Iho López Tobi
il16d

Assignment #5

(max = 95)

Read pages 178-196 in the *Computer Organization and Design* text. I have provided a set of notes ("Notes for Assignment #5) on this reading that can be found under Course Notes. Please refer to these notes as you carefully work through the assigned reading.

Afterwards, submit answers for the following problems:

1. Multiply 10_{10} by 11_{10} (the multiplier) using the hardware of Figure 3.3. Produce a table similar to Figure 3.6. As the text has done, use 4-bit (unsigned) numbers, rather than 32-bit numbers! (10 points)

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	1011 (11)	0000 1010 (10)	0000 0000
1	1a prod = prod + Mcand	1011	0000 1010	0000 1010
	2 : shift left Multipicand	1011	0001 0100	0000 1010
	3 : Shift Right Multiplier	0101	0001 0100	0000 1010
2	1a : prod = prod + Mcand	0101	0001 0100	0001 1110
	2 : Shift left Multipicand	0101	0010 1000	0001 1110
	3 : Shift Right Multipier	0010	0010 1000	0001 1110
3	1: 0 → No operation	0010	0010 1000	0001 1110
	2: Shift left Multipicand	0010	0101 0000	0001 1110
	3: shift Right Multipier	0001	0101 0000	0001 1110
4	1a: Prod = Prod + Multipicand	0001	0101 0000	0110 1110
	2: Shift left Multipicand	0001	1010 0000	0110 1110
	3: Shift Right Multipier	0000	1010 0000	0110 1110
5	1: 0 → no operation	0000	1010 0000	0110 1110
	2: Shift left Multipicand	0000	0100 0000	0110 .110
	3: Shift Right Multiplier	0000	0100 0000	0110 1110

Version 1 Figure 3.6 and 3.3

2. This time, multiply 11_{10} by 12_{10} (the multiplier). Use the refined version of the hardware given in Figure 3.5, producing a table similar to the one that appears in the course notes. Use 4-bit (unsigned) numbers. (10 points)

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	1011	1010	0000 0000
1	1a: prod = prod + Mcand (left half)	1011	1010	1010 0000
	2: Shift Right prod reg	1011	1010	0101 0000
	3: Shift Right Multiplier	0101	1010	0101 0000
2	1a prod + Mcand	0101	1010	1111 0000
	2: Shift Right prod	0101	1010	0111 1000
	3: shift Right multiplier	0010	1010	0111 1000
3	1: 0 → no operation	0010	1010	0111 1000
	2: Shift Right prod	0010	1010	0011 1100
	3: Shift Right multiplier	0001	1010	0011 1100
4	1a: prod = prod + Mcand	0001	1010	1101 1100
	2: shift Right prod	0001	1010	0110 1110
	3: shift Right multiplier	0000	1010	0110 1110
5	1: 0 → no operation	0000	1010	0110 1110
	2: shift Right prod	0000	1010	0011 0111
	3: shift Right multiplier	0000	1010	0011 0111

Version 2 Figure 3.5

3. Divide 14_{10} by 3_{10} using the hardware of Figure 3.8. Produce a table similar to Figure 3.10 (use my slightly modified algorithm that starts with Step 3 for the first iteration). Use 4-bit (unsigned) numbers. (10 points)

14/3 Quotient = 4 Remainder = (Dividend − Quotient × Divisor) = 14 − (4·3) = 2
Remainder = 2

Iteration	Step	Quotient	Divisor 3	Remainder
0	Initial values	0000	0011 0000	0000 1110
1	1: Rem = Rem − Div	0000	0011 0000	1101 1110
	2b shift Q left	0000	0011 0000	restore remainder 0000 1110
	3: shift Div right	0000	0001 1000	0000 1110
2	1: Rem = Rem − Div	0000	0001 1000	1111 0110
	2b: shift Q left	0000	0001 1000	restore rem 0000 1110
	3: shift Div right	0000	0000 1100	0000 1110
3	1: Rem = Rem − Div	0000	0000 1100	1 0000 0010
	2a: shift Q left add 1	0001	0000 1100	0000 0010
	3: Shift D right		0000 0110	0000 0010
4	1: Rem = Rem − Div	0001	0000 0110	1111 1100
	2b: shift Q left	0010	0000 0110	restore rem to step 3 0000 0010
	3: shift D right	0010	0000 0011	0000 0010
5	1: Rem = Rem − DIV	0010	0000 0011	1111 1111
	2b: shift Q left	0100	0000 0011	restore rem 0000 0010
	3: shift D Right	0100 = 4	0000 0001	0000 0010 = 2

Version 1 Figure 3.8 and 3.9

4. Divide 14_{10} by 3_{10} again. This time use the improved non-restoring version of the division algorithm. Produce a table like the one that appears in the course notes. Use 4-bit (unsigned) numbers. (10 points)

Iteration	Step	Divisor	Reminder
0	Initial values	0011	0 0000 1110
1	1 Rem = Rem - Div	0011	1 1101 1110
	2b: add div back shift left	0011	0 0001 1100
2	1 Rem = Rem - Div	0011	1 1110 1100
	2: Rem<0 => + Div	0011	0 0011 1000
3	1. Rem = Rem - Div	0011	0 0000 1000
	2: Sll Rem	0011	0 0001 0001
4	1. Rem = Rem - Div	0011	1 1110 0001
	2: Rem<0 => + Div	0011	0 0010 0010
5	1: Rem = Rem - Div	0011	1 1111 0010 2
	2: Rem<0 => + Div	0011	0 0100 0100 4

5. Consider the following sequence, which I'll refer to as the alternating Fibonacci sequence:

1 -1 2 -3 5 -8 13 …

Here $altfib_1 = 1$, $altfib_2 = -1$ and $altfib_n = altfib_{n-2} - altfib_{n-1}$ for $n > 2$. Write a MIPS program (call it **altfib.s**) that will produce and print numbers (5 per line) in the alternating Fibonacci sequence in such a way that the code detects when overflow takes place. The "offending" number should not be in your list of numbers, but you should display the bogus value that is produced [see my output; the next value in the list would have been 1134903170 – (-1836311903) = 1134903170 + 1836311903 = 2971215073, which is too large for a 32-bit 2's complement number; instead, it is interpreted as -1323752223]. You should use the elaboration on page 182 as a guide, but notice that you will need to alter things slightly since you are taking the difference of two numbers, not the sum. You might want to (carefully) use the "negu" instruction on page A-54. Here is output from my program:

```
# Iho Lopez Tobi
# altfib.s function for the reverse fibbonacci sequence
# 1 -1 2 -3 5 -8 13
# 1 -1 then n = (n-2) - (n-1) for n > 2

OVERFLOW:

    addi    $sp, $sp, -12    # save $ra and $s2-$s3
    sw    $ra, 8($sp)
    sw   $s3, 4($sp)
    sw   $s2, 0($sp)
    subu $t0, $s2, $s3          # overflow check
    negu $s3, $s3             # negate $s3
    xor $t3, $s2, $s3          # check signs
    slt $t3, $t3, $zero
    bne $t3, $zero, RESTORE
    xor $t3, $t0, $s2          # sign of sums match
    slt $t3, $t3, $zero        # if signs match set $t3 to 1
    bne $t3, $zero, out          # overflow
    lw    $s2, 0($sp)          # restore values from stack
    lw   $s3, 4($sp)
    lw   $ra, 8($sp)
    addi    $sp, $sp, 12
    jr $ra


RESTORE:
    lw    $s2, 0($sp)           # restore values
    lw   $s3, 4($sp)
    lw   $ra, 8($sp)
```

```
    addi    $sp, $sp, 12
    jr $ra

ALTFIB:
    addi    $sp, $sp, -20       # save $ra and $s0-$s3
    sw    $ra, 16($sp)
    sw   $s3, 12($sp)
    sw   $s2, 8($sp)
    sw   $s1, 4($sp)
    sw    $s0, 0($sp)
    addi $s2, $s2, 1            # initialize first and second terms
    addi $s3, $s3, 0
    move $s0, $a0
    li $t1, 0

LOOP:
    beq $t1, 5, NEW
    la    $a0, space            # print a space after each number
    li    $v0, 4
    syscall
    addi $s1, $s0, 0
    jal OVERFLOW
    sub $t0, $s2, $s3           # subtract the last two nums to get next
    addi $s2, $s3, 0            # shift nums for calculation
    addi $s3, $t0, 0
    move    $a0, $t0
    li    $v0, 1
    syscall
    addi $t1, $t1, 1
    j LOOP

NEW:
    la    $a0, newline          # print a new line after 5th number
    li    $v0, 4
    syscall
    move $a0, $s1               # move next number in sequence to return value
    li $t1, 0
    j LOOP                      # jump back to for loop

EXIT:
    la    $a0, newlinw          # print a new line
    li    $v0, 4
    syscall
    move $v0, $s1               # move next number in sequence to return value
    lw    $s0, 0($sp)           # restore values from stack
    lw   $s1, 4($sp)
    lw   $s2, 8($sp)
    lw   $s3, 12($sp)
    lw    $ra, 16($sp)
    addi    $sp, $sp, 20
    jr    $ra
```

```
main:
    la    $a0, welc
    li    $v0, 4
    syscall
    jal ALTFIB

out:

    la    $a0, newline
    li    $v0, 4
    syscall

    la    $a0, overflow
    li    $v0, 4
    syscall

    move    $a0, $t0
    li    $v0, 1
    syscall

    li    $v0, 10          # exit from the program
    syscall

.data
welc: .asciiz "Here are the alternating Fibonacci numbers that I produced:\n\n"
overflow: .asciiz "\nValue causing overflow: "
space: .asciiz    " "
newline: .asciiz "
```

```
Here are the alternating Fibonacci numbers that I produced:

1 -1 2 -3 5

-8 13 -21 34 -55

89 -144 233 -377 610

-987 1597 -2584 4181 -6765

10946 -17711 28657 -46368 75025

-121393 196418 -317811 514229 -832040

1346269 -2178309 3524578 -5702887 9227465

-14930352 24157817 -39088169 63245986 -102334155
165580141 -267914296 433494437 -701408733 1134903170
-1836311903

Value causing overflow = -1323752223
```

Don't forget to document your code! Submit a separate file called **altfib.s** as well as placing your code in this assignment submission; the Mentor will clarify what I mean by this. (50 points)

6. Recall that in question 9 in Assignment #2, we displayed the values of fact(n) for various values of n. We saw that you could only go up to a certain value of n and still expect to get a valid result. We also saw that eventually (as the value of n increased) the values being returned were simply zero. Now that you know how multiplication works, explain both of these phenomena (incorrect result and zero result). There is a way that you could have predicted the first value of n that would produce a result of zero. Explain that process. (5 points)

   To avoid overflow MIPS use arbitrary precision on arithmetic operations. This is naturally a fixed precision arithmetic, when the results from multiplication do not fit into the set precision the result is set to the nearest possible value. Since the results have been neared to the most representable value this results into a negative number. When results are larger they turn into 0.

Your assignment is due by 11:59 PM (Eastern Time) on the assignment due date (consult Course Calendar on course website).