

***Introduction***

The purpose of this lab is to further the students’ understanding of recursion, as well as to introduce the students to the concept of jump functions in the MIPS assembly language.

***Design Discussion***

The list of tasks were fully completed in this lab:

* Write a MIPS Assembly program to perform the recursive factorial computation described, given some source code
* Assemble the MIPS Assembly code and single-step execute through all of the instructions. Verify the contents of each relevant register.

***Task1***

The first task of this assignment was to convert the given C++ pseudo code into MIPS Assembly. Some aspects of the C++ pseudo code were already completed and provided to us, such as the creation of the array and the general framework for the recursive function. There was a relatively complex computation that involved elements of the given array. The final value of this computation was to be used as the input for the recursive factorial function the student was instructed to design.

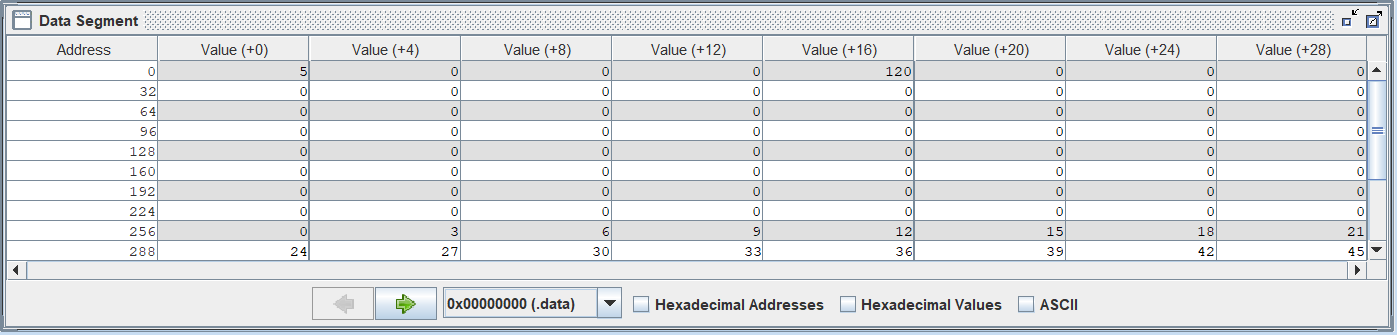


Figure 1: Final Register Values

***Task2***

The second task was to single-step through the assembly program written to verify the correct values being stored in registers, including computational values and return addresses stored by the stack pointer. This process allowed the student to verify the recursive nature of the function that has been written.

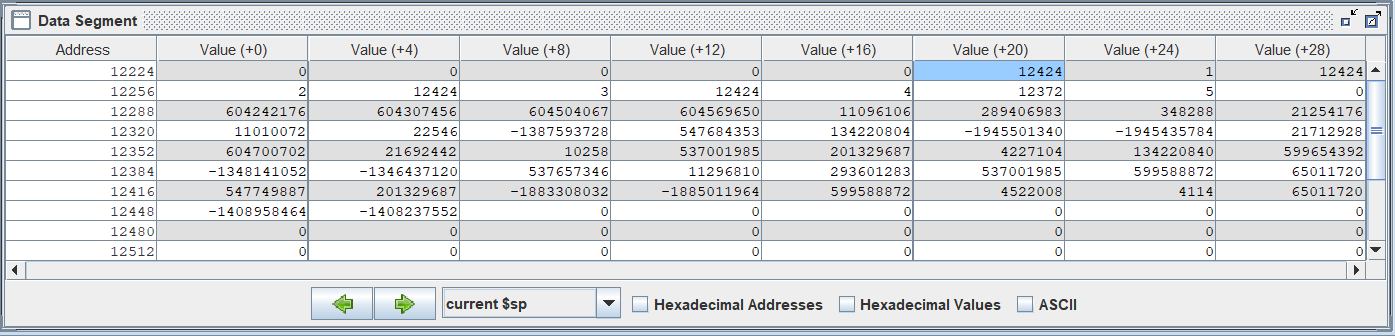


Figure 2: Register Values Mid-Recursive Process

| Addr | MIPS Instruction | Machine Code | Registers | | | | Memory Content | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $a1 | $sp | $ra | $v0 | [0x00] | [0x10] |
| 034 | Lw $t2, 356($0) | 8c0a0164 | 32 | 2ffc | 0 | 0 | 0 | 0 |
| 038 | Lw $t3, 376($0) | 8c0b0178 | 32 | 2ffc | 0 | 0 | 0 | 0 |
| 03c | Add $t2, $t2, $t3 | 014b5020 | 32 | 2ffc | 0 | 0 | 0 | 0 |
| 040 | Addiu $t3, $0, 30 | 240b001e | 32 | 2ffc | 0 | 0 | 0 | 0 |
| 044 | Div $t2, $t3 | 014b001a | 32 | 2ffc | 0 | 0 | 0 | 0 |
| 048 | Mflo $a1 | 00002812 | 5 | 2ffc | 0 | 0 | 0 | 0 |
| 04c | Addi $v0, $0, 1 | 20020001 | 5 | 2ffc | 0 | 1 | 0 | 0 |
| 050 | Jal factorial | 0c000c17 | 5 | 2ffc | 3054 | 1 | 0 | 0 |
| 054 | Add $s0, $v0, $0 | 00408020 | 5 | 2ffc | 3054 | 78 | 0 | 0 |
| 058 | J end | 08000c26 | 5 | 2ffc | 3054 | 78 | 0 | 0 |
| 05c | factorial: addi $s0, $v0, $0 | 00003098 | 5 | 2ff4 | 3054 | 1 | 0 | 0 |
| 060 | Sw $a1, 4($sp) | afa50004 | 5 | 2ff4 | 3054 | 1 | 0 | 0 |
| 064 | Sw $ra, 0($sp) | afbf0000 | 5 | 2ff4 | 3054 | 1 | 0 | 0 |
| 068 | Slti $t4, $a1, 2 | 28ac0002 | 5 | 2ff4 | 3054 | 1 | 0 | 0 |
| 070 | Addi $sp, $sp, 8 | 23bd0008 | 5 | 2ff4 | 3054 | 1 | 0 | 0 |
| 074 | Jr $ra | 03e00008 | 1 | 2fdc | 3080 | 1 | 0 | 0 |
| 078 | Addi $a1, $a1, -1 | 20a5ffff | 4 | 2ff4 | 3054 | 1 | 0 | 0 |
| 07c | Jal factorial | 0c000c17 | 4 | 2ff4 | 3080 | 1 | 0 | 0 |
| 080 | Lw $ra, 0($sp) | 8fbf0000 | 1 | 2fdc | 3080 | 1 | 0 | 0 |
| 084 | Lw $a1, 4($sp) | 8fa50004 | 2 | 2fdc | 3080 | 1 | 0 | 0 |
| 088 | Addi $sp, $sp, 8 | 23bd0008 | 2 | 2fe4 | 3080 | 1 | 0 | 0 |
| 08c | Mult $v0, $a1 | 00450018 | 2 | 2fe4 | 3080 | 1 | 0 | 0 |
| 090 | Mflo $v0 | 00001012 | 2 | 2fe4 | 3080 | 2 | 0 | 0 |
| 094 | Jr $ra | 03e00008 | 2 | 2fe4 | 3080 | 2 | 0 | 0 |
| 098 | Sw $a1, 0($0) | Ac050000 | 5 | 2ffc | 3054 | 78 | 5 | 0 |
| 09c | Sw $s0, 16($0) | Ac100010 | 5 | 2ffc | 3054 | 78 | 5 | 78 |

Table 1: Test Log of Mips Assembly Code

***Conclusion:***

This lab was a good exercise on the nature of recursive algorithms. In other courses, it is unclear how recursion functions at such a low level, but watching the registers act in the manner was a very eye-opening experience to a process that higher-level programmers take for granted. This lab was completed by both lab partners without any issues besides difficulties in articulation during the presentation.

***Appendix***

| MIPS Assembly Code |
| --- |
| Main:  li $a0, 0x100 # array base address = 0x100  li $a1, 0 # i = 0  li $t0, 3  li $t1, 50 # $t1 = 50  CreateArray\_Loop:  slt $t2, $a1, $t1 # i < 50?  beq $t2, $0, Exit\_Loop # if not then exit loop  sll $t2, $a1, 2 # $t2 = i \* 4  add $t2, $t2, $a0 # address of array[i]  mult $a1, $t0  mflo $t3 # $t3 = i \* 3  sw $t3, 0($t2) # save array[i]  addi $a1, $a1, 1 # i = i + 1  j CreateArray\_Loop  Exit\_Loop:  #my code  lw $t2, 356($0)  lw $t3, 376($0)  add $t2, $t2, $t3  addiu $t3, $0, 30  div $t2, $t3  mflo $a1  addi $v0, $0, 1  jal factorial  add $s0, $v0, $0  j end  factorial: addi $sp, $sp, -8 room on stack  sw $a1, 4($sp)  sw $ra, 0($sp)  # your code goes in here  addi $t4, $0, 2  slt $t4, $a1, $t4  beq $t4, $0, else  addi $v0, $0, 1  addi $sp, $sp, 8  jr $ra  else: addi $a1, $a1, -1  jal factorial  lw $ra, 0($sp)  lw $a1, 4($sp)  addi $sp, $sp, 8  mult $v0, $a1  mflo $v0  jr $ra  end:  sw $a1, 0($0)  sw $s0, 16($0) |

| C++ Pseudo Code |
| --- |
| void main()  {  int n, f;  int my\_array[50];  // Create the array  for(i=0; i<50; i=i+1)  {  my\_array[i] = i\*3;  }  /\*You will write MIPS code for the following parts\*/  // Arithmetic calculation  n = (my\_array[25]+ my\_array[30])/30;  // Factorial  f = Factorial(n);  return;  }  // Recursive factorial procedure  int Factorial(int n)  {  if (n <= 1)  return 1;  else  return (n\*Factorial(n-1));  } |