Arithmetic Logic

JACOB LEPERE
San Jose State University
jacob.lepere@sjsu.edu

Abstract: This report documents the logical operations performed for addition, subtraction, multiplication, and division arithmetic using MARS simulation tool. A test program is used to prove the results.

I. INTRODUCTION

This document explains the procedures necessary to download MARS simulation tool, upload the required files, and run the program. The procedures will also be demonstrated for each arithmetic operation. Our objective is to prove logical operations equal normal operations.

II. STEPS TO INSTALL MARS IDE

MARS is an open-source, lightweight interactive development environment (IDE) for MIPS assembly programming. MARS was developed by Pete Sanderson and Ken Volmer of Missouri State University to be used across campuses and in tandem with Patterson and Hennessy's *Computer Organization and Design*.

Following are the steps to get the free MARS software.

1) Open the following link in your browser http://courses.missouristate.edu/KenVollmar/mars/download.htm

2) Click the Download MARS at the top of the page to begin installation. Note that you may need to download or update the Java SDK on your computer. If this is the case, click the Download Java button near the bottom of the page.

3) Locate the .jar file in your downloads folder and move it to somewhere more appropriate.

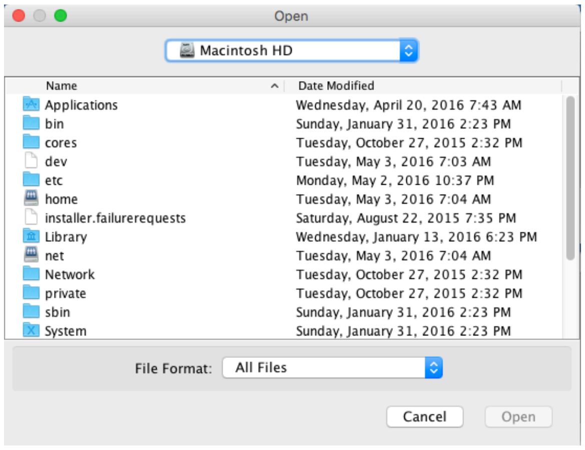
4) You should now be able to run the MIPS assembly language IDE.

III. STEPS TO SIMULATE THE PROJECT

This section contains the information required to successfully load and run the proper files onto MARS. Follow the ensuing steps to simulate addition, subtraction, multiplication, and division logical arithmetic on MARS. Note that the graphical interface may be slightly different due to updates to MARS since this document has been published.

1) Download the zip file attached to this report. Place and unzip this file into a new folder in an appropriate location. The zip consists of three .asm files.
- The first file 'cs47_proj_macros.asm' contains the required macros to run the program. This file deals mostly with common print operations to make the code easier to read and understand.
- The second file 'cs47_proj_proc.asm' contains the required protocols. Here, you will find protocols for each arithmetic operation as well as a printf protocol.
- The third file 'proj-auto-test.asm' contains the code to preform tests on the arithmetic protocols. This is where the main label is also found.

2) Open MARS and in the upper left hand corner, navigate to File > Open. You will be prompted a similar window as shown on Figure 3.1.



Open

Macintosh HD

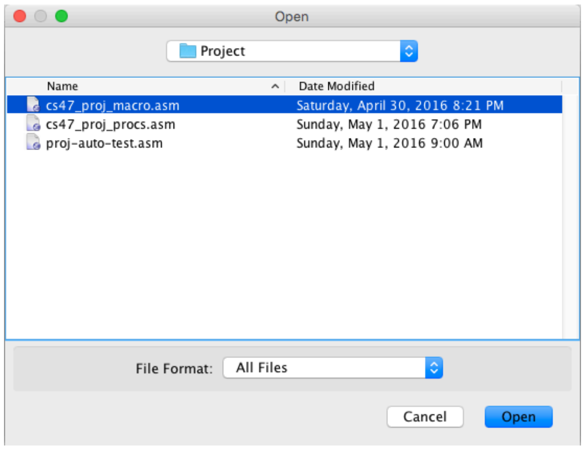
Name	Date Modified
Applications	Wednesday, April 20, 2016 7:43 AM
bin	Sunday, January 31, 2016 2:23 PM
cores	Tuesday, October 27, 2015 2:32 PM
dev	Tuesday, May 3, 2016 7:03 AM
etc	Monday, May 2, 2016 10:37 PM
home	Tuesday, May 3, 2016 7:04 AM
installer.failurerequests	Saturday, August 22, 2015 7:35 PM
Library	Wednesday, January 13, 2016 6:23 PM
net	Tuesday, May 3, 2016 7:04 AM
Network	Tuesday, October 27, 2015 2:32 PM
private	Tuesday, October 27, 2015 2:32 PM
sbin	Sunday, January 31, 2016 2:23 PM
System	Sunday, January 31, 2016 2:23 PM

File Format: All Files

Cancel Open

Figure 3.1 Select Folder Window

3) Navigate to the folder containing the three .asm files. Once in the correct folder, select one of the three files as shown on Figure 3.2. Click Open once the file has been selected.



Open

Project

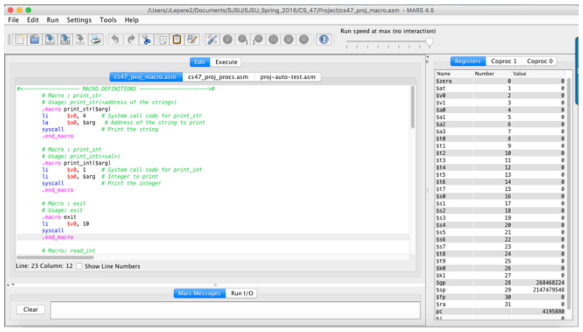
Name	Date Modified
cs47_proj_macro.asm	Saturday, April 30, 2016 8:21 PM
cs47_proj_procs.asm	Sunday, May 1, 2016 7:06 PM
proj-auto-test.asm	Sunday, May 1, 2016 9:00 AM

File Format: All Files

Cancel Open

Figure 3.2 Selecting A File

4) Continue this process until you have successfully loaded three individual files onto MARS. Figure 2.3 documents the correct graphical interface after this step. Notice the three tabs under the Edit and Execute tabs. These are the three uploaded files.



/Users/JLepere2/Documents/SJSU/SJSU_Spring_2016/CS_47/Project/cs47_proj_macro.asm - MARS 4.5

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

cs47_proj_macro.asm cs47_proj_procs.asm proj-auto-test.asm

```
#<------------------ MACRO DEFINITIONS ---------------------->#
        # Macro : print_str
        # Usage: print_str(<address of the string>)
        .macro print_str($arg)
        li      $v0, 4      # System call code for print_str
        la      $a0, $arg   # Address of the string to print
        syscall             # Print the string
        .end_macro

        # Macro : print_int
        # Usage: print_int(<val>)
        .macro print_int($arg)
        li      $v0, 1      # System call code for print_int
        li      $a0, $arg   # Integer to print
        syscall             # Print the integer
        .end_macro

        # Macro : exit
        # Usage: exit
        .macro exit
        li      $v0, 10
        syscall
        .end_macro

        # Macro: read_int
```

Line: 23 Column: 12 Show Line Numbers

Mars Messages Run I/O

Clear

Registers Coproc 1 Coproc 0

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4195880
hi		0

Figure 3.3 Interface After Uploaded Files

5) Press the assemble button, as shown on Figure 3.4, located on the tool bar above the Edit and Execute tabs.

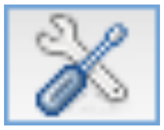


Figure 3.4 Assemble Button

6) Now that the files are assembled, press the Run button, as shown on Figure 3.5. This button is located just to the right of the assemble button.

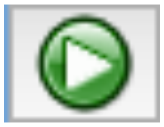


Figure 3.5 Run Button

7) Allow the program a few seconds to run. At the bottom of the screen you should see two tabs, Mars Messages and Run I/O. The final results will be under the Run I/O tab.

IV. Addition and Subtraction Logic

The similarities of addition and subtraction allow us to preform both procedures on one circuit by manipulating he “Carry bit” (see Figure 4.1). The Carry bit is initialized to 0 for addition and 1 for subtraction. The Carry bit is then set as the algorithm runs pending on the inputs of the two terms.

	1	1	0	0	Carry Bit
	0	1	1	1	A
+	0	1	1	0	B
0	1	1	0	1	Result

Figure 4.1. Addition/Subtraction Carry bit

Figure 4.2 represents the algorithm for addition and subtraction logic. Steps are preformed for each index less than thirty-two to retrieve the result and carry bit.

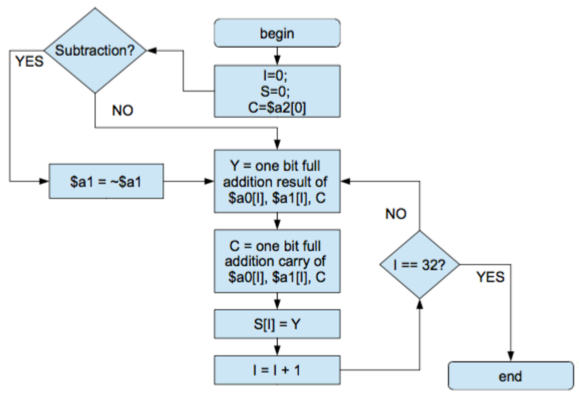

begin
I=0;
S=0;
C=$a2[0]
Subtraction?
YES
NO
$a1 = ~$a1
Y = one bit full addition result of $a0[I], $a1[I], C
C = one bit full addition carry of $a0[I], $a1[I], C
S[I] = Y
I = I + 1
I == 32?
NO
YES
end

Figure 4.2. Addition and Subtraction Logic Algorithm (Patra, 2016)

The algorithm then sets the final result to the result from the previous step at the index bit. Once we have initialized the result at each bit, the loop ends and our result is stored in register $v0.

Register	Usage
$t0	$a0[I]
$t1	$a1[I]
$t2	Index (I)
$t3	Carry bit (C)
$t4	Stores $a0[I] + $a1[I] + C
$t5	Y
$t6	Mask register
$t7	Stores the value 0x1
$a0	The first term
$a1	The second term
$a2	0x00000000 if addition 0xFFFFFFFF if subtraction
$v0	The result (S)
$v1	The final carry bit

Figure 4.3. Registers Used for Addition and Subtraction Logic

V. Multiplication Logic

Multiplication logic gives rise to a few more complications. First, multiplying a 32-bit number by another 32-bit number gives us a 64-bit result. Therefor, we must have a High (H) and a Low (L) register for our answer.

			1	0	1	MCND
		x	0	1	1	MPLR
			1	0	1	
		1	0	1	0	
+	0	0	0	0	0	
0	0	1	1	1	1	Result

Figure 5.1. Multiplication Logic

By taking a closer look at Figure 5.1, we see that the result is an addition of the multiplicand (MCND) shifted some amount. The shift amount is equal to the bit number of the multiplier (MPLR) that we are using to calculate the solution. If the bit number is 0, the result for that bit will be zero. Therefor, we need only to take the sum of those results when the MPLR bit is 1.

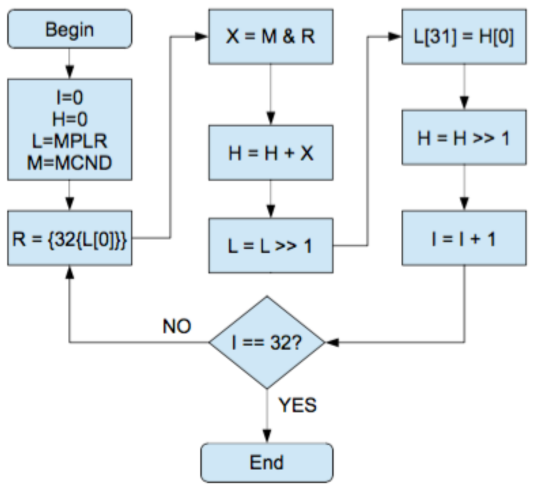


Figure 5.2. Multiplication Logic Algorithm (Patra, 2016)

Negative numbers give us another problem. It is well established that to get a negative number after multiplication, we must have exactly one positive and one negative number. Figure 5.3 represents this theory logically. Therefor, before we run the multiplication logic algorithm we must take into account the sign of the MPLR and MCND. We invert these variable if they are negative, and keep this into account for our final result.

MPLR	MCND	Result
0	0	0
0	1	1
1	0	1
1	1	0

Figure 5.3. Multiplication of Positive and Negative Numbers. (1 is negative)

Therefor, the result of the XOR operation on the sign bits of the MPLR and MCND will tell us whether or not the final result is positive or negative. We can then quickly invert the final result by means of the 2's complement notation.

Register	Usage
$a0	Multiplicand (MCND). Initial
$a1	Multiplier (MPLR). Initial
$t0	Index (I)
$t1	High Value (H)
$t2	Low Value (L)
$t3	M
$t4	X
$t5	Stores $t1[0] (H[0])
$t6	Stores the value 0x1F (31)
$t7	Mask register
$a0	Holder for L[0]
$v0	Holder for R
$v0	Final Low Result
$v1	Final High Result

Figure 5.4. Registers Used for Multiplication Logic

VI. Division

Division Logic is similar to multiplication logic, as far as needing two separate registers for the result. In this case, we need a register for the quotient (Q) and the remainder (R). We manipulate the registers by shifting, extracting, and inserting bits. Figure 6.1 explains the algorithm for division logic.

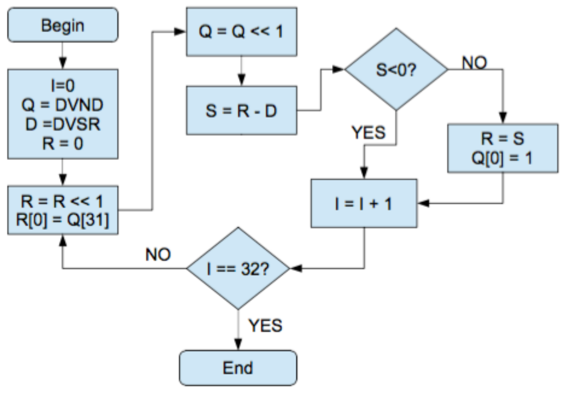


Figure 6.1. Division Logic Algorithm (Patra, 2016)

The method to solve the issue of multiplying negative numbers applies to division logic as well. To find the sign of the quotient, we XOR the sign bit of the dividend and the divisor. To find the sign of the remainder, we only need to extract the sign of the dividend. If the result is one, we need to invert the quotient or remainder respectively using 2's complement.

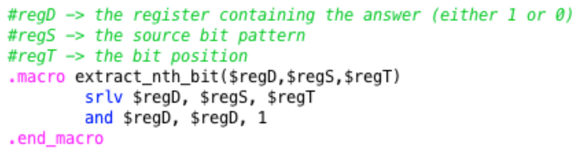
Register	Usage
$a0	Dividend (DVND). Initial
$a1	Divisor (DVSR). Initial
$s0	Index (I)
$s1	Quotient (Q)
$s2	Divisor (D)
$s3	Remainder (R)
$s4	Result of R – D. (S)
$s5	Holder for $s1[0x1F] (Q[31])
$s6	Stores 0x1F. (31)

$s7	Mask Register
$t8	Stores 0x1
$v0	Final Quotient Result
$v1	Final Remainder Result

Figure 6.2. Registers Used for Division Logic

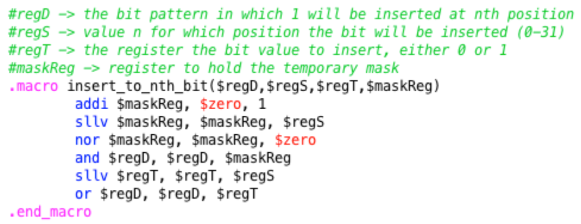
VII. Helper Macros

A macro allows the programmer to save time and space when preforming the same lines of code multiple times. Rather than write the same lines over and over, one can write a macro with different parameters to preform the same task. Therefor, visually you preform the same task with only one line of code. This makes the code essentially shorter and easier to read.



```
#regD -> the register containing the answer (either 1 or 0)
#regS -> the source bit pattern
#regT -> the bit position
.macro extract_nth_bit($regD,$regS,$regT)
        srlv $regD, $regS, $regT
        and $regD, $regD, 1
.end_macro
```

Figure 7.1. Extract Nth Bit Macro



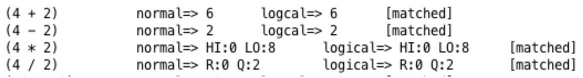
```
#regD -> the bit pattern in which 1 will be inserted at nth position
#regS -> value n for which position the bit will be inserted (0-31)
#regT -> the register the bit value to insert, either 0 or 1
#maskReg -> register to hold the temporary mask
.macro insert_to_nth_bit($regD,$regS,$regT,$maskReg)
        addi $maskReg, $zero, 1
        sllv $maskReg, $maskReg, $regS
        nor $maskReg, $maskReg, $zero
        and $regD, $regD, $maskReg
        sllv $regT, $regT, $regS
        or $regD, $regD, $regT
.end_macro
```

Figure 7.2. Insert to Nth Bit Macro

To preform arithmetic logically, we constantly need to remove and insert bits to a given register. Therefor, it is beneficial to create a macro to preform these tasks.

VIII. Testing

The file titled "proj-auto-test.asm" contains the code to test the logic operations. The program provides 10 sets of two terms that are each tested on addition, subtraction, multiplication, and division logic. The logical result is than compared to the normal result, that is, using "+", "-", "x", and "/" on the terms.

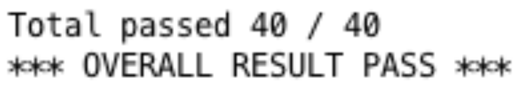


```
(4 + 2)         normal=> 6      logcal=> 6      [matched]
(4 - 2)         normal=> 2      logcal=> 2      [matched]
(4 * 2)         normal=> HI:0 LO:8      logical=> HI:0 LO:8     [matched]
(4 / 2)         normal=> R:0 Q:2        logical=> R:0 Q:2       [matched]
```

Figure 8.1. Sample output for values 4 and 2

IX. Conclusion

After all of the 10 sets of terms have been run through the logical operations, the final result is displayed. This shows the total correct / total tested.



```
Total passed 40 / 40
*** OVERALL RESULT PASS ***
```

Figure 9.1. The Final Result

X. Resources

Patra, K. CS47 – Lecture 18 [PDF document]. Retrieved from Lecture Notes Online Web site: https://sjsu.instructure.com/courses/1185206

Patra, K. CS47 – Lecture 19 [PDF document]. Retrieved from Lecture Notes Online Web site: https://sjsu.instructure.com/courses/1185206

Patra, K. CS47 – Lecture 20 [PDF document]. Retrieved from Lecture Notes Online Web site: https://sjsu.instructure.com/courses/1185206

Patterson, D. A., & Hennessy, J. L. (2009). Computer organization and design: The hardware/software interface. Burlington, MA: Morgan Kaufmann.

Vollmar, K. (2014, October 28). MARS MIPS simulator - Missouri State University. Retrieved May 05, 2016, from http://courses.missouristate.edu/KenVollmar/mars/download.htm