

**Course Name**: Computer Architecture Lab

**Course Number and Section**: **14:332:333:02**

**Experiment**: Lab 1 – Machine Language, SPIM, High-to-Low Level Conversion

**Lab Instructor**: Bangtian Liu

**Date Performed**: 1/23/18

**Date Submitted**: 2/4/18

**Submitted by**: Yuqing Feng (yf184)

Purpose

This laboratory exercise taught me how to perform basic operations in MIPS assembly code and to run programs on QTSpim. Assembly language is important for understanding what is happening in the memory when higher level languages run. I learned basic arithmetic operations and system calls that display text to the user as well as get user input. Finally, I figured out how to use QTSpim and run it step by step for debugging.

Approach

Before beginning assignments, I needed to learn MIPS. Instructions are divided into sections from 32 bits depending on the type of format. For I-type instruction, it is split up as op, rs, rt and then a constant or address. For example, the add command adds together rs and rt and puts the result into that address. For R-type, op is also 0’s while there is still rs and rt. Instead of a constant or address, the remaining 16 bits are divided as rd, shamt, and funct. These determine which type of operation is being performed.

Next, RAM memory transfer is an important part of loading and saving data, especially for assigning array elements values. “lw $destination, offset($base)” loads the destination with the value at the offset($base) address. “sw $source, offset($base)” saves the value in source into offset($base).

I also learned the importance of understanding the bytes of elements in arrays. If the array elements are 1 byte each, incrementing by one for the address will get to the next element, but if they are 4 bytes each, I need to increment by four to get to the next element. While C automatically does this, it is manual in assembly.

Important operations were “add”, “addi” (for adding an immediate number, not a register), “sub”, “div”, and “mult” (and retrieval of the value with “mflo $dest”).

For reading and printing, “syscall” is used. For syscall to know what to do, $v0 must be loaded with a code. When reading values, the value the user inputs is put into $v0 and to put it into a register, one must use “move $t0, $v0” so the value is stored in $t0. The codes below specify what syscall will do:

|  |  |  |
| --- | --- | --- |
| $v0 code | What it does | Arguments needed |
| 1 | Print int | Value in $a0 |
| 2 | Print float | Value in $f12 |
| 3 | Print double | Value in $f12 |
| 4 | Print string | Value in $a0 |
| 5 | Read int | - |
| 6 | Read float | - |
| 7 | Read double | - |
| 8 | Read string | Addr. to store $a0  No. of char $a1 |
| 9 | Memory allocation | Bytes storage $a0 |
| 10 | Exit program |  |
| 11 | Print character | Integer in $a0 |
| 12 | Read character |  |

For Assignment 5, I had to translate for loops from C to MIPS. The syntax for “for loops” in MIPS is:

loopA:

beq $t0, $s0, end # if i == a, end

…

addi $t0, $t0, 1 # add 1 to i

j loopA # jump back to the top

end:

The incrementing variable and the limit of i can be set through this. The loop is named in case of nested loops, so that they can be distinguished when jumping up to the top.

Using everything I learned above during the lab, I completed the below assignments and checked the coded ones (pasted below and attached as .asm files) in QTSpim.

Results

Assignment 1 – Machine Language Instructions in Binary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | op | rs | rt | constant |
| addi $3, $0, 15 | 001000 | 00011 | 00000 | 0000000000001111 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | op | rs | rt | rd | shamt | funct |
| mult $8, $12 | 000000 | 01000 | 01100 | 00000 | 00000 | 011000 |

Assignment 2 – C to MIPS

C code:

A[8\*i] =A[2\*i] – B[2\*i]

Variables f, g, h, i, j are $s0, $s1, $s2, $s3, $s4

Arrays A and B are $s6 and $s7

\*temporary variables assumed: $a0, $a1, $a2 and $s5

1. 1 byte

sll $s0, $s3, 3 # f = 8\*i – shift 3 for 1 byte is 2^3 = 8 times

sll $s1, $s3, 1 # g = 2\*i – shift 1 for 1 byte is 2^1 = 2 times

add $a0, $s6, $s1 # stores address of the element at 2\*i in A

add $a1, $s7, $s1 # stores address of the element at 2\*i in B

lb $s2, 0($a0) #h = content in the address (h = A[2\*i])

lb $s4, 0($a1) #j = content in the address (j = B[2\*i])

sub $s5, $s2, $s4 #stores $s2 - $s4 (A[2\*i] – B[2\*i])

add $a2, $s6, $s0 #gets address of A[8\*i]

sw $s5, 0($a2) #saves contents of $s5 into A[8\*i] address

1. 4 bytes

sll $s0, $s3, 5 # f = 8\*i – shift 5 for 4 bytes, need to shift (2^2)\*(2^3), so 2^5

sll $s1, $s3, 3 # g = 2\*i – shift 3 for 4 bytes, shift (2^2)(2^1) = 2^3

add $a0, $s6, $s1 # stores address of the element at 2\*i in A

add $a1, $s7, $s1 # stores address of the element at 2\*i in B

lw $s2, 0($a0) #h = content in the address (h = A[2\*i])

lw $s4, 0($a1) #j = content in the address (j = B[2\*i])

sub $s5, $s2, $s4 #stores $s2 - $s4 (A[2\*i] – B[2\*i])

add $a2, $s6, $s0 #gets address of A[8\*i]

sw $s5, 0($a2) #saves contents of $s5 into A[8\*i] address

Assignment 3 – MIPS to C

Variables f, g, h, i, j are $s0, $s1, $s2, $s3, $s4

Arrays A and B are $s6 and $s7

MIPS code 🡪 C code

addi $t0, $s6, 8 🡪 t0 = A + 2;

add $t1, $s3, $0 🡪 t1 = i;

sw $t1, 12($s7) 🡪 B[3] = t1;

lw $t0, 12($s6) 🡪 t0 = A[3];

add $s0, $t1, $t0 🡪 s0 = t1 + A[3];

Assignment 4 – Syscall and MIPS Program

1. $v0 in syscall

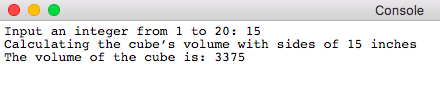
$v0 tells the syscall what to do. The codes assigned to $v0 have different meanings.

|  |  |  |
| --- | --- | --- |
| $v0 code | What it does | Arguments needed |
| 1 | Print int | Value in $a0 |
| 2 | Print float | Value in $f12 |
| 3 | Print double | Value in $f12 |
| 4 | Print string | Value in $a0 |
| 5 | Read int | - |
| 6 | Read float | - |
| 7 | Read double | - |
| 8 | Read string | Addr. to store $a0  No. of char $a1 |
| 9 | Memory allocation | Bytes storage $a0 |
| 10 | Exit program |  |
| 11 | Print character | Integer in $a0 |
| 12 | Read character |  |

1. MIPS Program

**Objective:** ask user to input integer between 1 and 20, report that it is calculating volume of cube, calculate, and report the volume of the cube with a message

**Running result:**



**MIPS Program (assignment4.asm file):**

.data

inputStr: .asciiz "Input an integer from 1 to 20: "

calcStr: .asciiz "Calculating the cube’s volume with sides of "

inchStr: .asciiz " inches"

outputStr: .asciiz "\nThe volume of the cube is: "

.text

.globl main

main:

li $v0, 4 # print input string - prompt

la $a0, inputStr

syscall

li $v0, 5 #command to read integer

syscall

move $t0, $v0 #store user input in $t0 register

li $v0, 4 #print calculating string

la $a0, calcStr

syscall

li $v0, 1 #print user input

move $a0, $t0

syscall

li $v0, 4 #finish printing calculating string

la $a0, inchStr

syscall

mult $t0, $t0 #multiply by itself and store in $t1

mflo $t1

mult $t1, $t0 #multiply the square $t1 by $t0 and store in $t2

mflo $t2

li $v0, 4 #print output string

la $a0, outputStr

syscall

li $v0, 1 #print result to user

move $a0, $t2

syscall

li $v0, 10 #exit program

syscall

Assignment 5 – C to MIPS

a -> $s0, b -> $s1, i -> $t0, j -> $t1

for (i=0; i<a; i++){

for (j=0; j<b; j++){

D[j] = i\*\*2 + j\*\*2;

}

}

**MIPS Code (assignment5.asm attached):**

loopA:

beq $t0, $s0, end # if i == a, end

loopB:

beq $t1, $s1, end # if j == b, end

mult $t0, $t0 #$t2 = i^2

mflo $t2

mult $t1, $t1 #$t3 = j^2

mflo $t3

add $t2, $t2, $t3 #$t2 = i^2 + j^2

li $t3, 4 #can't use mult with just immediate, load 4 in

mult $t1, $t3

mflo $t3 #$t3 = j\*4 (4 bytes assumed)

add $a0, $s2, $t3 #get the address of the jth element in array D

sw $t2, 0($a0) #save i^2+j^2 into $a0, which is the jth element of D

addi $t1, $t1, 1 # add 1 to j

j loopB # jump back to the top

end:

addi $t0, $t0, 1 # add 1 to i

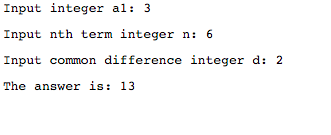
j loopA # jump back to the top

end:

Assignment 6 – Arithmetic Progression

Arithmetic progression: a1 + (n-1)\*d

**Running code:**



3, 5, 7, 9, 11, **13** <- 6th in the progression

**MIPS Code (assignment6.asm attached):**

.data

a1Str: .asciiz "Input integer a1: "

nStr: .asciiz "\nInput nth term integer n: "

dStr: .asciiz "\nInput common difference integer d: "

outStr: .asciiz "\nThe answer is: "

.text

.globl main

main:

li $v0, 4 #command to print a1Str

la $a0, a1Str

syscall

li $v0, 5 #command to read a1

syscall

move $t0, $v0 #store input in $t0 (a1) register

li $v0, 4 #command to print nStr

la $a0, nStr

syscall

li $v0, 5 #command to read n

syscall

move $t1, $v0 #store input in $t1 (n) register

li $v0, 4 #command to print dStr

la $a0, dStr

syscall

li $v0, 5 #command to read n

syscall

move $t2, $v0 #store input in $t2 (d) register

#solving

sub $t1, $t1, 1 #n = n-1

mult $t1, $t2

mflo $t3 #$t3 = (n-1)\*d

add $t4, $t0, $t3 #$t4 = a1 + (n-1)\*d

li $v0, 4 #command to print outStr

la $a0, outStr

syscall

li $v0, 1 #print the answer (stored in $t4)

move $a0, $t4

syscall

li $v0, 10 #exit program

syscall

Conclusion

Assignment 1 was important in showing the difference between I type and R type format. It also shows me the connection between assembly and binary, as assembly language is much closer to binary than higher level languages. Assignment 2 taught me how to locate array elements when there are different element byte sizes. Using sll to shift the i, variable I learned that the shift left value is different if it is 1 byte or 4 bytes. Load byte “lb” is used when it is just one byte, but load word “lw” must be used when there are 4 bytes. To get the addresses of the elements, I had to add the shifted values to the starting address of the array. Assignment taught me to translate MIPS into C code. Assignment 4 let me explore the uses of syscall and $v0 values as well as write a program using these concepts. The program printed and read strings and integers, which allowed me to use the concepts to create a working program. Assignment 5 taught me how to use for loops and nested for loops. I further solidified my understanding of getting an array element and assigning it a value. Finally, Assignment 6 allowed me to test everything I learned with an arithmetic challenge. I used printing and integer requesting as well as arithmetic operations to get the answer.

For the runnable assignments, I ran them on QTSpim and confirmed the correct outputs of my programs. I also used the running step by step feature for debugging purposes. Overall, I learned so much about programming in MIPS and how the computer actually makes the higher level code run. This was a great lesson in memory access, basic operations, and system calls in MIPS.