7 | Integer Multiplication and Division

Multiplication

How can we multiply whole numbers? Well, what is multiplication? It's repeated addition. Think of how it was conceptualized in elementary school: 2 * 3 = 3 + 3 = 6.

How large can the result of a multiply be given two whole numbers?

- In decimal, if we have 3 digit numbers multiplied together (ex. 999 * 999 = 998001), we can have at most 6 as a result.
- When multiplying two *n*-bit numbers, the result could require 2*n* bits (two 8-bit words multiplied = two 16-bit words)

Function to perform multiplication using repeated addition:

```
## Pre:         $a0 and $a1 store the numbers to be multiplied
##              $a0 is not negative
##              Both numbers fit in a half word (16 bits)
##
## Post:        $v0 contains the product
##              $a0 is 0
mult:
        addi    $sp, $sp, -4
        sw      $ra, 0($sp)             # save return address

        li      $v0, 0                  # product

# multiplier, $a0 is used as a counter
# using $v0 as an accumulator (result store-r)
lp_mult:
        beq     $a0, 0, done_mult       # finished? ($a0 == 0?)
        add     $v0, $v0, $a1           # add multiplicand to product
        addi    $a0, $a0, -1            # decrement counter
        j       lp_mult                 # jump to beginning again

done_mult:
        lw      $ra, 0($sp)
        add     $sp, $sp, 4             # get the return address from stack
        jr      $ra
```

A faster way to multiply whole numbers, use shift and add (imagine using powers of 2). For example, $200x = 128x + 64x + 8x$ for any value of x. There are 2 additions and 3 shifts instead of doing 200 additions.

Also see binary multiplication (right); notice how it shifts to the left each time and see the diagonals. See below for how this would work:

				1	0	1	=	5
		x	1	0	1	1	=	11
				1	0	1	1	
			1	0	1		1	
		0	0	0			0	
+	1	0	1				1	
	1	1	0	1	1	1	=	55

```
######################## Begin mult function
### Multiply two whole numbers using
###   a shift-and-add algorithm
### Pre: The values to be multiplied are in registers
###      $a0 and $a1
### Pre: $a0 is not negative
### Return product in $v0
mult:
      addi      $sp, $sp, -4
      sw        $ra, 0($sp)

      li        $v0, 0                 # Accumulator for product
lp_mult:
      ble       $a0, $0, done_mult
      andi      $t0, $a0, 1            # Test low order bit
      beq       $t0, $0, even_mult
      add       $v0, $v0, $a1          # Add in multiplicand
even_mult:
      sll       $a1, $a1, 1            # Shift multiplicand
      srl       $a0, $a0, 1            # Shift multiplier
      j         lp_mult                # Repeat the loop

done_mult:
      lw        $ra, 0($sp)            # return to calling function
      addi      $sp, $sp, 4
      jr        $ra
########################## End function mult
```

MIPS multiply instruction uses two extra 32-bit registers, hi and lo, which are separate from the 32 that we've already seen. Those two together form a 64-bit register pair (think high-order and low-order 32 bits for each).

Format: **mult** **$rs, $rt** # [Hi, Lo] ← $rs * $rt

How to obtain the results of mult? We can move the Hi value into some CPU register, and the same way with the low:

Format: **mfhi** **$rd** # $rd ← Hi
Format: **mflo** **$rd** # $rd ← Lo

Example: Find the area of a rectangle, given the lengths of its sides.

```
        .text
################################  Begin function: area
## Find the area of a rectangle.
##
## Pre:         $a0 and $a1 store the length and width, in inches
##              $a0 and $a1 both fit in a half word, 0x0000ffff
##
## Post:        $v0 contains the number of square inches in the result
area:
        addi    $sp, $sp, -4
        sw      $ra, 0($sp)             # save return address

        mult    $a0, $a1                # lo = len + width
        mflo    $v0                     # result

        lw      $ra, 0($sp)
        add     $sp, $sp, 4             # get return address from stack
        jr      $ra

################################ End function
```

Division

Division is repeated subtraction (for strictly whole numbers). For example, 17 / 5 = 17 - 5 = 12 - 5 = 7 - 5 = 2. We subtracted 5 from 17 three times, making 3 the quotient. What about the remainder? It's what we're left with after pulling out 5 as many times as we can. It's 2 here.

→ What does this tell us? ***Division of whole numbers has 2 results.***

→ We will not be able to do this with negative numbers since the modular operation messes up afterwards, so we'll avoid this. (Repeated subtraction in book.)

There is a faster way to divide whole numbers; shift and subtract (shown below):

```
        .text
######################### Begin div function
```

```
### Perform division using shift and suubtract algorithm
### The dividend is in register $a0
### The divisor is in register $a1
### The quotient is left in $v0
### The remainder is left in $v1
### Pre:  The divisor is positive
### Pre   The dividend is not negative
### Author: sdb
div:
       addi     $sp, $sp, -4
       sw       $ra, 0($sp)

       li       $v0, 0                  # quotient
       li       $v1, 0                  # remainder
       li       $t0, 32                 # loop counter
lp_div:
       beq      $t0, $0, done_div
       sll      $v0, $v0, 1             # shift quotient
       sll      $v1, $v1, 1             # shift remainder,dividend
       bge      $a0, $0, notNeg_div
       addi     $v1, $v1, 1
notNeg_div:
       sll      $a0, $a0, 1             # shift dividend
       blt      $v1, $a1, noSubtr_div   # subtract?
       sub      $v1, $v1, $a1
       addi     $v0, $v0, 1
noSubtr_div:
       addi     $t0, $t0, -1
       j        lp_div                  # Repeat the loop
done_div:
       lw       $ra, 0($sp)             # return to calling function
       addi     $sp, $sp, 4
       jr       $ra
########################### End function div
```

MIPS division instruction: The Lo stores the result, the Hi stores the remainder.

Format:	**div**	**$rs, $rt**	# Lo ← $rs (dividend) / $rt (divisor)
			# Hi ← $rs (dividend) % $rt (divisor)

Example: A swimmer swims several laps in the pool. Given the number of laps, and the length of one lap, in yards, write a function to find the total distance of the swim, in whole miles and remaining yards (1,760 yards in a mile).

```
############################### Begin function: swim
## Find the swimmer's distance
##
```

```
## Pre:        $a0 is the length of a lap, in yards
##             $a1 is the number of laps completed
##             The two arguments fit in a half word
##
## Post:       $v0 contains the number of miles in the result
##             $v1 contains the number of yards remaining in the resul
swim:
        addi    $sp, $sp, -4
        sw      $ra, 0($sp)             # save return address

        mult    $a0, $a1                # total yards
        mflo    $t0
        li      $t1, 1760               # yards per mile
        div     $t0, $t1                # quotient and remainder
        mflo    $v0                     # quotient = miles
        mfhi    $v1                     # remainder = yards

        lw      $ra, 0($sp)
        addi    $sp, $sp, 4             # get return address from stack
        jr      $ra

############################### End function
```