**FORMAN CHRISTIAN COLLEGE (A CHARTERED UNIVERSITY)**

****

**Computer Organization and Assembly Language – COMP 300 B**

**Spring 21**

**Programming Assignment 1**

**Muhammad Sameed Gilani - 231488347**

**You should attach the lab / assignment handout as second page of this report.**

**From third page onwards following headings should be included:**

* **Introduction**
  + **Should carry information of all major library functions.**
* **Your logic / algorithm in simple English. Bullet points are appreciated.**
* **Your code**
* **Screen shots of at least three outputs of your code with appropriate inputs.**
* **References**

**INTRODUCTION**

* li – Load immediate. 🡪 It is used to set the register to the immediate value we enter.

Ex:

li $v0,1  
This sets the register $v0, to 1

* la – Load address 🡪 It is used to set the register to the contents of another register or to an immediate value we enter.

Ex:

la $a0,$t0  
This loads the contents of $t0 onto $a0

* lw - Load Word 🡪 Set a register to contents of effective memory word address,

Ex:

lw $a0,input

This loads the address of the .word input, we created in the data segment.

* .asciiz 🡪 Store the string in the Data segment and add null terminator. Used in the program to store strings.

Ex:

x: .asciiz " Enter a value for x: "

In the data segment, this string is stored in x.

* .word 🡪 Store the listed value(s) as 32 bit words on word boundary

Ex:

c: .word 3

In the data segment, 3 is stored in c.

* .space 🡪 Reserve the next specified number of bytes in Data segment. It is used in the program to assign specific space for the input user will enter.

Ex:

input1: .space 8

In the data segment, assigns space to the input.

* move 🡪 Move the contents of one register to another

Ex:

move $t0,$t1

Contents of $t1 are moved to $t0

* addi 🡪 Used to add an immediate value to a register and store the value in another.

Ex:

add $t0, $t1, 5

$t1 and 5 are added and answer is stored in $to

* jal ( Jump and link ) 🡪 Set $ra to Program Counter (return address) then jump to statement at target address.

Used to jump and link to a function. $ra can be used to return back to the position we jumped from.

* beq 🡪 when the two subsequent mentioned registers are equal in value, we branch to the function that is mentioned.

Ex:

beq $t0, $t1, label

$t1 and $t0 are equal, we branch to the label

* add 🡪 Used to add the values in 2 registers and store it in a register

Ex:

add $t0, $t1, $t2

$t1 and $t2 are added and answer is stored in $to

* b 🡪 Used to branch to a specific label

Ex:

b label

* sw 🡪 Used to store a word into the mentioned memory address.

Ex:

sw $t0, ($t1)

* jr 🡪 Jump register unconditionally : Jump to statement whose address is in $t1

Ex:

jr $ra

* sll (Shift left logical) 🡪 Set $t1 to result of shifting $t2 left by number of bits specified by immediate value.

Ex:

sll $t0,$t1,1

it will shift $t1 left by 1 bit and store it in $t0

* srlv (Shift right logical variable) 🡪 Set $t1 to result of shifting $t2 right by number of bits specified by value in low-order 5 bits of $t3

Ex:

srlv $t1,$t2,$t3

* subi 🡪 Used to subtract an immediate value to a register and store the value in another.

Ex:

sub $t0, $t1, 5

$t1 and 5 are subtracted and answer is stored in $to

* srl (Shift right logical) 🡪 Set $t1 to result of shifting $t2 right by number of bits specified by immediate .

Ex:

srl $t1,$t2,1

it will shift $t1 right by 1 bit and store it in $t0

* or 🡪 Set $t1 to bitwise OR of $t2 and $t3

Ex:

or $t1,$t2,$t3

|  |  |  |
| --- | --- | --- |
| print integer | 1 | $a0 = integer to print |

* Service numbers used are,
  + 1 🡪

|  |  |  |
| --- | --- | --- |
| print string |  | $a0 = address of null-terminated string to print |

* + 4 🡪

|  |  |  |  |
| --- | --- | --- | --- |
| read integer |  |  | $v0 contains integer read |

* + 5 🡪
  + 10 🡪 exit (terminate execution)

|  |  |  |  |
| --- | --- | --- | --- |
| print integer in binary |  | $a0 = integer to print | Displayed value is 32 bits, left-padding with zeroes if necessary. |

* + 35 🡪

|  |  |  |  |
| --- | --- | --- | --- |
| print integer in hexadecimal |  | $a0 = integer to print | Displayed value is 8 hexadecimal digits, left-padding with zeroes if necessary. |

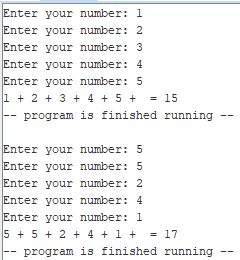
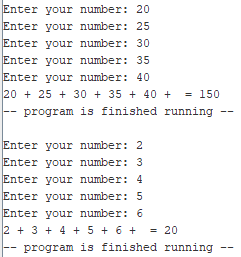
* + 34 🡪

**LOGIC**

*Problem 1:*

* In the data section we create the asciiz texts that we require. And we specify space for an array and how many items it can hold. Since we are using .word, each number requires 4 spaces. And since we are storing 6 numbers in the array, space is 24 and size is 6.
* In main, we call all the relevant functions, by jumping to them and returning back to main.
* We first call get\_input. Here we initialize the starting address of the array, the loop counter and number of iterations. We also store $ra in $s3, in order to return back to main at the position we left. As $ra will be overwrtitten when we call print\_prompt later.
* We move forward to, loop\_get\_input. In this loop we ask the user to enter a value, through print print\_prompt. Store it in a register and then store it into the array. Then we increment the loop counter and array position. In order to store the next value at the next position in the array.
* Once done we branch to exit\_func\_special. Which uses $s3 ( where we earlier stored $ra ) to jump back to main.
* Now we jump to add\_num. Here we initialize the starting address of the array, the loop counter, number of iterations and the 6th position in the array.
* We then move to loop\_add\_num. Initially the 6th position in the array is 0. To 0 we add the first number from the array. And overwrite it at the 6th position of the array.  
  Similarly we keep adding each value from 1 to 5 from the array to the 6th position and obtain the sum of all the values at position 6. Unfortunately I had to use $t2 in order to acheive this.  
  Once done we branch to exit\_func, which uses $ra to return to main.
* Now we jump to display\_output. Here we initialize the starting address of the array, the loop counter and number of iterations.
* We then move to loop\_display\_output, in this loop we access the array and we print each number from it. After each number we subsequently print a plus sign inbetween.  
  Once we have printed the 5th number, the loop counter is equal to 5, we branch to eq\_sign. Which prints an equal sign and the 6th number from the array. Which is the sum of the numbers.
* eq\_sign prints an equal sign and the 6th value of the array and uses exit\_func to go back to main.
* From here we jump to exit\_program. Which terminates the program.

*Sample outputs – Problem 1:*

*Problem 2:*

* In the data section I stored all the relevant asciiz texts, stored the hexadecimal in .word format and specified space for an array to store the binary value of the hexacdimal input. Since the binary is 32 bits long i assigned 128 space for the array. As each number will take 4 spaces in the array.
* In main I firstly employ the use of some code I discoverd online. Ive added the link to the code as refrence. I was unable to fully write code that could convert a complete hexadecimal value to binary so unfortunaetly I had to use this pre-written piece of code.
* From line #25 – 50 the code is used. As also marked by comments. I, however did make some changes to the code to make it useful in context of this problem.
* After the hexadecimal number is converted to binary, each bit is stored into an array. Once done, we branch to, print\_hex. Which prints text for the hexadecimal value, and the value itself.
* From here we move to display, where we initialize the starting address of the array, the loop counter and number of iterations. Moving on to display\_loop.
* This loop accesss the array and prints each bit of the 32 bit binary number we previously stored.
* And corresposing to R-type instruction, each set of bits is printed seperately.
* Once done we branch to, opcode\_binary. Where I first initialize $t9, with $zero.  
  Now I access the array and store each bit into a register and increment the array position by 4.
* Once I have each bit stored into a register, corresponding to the respective section. In this case, opcode need 6bits. So its 6 bits are stored in 6 seperate registers.
* Now using each bit I begin to combine these bits into a single register.   
  This is done by using, bitwise OR. I, OR each bit with $t9(which is intially 0). This allows me to store that bit into $t9 on the leftmost place.  
  After doing this I use, sll (shift left logical) to shift $t9 left by 1 bit, to make space for the next bit to be added to $t9.   
  To explain this Ive added images as well. \*
* This process is performed for each part of the R-type instruction. This couldve been made more efficient by using a function, allowing me to not have to print the whole block of code each instance. However I could not do that in the time alloted..
* So now i have the respective set of bits in $t9. I then print text respective to the section of the R-type instruction, then print the binary value as integer using, syscall 🡪 1 .
* After each section of the R-type instruction is printed, the program goes to exit\_prog and terminates.

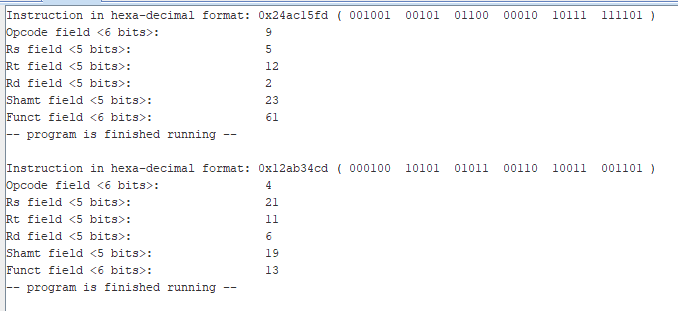
\*

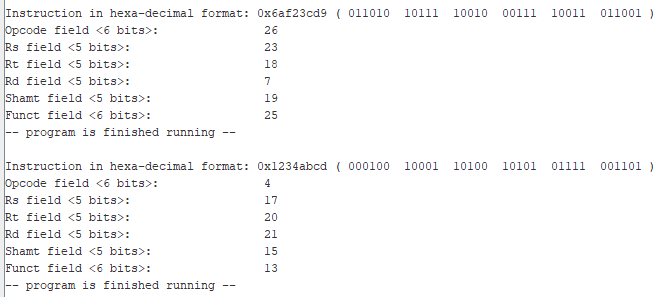
A piece of paper with writing

Description automatically generated with low confidence A piece of paper with writing

Description automatically generated with medium confidence

*Sample outputs – Problem 2:*

**



**Refrences:**

[**https://www.quora.com/Can-someone-write-me-a-function-in-mips-assembly-language-that-will-take-an-ascii-hexadecimal-value-and-convert-it-into-a-binary-value**](https://www.quora.com/Can-someone-write-me-a-function-in-mips-assembly-language-that-will-take-an-ascii-hexadecimal-value-and-convert-it-into-a-binary-value)

(comment by, Pavlos Fragkiadoulakis )

**CODE**

*Problem 1:*

.data

#Program Name: a1\_pb1.asm

#Programmer Name: Muhammad Sameed Gilani

#Programmer Roll Number: 231488347

prompt: .asciiz "Enter your number: "

list: .space 24 #allocate this space for a list

size: .word 6 #number of elements in the list

plus: .asciiz " + "

equal: .asciiz " = "

.text

main:

jal get\_input

jal add\_num

jal display\_output

jal exit\_program

display\_output:

li $t0,0 #loop counter

la $s1,list #starting address of array

lw $s0,size #size of list/number of iterations

loop\_display\_output:

beq $t0,5,eq\_sign # to print out an equal sign before the last number. ie the answer, is displayed

lw $t1,($s1) #load the first number from array

li $v0,1 # print that number

move $a0,$t1

syscall

addi $s1,$s1,4 #increment the array index and loop counter

addi $t0,$t0,1

li $v0,4 # prints + , after every number is printed, except after the last and second last number

la $a0,plus

syscall

b loop\_display\_output #restarts the loop

add\_num:

li $t0,1 #loop counter

la $s1,list #starting address of array

lw $s0,size #size of list/number of iterations

la $s2,list #starting address of array (This is to store the 6th position)

addi $s2,$s2,20 #6th position

loop\_add\_num:

beq $t0,$s0,exit\_func # when counter reaches list size, it goes to exit func which sends it back to main

lw $t1,($s1) # number from array

lw $t2,($s2) #6th number (initially it is 0)

add $t2,$t1,$t2 #adding the each number from 1st to 5th position, to the number in the 6th position

sw $t2,($s2) #storing that new number at the 6th position of the array

addi $s1,$s1,4 #increment the array index and loop counter

addi $t0,$t0,1

b loop\_add\_num

get\_input:

li $t0,1 #loop counter

la $s1,list #starting address of array

lw $s0,size #size of list/number of iterations

move $s2,$ra # This is to keep track of the return address back to main.

# As when we jal print\_prompt, $ra is overwritten.

loop\_get\_input:

beq $t0,$s0,exit\_func\_special # if $t0 == $s0. we go to exit\_func\_special only for this function.

# as it uses $s2 (where we stored $ra) to return back to main

jal print\_prompt # prompt user to enter values

# read user response

li $v0,5

syscall

move $t1,$v0

# store this number in array

sw $t1,($s1)

addi $s1,$s1,4 #increment the array index and loop counter

addi $t0,$t0,1

b loop\_get\_input

print\_prompt:

# Prints the prompt to enter a value, and returns back to loop\_get\_num, where it jumped from.

li $v0,4

la $a0,prompt

syscall

jr $ra

exit\_func:

# This is to allow functions to branch here and return back to main

jr $ra

exit\_func\_special:

#this is only for loop\_get\_input. As it uses $s2(where we stored $ra, to return back to main)

jr $s2

eq\_sign:

# only brached from loop\_display\_output, after the 5th number is displayed. This prints an equal sign after it.

# this also prints the 6th number. ie the sum after printing the equal sign

la $s1,list #starting address of array

addi $t1,$s1,20 #6th position

lw $t0,($s2) #6th number from array

#Prints equal sign

li $v0,4

la $a0,equal

syscall

# prints the 6th number from array. ie the final sum

li $v0,1

move $a0,$t0

syscall

b exit\_func # returns back to loop\_display\_output, from where it left off

exit\_program:

# to exit the program gracefully

li $v0,10

syscall

*Problem 2:*

.data

#Program Name: a1\_pb2.asm

#Programmer Name: Muhammad Sameed Gilani

#Programmer Roll Number: 231488347

hex: .word 0x24ac15fd

list: .space 128

siz: .word 32

hex\_prompt: .asciiz "Instruction in hexa-decimal format: "

opcode: .asciiz "\nOpcode field <6 bits>: "

rs: .asciiz "\nRs field <5 bits>: "

rt: .asciiz "\nRt field <5 bits>: "

rd: .asciiz "\nRd field <5 bits>: "

shamt: .asciiz "\nShamt field <5 bits>: "

funct: .asciiz "\nFunct field <6 bits>: "

bracket\_open: .asciiz " ( "

bracket\_close: .asciiz " ) "

space: .asciiz " "

.text

##start of refrenced code

main:

lw $s0,hex #s0 = x

li $t0,31 #(t0) i == 31 (the counter)

li $t1,1 #(t1) mask

sll $t1,$t1,31

la $s3,list # starting address of array

loop:

beq $t0,-1,print\_hex #if t0 == -1, we go to print\_hex, which prints the hex value.

and $t3,$s0,$t1 #isolate the bit

beq $t0,$0,after\_shift #shift is needed only if t0 > 0

srlv $t3,$t3,$t0 #right shift before display

after\_shift:

sw $t3,($s3) # add the single bit into the array

addi $s3,$s3,4 #increment the array to the next position

subi $t0, $t0, 1 #decrease the counter

srl $t1,$t1, 1 #right shift the mask

j loop

##end of refrenced code

print\_hex:

#prints prompt for hex value

li $v0,4

la $a0,hex\_prompt

syscall

#prints hex value

li $v0,34

lw $a0,hex

syscall

#prints an opening bracket to contain the binary value of the hexadecimal number

li $v0,4

la $a0,bracket\_open

syscall

display:

li $t0,0 #loop counter

la $s0,list # starting address of array

lw $s1,siz # number of iterations

display\_loop:

beq $t0,$s1,opcode\_binary #when $t0 == $s1, we go to opcode\_binary.

lw $t1,($s0) #load value from array

# print that value from array

li $v0,1

move $a0,$t1

syscall

addi $s0,$s0,4 #increment array index and loop counter

addi $t0,$t0,1

# this block prints a space after every set of bits correspoding to the R-type instruction.

# ie. space after 6 bits, then 5,then 5,then 5,then 5.

beq $t0,6,spc

beq $t0,11,spc

beq $t0,16,spc

beq $t0,21,spc

beq $t0,26,spc

b display\_loop

spc:

# to print the spaces in the binary number

li $v0,4

la $a0,space

syscall

j display\_loop

opcode\_binary:

#print closing bracket for what was opened previously

li $v0,4

la $a0,bracket\_close

syscall

#intialize $t9 to have 0 in it

move $t9,$zero

la $s1,list #starting address of array. 0th position in array

lw $t1,($s1) #1st number from array

addi $s1,$s1,4 #4th position

lw $t2,($s1) # 2nd number from array

addi $s1,$s1,4 #8th position

lw $t3,($s1) # 3rd number from array

addi $s1,$s1,4 #12th position

lw $t4,($s1) # 4th number from array

addi $s1,$s1,4 #16th position

lw $t5,($s1) # 5th number from array

addi $s1,$s1,4#20th position

lw $t6,($s1) # 6th number from array

# in this block, we in a way merge the values stored in the previous registers.

# by using OR with $t9 we first add the value on the left hand side of the binary number

# then we sll, to shift the binary number left by 1 position, to make space for the next value to be added in that position

or $t9,$t9,$t1

sll $t9,$t9,1

or $t9,$t9,$t2

sll $t9,$t9,1

or $t9,$t9,$t3

sll $t9,$t9,1

or $t9,$t9,$t4

sll $t9,$t9,1

or $t9,$t9,$t5

sll $t9,$t9,1

or $t9,$t9,$t6

# Now we have the specified binary number, stored in $t9

#print text for opcode

li $v0,4

la $a0,opcode

syscall

#print the binary number that we just stored into $t9, as an integer

li $v0,1

move $a0,$t9

syscall

rs\_binary:

#intialize $t9 to have 0 in it

move $t9,$zero

addi $s1,$s1,4 #24th position

lw $t1,($s1) # 7th number from array

addi $s1,$s1,4 #28th position

lw $t2,($s1) # 8th number from array

addi $s1,$s1,4 #32nd position

lw $t3,($s1) # 9th number from array

addi $s1,$s1,4 #36th position

lw $t4,($s1) # 10th number from array

addi $s1,$s1,4 #40th position

lw $t5,($s1) # 11th number from array

# in this block, we in a way merge the values stored in the previous registers.

# by using OR with $t9 we first add the value on the left hand side of the binary number

# then we sll, to shift the binary number left by 1 position, to make space for the next value to be added in that position

or $t9,$t9,$t1

sll $t9,$t9,1

or $t9,$t9,$t2

sll $t9,$t9,1

or $t9,$t9,$t3

sll $t9,$t9,1

or $t9,$t9,$t4

sll $t9,$t9,1

or $t9,$t9,$t5

# Now we have the specified binary number, stored in $t9

#print text for rs

li $v0,4

la $a0,rs

syscall

#print the binary number that we just stored into $t9, as an integer

li $v0,1

move $a0,$t9

syscall

rt\_binary:

#intialize $t9 to have 0 in it

move $t9,$zero

addi $s1,$s1,4 #44th position

lw $t1,($s1) # 12th number from array

addi $s1,$s1,4 #48th position

lw $t2,($s1) # 13th number from array

addi $s1,$s1,4 #52nd position

lw $t3,($s1) # 14th number from array

addi $s1,$s1,4 #56th position

lw $t4,($s1) # 15th number from array

addi $s1,$s1,4 #60th position

lw $t5,($s1) # 16th number from array

# in this block, we in a way merge the values stored in the previous registers.

# by using OR with $t9 we first add the value on the left hand side of the binary number

# then we sll, to shift the binary number left by 1 position, to make space for the next value to be added in that position

or $t9,$t9,$t1

sll $t9,$t9,1

or $t9,$t9,$t2

sll $t9,$t9,1

or $t9,$t9,$t3

sll $t9,$t9,1

or $t9,$t9,$t4

sll $t9,$t9,1

or $t9,$t9,$t5

# Now we have the specified binary number, stored in $t9

#print text for rt

li $v0,4

la $a0,rt

syscall

#print the binary number that we just stored into $t9, as an integer

li $v0,1

move $a0,$t9

syscall

rd\_binary:

move $t9,$zero

addi $s1,$s1,4 #64th position

lw $t1,($s1) # 17thnumber from array

addi $s1,$s1,4 #68th position

lw $t2,($s1) # 18th number from array

addi $s1,$s1,4 #72nd position

lw $t3,($s1) # 19th number from array

addi $s1,$s1,4 #76th position

lw $t4,($s1) # 20th number from array

addi $s1,$s1,4 #80th position

lw $t5,($s1) # 21st number from array

# in this block, we in a way merge the values stored in the previous registers.

# by using OR with $t9 we first add the value on the left hand side of the binary number

# then we sll, to shift the binary number left by 1 position, to make space for the next value to be added in that position

or $t9,$t9,$t1

sll $t9,$t9,1

or $t9,$t9,$t2

sll $t9,$t9,1

or $t9,$t9,$t3

sll $t9,$t9,1

or $t9,$t9,$t4

sll $t9,$t9,1

or $t9,$t9,$t5

# Now we have the specified binary number, stored in $t9

#print text for rd

li $v0,4

la $a0,rd

syscall

#print the binary number that we just stored into $t9, as an integer

li $v0,1

move $a0,$t9

syscall

shamt\_binary:

move $t9,$zero

addi $s1,$s1,4 #84th position

lw $t1,($s1) # 22nd number from array

addi $s1,$s1,4 #88th position

lw $t2,($s1) # 23rd number from array

addi $s1,$s1,4 #92nd position

lw $t3,($s1) # 24th number from array

addi $s1,$s1,4 #96th position

lw $t4,($s1) # 25th number from array

addi $s1,$s1,4 #100th position

lw $t5,($s1) # 26th number from array

# in this block, we in a way merge the values stored in the previous registers.

# by using OR with $t9 we first add the value on the left hand side of the binary number

# then we sll, to shift the binary number left by 1 position, to make space for the next value to be added in that position

or $t9,$t9,$t1

sll $t9,$t9,1

or $t9,$t9,$t2

sll $t9,$t9,1

or $t9,$t9,$t3

sll $t9,$t9,1

or $t9,$t9,$t4

sll $t9,$t9,1

or $t9,$t9,$t5

# Now we have the specified binary number, stored in $t9

#print text for shamt

li $v0,4

la $a0,shamt

syscall

#print the binary number that we just stored into $t9, as an integer

li $v0,1

move $a0,$t9

syscall

funct\_binary:

move $t9,$zero

addi $s1,$s1,4 #104th position

lw $t1,($s1) # 27th number from array

addi $s1,$s1,4 #108th position

lw $t2,($s1) # 28th number from array

addi $s1,$s1,4 #112th position

lw $t3,($s1) # 29th number from array

addi $s1,$s1,4 #116th position

lw $t4,($s1) # 30th number from array

addi $s1,$s1,4 #120th position

lw $t5,($s1) # 31st number from array

addi $s1,$s1,4 #124th position

lw $t6,($s1) # 32nd number from array

# in this block, we in a way merge the values stored in the previous registers.

# by using OR with $t9 we first add the value on the left hand side of the binary number

# then we sll, to shift the binary number left by 1 position, to make space for the next value to be added in that position

or $t9,$t9,$t1

sll $t9,$t9,1

or $t9,$t9,$t2

sll $t9,$t9,1

or $t9,$t9,$t3

sll $t9,$t9,1

or $t9,$t9,$t4

sll $t9,$t9,1

or $t9,$t9,$t5

sll $t9,$t9,1

or $t9,$t9,$t6

# Now we have the specified binary number, stored in $t9

#print text for funct

li $v0,4

la $a0,funct

syscall

#print the binary number that we just stored into $t9, as an integer

li $v0,1

move $a0,$t9

syscall

j exit\_prog

exit\_prog:

#ends program gracefully

li $v0,10

syscall