Project 4 Lab Report

1.0 Program Input/Output

The first program we created took in three inputs from the user. All three inputs were signed numbers that would be used to complete the program. The program then utilized these inputs in a function. The value of this function was then output back out to the user. The only other output let the user know if there was overflow or no overflow as a result of the function. The second program prompted the user to input multiple different integers for data on two different processors. This data included the Instruction Count, the CPI, and the Clock Rate for each one respectively. The program collected this by outputting individual statements for each individual value. The final output of the program relied on which processor's CPU Time was greater and by how much. The third program we created only involved a single output, which was the sum of the array established in the program. We took in no input from the user.

2.0 Program Design

I began the first program by asking the user to input 3 different values for 3 variables. I then utilized mathematical operations to perform the equation put forth in the report. by the equation. I then printed the solution to the equation back out the user. I started the second program by first storing the CPU Time and “Zero” into floating point registers. I then prompted the user for all of the values I needed to complete the equation for Processor A. I then stored these values into individual floating point registers. I then jumped to the Calculate function, which served the purpose of completing the operations within the equation. We then returned to the original jump location and continued on. We then moved on to collect and store the values for the second processor, B, into floating point registers. The mathematical operations were conducted the same way as for processor A, utilizing a jump and return. The program then moved to the higherValue function in order to compare the two calculated CPU Times. The program did this by utilizing a single precision comparison in order to find which one was greater. Whichever one was established to be bigger would be established as the larger value in the program. This larger value would be divided by the lower value. This calculated value would be printed back out to the user in order to identify how much greater the bigger one was then the smaller one. The third program started by establishing an array with values along with a size for the array. This array was then taken to a loop function. This function utilized a stack in order to add each individual value from the array onto the prior one. Once the loop had finished, the jump returned to its prior position. The calculated sum was then returned to the user prefaced with a statement informing the user of the operation and situation.

3.0 Symbol Table

Registers	Usage
$v0	**Contains address of allocated memory**
$a0	**The number of bytes to be allocated**
$s0	**Used to store input along with output**
$s1	**Used to store input along with outputs**
$s2	**Used to store user input along with outputs**
$s3	**Used to store input along with output**
$s4	**Used to store input along with output**
$s5	**Used to store input along with output**
$s6	**Used to store input along with output**
$t0	**Primarily used to store variables, sums, products, differences, and quotients between registers**

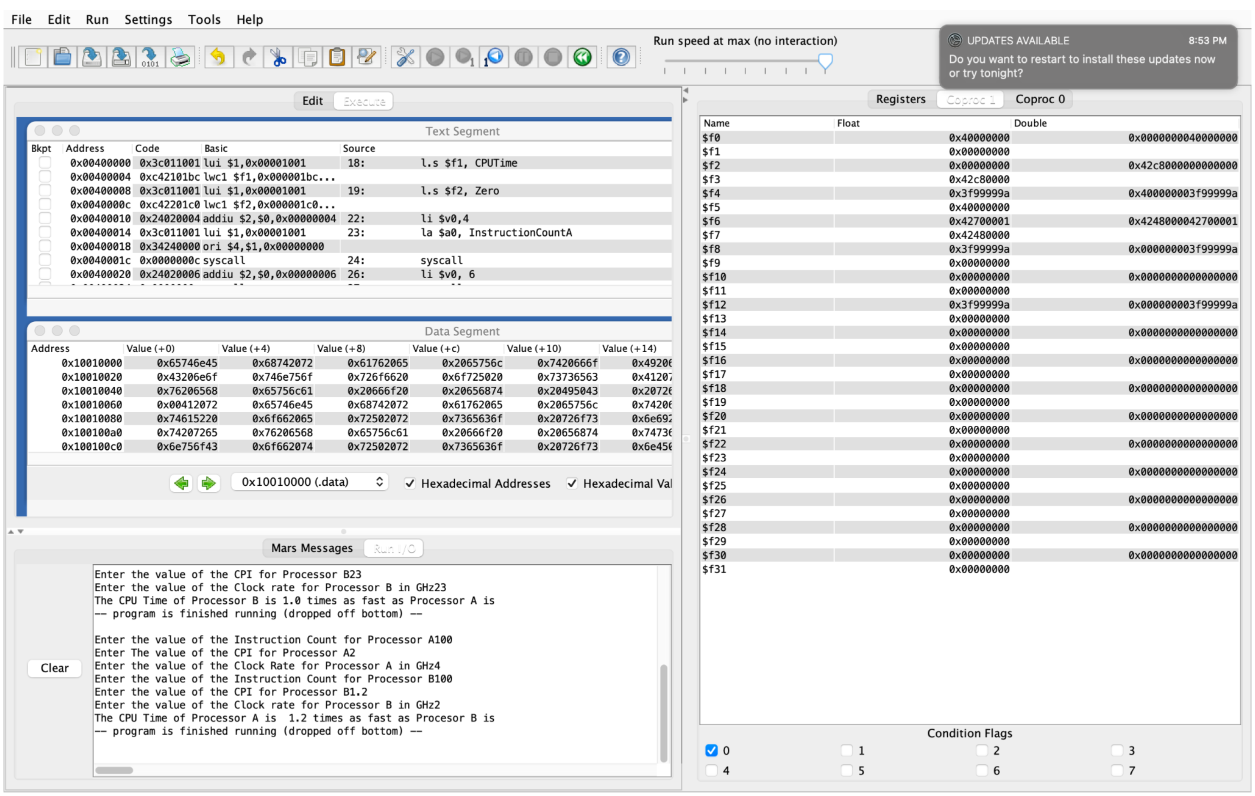
$t1	**Primarily used to store variables, sums, products, differences, and quotients between registers**
$t2	**Primarily used to store variables, sums, products, differences, and quotients between registers**
$f0	**Single precision floating point register**
$f1	**Single precision floating point register**
$f2	**Single precision floating point register**
$f3	**Single precision floating point register**
$f4	**Single precision floating point register**
$f5	**Single precision floating point register**
$f6	**Single precision floating point register**

4.0 Learning Coverage

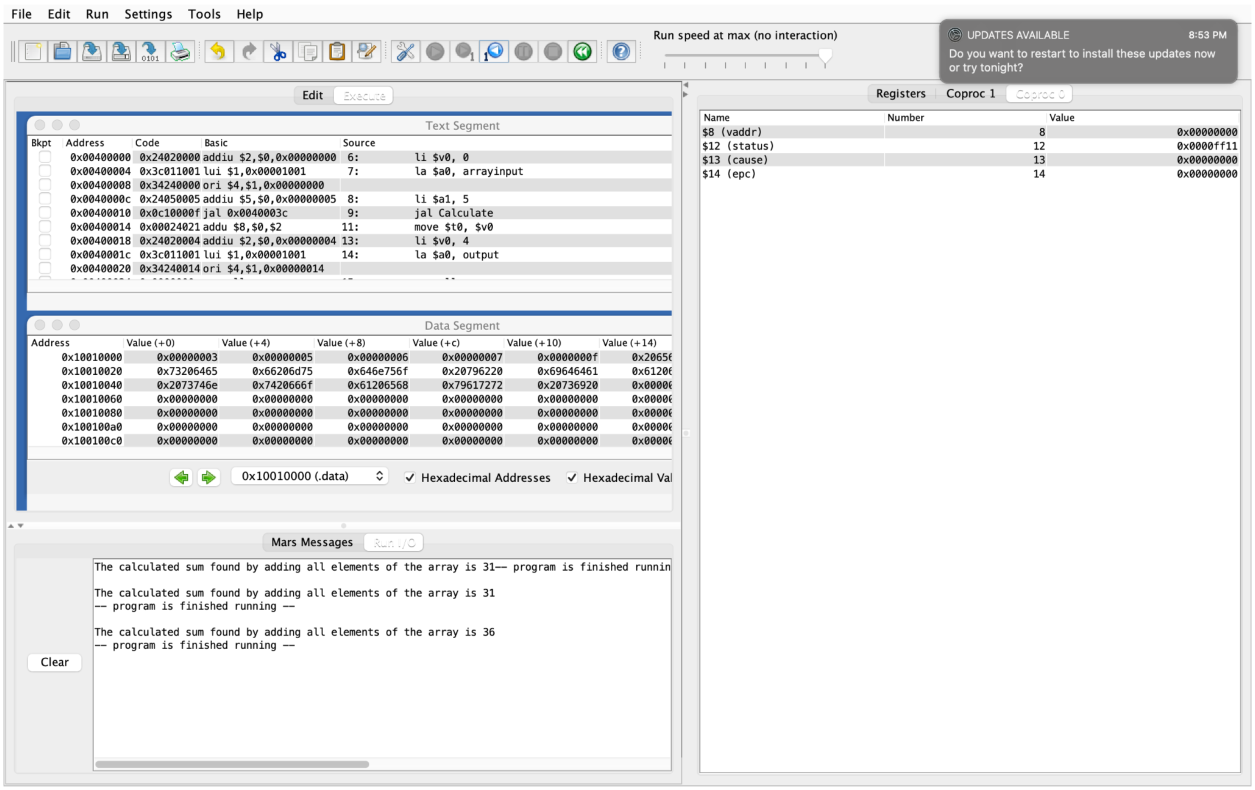
1. How to utilize a stack in order to push information on top of each and how to pop it off
2. Utilizing two inputted values in order to complete exponent operations
3. How to take two calculated values, compare them, and then conduct operations on them based on this comparison
4. Expansion on how to utilize floating points
5. The ability to utilize jump in MIPS and how to return this jump point

5.0 Test Results

Project 2



Project 3


File Edit Run Settings Tools Help
Run speed at max (no interaction)
UPDATES AVAILABLE 8:53 PM
Do you want to restart to install these updates now or try tonight?
Edit Execute
Text Segment
Bkpt Address Code Basic Source
0x00400000 0x24020000 addiu $2,$0,0x00000000 6: li $v0, 0
0x00400004 0x3c011001 lui $1,0x00001001 7: la $a0, arrayinput
0x00400008 0x34240000 ori $4,$1,0x00000000
0x0040000c 0x24050005 addiu $5,$0,0x00000005 8: li $a1, 5
0x00400010 0x0c10000f jal 0x0040003c 9: jal Calculate
0x00400014 0x00024021 addu $8,$0,$2 11: move $t0, $v0
0x00400018 0x24020004 addiu $2,$0,0x00000004 13: li $v0, 4
0x0040001c 0x3c011001 lui $1,0x00001001 14: la $a0, output
0x00400020 0x34240014 ori $4,$1,0x00000014
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+c) Value (+10) Value (+14)
0x10010000 0x00000003 0x00000005 0x00000006 0x00000007 0x0000000f
0x10010020 0x73206465 0x66206d75 0x646e756f 0x20796220 0x69646461
0x10010040 0x2073746e 0x7420666f 0x61206568 0x79617272 0x20736920
0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values
Mars Messages Run I/O
The calculated sum found by adding all elements of the array is 31-- program is finished running
The calculated sum found by adding all elements of the array is 31
-- program is finished running --
The calculated sum found by adding all elements of the array is 36
-- program is finished running --
Clear
Registers Coproc 1 Coproc 0
Name Number Value
$8 (vaddr) 8 0x00000000
$12 (status) 12 0x0000ff11
$13 (cause) 13 0x00000000
$14 (epc) 14 0x00000000