5 Remainder = +2
3. **Negative** Dividend and **Positive** Divisor
   - ✧ Example: –17 / +3 Quotient = –5 Remainder = –2
4. **Negative** Dividend and **Negative** Divisor
   - ✧ Example: –17 / –3 Quotient = +5 Remainder = –2

The following equation must always hold:

Dividend = Quotient × Divisor + Remainder

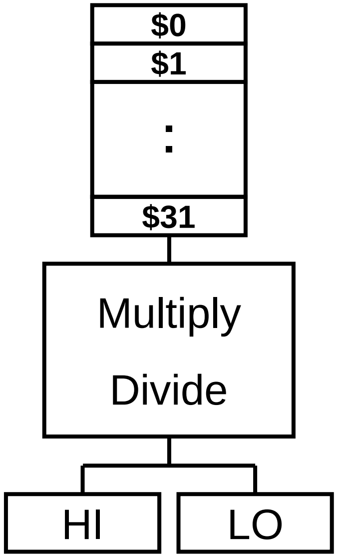
Integer Multiplication in MIPS

- Multiply instructions
  - `mult  Rs, Rt` **Signed multiplication**
  - `multu Rs, Rt` **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  Rd, Rs, Rt` `Rd = Rs × Rt`
  - Copy **LO** into destination register **Rd**
  - Useful when the product is small (32 bits) and **HI** is not needed



Integer Division in MIPS

- Divide instructions
  - `div  Rs, Rt` **Signed division**
  - `divu Rs, Rt` **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` (**Rd = HI**)
  - `mflo Rd` (**Rd = LO**)



Integer Multiply and Divide Instructions

Instruction	Meaning	Format					
mult Rs, Rt	HI, LO = Rs $\times_s$ Rt	Op = 0	Rs	Rt	0	0	0x18
multu Rs, Rt	HI, LO = Rs $\times_u$ Rt	Op = 0	Rs	Rt	0	0	0x19
mul Rd, Rs, Rt	Rd = Rs $\times_s$ Rt	0x1c	Rs	Rt	Rd	0	2
div Rs, Rt	HI, LO = Rs $/_s$ Rt	Op = 0	Rs	Rt	0	0	0x1a
divu Rs, Rt	HI, LO = Rs $/_u$ Rt	Op = 0	Rs	Rt	0	0	0x1b
mfhi Rd	Rd = HI	Op = 0	0	0	Rd	0	0x10
mflo Rd	Rd = LO	Op = 0	0	0	Rd	0	0x12
mthi Rs	HI = Rs	Op = 0	Rs	0	0	0	0x11
mtlo Rs	LO = Rs	Op = 0	Rs	0	0	0	0x13

$\times_s$ = Signed multiplication, $\times_u$ = Unsigned multiplication

$/_s$ = Signed division, $/_u$ = Unsigned division

NO arithmetic exception can occur

String to Integer Conversion

- Consider the conversion of string "91052" into an integer

`'9'`	`'1'`	`'0'`	`'5'`	`'2'`

- How to convert the string into an integer?
- Initialize: **sum = 0**
- Load each character of the string into a register
  - Check if the character is in the range: **`'0'`** to **`'9'`**
  - Convert the character into a **digit** in the range: **`0`** to **`9`**
  - Compute: **sum = sum * 10 + digit**
  - Repeat until end of string or a non-digit character is encountered
- To convert "91052", initialize sum to 0 then …
  - sum = 9, then 91, then 910, then 9105, then 91052

String to Integer Conversion Function

```
#-----------------------------------------------------------
# str2int:  Convert a string of digits into unsigned integer
# Input:    $a0 = address of null terminated string
# Output:   $v0 = unsigned integer value
#-----------------------------------------------------------
str2int:
      li     $v0, 0            # Initialize: $v0 = sum = 0
      li     $t0, 10           # Initialize: $t0 = 10
L1:   lb     $t1, 0($a0)       # load $t1 = str[i]
      blt    $t1, '0', done    # exit loop if ($t1 < '0')
      bgt    $t1, '9', done    # exit loop if ($t1 > '9')
      addiu  $t1, $t1, -48     # Convert character to digit
      mul    $v0, $v0, $t0     # $v0 = sum * 10
      addu   $v0, $v0, $t1     # $v0 = sum * 10 + digit
      addiu  $a0, $a0, 1       # $a0 = address of next char
      j      L1                # loop back
done: jr     $ra               # return to caller
```

Integer to String Conversion

- Convert an unsigned 32-bit integer into 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 Conversion Function

```
#--------------------------------------------------------------
# int2str:  Converts an unsigned integer into a string
# Input:    $a0 = value, $a1 = buffer address (12 bytes)
# Output:   $v0 = address of converted string in buffer
#--------------------------------------------------------------
int2str:
      li     $t0, 10          # $t0 = divisor = 10
      addiu  $v0, $a1, 11     # start at end of buffer
      sb     $zero, 0($v0)    # store a NULL character
L2:   divu   $a0, $t0         # LO  = value/10, HI = value%10
      mflo   $a0              # $a0 = value/10
      mfhi   $t1              # $t1 = value%10
      addiu  $t1, $t1, 48     # convert digit into ASCII
      addiu  $v0, $v0, -1     # point to previous byte
      sb     $t1, 0($v0)      # store character in memory
      bnez   $a0, L2          # loop if value is not 0
      jr     $ra              # return to caller
```