Computer Architecture - Lab 5

Integer Multiplication, Division and Arithmetic Shift Instructions

April 21, 2019

1 Instruction

This lab topics:

- Integer multiplication and division
- The hi and lo registers
- The mult and multu instructions
- The div and divu instructions
- The mfhi and mflo instructions
- Arithmetic shift right
- The sra Instruction

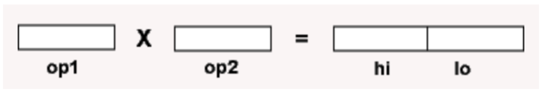
The product of two N-place decimal integers may need $2N$ places. This is true for numbers expressed in any base. In particular, the product of two integers expressed with N-bit binary may need $2N$ bits.

```
       10110111            B7          183
       10100010            A2          162
       ________            __          ___
       00000000           16E          366
      10110111.          726.        1098
     00000000..                      183
    00000000...
   00000000....
  10110111.....
 00000000......
10110111.......
_______________          ____        _____
111001111001110          73CE        29646
```

The two 8-bit operands result in a 15-bit product. Also shown is the same product done with base 16 and base 10 notation.

The multiply unit of MIPS contains two 32-bit registers called hi and lo. These are not general purpose registers. When two 32-bit operands are multiplied, hi and lo hold the 64 bits of the result. Bits 32 through 63 are in `hi` and bits 0 through 31 are in `lo`.

MIPS Multiplication



Here are the instructions that do this. The operands are contained in general-purpose registers.

```
mult    s,t         # hilo <- $s * $t   (two's comp operands)
multu   s,t         # hilo <- $s * $t   (unsigned operands)
```

There is a multiply instruction for unsigned operands, and a multiply instruction for signed operands (two's complement operands).

The `mfhi` and `mflo` Instructions

There are two instructions that move the result of a multiplication into a general purpose register:

```
mfhi    d         #  d <- hi.  Move From Hi
mflo    d         #  d <- lo.  Move From Lo
```

The `hi` and `lo` registers cannot be used with any of the other arithmetic or logic instructions. If you want to do something with a product, it must first be moved to a general purpose register. However there is a further complication on MIPS hardware:

Rule: Do not use a multiply or a divide instruction within two instructions after `mflo` or `mfhi`. The reason for this involves the way the MIPS pipeline works. On the SPIM simulator this rule does not matter.

Example Program

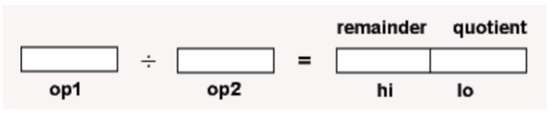
```
## newMult.asm
##
## Program to calculate 5*x - 74
##
## Register Use:
##  $t0   x
##  $t1   result

        .text
        .globl  main

main:
        ori      $t0,   $0, 12           # put x into $t0
        ori      $t2,   $0,  5           # put 5 into $t2
        mult     $t0,   $t2              # 5x -> hi & lo
        mflo     $t3                     # $t3 = 5x
        addiu    $t1,   $t3,-74          # $t1 = 5x - 74

## End of file
```

The div and the divu Instructions



With N-digit integer division there are two results, an N-digit quotient and an N-digit remainder. With 32-bit operands there will be (in general) two 32-bit results. MIPS uses the `hi` and `lo` registers for the results:
Here are the MIPS instructions for integer divide. The "u" means operands and results are in unsigned binary.

```
div    s,t         #  lo <- s div t
                   #  hi <- s mod t
              #  operands are two's complement
divu   s,t         #  lo <- s div t
                   #  hi <- s mod t
                   #  operands are unsigned
```

Example Program

```
## divEg.asm
##
## Program to calculate (y + x) / (y - x)
##
## Register Use:
##  $t1   x
##  $t2   y
##  $t3  y+x
##  $t4  y-x

        .text
        .globl  main

main:
        ori      $t1,   $0,    8         # put x into $8
        ori      $t2,   $0,   36         # put y into $9
        addu     $t3,   $t2, $t1         # $10  = (y+x)
        subu     $11,   $t2, $t1         # $11  = (y-x)
        div      $t3,   $t4              # hilo = (y+x)/(y-x)
        mflo     $t3                     # $10  = quotient
        mfhi     $t4                     # $11  = remainder

## End of file
```

The sra Instruction MIPS has a shift right arithmetic instruction:

```
    sra    d,s,shft   #  $d <- s shifted right
                      #  shft bit positions.
                      #  0 <= shft <= 31
```

Sometimes you need to divide by two. This instruction is faster and more convenient than the `div` instruction.

2 Exercises

1. **Exercise 1**
   Write a program to evaluate a polynomial, similar to *newMult.asm* from the chapter. Evaluate the polynomial:
   $$3x^2 + 5x - 12$$
   Pick a register to contain x and initialize it to an integer value (positive or negative) at the beginning of the program. Assume that x is small enough so that all results remain in the `lo` result register. Evaluate the polynomial and leave its value in a register.
   Verify that the program works by using several initial values for x. Use $x = 0$ and $x = 1$ to start since this will make debugging easy.

   Optional: write the program following the hardware rule that two or more instructions must follow a mflo instruction before another `mult` instruction. Try to put useful instructions in the two slots that follow the `mflo`. Otherwise put *no-op instructions*, `sll $0,$0,0`, in the two slots.

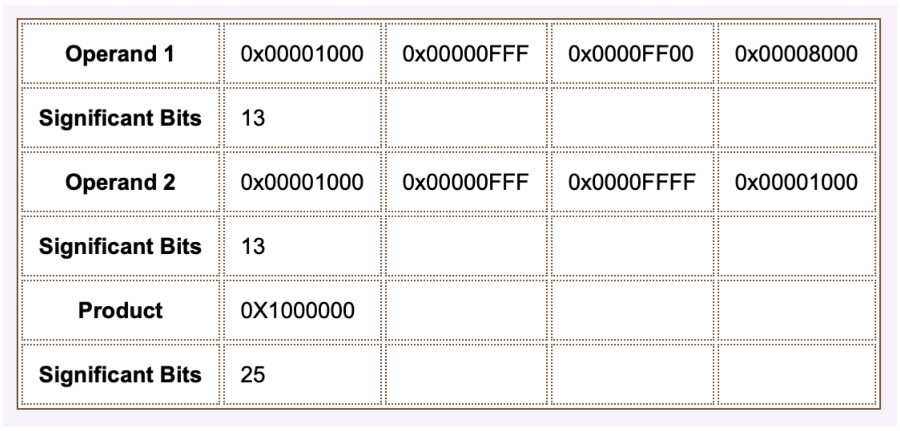
2. **Exercise 2** Write a program similar to *divEg.asm* in to evaluate a rational function:
   $$(3x + 7)/(2x + 8)$$
   Verify that the program works by using several initial values for x. Use $x = 0$ and $x = 1$ to start since this will make debugging easy. Try some other values, then check what happens when $x = -4$.

3. **Exercise 3** Write a program that determines the value of the following expression:
   $$(x * y)/z$$
   Use $x = 1600000(= 0x186A00)$, $y = 80000(= 0x13880)$, and $z = 400000(= 0x61A80)$. Initialize three registers to these values. Since the immediate operand of the `ori` instruction is only 16 bits wide, use shift instructions to move bits into the correct locations of the registers.

   Choose wisely the order of multiply and divide operations so that the *significant bits* always remain in the `lo` result register.

   Note: read this https://chortle.ccsu.edu/AssemblyTutorial/Chapter-14/ass14_4.html for definition of **significant bits**.

4. **Exercise 4**
   Write a program that multiplies the contents of two registers which you have initialized using an immediate operand with the `ori` instruction. Determine (by inspection) the number of significant bits in each of the following numbers, represented in two's complement. Use the program to form their product and then determine the number of significant bits in it.



Operand 1	0x00001000	0x00000FFF	0x0000FF00	0x00008000
Significant Bits	13			
Operand 2	0x00001000	0x00000FFF	0x0000FFFF	0x00001000
Significant Bits	13			
Product	0X1000000			
Significant Bits	25			

3 References

1. https://chortle.ccsu.edu/AssemblyTutorial/Chapter-14/ass14_1.html