Laboratory 6: Prelab Exercise #2

Date ______________________ Section ______________________

Name ____

Integer Arithmetic in MIPS (cont' d)

During this prelab exercise you will become familiar with integer multiplication and division in MIPS.

Integer multiplication

Integer multiplication can be done using both signed and unsigned numbers. The architecture specifies two special registers, **`hi`** and **`lo`** (32 bit each), which are the destination for integer multiplication and division.

Multiplying two n-bit unsigned integers may yield a result that requires $2 \cdot n$ bits to be represented.

Ex 1:

Assume 2 bit unsigned integers. Then the largest integer that can be represented using 2 bits is 3 (its binary representation is 11).

	Multiplication	Comment
Decimal	$3 \cdot 3 = 9$	
Binary	$11 \cdot 11 = 1101$	Result requires 4 bits for representation

■

For signed multiplication the two numbers to be multiplied are in 2's complement representation. The result is also in 2's complement representation. The result of a signed multiplication may require $2 \cdot n - 1$ bits for representation: there are $2 \cdot (n - 1)$ bits for the magnitude and one bit for the sign.

Native instructions for multiplication

Instruction	Comment
`mult Rsrc1, Rsrc2`	Multiply the signed numbers in `Rsrc1` and `Rsrc2`. The higher 32 bits of the result go in register **`hi`**. The lower 32 bits of the result go in register **`lo`**
`multu Rsrc1, Rsrc2`	Multiply the unsigned numbers in `Rsrc1` and `Rsrc2`. The higher 32 bits of the result go in register **`hi`**. The lower 32 bits of the result go in register **`lo`**

Since both signed and unsigned integer multiplication never require more than 64 bits for the result, an overflow will never occur during integer multiplication using the native multiply instructions.

The virtual machine provides multiply instructions whose result is the size of a word (the destination register is some general purpose integer register). Since the destination register is only 32 bit wide, these synthetic instructions may overflow.

Synthetic instructions for multiplication

Instruction	Comment
`mul Rdest, Rsrc1, Rsrc2`	Multiply the signed numbers in `Rsrc1` and `Rsrc2`. The lower 32 bits of the result go in register `Rdest`.
`mulo Rdest, Rsrc1, Rsrc2`	Multiply the signed numbers in `Rsrc1` and `Rsrc2`. The lower 32 bits of the result go in register `Rdest`. Signal overflow
`mulou Rdest, Rsrc1, Rsrc2`	Multiply the unsigned numbers in `Rsrc1` and `Rsrc2`. The lower 32 bits of the result go in register `Rdest`. Signal overflow

Integer division

The result of an integer division is a quotient (stored in register **`lo`**) and a remainder (stored in register **`hi`**), both integer numbers.

- There is one complication related to integer division: the sign of the remainder. There are two approaches to this problem:
- follow the division theorem from mathematics
- use the computer science view

By dividing an integer *DD* called dividend, by a *positive* integer *DR* called divisor, the following conditions are true (the division theorem):

$$DD = DR \cdot Q + RM \qquad (1)$$
$$0 \le RM < DR \qquad (2)$$

where *Q* (the quotient) and *RM* (the remainder) are unique integers. Note that the remainder is always positive.

Ex 2:

What is the result of dividing -22 by 3?

DD = -22, DR = 3, Q = -8, RM = +2

Note that relation (1) could be satisfied by Q' = -7, RM' = -1. However this solution is not valid since it does not satisfy the condition that the remainder must be positive (2).

What is the result of dividing +22 by 3?

DD = +22, DR = 3, Q = +7, RM = +1

In this approach $-\left(\frac{22}{7}\right) \neq \left(\frac{-22}{7}\right)$. If this was implemented on computers, then programming would definitely be more fun than it is now. ■

The way division is usually implemented (the ‘computer science way’), the following relations are true:

$$DD = DR \cdot Q + RM \qquad (3)$$
$$|RM| < |DR| \qquad (4)$$
$$sign(RM) = sign(DD) \qquad (5)$$

If the dividend and the divisor have the same sign then the quotient is positive; otherwise it is negative. A nonzero remainder has always the same sign as the dividend.

Q 1:

What are the results of the following integer divisions?

Dividend	Divisor	Quotient	Remainder
22	7		
-22	7		
22	-7		
-22	-7		

In MIPS, if one of the operands in a division is negative, then the remainder is **unspecified**. The SPIM simulator will return a result based on the conventions on the machine it is run on.

To obtain the correct result for division, extra steps need to be performed:

- convert both operand to positive integers
- perform the division
- set the result to its actual representation based on the initial signs of the dividend and divisor.

A signed division may overflow since the 2's complement representation of integers is asymmetric: there is one negative number more than positive numbers. Note however that the overflowing division does not signal the overflow. It is the compiler's or programmer's task to generate proper code that detects the overflow.

Dividing an integer by zero is an illegal operation. However, even if the divisor is zero, the division will report no error. It is again the compiler's or programmer's task to generate proper code that detects the illegal operation.

Native instructions for division

Instruction	Comment
`div Rsrc1, Rsrc2`	Divide the signed integer in `Rsrc1` by the signed integer in `Rsrc2`. The quotient (32 bits) goes in register **`lo`**. The remainder goes in register **`hi`**. Can overflow.

Native instructions for division

Instruction	Comment
`divu Rsrc1, Rsrc2`	Divide the unsigned integer in `Rsrc1` by the unsigned integer in `Rsrc2`. The quotient (32 bits) goes in register **`lo`**. The remainder goes in register **`hi`**. Does not overflow.

The virtual machine provides divide instructions whose result is stored in a general purpose register.

Synthetic instructions for division

Instruction	Comment
`div Rdest, Rsrc1, Rsrc2`	Divide the signed integer in `Rsrc1` by the signed integer in `Rsrc2`. Store the quotient (32 bits) in register `Rdest`. Can overflow.
`divu Rdest, Rsrc1, Rsrc2`	Divide the unsigned integer in `Rsrc1` by the unsigned integer in `Rsrc2`. Store the quotient in Rdest. Does not overflow.
`rem Rdest, Rsrc1, Rsrc2`	Divide the signed integer in `Rsrc1` by the signed integer in `Rsrc2`. Store the remainder[a] (32 bits) in register `Rdest`. Can overflow.
`remu Rdest, Rsrc1, Rsrc2`	Divide the signed integer in `Rsrc1` by the signed integer in `Rsrc2`. Store the remainder[a] (32 bits) in register `Rdest`. Does not overflow.

a. If one of the operands is negative then the remainder is undefined.

Step 1

Write a program, called *lab6.2.c* which:

- declares two integer variables, `a` and `b`. The initial value for `a` and `b` is the maximum possible value for an integer (INT_MAX if you include the *limits.h* file)
- multiplies `a` and `b` and prints the result

Q 2:

Do you think the multiplication in *lab6.2.c* overflows? Explain.

Step 2

Compile and run the executable created from *lab6.2.c*.

`a`	`b`	Printed result

Q 3:

Is the result your program prints correct? Explain.

Most compilers generate code that does not signal integer overflows. An unsuspecting programmer may easily generate incorrect output without any error message to indicate it.

Step 3

Create the program *lab6.5.asm* based on the following description:

- in 'main' prompts the user to enter two integers; store them in **`$t0`** and **`$t1`**
- multiplies the two numbers using the native signed instruction
- prints the result

Run the program and fill out the following test plan. Use the `print` command in the simulator to see the content of registers **`hi`** and **`lo`**. In the last two rows of the table enter the numbers that represent the largest possible signed integer and the smallest possible one respectively.

Test plan for *lab6.5.asm* (signed multiplication): register's content

Operand_1	Operand_2	Register **`hi`** (hexadecimal)	Register **`lo`** (hexadecimal)
2	1		
2	-1		
262,144 (=2^{18})	16,384 (=2^{14})		

Q 4:

In the second row of this test plan the register **`hi`** is all 1s. Why is that?

Step 4

Run the program again and fill out the next test plan. Use the same numbers you have used in the previous step.

Test plan for *lab6.5.asm* (signed multiplication): printed results

Operand_1	Operand_2	Expected result	Printed result

Q 5:

Some of the results your program prints are not correct. Why?

Step 5

Create the program *lab6.6.asm* based on the following description:

- in 'main' prompts the user to enter two integers; store them in **`$t0`** and **`$t1`**
- multiplies the two numbers using the native unsigned instruction

Run the program and fill out the following test plan.

Test plan for *lab6.6.asm* (unsigned multiplication): register's content

Operand_1	Operand_2	Register **`hi`** (hexadecimal)	Register **`lo`** (hexadecimal)
`2`	`1`		
2	-1		
262,144 (=2^{18})	16,384 (=2^{14})		

Q 6:

What numbers are in reality multiplied when the input numbers are 2 and -1? Remember that a negative integer will mean a large positive unsigned integer.

Number	Bit pattern	Value if bit pattern represents an unsigned integer
`2`		

Number	Bit pattern	Value if bit pattern represents an unsigned integer
`-1`		

Step 6

Create the program *lab6.7.asm* based on the following description:

- in 'main' prompts the user to enter two integers; store them in **`$t0`** and **`$t1`**
- divides the first number by the second one, using the native signed instruction
- prints the quotient and the remainder

Run the program and fill out the following test plan. Use small numbers that would generate a nonzero remainder. For the last row use a pair of numbers that would make the division overflow. Note that division by zero (which is an illegal operation) and overflow are different things. In the 'Comment' column mark with a star those cases where an error is reported.

Test plan for *lab6.7.asm* **(signed division)**

Operand_1	Operand_2	Printed Quotient	Printed remainder	Comment
+	+			
+	-			
-	+			
-	-			
	0			

Highlight those cells that contain incorrect results.

Step 7

Create the program *lab6.8.asm* based on the following description:

- in 'main' prompts the user to enter two integers; store them in **`$t0`** and **`$t1`**
- divides the first number by the second one, using the native unsigned instruction
- prints the quotient and the remainder

Run the program and fill out the following test plan. Use small numbers that would generate a nonzero remainder. For the last row use a pair of numbers that would make the division overflow. In the 'Comment' column mark with a star those cases where an error is reported.

Test plan for *lab6.7.asm* (unsigned division)

Operand_1	Operand_2	Printed Quotient	Printed remainder	Comment
+	+			
+	-			
-	+			
-1	+2			
-	-			
	0			

Highlight divisions where incorrect results are printed.

Q 7:

What numbers are in reality divided when the input numbers are -1 and +2?

Number	Bit pattern	Value if bit pattern represents an unsigned integer
`-1`		
`+2`		