Cover Page

Computer Architecture

CPE 315 - Section 05

Lab Section 06

Todo: 4-7-16

Chad Benson

Nghia Nguyen

Chad Benson

Nghia Nguyen

Introduction (& purpose of project):

Lab 2 continues to develop and expand on programming in MIPS. Here, more advanced functions are implemented, including more advanced instructions and methods that will allow the student to push and pop to the stack, bit shift, recurse through addition, and the like.

Functional Requirements (What is to be accomplished):

Four functions will be implemented. The first, bintohex, converts a 32 bit value into a null terminating string of 8 characters. The second function, fibonacci, finds a fibonacci value given an index. Next, adder, must add two pairs of 32 bit values to form a 64 bit result. The last function, shifter, takes a 32 bit value, validates it, making sure it matches a beginning field format, and then performs a series of bit shifts to match a final field format.

Approach used (algorithms & methods):

Function 1, bintohex:
We first allocated space for our result of 8 characters. Then we made a table to hold each hex value. An input value is given to the function to parse, each character at a time. To parse the characters from the input, we looped 8 times, using a series of bit shifts, storing each character in a “buffer”. To do this, we used to the 8 bit binary value to “index” into a table, which gave us the correct hex value. Once all the characters are stored into the buffer, final result is printed to console.

Function 2, fibonacci:
We begin by prompting the user for an index, n. The index is passed to a recursive function where sequential additions of n - 1 and n - 2 are summed together until the base cases of 0 or 1 are reached. The final fibonacci value is then returned and printed to console.

Function 3, adder:
This function loads two pairs of 32 bit values into registers a0 through a3, which are then passed to the function. The lower 16 bit pairs are added together, checking to see if a carry occurs, which is then added to the sum of the higher 16 bit pairs. The bintohex function is then used to print the final result. Here, we ignore any carry that occurs in the upper 16 bit pairs.

Function 4, shifter:
This function assumes a 32 bit value of the following format:
fff0 0nn0 0000 x000 yyyy 0000 0000 0000
We first check to see if the input matches this format; if it does not, we print an error message and halt the program. If the input is valid, we shift the upper 7 bits to the lower bottom and store in a register. Then we add a bit mask to keep the bits that do not get shifted and store in a separate register. Then bit 19 is moved to its final position and saved in a different register. Once, all the bits have been shifted properly, the three registers are added together to obtain the final result, which is printed to console in this format:
0000 0000 0000 0000 yyyy 000x 0fff 00nn

<u>Source Code:</u>

```
# Chad Benson and Nghia Nguyen
# Lab 2 - Creating subroutines
# Function 1- Use a 32 bit value to print hex characters

.data

   # buff: .word 0, 0, 0  # option 2, storage for result, 12 bytes
   buff: .space 9 # storage for 9 bytes, less spaces
   table: .byte 0x30, 0x31, 0x32, 0x33, 0x34, 0x35 0x36, 0x37, 0x38, 0x39, 0x41, 0x42,
0x43, 0x44, 0x45, 0x46  # table of character values

.text

############################################
main:

   la $t1 table # load address of table
   la $a1 buff  # load address of buffer
   li $a0 0x1A0B8F03 # test value

    # call bintohex func
   jal bintohex

    # a1 will be at the last index in buffer when ret
    sb $zero 0($a1)
```

```
   la $a0 buff
    li $v0 4
    syscall

   li $v0 10 # halt program
   syscall

##############################################
# function bintohex
# function makes the following assumptions:
# a0 => value
# a1 => buffer location
# t1 => table
# t5 => loop counter
# ther is no return value

bintohex:
   addi $sp $sp -4 # build stack, ra, fp, vars, etc
   sw $ra 0($sp)
   addi $sp $sp -4
   sw $fp 0($sp)
   move $fp $sp
   addi $sp $sp -4 # save temp register $t0
    sw $t0 0($sp)

   li $t5 8                    # set counter to 8
loop:

   srl $t0 $a0 28     # get the first 4 bits of a0 into t0
    sll $a0 $a0 4          # shift left 4 bits to delete the first 4 bits we got above
    add $t0 $t0 $t1     # t0 will not be at the right index at the table
    lb  $t0 0($t0)         # the hex value (character) is saved in t0
    sb  $t0 0($a1)       # save the hex value (character) into the buff
    addi $a1 $a1 1      # increase to the next location in the buff
    addi $t5, $t5, -1 # decrease the counter by 1
    bne $t5 $zero loop    # run until the counter reach 0 (run 8 times)

   lw $t0 0($sp)         # restore the temp t0
    addi $sp $sp 4
```

```
    lw $fp 0($sp)        # restore frame
    addi $sp $sp 4
    lw $ra 0($sp)        # retore the ret address
    addi $sp $sp 4     # stack is now has nothing
    jr $ra

.end

##########################################################################
# Chad Benson and Nghia Nguyen
# Lab 2 - Creating subroutines
# Function 2: Use recursion to find Fibonacci value

.data
    prompt1:    .asciiz  "Enter number: "
   prompt2: .asciiz  "Result: "

.text

main:
    la $a0, prompt1    #load address of prompt1 into a0
   li $v0, 4                 #set print_string mode
    syscall

    li $v0, 5                #  read in int
    syscall
    move $a0, $v0      # store argument

    jal fib
    # get the result from ret value and pop stack
    lw $t0, 0($sp)
    addi $sp, $sp, 8

    la $a0, prompt2
    li $v0, 4
    syscall
    #print out the result
    move $a0, $t0
    li $v0, 1
```

```
    syscall

    li $v0 10           # halt
    syscall

# function
# Stack
#################################
# local t0, t1 <- top of stack
# caller frame pointer   <- new frame pointer
# caller return address
# Space for ret value
# $a0 contains fib number to find
##################################
fib:
    #callee setup
    addi $sp, $sp, -4           # push argument
    sw $a0, 0($sp)
    addi $sp, $sp, -8 # save space for ret value and push ra
    sw $ra, 0($sp)
    addi $sp, $sp, -4
    sw $fp, 0($sp)
    move $fp, $sp       # set frame ptr before any local
    addi $sp, $sp, -4
    sw $t0, 0($sp)      # store t0 for local use
    addi $sp, $sp, -4
    sw $t1, 0($sp)      #store t1 for local use

    li $t0, 1
    bgt $a0, $t0, Recursive
    beq $a0, $t0, BaseCase1

BaseCase0:
    sw $zero, 8($fp)    # store 0 to ret value
    j Done

BaseCase1:
    sw $t0, 8($fp)    # store 1 to ret value
    j Done
```

```
Recursive:
    #passing argument n-1
    addi $a0, $a0, -1
    jal fib
    #getting the return value
    lw $t0, 0($sp)        # ret value from callee is at top of stack
    addi $sp, $sp, 8      # pop the ret value and argument

    #passing argument n-2
    lw $a0, 12($fp)    #load the argument because $a0 changed
    addi $a0, $a0, -2
    jal fib
    #getting return value
    lw $t1, 0($sp)
    addi $sp, $sp, 8

    #done with 2 argument
    add $t0, $t1, $t0    # add the 2 ret value together
    sw $t0, 8($fp)        # store sum into the ret value

Done:
    #Callee teardown
    lw $t1, 0($sp)
    addi $sp, $sp, 4
    lw $t0, 0($sp)
    addi $sp, $sp, 4
    lw $fp, 0($sp)
    addi $sp, $sp, 4
    lw $ra, 0($sp)
    addi $sp, $sp, 4

    j $ra
    # return to caller
    # top of stack will point to ret value

.end
```

```
#############################################################
# Chad Benson and Nghia Nguyen
# Lab 2 - Creating subroutines
# Function 3: adds two sets of 32 bit values to get a 64 bit value

.data
    ahi: .word 0x10000000
    alo: .word 0x842A0000
    bhi: .word 0x1CDA0000
    blo: .word 0xA2410000
    buff: .space 9 # storage for 9 bytes, less spaces
   table: .byte 0x30, 0x31, 0x32, 0x33, 0x34, 0x35 0x36, 0x37, 0x38, 0x39, 0x41, 0x42,
0x43, 0x44, 0x45, 0x46  # table of character values

.text

#######################################################
main:
    # load ahi, alo, bhi, blo into arguments
    lw $a0, ahi
    lw $a1, alo
    lw $a2, bhi
    lw $a3, blo

    jal doubleAdd

    # print the upper
    move $a0, $v0
    la $t1 table # load address of table
   la $a1 buff  # load address of buffer
    jal bintohex

    # append \0 at the end
    sb $zero 0($a1)
   la $a0 buff
    li $v0 4
    syscall
```

```
    # print the lower
    move $a0, $v1
    la $t1 table # load address of table
   la $a1 buff  # load address of buffer
    jal bintohex

    # append \0 at the end
    sb $zero 0($a1)
   la $a0 buff
    li $v0 4
    syscall

    li $v0 10
    syscall

##################################################
# function doubleAdd
# assumes there will valid values in a0 - a3
# return value placed in v0 and v1
doubleAdd:

    addu $v1, $a1, $a3    # sum of 2 low
    sltu $v0, $v1, $a1    # check if the sum is less than either 2 arguments
                                                                   # and put the result 1 if carry 0
otherwise into v0
    # add v0 with each high and get the final result
    addu $v0, $v0, $a0
    addu $v0, $v0, $a2

    j $ra

# Result will be saved in v0 (upper) and v1 (lower)
# function bintohex
# a0 => value
# a1 => buffer location
# t1 => table
# t5 => loop counter
#
```

```
bintohex:
   addi $sp $sp -4 # build stack, ra, fp, vars, etc
   sw $ra 0($sp)
   addi $sp $sp -4
   sw $fp 0($sp)
   move $fp $sp
   addi $sp $sp -4 # save temp register $t0
    sw $t0 0($sp)


   li $t5 8                    # set counter to 8
loop:
   srl $t0 $a0 28     # get the first 4 bits of a0 into t0
    sll $a0 $a0 4          # shift left 4 bits to delete the first 4 bits we got above
    add $t0 $t0 $t1     # t0 will not be at the right index at the table

    lb  $t0 0($t0)         # the hex value (character) is now save in t0
    sb  $t0 0($a1)        # save the hex value (character) into the buff
    addi $a1 $a1 1      # increase to the next location in the buff
    addi $t5, $t5, -1 # decrease the counter by 1
    bne $t5 $zero loop    # run until the counter reach 0 (run 8 times)

   lw $t0 0($sp)          # restore the temp t0
    addi $sp $sp 4
    lw $fp 0($sp)         # restore frame
    addi $sp $sp 4
    lw $ra 0($sp)         # retore the ret address
    addi $sp $sp 4     # stack is now has nothing
    jr $ra

.end

#
# Nghia Nguyen
# Chad Benson
# Lab 2 - Creating Functions in mips
# Functions takes a 32 bit value and
# shifts it according to a bit field format,
# from   fff0 0nn0 0000 x000 yyyy 0000 0000 0000
```

```
# to     0000 0000 0000 0000 yyyy 000x 0fff 00nn
#

.data

   val1: .word 0x6608C000
   val2: .word 0xC2008000
   inputMask: .word 0x19F70FFF
   shiftMask1: .word 0x0000F000  # used for bits 15 - 12
   shiftMask2: .word 0x00080000  # used for bit 19
   msg: .asciiz   "Sorry, format of input is not valid."
   format: .asciiz "\n"
        buff: .space 9 # storage for 9 bytes, less spaces
   table: .byte 0x30, 0x31, 0x32, 0x33, 0x34, 0x35 0x36, 0x37, 0x38, 0x39, 0x41, 0x42,
0x43, 0x44, 0x45, 0x46  # table of character values

.text

###################################################
main:

   lw $a0, val1    # load first input value
   jal shifter

   la $t1 table # load address of table
   la $a1 buff  # load address of buffer
   move $a0, $v0  # load result to print

   # call bintohex func
   jal bintohex

   # a1 will be at the last index in buffer when ret
   sb $zero 0($a1)
   la $a0 buff
   li $v0 4
   syscall

   li $v0, 4
   la $a0, format
```

```
   syscall

   lw $a0, val2   # load second input value
   jal shifter

   la $t1 table # load address of table
   la $a1 buff  # load address of buffer
   move $a0, $v0  # load result to print

   # call bintohex func
   jal bintohex

   # a1 will be at the last index in buffer when ret
   sb $zero 0($a1)
   la $a0 buff
   li $v0 4
   syscall

   li $v0 10
   syscall


 #  lw $a0, val2
 #  jal shifter



##################################################
# function valid checks whether input matches format
# assumes an input value in a0
# does not return anything

validate:

   addi $sp, $sp, -4       # push stack
   sw $ra, 0($sp)
   addi $sp, $sp, -4
   sw $fp, 0($sp)
   move $fp, $sp
```

```
  addi $sp, $sp, -4
  sw $a0, 0($sp)

  lw $t1, inputMask       # load mask
  and $a0, $a0, $t1       # mod input to test format
  beq $a0, $zero, valid   # branch to valid if a0 is zero

  la $a0, msg             # if not valid, alert user, and halt program
  li $v0, 4               # print error
  syscall

  li $v0, 10
  syscall

valid:

  lw $a0, 0($sp)          # pop stack
  addi $sp, $sp, 4
  lw $fp, 0($sp)
  addi $sp, $sp, 4
  lw $ra, 0($sp)
  addi $sp, $sp, 4
  jr $ra

#####################################################
# function shifter assumes value in a0, and then validates
# t0 for result
# t1 handles top 7 bits
# t2 handles bits 12 - 15
# t3 handles bit 18
# t4 stores bit masks
# returns shifted word in v0

shifter:

  addi $sp, $sp, -4       # push stack
  sw $ra, 0($sp)
  addi $sp, $sp, -4
  sw $fp, 0($sp)
```

```
  move $fp, $sp
  addi $sp, $sp, -4
  sw $a0, 0($sp)

  jal validate        # test input format, halt if incorrect
                      # input ok, compute shift sequence

  srl $t1, $a0, 25     # shift high 7 bits to bottom
  lw $t4, shiftMask1
  and $t2, $t4, $a0    # save bits 12 through 15
  lw $t4, shiftMask2
  and $t3, $t4, $a0    # save bit 19
  srl $t3, $t3, 11     # move bit 19 down
  add $t1, $t2, $t1    # sum registers
  add $t1, $t3, $t1
  move $v0, $t1        # store result

  lw $a0, 0($sp)
  addi $sp, $sp, 4
  lw $fp, 0($sp)       # pop stack
  addi $sp, $sp, 4
  lw $ra, 0($sp)
  addi $sp, $sp, 4
  jr $ra

###############################################
# function bintohex
# function makes the following assumptions:
# a0 => value
# a1 => buffer location
# t1 => table
# t5 => loop counter
# ther is no return value

bintohex:
  addi $sp $sp -4 # build stack, ra, fp, vars, etc
  sw $ra 0($sp)
  addi $sp $sp -4
  sw $fp 0($sp)
```

```
  move $fp $sp
  addi $sp $sp -4 # save temp register $t0
  sw $t0 0($sp)


  li $t5 8                      # set counter to 8
loop:
  srl $t0 $a0 28      # get the first 4 bits of a0 into t0
  sll $a0 $a0 4                 # shift left 4 bits to delete the first 4 bits we got above
  add $t0 $t0 $t1     # t0 will not be at the right index at the table

  lb  $t0 0($t0)                # the hex value (character) is now save in t0
  sb  $t0 0($a1)                # save the hex value (character) into the buff
  addi $a1 $a1 1                # increase to the next location in the buff
  addi $t5, $t5, -1 # decrease the counter by 1
  bne $t5 $zero loop            # run until the counter reach 0 (run 8 times)

  lw $t0 0($sp)                 # restore the temp t0
  addi $sp $sp 4
  lw $fp 0($sp)       # restore frame
  addi $sp $sp 4
  lw $ra 0($sp)                 # retore the ret address
  addi $sp $sp 4    # stack is now has nothing
  jr $ra

 .end
```

Discussion (difficulties and or concerns with reliability or security) :

Most of the implementation went smoothly. Perhaps, the most challenging part was translating algorithms that we were already familiar with in higher level languages into their MIPS counterparts. Working as a team was of great value in this regard, as we were both able to help each other work through to a solution.

Summary:

In this lab we expanded on our understanding of a stack and how to implement one. We also learned a little about i/o in mips, as well as indexing into a symbol table. Overall, this lab was highly effective in enhancing our understanding of the mips assembly language.