Zack Fravel

010646947

Lab 1 : M 4:10 - 5:55 PM

Lab 5 - SPIM Simulator

3/28/16

## Introduction

The main purpose of this lab was our introduction to writing assembly code from given C++ examples. We were given a pre lab example that was a simple function call with two variables that called upon another function. The code for the pre lab is included at the end of the report. Once we got the pre lab working, we were to change the main program to accommodate for arrays and use the same functions to perform the same operation multiple times using a for loop.

**Approach**

The following is the program we wish to implement in MIPS using the standard 32 register architecture.

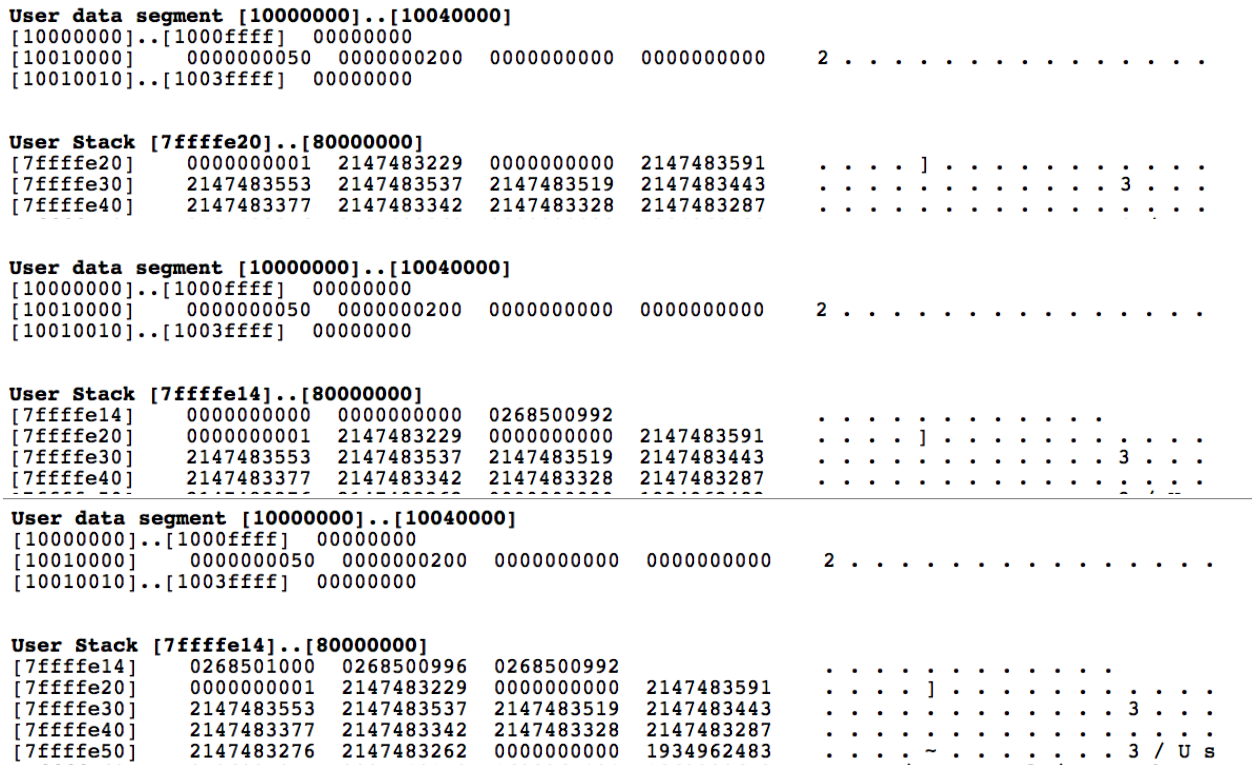
```
void swap (int a, int b)
{
   int temp=a;
   a=b;
   b=temp;
}


int distance (int a, int b)
{
   if (b > a)
      swap (a,b);

  return (a-b)
}

void  main(void)
{
  int var1 =50;
  int var2= 200;
  int result=0;
  int sum =0;

  result = distance (var1, var2);


  return;

}
```

```
int main(void)
{
  int var1[4] ={4, 7, 12, 5};
  int var2[4]= {15, 3, 6, 14};
  int result[4]={0};

  for (int i=0 ; i< 4; i++)
    result[i] = distance (var1[i], var2[i]);

}
```

The code on the left is the pre lab code we were assigned, and the segment on the right is the new main program we were asked to implement in the lab.

**Experimentation**

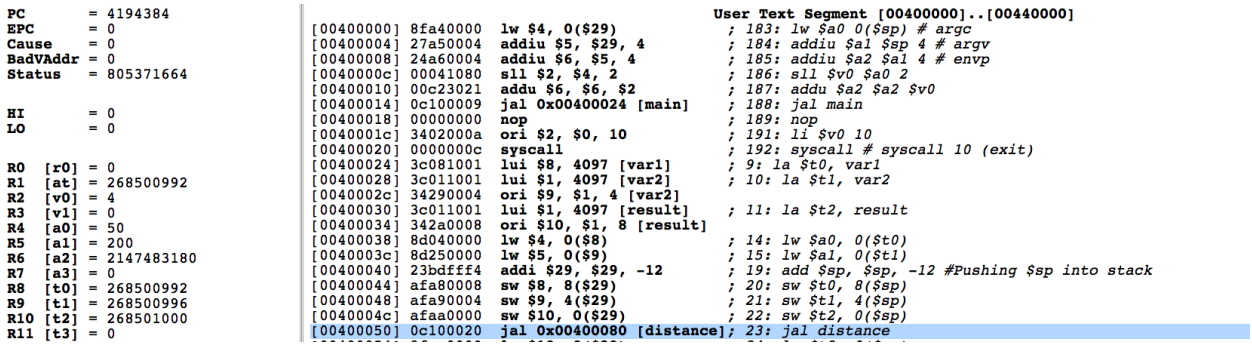
In my pre lab example, I have around 32 instructions that MIPS reads to complete the entire program. Of these instructions, about 31% are lw and sw functions, 32% arithmetic, and the rest being control (branch) instructions. One of the main things I had to work on was the use of the $sp when calling different functions. We push the $sp by 12 to account for 3 different variables when calling the distance function, then store the 3 addresses we had in the registers to memory so we can call upon them after we call the distance function.




When we do this, the stack increases and allows us to store 3 more 32 bit addresses on the top of the stack.

When we use the jump-and-link (jal) and jump-register (jr), this effects the program counter (PC) as well as our $ra. Whenever you call a function, the $ra is saved so you can call upon it at the end of your function to return to wherever you were in the main program, therefore whenever you use a (jal) or (jr), the $ra becomes a new address. This is why it is necessary to

save the $ra when calling another function. The PC changes as well when you use these instruction types, because a jump indicates jumping to another instruction.



```
PC       = 4194384
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 805371664

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 268500992
R2  [v0] = 4
R3  [v1] = 0
R4  [a0] = 50
R5  [a1] = 200
R6  [a2] = 2147483180
R7  [a3] = 0
R8  [t0] = 268500992
R9  [t1] = 268500996
R10 [t2] = 268501000
R11 [t3] = 0
```

```
                                                User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)           ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4        ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4         ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2           ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2         ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]   ; 188: jal main
[00400018] 00000000  nop                     ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10          ; 191: li $v0 10
[00400020] 0000000c  syscall                 ; 192: syscall # syscall 10 (exit)
[00400024] 3c081001  lui $8, 4097 [var1]     ; 9: la $t0, var1
[00400028] 3c011001  lui $1, 4097 [var2]     ; 10: la $t1, var2
[0040002c] 34290004  ori $9, $1, 4 [var2]
[00400030] 3c011001  lui $1, 4097 [result]   ; 11: la $t2, result
[00400034] 342a0008  ori $10, $1, 8 [result]
[00400038] 8d040000  lw $4, 0($8)            ; 14: lw $a0, 0($t0)
[0040003c] 8d250000  lw $5, 0($9)            ; 15: lw $a1, 0($t1)
[00400040] 23bdfff4  addi $29, $29, -12      ; 19: add $sp, $sp, -12 #Pushing $sp into stack
[00400044] afa80008  sw $8, 8($29)           ; 20: sw $t0, 8($sp)
[00400048] afa90004  sw $9, 4($29)           ; 21: sw $t1, 4($sp)
[0040004c] afaa0000  sw $10, 0($29)          ; 22: sw $t2, 0($sp)
[00400050] 0c100020  jal 0x00400080 [distance]; 23: jal distance
```

```
PC       = 4194432
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 805371664

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 268500992
R2  [v0] = 4
R3  [v1] = 0
R4  [a0] = 50
R5  [a1] = 200
R6  [a2] = 2147483180
R7  [a3] = 0
R8  [t0] = 268500992
R9  [t1] = 268500996
R10 [t2] = 268501000
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
```

```
                                                User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)           ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4        ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4         ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2           ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2         ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]   ; 188: jal main
[00400018] 00000000  nop                     ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10          ; 191: li $v0 10
[00400020] 0000000c  syscall                 ; 192: syscall # syscall 10 (exit)
[00400024] 3c081001  lui $8, 4097 [var1]     ; 9: la $t0, var1
[00400028] 3c011001  lui $1, 4097 [var2]     ; 10: la $t1, var2
[0040002c] 34290004  ori $9, $1, 4 [var2]
[00400030] 3c011001  lui $1, 4097 [result]   ; 11: la $t2, result
[00400034] 342a0008  ori $10, $1, 8 [result]
[00400038] 8d040000  lw $4, 0($8)            ; 14: lw $a0, 0($t0)
[0040003c] 8d250000  lw $5, 0($9)            ; 15: lw $a1, 0($t1)
[00400040] 23bdfff4  addi $29, $29, -12      ; 19: add $sp, $sp, -12 #Pushing $sp into stack
[00400044] afa80008  sw $8, 8($29)           ; 20: sw $t0, 8($sp)
[00400048] afa90004  sw $9, 4($29)           ; 21: sw $t1, 4($sp)
[0040004c] afaa0000  sw $10, 0($29)          ; 22: sw $t2, 0($sp)
[00400050] 0c100020  jal 0x00400080 [distance]; 23: jal distance
[00400054] 8faa0000  lw $10, 0($29)          ; 24: lw $t2, 0($sp)
[00400058] 8fa90004  lw $9, 4($29)           ; 25: lw $t1, 4($sp)
[0040005c] 8fa80008  lw $8, 8($29)           ; 26: lw $t0, 8($sp)
[00400060] 23bd000c  addi $29, $29, 12       ; 27: add $sp, $sp,12
[00400064] ad420000  sw $2, 0($10)           ; 30: sw $v0, 0($t2)
[00400068] 3402000a  ori $2, $0, 10          ; 31: li $v0,10
[0040006c] 0000000c  syscall                 ; 32: syscall
[00400070] 00804020  add $8, $4, $0          ; 36: add $t0, $a0, $0
[00400074] 00a02020  add $4, $5, $0          ; 37: add $a0, $a1, $0
[00400078] 01002820  add $5, $8, $0          ; 38: add $a1, $t0, $0
[0040007c] 03e00008  jr $31                  ; 39: jr $ra
[00400080] 00804020  add $8, $4, $0          ; 42: add $t0, $a0, $0
```

The PC changes by whatever amount is indicated in the offset in the instruction, making the next instruction to be executed whatever was specified.

There are 3 segments in the data section, our data is stored per byte of information.

**Results and Conclusion**

I was not able to get the fully functioning main program from the lab working, however the pre lab works as is. Both the lab code and the pre lab code are included on the back.

```
.data
   var1: .word 50
   var2: .word 200
   result: .word 0

.text

main:
  la $t0, var1
  la $t1, var2
  la $t2, result


  lw $a0, 0($t0)
  lw $a1, 0($t1)



  add $sp, $sp, -12 #Pushing $sp into stack
  sw $t0, 8($sp)
  sw $t1, 4($sp)
  sw $t2, 0($sp)
  jal distance
  lw $t2, 0($sp)
  lw $t1, 4($sp)
  lw $t0, 8($sp)
  add $sp, $sp,12


  sw $v0, 0($t2)
  li $v0,10
  syscall


swap:
  add   $t0, $a0, $0
  add   $a0, $a1, $0
  add   $a1, $t0, $0
  jr    $ra

distance:
  add   $t0, $a0, $0
  add   $t1, $a1, $0

  slt   $t2, $t0, $t1
  beq   $t2, $0, if_done


  add   $sp, $sp, -4 #Pushing $sp into stack
  sw    $ra, 0($sp)

  add   $a0, $t0, $0  #call the swap function
  add   $a1, $t1, $0
  jal   swap
```

```
.data
   var1: .word 4, 7, 12, 5
   var2: .word 15, 3, 6, 14
   result: .space 4

.text

main:
  la $t0, var1
  la $t1, var2
  la $t2, result
  or $t3, $0, $0        # initialize i

 for_loop:
  slti $s0, $s3, 4
  beq  $s0, $0, for_done

  sll  $t4, $t3, 2      # t2 goes from i to i*4
  add  $s0, $t4, $t0    # s0 is var1[i] address
  add  $s1, $t4, $t1    # s1 is var2[i] address
  add  $s2, $t4, $t2    # s2 is result[i] address

  lw $a0, 0($s0)
  lw $a1, 0($s1)
  lw $a2, 0($s2)

  jal distance

  addi $s3, 1
  j for_loop

 for_done:
  li $v0,10
  syscall
```