EECE 417 Computer Systems Architecture

Department of Electrical and Computer Engineering
Howard University

Charles Kim

Spring 2007

Computer Organization and Design (3rd Ed)

-The Hardware/Software Interface

by

David A. Patterson
John L. Hennessy

Chapter Three

Arithmetic for Computers -Part A

Numbers

- **Bits are just bits (no inherent meaning)**
  — conventions define relationship between bits and numbers
- **Binary numbers (base 2)**
  0000 0001 0010 0011 0100 0101 0110 0111 1000 1001...
  decimal: 0...2^n-1
- **Of course it gets more complicated:**
  numbers are finite (overflow)
  fractions and real numbers
  negative numbers
  e.g., no MIPS subi instruction; addi can add a negative number
- **How do we represent negative numbers?**
  i.e., which bit patterns will represent which numbers?

Possible Representations

Sign Magnitude:	One's Complement	Two's Complement
000 = +0	000 = +0	000 = +0
001 = +1	001 = +1	001 = +1
010 = +2	010 = +2	010 = +2
011 = +3	011 = +3	011 = +3
100 = -0	100 = -3	100 = -4
101 = -1	101 = -2	101 = -3
110 = -2	110 = -1	110 = -2
111 = -3	111 = -0	111 = -1

- **Issues: balance, number of zeros, ease of operations**
- **Which one is best? Why?**

MIPS

- **32 bit signed numbers:**

0000 0000 0000 0000 0000 0000 0000 0000$_{two}$ = 0$_{ten}$
0000 0000 0000 0000 0000 0000 0000 0001$_{two}$ = + 1$_{ten}$
0000 0000 0000 0000 0000 0000 0000 0010$_{two}$ = + 2$_{ten}$
...
0111 1111 1111 1111 1111 1111 1111 1110$_{two}$ = + 2,147,483,646$_{ten}$
0111 1111 1111 1111 1111 1111 1111 1111$_{two}$ = + 2,147,483,647$_{ten}$ — *maxint*
1000 0000 0000 0000 0000 0000 0000 0000$_{two}$ = – 2,147,483,648$_{ten}$ — *minint*
1000 0000 0000 0000 0000 0000 0000 0001$_{two}$ = – 2,147,483,647$_{ten}$
1000 0000 0000 0000 0000 0000 0000 0010$_{two}$ = – 2,147,483,646$_{ten}$
...
1111 1111 1111 1111 1111 1111 1111 1101$_{two}$ = – 3$_{ten}$
1111 1111 1111 1111 1111 1111 1111 1110$_{two}$ = – 2$_{ten}$
1111 1111 1111 1111 1111 1111 1111 1111$_{two}$ = – 1$_{ten}$

Two's Complement Operations

- **Negating a two's complement number: invert all bits and add 1**
  - **remember: “negate” and “invert” are quite different!**
- **Converting n bit numbers into numbers with more than n bits:**
  - **MIPS 16 bit immediate gets converted to 32 bits for arithmetic**
  - **copy the most significant bit (the sign bit) into the other bits (“sign extension”)**

```
0010  -> 0000 0010
1010  -> 1111 1010
```

  - **"sign extension" (lbu vs. lb) (u -- ‘unsigned’) (load byte)**
    - *lb* - (upper 24 bits determined by the sign bit of the byte)’(byte)
    - *lbu* - (upper 24 bits all zeros)’(byte)

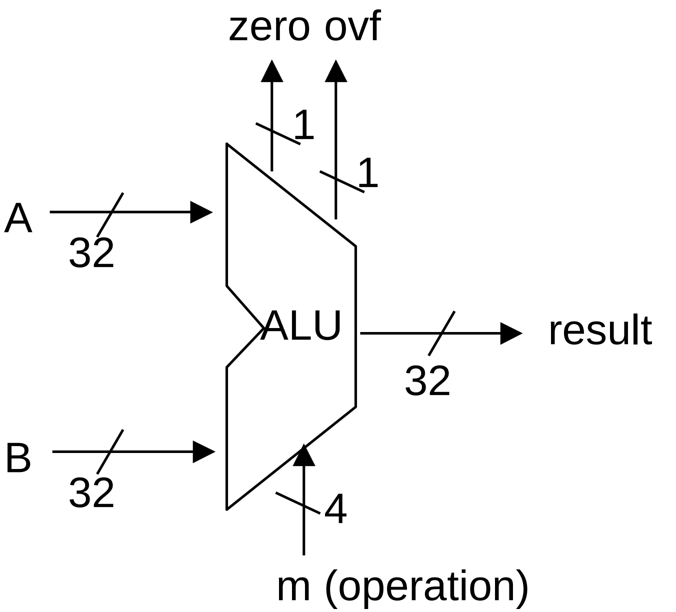
Teasing Problem involving *lb* and *lbu*

```
#lb-lbu.asm
#lb and lbu (unsigned) comparison
#
        .data   0x10010000
        .word   0xFF8F7F3F, 0x708090A0
        .text
main:
        lui     $a0, 0x1001
        or      $a0, $a0, $zero
        lw      $s0, 0($a0)          #1
        lw      $t0, 0($a0)          #2
        lw      $s1, 0($a0)          #3
        lw      $t1, 0($a0)          #4
        lw      $s2, 0($a0)          #5
        lw      $t2, 0($a0)          #6
        lw      $s3, 0($a0)          #7
        lw      $t3, 0($a0)          #8
        lw      $s4, 0($a0)          #9
        lw      $t4, 0($a0)          #10
        lw      $s5, 0($a0)          #11
        lw      $t5, 0($a0)          #12
        lb      $s0, 0($a0)          #21
        lbu     $t0, 0($a0)          #22
        lb      $s1, 1($a0)          #23
        lbu     $t1, 1($a0)          #24
        lb      $s2, 2($a0)          #25
        lbu     $t2, 2($a0)          #26
        lb      $s3, 3($a0)          #27
        lbu     $t3, 3($a0)          #28
        lb      $s4, 4($a0)          #29
        lbu     $t4, 4($a0)          #30
        lb      $s5, 5($a0)          #31
        lbu     $t5, 5($a0)          #32
```

R8	(t0)	=	ff8f7f3f	R16	(s0)	=	ff8f7f3f
R9	(t1)	=	ff8f7f3f	R17	(s1)	=	ff8f7f3f
R10	(t2)	=	ff8f7f3f	R18	(s2)	=	ff8f7f3f
R11	(t3)	=	ff8f7f3f	R19	(s3)	=	ff8f7f3f
R12	(t4)	=	ff8f7f3f	R20	(s4)	=	ff8f7f3f
R13	(t5)	=	ff8f7f3f	R21	(s5)	=	ff8f7f3f

Arithmetic Logic Operation Core Instructions

- **MIPS Arithmetic/Logic operations**
  `add, addi, addiu, addu`
  `sub, subu`
  `mult, multu, div, divu`
  `and, andi, nor, or, ori,`
  `beq, bne, slt, slti, sltiu, sltu`



- **Special Attention Required for these!**
  - **sign extend – `addi, addiu andi, ori, slti, sltiu`**
  - **zero extend – `lbu, addiu, sltiu`**
  - **no overflow detected – `addu, addiu, subu, multu, divu, sltiu, sltu`**

Addition & Subtraction

- **Just like in grade school (carry/borrow 1s)**

```
   0111        0111        0110
 + 0110      - 0110      - 0101
 ---------------------------------
```

- **Two's complement operations easy**
  - **subtraction using addition of negative numbers**

```
   0111
 + 1010
 ------
```

- **Overflow (result too large for finite computer word):**
  - **e.g., adding two n-bit numbers does not yield an n-bit number**

```
   0111
 + 0001
 ------
_  1000
```

note that overflow term is somewhat misleading,
it does not mean a carry "overflowed"

Detecting Overflow

- **No overflow when adding a positive and a negative number**
- **No overflow when signs are the same for subtraction**
- **Overflow occurs when the value affects the sign:**
  - **overflow when adding two positives yields a negative**
  - **or, adding two negatives gives a positive**
  - **or, subtract a negative from a positive and get a negative**
  - **or, subtract a positive from a negative and get a positive**
- **Consider the operations A + B, and A – B**
  - **Can overflow occur if B is 0 ?**
  - **Can overflow occur if A is 0 ?**

OPR	A	B	Result Indicating ovf
A+B	>=0	>=0	<0
A+B	<0	<0	>=0
A-B	>=0	<0	<0
A-B	<0	>=0	>=0

Effects of Overflow

- An exception (interrupt) occurs
  - Control jumps to predefined address for exception
  - Interrupted address is saved for possible resumption
- Details based on software system / language
- Don't always want to detect overflow
  — new MIPS instructions: `addu, addiu, subu`

note: `addiu` *still sign-extends!*
note: `sltu`, `sltiu` *for unsigned comparisons*

Multiplication

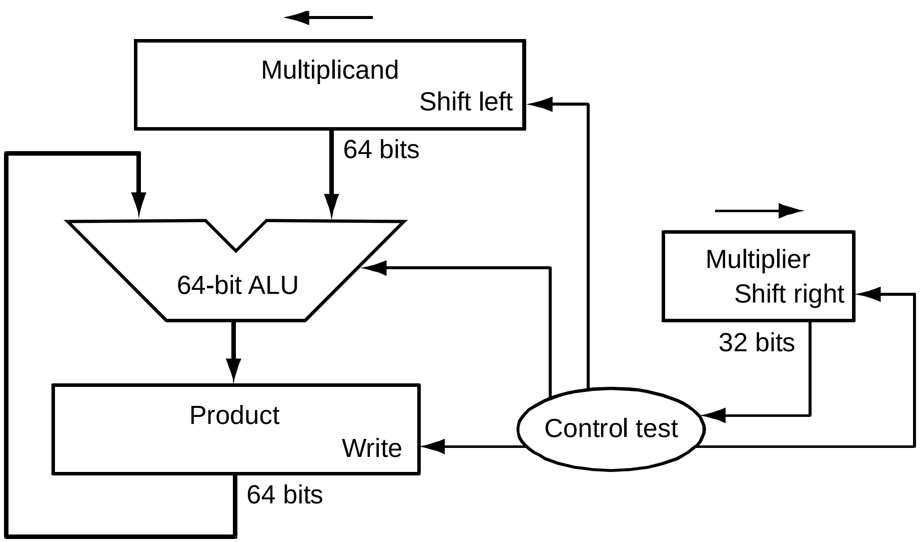
- Let's look at multiplication based on a gradeschool algorithm (decimal representation)

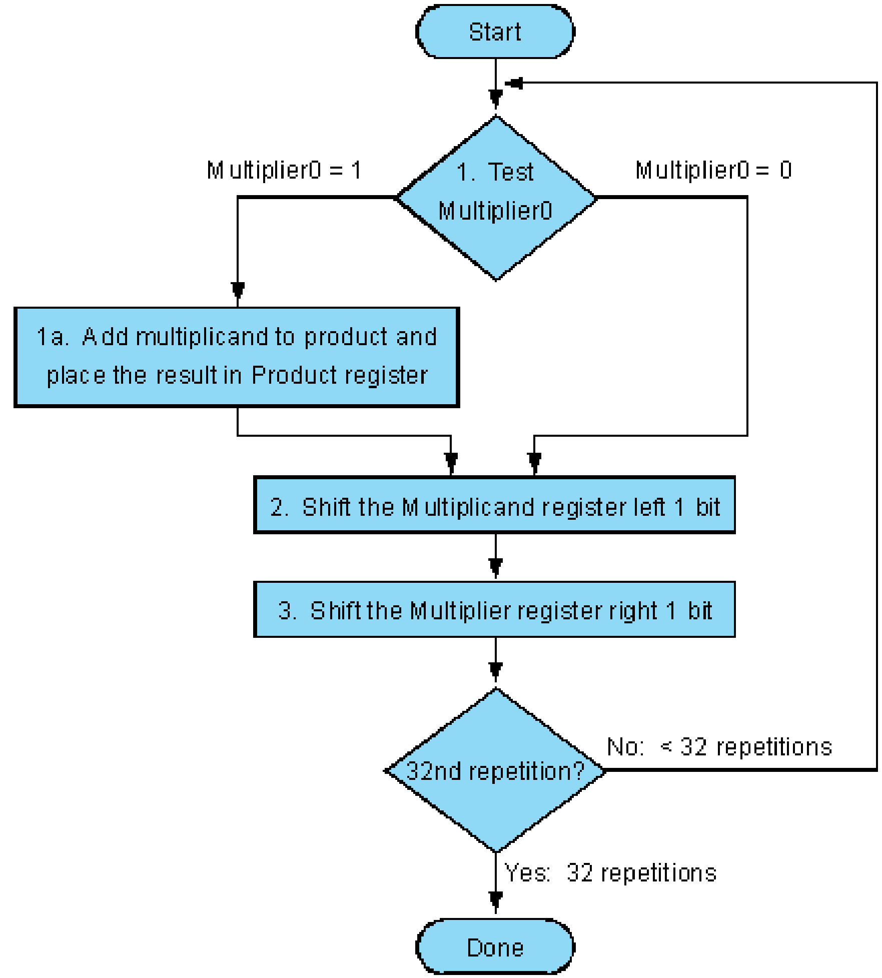
```
      1000   (multiplicand)
   x  1001   (multiplier)
      1000
     0000
    0000
   1000
   1001000   (Product)
```

- More complicated than addition
  - accomplished via shifting and addition
- More time and more area
  - n-bit multiplcand & m-bit multiplier needs n+m bit long product
- Algorithm (Binary Case)
  - At each step
    - Copy of the multiplicand in the proper place of the multiplier digit is 1, or
    - Place 0 in the proper place if the digit is 0
- Three types of Multiplication Hardware
  - Sequential Version
  - Refined Version
  - Fast Version

Multiplication Implementation: Sequential Version

Data flows top to bottom





Datapath

Control

Multiplication Algorithm Exercise

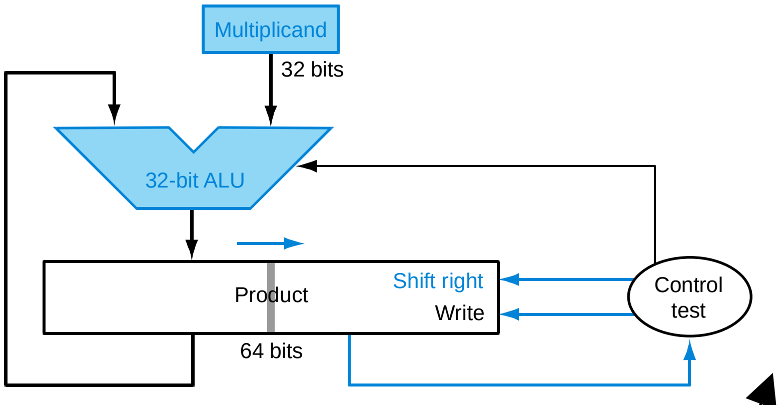
- Binary Number multiplication by the (sequential) algorithm.
- 0010 (multiplicand)
- x 0011 (multiplier)
- multiplier (8-bit), multiplicand (16-bit) and Product (8-bit)

0010
x 0011

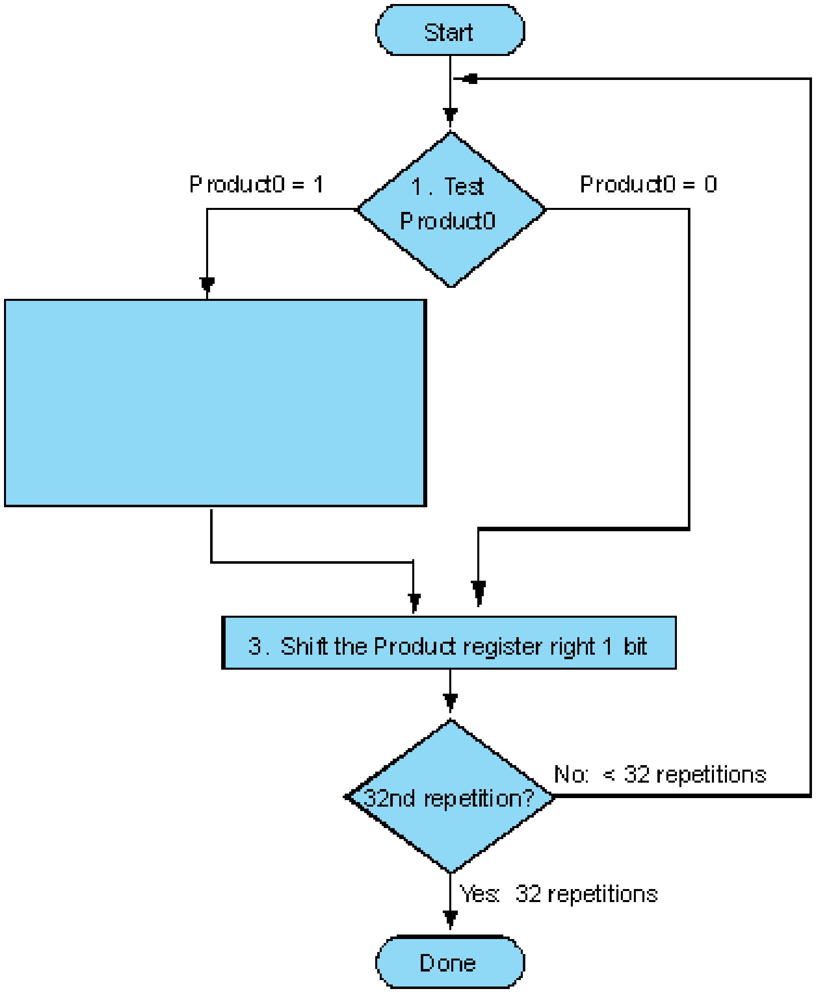
	Multipler	Multiplicand		Product
srl	0011 →	0000 0010		0000 0000
srl	1	0000 0010	Add	0000 0010
		Shift		
srl	1	0000 0100	Add	0000 0110
		Shift		
srl	0	0000 1000		0000 0110
		Shift		
srl	0	0001 0000		0000 0110
		Shift		
end		0010 0000		0000 0110

Multiplication: Revised Version

- Multiplier starts in right half of product

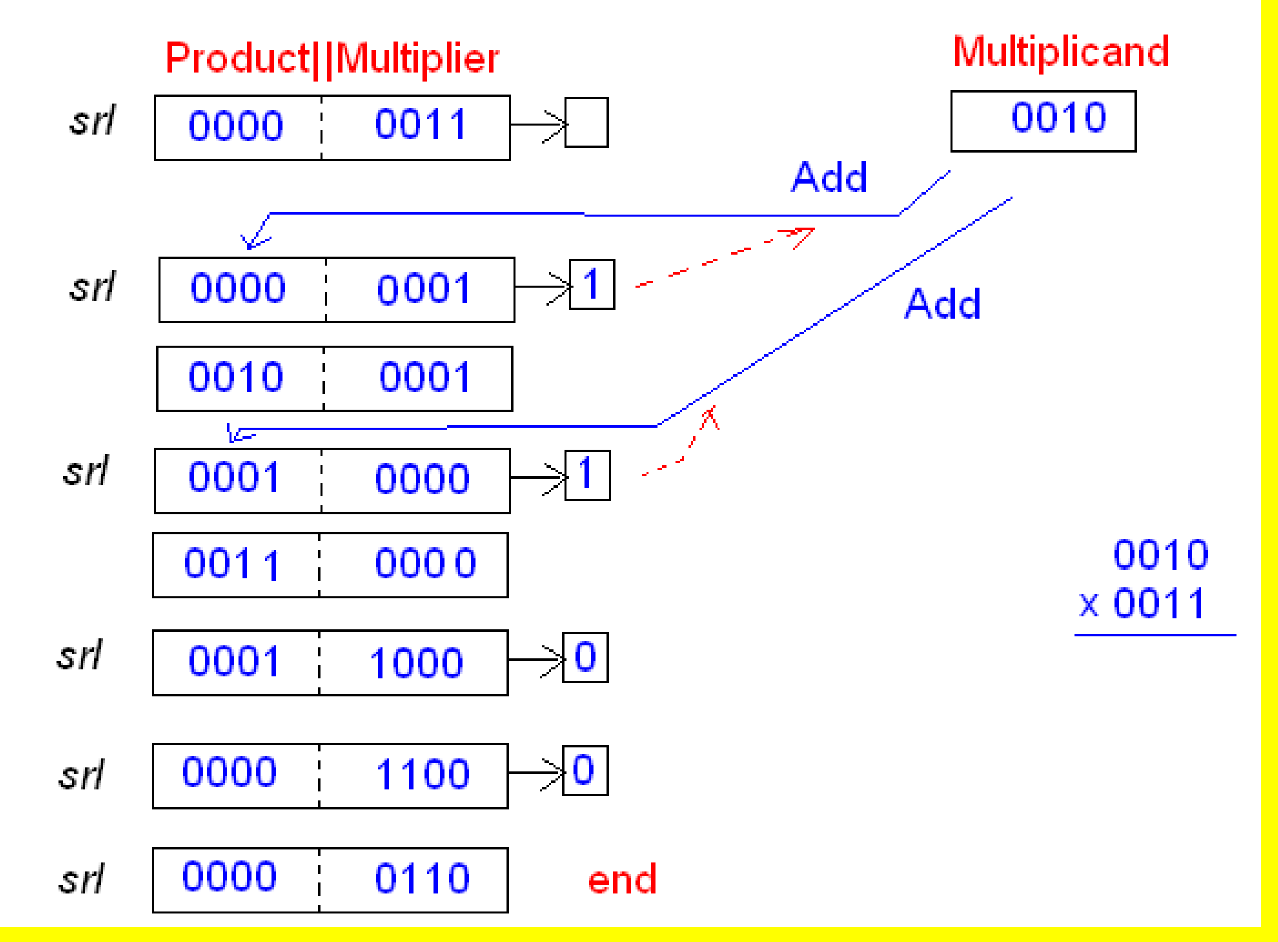


What goes here.



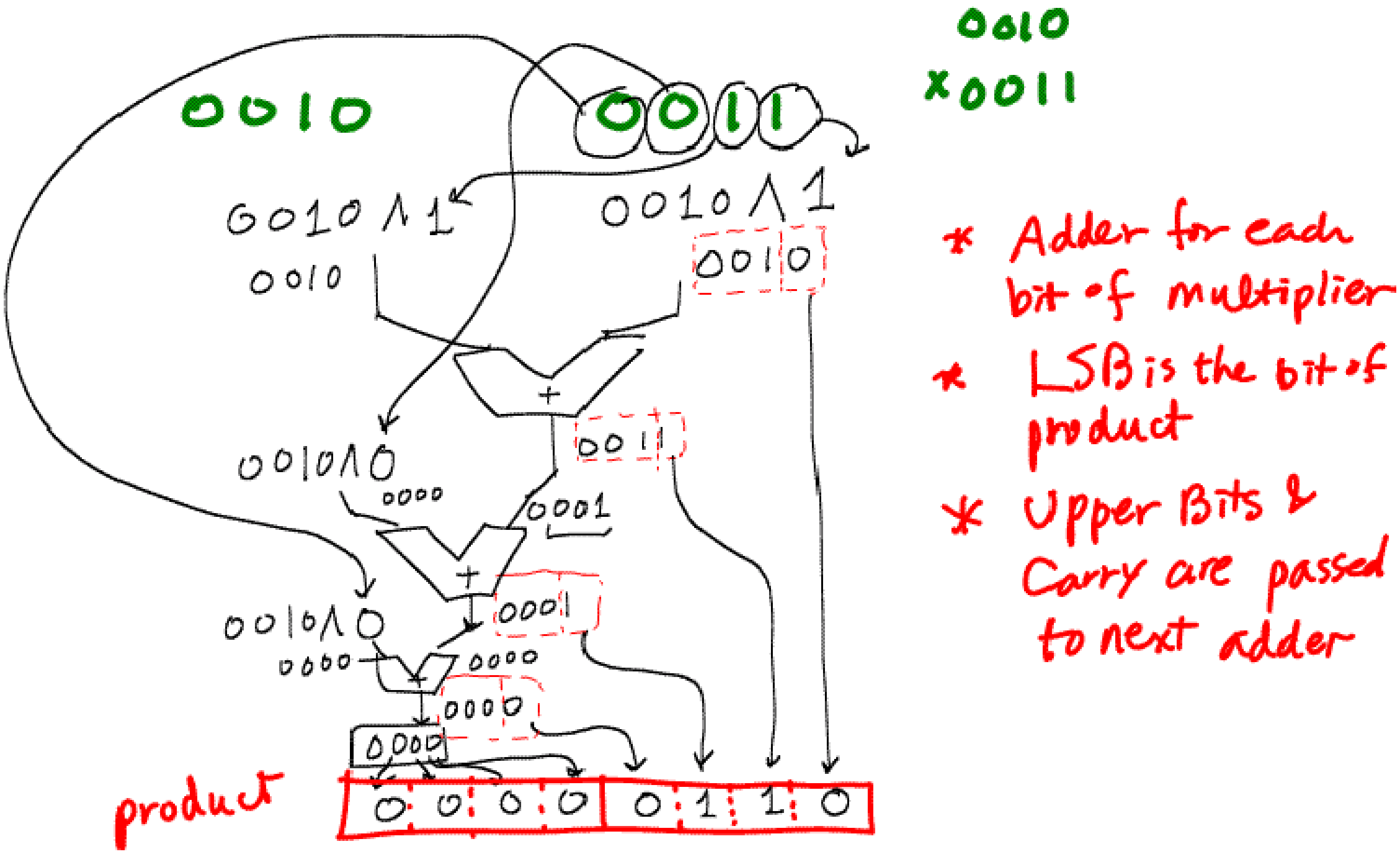
Multiplication Algorithm Exercise (2) - Revised Version

- multiplicand (8-bit) and [Product (8-bit) ||multiplier (8-bit)]



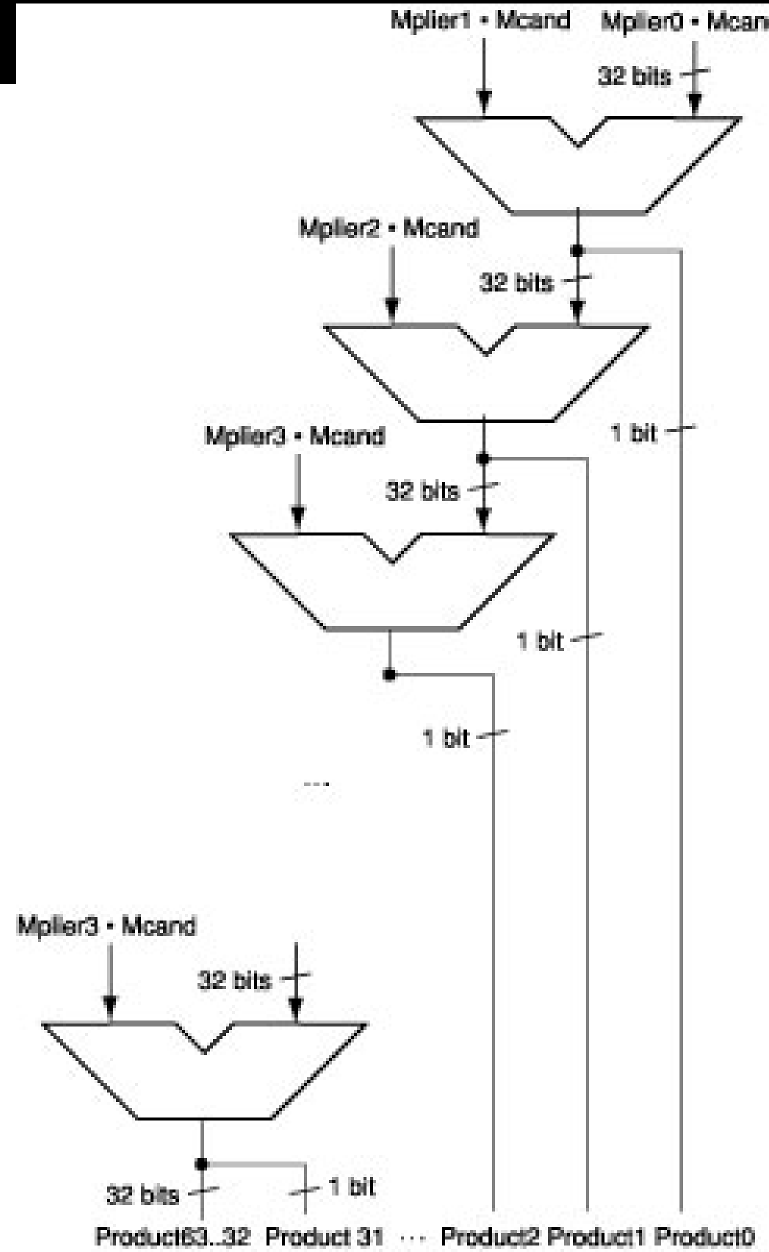
Faster Multiplication

- Uses 31 adders
- Each adder produces a 32-bit sum and a carry
- Algorithm Illustration for the 4-bit by 4-bit multiplication



Multiplication in MIPS

- **31 32-bit adders**



Multiplication in MIPS

- `mult` : multiply
- `multu`: multiply unsigned
- 32bit * 32 bit ---> 64-bit result
  - upper 32bit result ---> stored at Hi
  - lower 32bit result --->stored at Lo
- Fetching the result
  - `mflo` (move from lo) ----> get Lo
  - `mfhi` (move from hi) -----> get Hi

Multiplication Example

p181.asm - Notepad

File Edit Format View Help

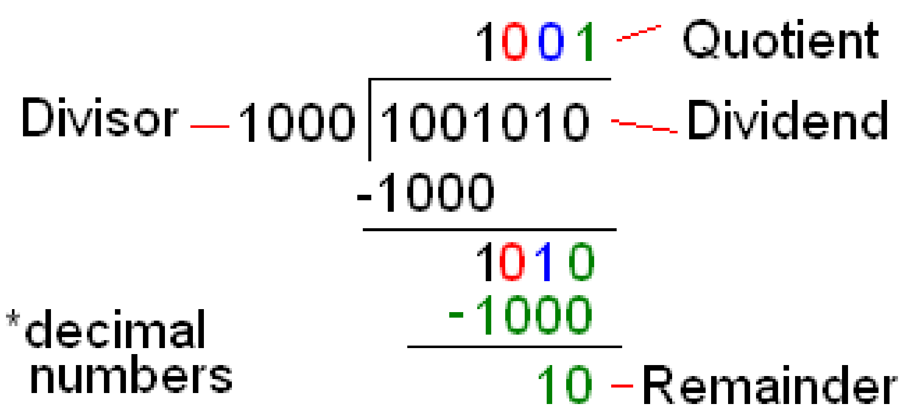
```
#p181.asm
#tmultiplication in MIPS
#       f=g*h
#
#   f <----$s0||$s1
#   g <----$s2
#   h <----$s3
main:
        .data   0x10010000
        .asciiz "\n\n\nMultiplicator\n "  #Banner
        .data   0x10010100
        .asciiz "\nType the multiplicand (first number): "
        .data   0x10010200
        .asciiz "\nType the multiplier (second number): "
        .data   0x10010300
        .asciiz "\nThe product is: "   #msg3
        .text
#Read varibales from key-in
again:  ori     $v0, $zero, 4
        lui     $a0, 0x1001
        ori     $a0, $a0, 0
        syscall                   #Print msg1
        ori     $v0, $zero, 4
        lui     $a0, 0x1001
        or      $a0, $a0, 0x0100
        syscall                   #Print msg2
```

Multiplication -conti

```
        ori     $v0, $zero, 5   #read multiplicand
        syscall                 #now type-in is in v0
        or      $s2, $zero,$v0      #$s2 <---- multiplicand

        ori     $v0, $zero, 4
        lui     $a0, 0x1001
        ori     $a0, $a0, 0x0200
        syscall                 #Print msg3
        ori     $v0, $zero, 5   #read input multiplier
        syscall                 #now type-in is in v0
        or      $s3, $zero,$v0  #$s3 <---- multiplier

        mult    $s2, $s3        #[Hi||Lo]=$s2*$s3
        mfhi    $s0             #$s0 <---Hi
        mflo    $s1             #$s1 <---Lo


        ori $v0, $zero, 4       #msg3
        lui $a0, 0x1001         # Upper part of msg1 addr (ie a0=1
        ori $a0, $a0, 0x0300    # now a0 has 1 word addr 10010200
        syscall                 #Print msg1

#print the result
        ori $v0, $zero,1        #request for print
        or  $a0,  $zero, $s1    #result (lower part)
        syscall
        j   main
```

Division

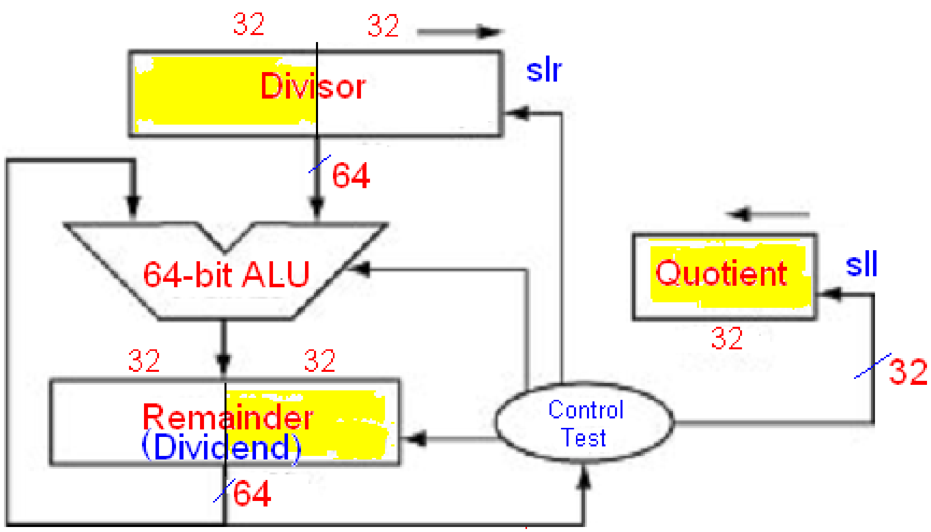
- The reciprocal operation of multiplication
- Less frequent
- More quirky
- Includes “Dividing by 0”
- Long division example



- **Observation**
  - Divisor is shifted to the right 1 bit at each subtraction
  - Quotient bit = 1 when Dividend>Divisor
  - Quotient bit=0 when Dividend>Divisor

Division Algorithm

```
(1)Rem - Divisor
(2)If Rem<0,
     Rem=Rem(old)
     sll Q and Q0=0
    Else
     sll Q and Q0=1
(3) slr Divisor
```



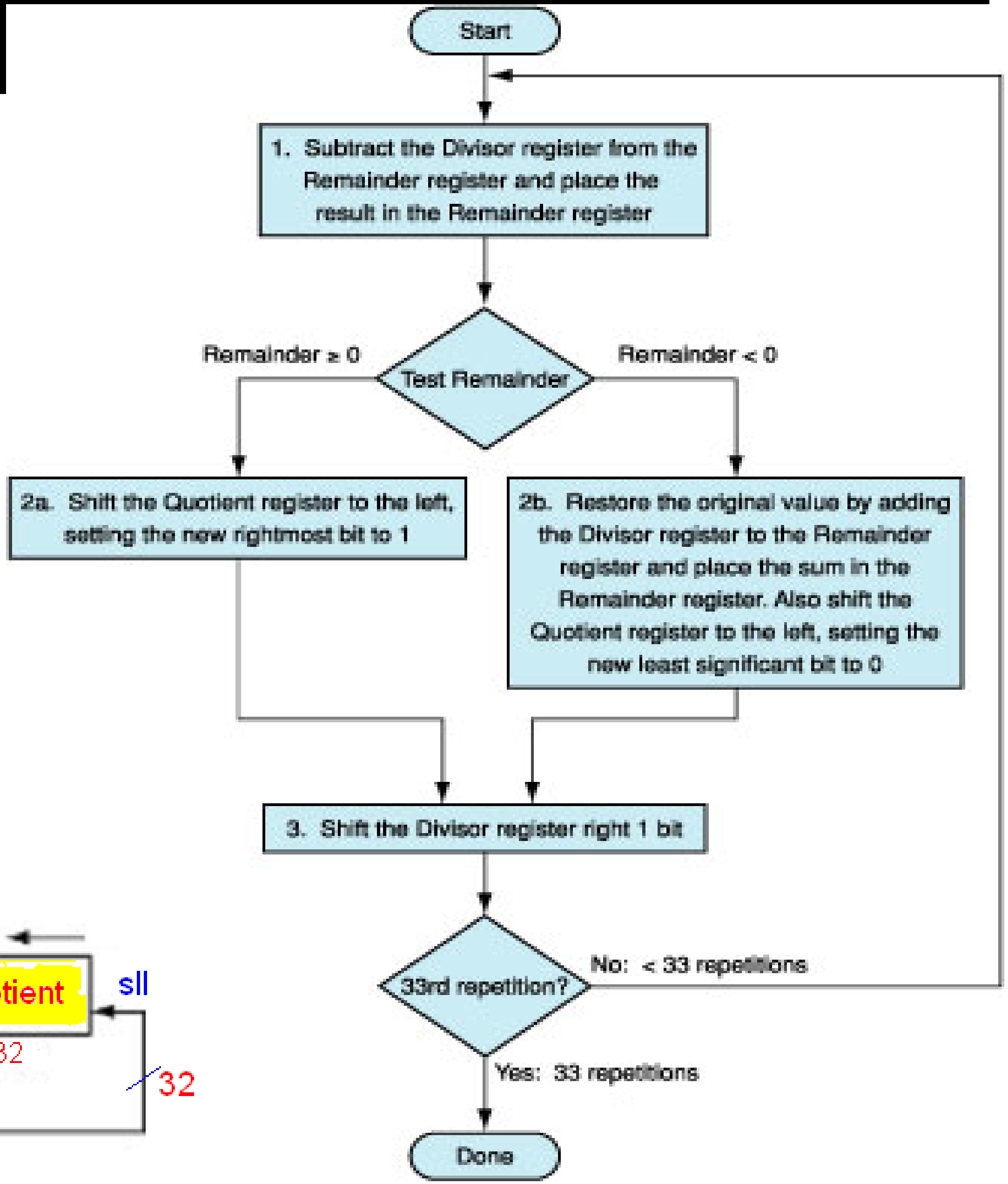
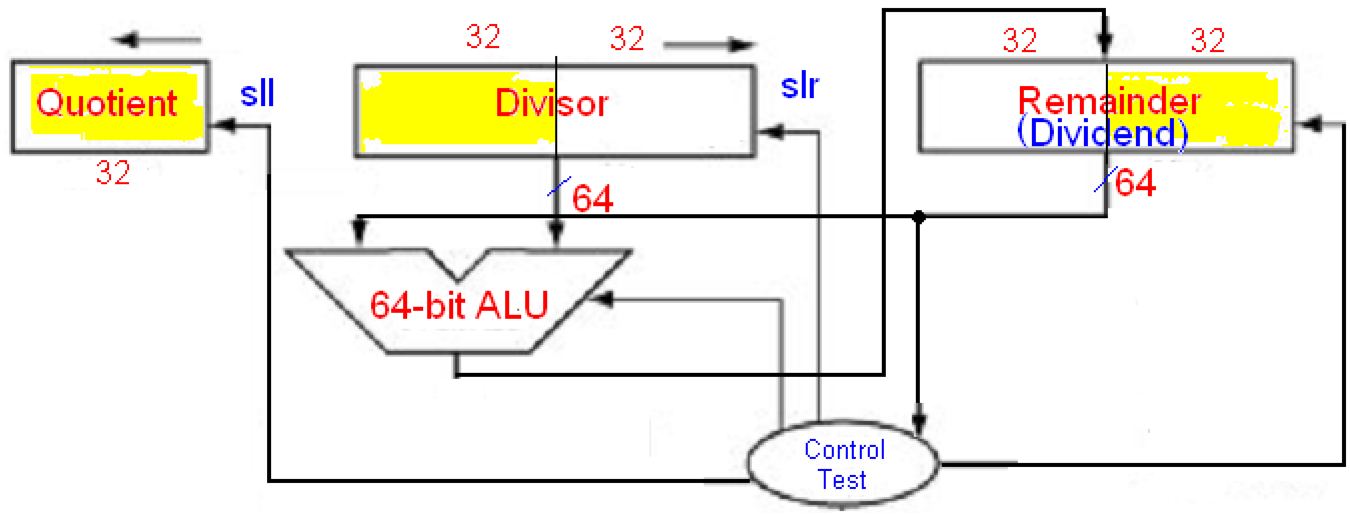


Illustration of Division Algorithm with 4-bit

0111/0010 example



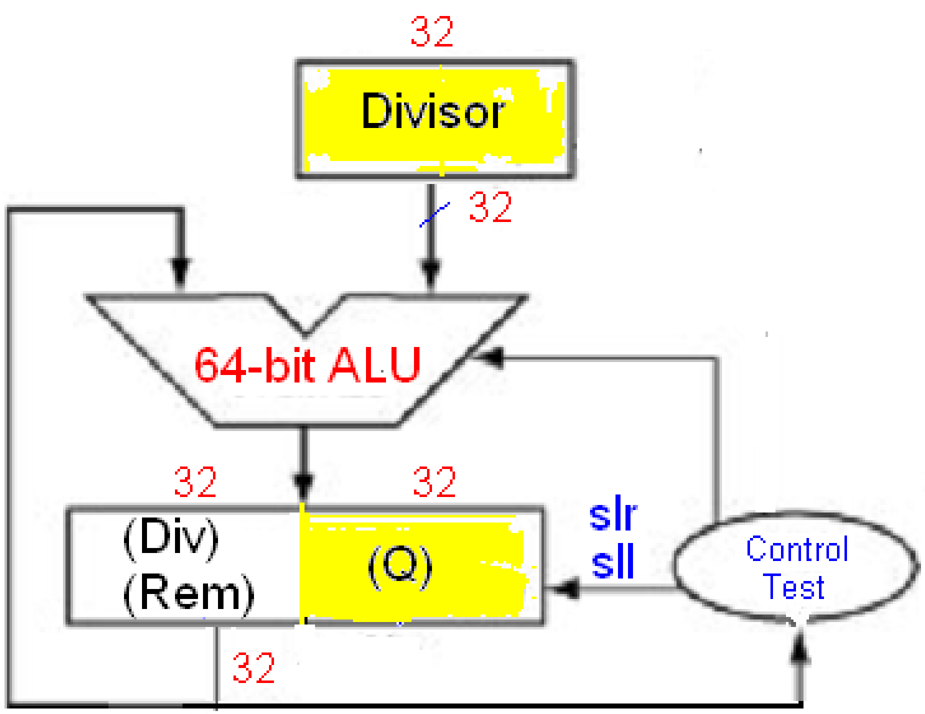

```
(1)Rem - Divisor
(2)If Rem<0,
     Rem=Rem(old)
     sll Q and Q0=0
    Else
     sll Q and Q0=1
(3) slr Divisor
```

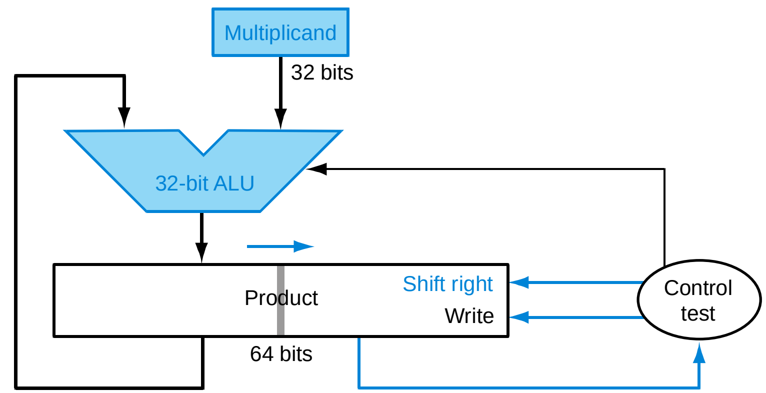
5 steps
(i.e., n+1 steps)

	Quotient	Divisor		Remainder	
0	0000	0010	0000	0000	0111
1	0000	0001	0000	0000	0111
2	0000	0000	1000	0000	0111
3	0000	0000	0100	0000	0111
4	0001	0000	0010	0000	0011
5	0011	0010	0001	0000	0001

Faster Division

- Do not shift Divisor
- Shift Remainder and Quotient at the same time
- Same as the multiplier?





Division in MIPS

- `div`: divide

```
div  $s2, $s3  #Lo = $s2/$s3
               #Hi = $s2 mod $s3
```

- `divu`: divide unsigned
- Result
  - Quotient ----> stored in Lo
  - Remainder ---> stored in Hi
- Fetching the results
  - `mflo` ---> for Quotient
  - `mfhi` ---> for Remainder

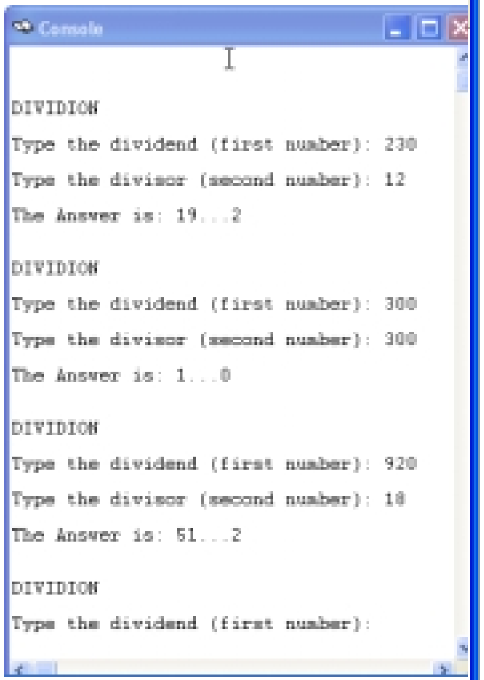
Division Example (p.188)

p188.asm - Notepad

File Edit Format View Help

```
#p188.asm
#Division in MIPS
#       f=g/h=Q---R
#   g <----$s2
#   h <----$s3
main:
        .data   0x10010000
        .asciiz "\n\n\nDIVIDION\n "  #Banner
        .data   0x10010100
        .asciiz "\nType the dividend (first number): " #
        .data   0x10010200
        .asciiz "\nType the divisor (second number): "
        .data   0x10010300
        .asciiz "\nThe Answer is: "   #msg3
        .data   0x10010400
        .asciiz "..."                  #separator
        .text
#Read varibales from key-in
again:  ori     $v0, $zero, 4
        lui     $a0, 0x1001
        ori     $a0, $a0, 0
        syscall                 #Print banner
        ori     $v0, $zero, 4
        lui     $a0, 0x1001
        or      $a0, $a0, 0x0100
        syscall                 #Print msg1
        ori     $v0, $zero, 5   #dividend
        syscall                 #now type-in is in v0
        or      $s2, $zero,$v0      #$s2 <---- dividend
```

Division - conti.




```
        or      $s2, $zero,$v0      #$s2 <---- dividend
        ori     $v0, $zero, 4
        lui     $a0, 0x1001
        ori     $a0, $a0, 0x0200
        syscall                 #Print msg2
        ori     $v0, $zero, 5   #read divisor
        syscall                 #now type-in is in v0
        or      $s3, $zero,$v0  #$s3 <---- divisor
        divu    $s2, $s3        #Lo=$s2/$s3
                                #Hi = remainder
        mfhi    $s0             #$s0 <---Remainder
        mflo    $s1             #$s1 <---Quotient
        ori     $v0, $zero, 4   #msg3
        lui     $a0, 0x1001             # Upper part of msg1
        ori     $a0, $a0, 0x0300        # now a0 has 1 word a
        syscall                 #Print
#print the result
        ori     $v0, $zero,1    #request for print
        or      $a0,  $zero, $s1        #result (lower part)
        syscall
        ori     $v0, $zero, 4
        lui     $a0, 0x1001
        ori     $a0, $a0, 0x0400
        syscall                 #...
        ori     $v0, $zero, 1
        or      $a0, $zero, $s0
        syscall
        j       main
```