Lab1: Basics of MIPS

```
.data
myMsg1: .asciiz "ENTER A NUMBER: "
myMsg2: .asciiz "ENTER ANOTHER NUMBER: "
myMsg3: .asciiz "THE SUM IS: "
.text
main:

la $a0 myMsg1
li $v0 4
syscall

li $v0 5
syscall

move $t0,$v0

la $a0 myMsg2
li $v0 4
syscall

li $v0 5
syscall

move $t1,$v0

la $a0 myMsg3
li $v0 4
syscall

add $a0 $t0 $t1

li $v0 1
syscall

li $v0 10
Syscall
```

Lab2: Basics of MIPS

.word : 32-bit quantities in successive memory words
.half : 16-bit quantities in successive memory half-word
.byte : 8-bit quantities in successive memory bytes
.ascii str : store string in memory but do not null terminate it
    Strings are represented by in double quotes str
    Special characters like \n \t follow C convention.
.asciiz str : store the string in memory and null terminate it
.float f1,...fn : store n floating point single precision numbers in successive memory
.double d1,...dn: store n floating point double precision number in successive memory locations.
.space n : reserves n successive bytes of space.

Layout of code in QTSPIM

.data -> variable declaration follows this line
main: -> entry point of program

LAB2 CODE

```
.data
myMsg: .asciiz "Enter a number: "
myMsg2: .asciiz "Enter another number: "
myMsg3: .asciiz "Sum of the two numbers is: "
.text
main:
#this is used to print myMsg
la $a0 myMsg
li $v0 4
syscall

#this is used to read input into v0
li $v0 5
syscall

#add number to a1
addi $a1 $v0 0

la $a0 myMsg2
li $v0 4
syscall
```

```
li $v0 5
syscall

#add numbers from a1 and v0
add $a1 $v0 $a1

#print myMsg3
la $a0 myMsg3
li $v0 4
syscall

#v0 = 1 will print content of a0
addi $a0 $a1 0
li $v0 1
syscall

li $v0 10
syscall
```

Code 2: sum and difference

```
.data
myMsg: .asciiz "Enter a number: "
myMsg2: .asciiz "Enter another number: "
myMsg3: .asciiz "Sum of the two numbers is: "
myMsg4: .asciiz "\nDifference of the two numbers is: "
.text
main:
la $a0 myMsg
li $v0 4
syscall

li $v0 5
syscall

addi $a1 $v0 0

la $a0 myMsg2
```

```
li $v0 4
syscall

li $v0 5
syscall

add $a2 $a1 $v0

sub $a3 $a1 $v0

la $a0 myMsg3
li $v0 4
syscall

addi $a0 $a2 0
li $v0 1
syscall

la $a0 myMsg4
li $v0 4
syscall

addi $a0 $a3 0
li $v0 1
syscall

li $v0 10
syscall
```

CODE3 : solve a set of linear equations in 2 vars
ax + by = c
dx + ey = f
Get values of x,y for given a,b,c,d,e,f
Y = (cd-ef)/(bd-ea)
X = (ce-fb)/(ae-db)

```
.data
myMsg: .asciiz "Value of Y= "
```

```
myMsg2: .asciiz "\nValue of X= "
.text
main:
li $t0 3
li $t1 5
li $t2 31
li $t3 4
li $t4 3
li $t5 23

mult $t2 $t3
mflo $a1
mult $t4 $t5
mflo $a0
sub $a1 $a1 $a0

mult $t1 $t3
mflo $a2
mult $t4 $t0
mflo $a0
sub $a2 $a2 $a0

div $a1 $a2

mflo $a1
la $a0 myMsg
li $v0 4
syscall

addi $a0 $a1 0
li $v0 1
syscall

mult $t2 $t4
mflo $a1
mult $t1 $t5
mflo $a0
sub $a1 $a1 $a0
```

```
mult $t4 $t0
mflo $a2
mult $t1 $t3
mflo $a0
sub $a2 $a2 $a0

div $a1 $a2
mflo $a2

la $a0 myMsg2
li $v0 4
syscall

addi $a0 $a2 0
li $v0 1
syscall

li $v0 10
syscall
```

When two registers are multiplied together, the overflow will be stored in the HI register and the actual value (without the overflow) is stored in the LO register.
When two registers are divided the quotient is stored in the LO register and the remainder is stored in the HI register.
Mult <reg> <reg> -> multiply
Div <reg1> <reg2> -> divide reg1 by reg2.
Mflo <reg> -> move from lo to reg.
Mfhi <reg> -> move from hi to reg.

Lab 3: Basics of MIPS

Add 2 numbers form memory and store result in memory:

```
.data
num1: .word 20
num2: .word 10
num3: .word 0
.text
main:
lw $a0 num2
lw $a1 num1
add $a1 $a0 $a1
sw $a1 num3
lw $a2 num3
li $v0 10
syscall
```

*Print A*B, A/B and A%B for 2 ints A,B*

```
.data
msg1: .asciiz "Enter A: \n"
msg2: .asciiz "\nEnter B: \n"
msg3: .asciiz "\nA*B: \n"
msg4: .asciiz "\nA/B: \n"
msg5: .asciiz  "\nA%B: \n"
.text
main:
la $a0 msg1
li $v0 4
syscall
li $v0 5
syscall
addi $a1 $v0 0
la $a0 msg2
li $v0 4
syscall
li $v0 5
```

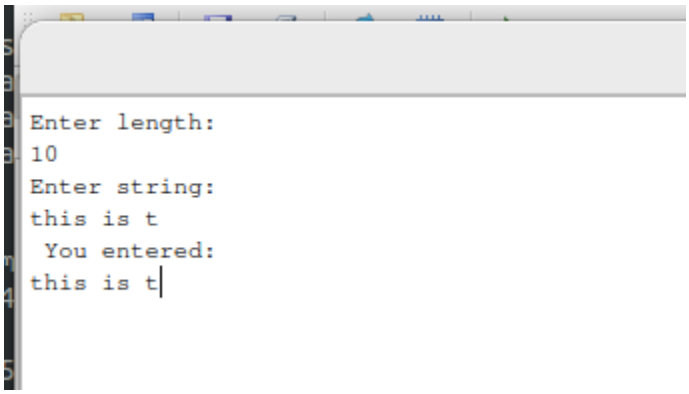
```
syscall
addi $a2 $v0 0
mult $a1 $a2
la $a0 msg3
li $v0 4
syscall
mflo $a0
li $v0 1
syscall
div $a1 $a2
la $a0 msg4
li $v0 4
syscall
mflo $a0
li $v0 1
syscall
la $a0 msg5
li $v0 4
syscall
mfhi $a0
li $v0 1
syscall
li $v0 10
syscall
```

Take length of string and the string as input and print string of that length:

(Here a1 should hold the length of str and a0 will hold the string.)

```
.data
str: .asciiz ""
msg1: .asciiz "Enter length: \n"
msg2: .asciiz "Enter string: \n"
msg3: .asciiz "\n You entered: \n"
.text
main:
la $a0 msg1
li $v0 4
```

```
syscall
li $v0 5
syscall
addi $a1 $v0 0
la $a0 msg2
li $v0 4
syscall
li $v0 8 #take input
la $a0 str #memory address where we store string
move $a2 $a0 #move string to a2
syscall
la $a0 msg3
li $v0 4
syscall
move $a0 $a2
li $v0 4
syscall
li $v0 10
syscall
```



Branching:

Conditional branch: beq -> branch if equal (==)(I-type)
bne -> branch if not equal (!=)(I-type)

Unconditional Branch: j -> jump (J-type)
jr -> jump register (R-type)
jal -> jump and link (J-type)

Eg.
```
beq $s0 $s1 target #if s1==s0 then go to label “target”
#any code after the beq wont we executed as it directly jumps to target
target:
add $s1 $s1 $s0
```

C-code to MIPS

C:
```
if(i==j) f = g+h;
f=f-i;
```

MIPS:
```
#S0 = f, s1 =g, s2 = h, s3=i,s4=j
bne $s3 $s4 L1
add $s0 $s1 $s2
L1: sub $s0  $s0 $s3
```

C:
```
if(i==j) f = g+h;
else
f=f-i;
```

MIPS:
```
#S0 = f, s1 =g, s2 = h, s3=i,s4=j
bne $s3 $s4 L1
add $s0 $s1 $s2
j done
L1: sub $s0  $s0 $s3
done:
```

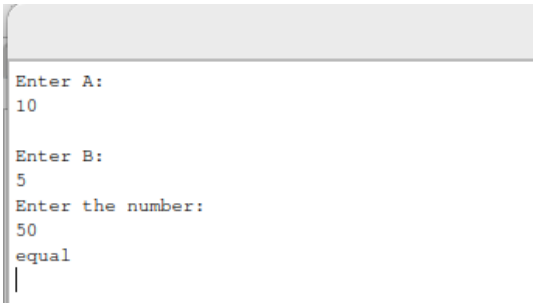
C:
```
int pow = 1;
int x= 0;
while(pow!=128){
      pow = pow*2;
      x = x+1;
}
```

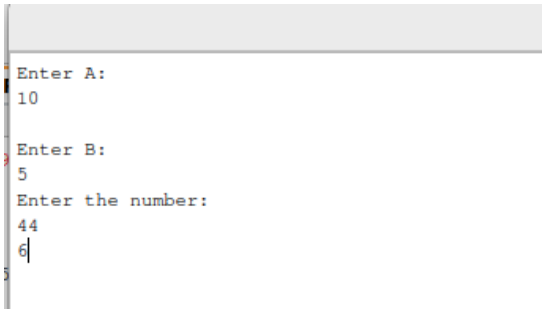
```
MIPS:
S0 = pow s1 = x
addi $s0 $0 1
Add $s1 $0 $0
addi $t0 $0 128
while: add $s0 $s0 $s0
        addi $s1 $s1 1
       bne $s0 $t0 while
```

Take 2 inputs and compare their product with a predefined number and check if they are equal
If they are equal print “equal”
Else print the predefined number/product

```
.data
msg1: .asciiz "Enter A: \n"
msg2: .asciiz "\nEnter B: \n"
msg3: .asciiz "Enter the number: \n"
msg4: .asciiz "equal\n"
.text
main:
la $a0 msg1
li $v0 4
syscall
li $v0 5
syscall
addi $a1 $v0 0
la $a0 msg2
li $v0 4
syscall
li $v0 5
syscall
addi $a2 $v0 0
la $a0 msg3
li $v0 4
syscall
```

```
li $v0 5
syscall
addi $a3 $v0 0
mult $a1 $a2
mflo $a1
beq $a3 $a1 equal
div $a1 $a3
mfhi $a0
li $v0 1
syscall
j done
equal: la $a0 msg4
li $v0 4
syscall
done:
li $v0 10
syscall
```





Print the integral part of average of N consecutive positive integers

```
.data
str1: .asciiz "Enter N:\n"
```

```
str2: .asciiz "Avg is = \n"
.text
main:
la $a0 str1
li $v0 4
syscall
li $v0 5
syscall
add $t0 $v0 $0
addi $t1 $0 0
addi $t2 $0 0
avg:
beq $t0 $t1 exit
addi $t1 $t1 1
add $t2 $t1 $t2
j avg
exit:
la $a0 str2
li $v0 4
syscall
div $t2 $t0
mflo $a0
li $v0 1
syscall
li $v0 10
syscall
```

Lab 4: Floating Point numbers

Ref:
https://ww2.cs.fsu.edu/~dennis/teaching/2013_summer_cda3100/week5/week5-day2.pdf

Floating point numbers and instructions:

Floating point numbers are used to describe real numbers to some degree of precision.
32-bits (single precision/4-bytes) or 64-bits (double precision/8-bytes) are used to represent floats.
Floating-point numbers are represented as a*(10^b)
Floating point registers and the instructions that operate on them are on a separate chip referred to as coprocessor 1.
Uses special registers $f0-$f31
Each reg is used for single precision and 2 reg together are used for double precision.
For double precision: $f0 $f1 are used together. Therefore 32 single precision and 16 double precision are available.

Load and store float from memory:
(use lwc1 type instr for QTSpim)

```
#single precision
lwc1 $f0, 0($t0)
l.s $f0, 0($t0)
( 0($t0) -> t0 has base addr and the number after that is shift from t0)

swc1 $f0 , 0($t0)
s.s $f0, 0($t0)

#double precision
ldc1 $f0, 0($t0)
l.d $f0, 0($t0)

sdc1 $f0 , 0($t0)
s.d $f0 , 0($t0)
```

Usually .s after instructions for single precision and .d for double precision.
Load immediate (pseudo instruction):

```
li.s  $f0, 0.5
li.d $f0 , 0.5
```

To use any floating point operation on 2 non floating point numbers we need to shift the numbers to floating point registers.
FLOPS: Floating point operations per second (rate)
FLOPs: Floating point operations (number)

Print floating point:

```
li $vo ,2(single) /3 (double)
li.s/d $f12 , 0.5
syscall
```

Read floating point:

```
li $v0, 6(single)/7(double)
syscall
```

(read will be stored in $f0)

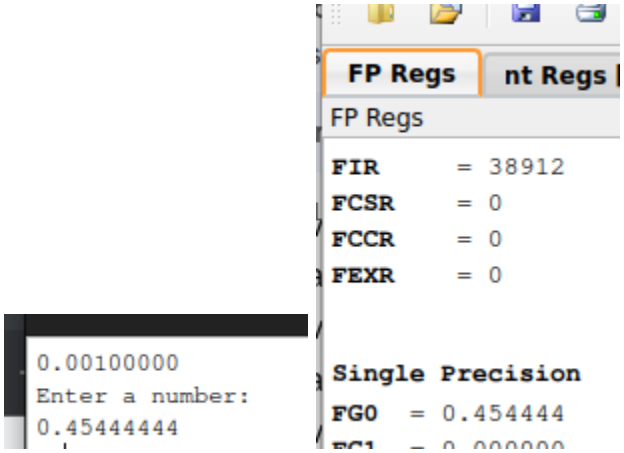
mov.s and mov.d is only between 2 floating point registers
mfc1 $t0, $f0 is from floating point to normal register
mtc1 $t0 , $f0 is from normal register to floating point register.

cvt.s.w $f0,$f1 -> convert 32 bits in f1 to float and store in f0
cvt .w.s $f0 $f1 -> convert float in f1 to integer and store in f0

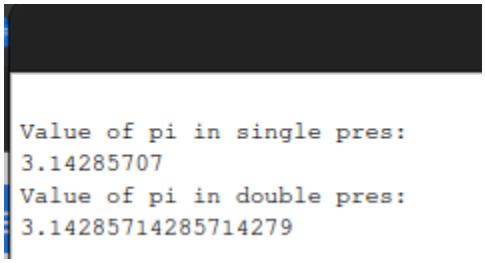
Q1. Print a floating point number from data segment and also read user input for floating point number

```
.data
val1: .float 0.001
str1: .asciiz "\nEnter a number:\n"
.text
main:
l.s $f12,val1
li $v0 2
syscall
la $a0,str1
```

```
li $v0 4
syscall
li $v0,6
syscall
li $v0,10
syscall
```



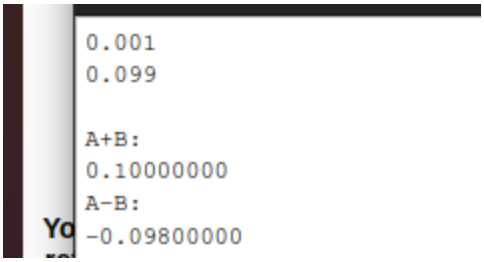
Q2) Print the value of 22/7 in single and double precision:




```
.data
val1: .float 0.001228651
str1: .asciiz "\nValue of pi in single pres: \n"
str2: .asciiz "\nValue of pi in double pres: \n"
.text
main:
```

```
li $t0,22
li $t1,7
mtc1 $t0,$f10
mtc1 $t1,$f12
cvt.s.w $f10 $f10
cvt.s.w $f12 $f12
la $a0,str1
li $v0 4
syscall
div.s $f12,$f10,$f12
li $v0,2
syscall
mtc1 $t0,$f10
mtc1 $t1,$f12
cvt.d.w $f0,$f10
cvt.d.w $f2,$f12
div.d $f12 $f0,$f2
la $a0,str2
li $v0 4
syscall
li $v0,3
syscall
li $v0 10
syscall
```

Q3) Add and subtract 2 numbers that are taken as input from user




```
.data
str1: .asciiz "\nA+B: \n"
str2: .asciiz "\nA-B: \n"
.text
main:
li $v0,6
syscall
mov.s $f1,$f0
li $v0,6
syscall
mov.s $f2,$f0
add.s $f12,$f1,$f2
la $a0,str1
li $v0 4
syscall
li $v0 2
syscall
sub.s $f12,$f1,$f2
la $a0,str2
li $v0 4
syscall
li $v0 2
syscall
li $v0 10
syscall
```

Convert a positive integer to IEEE format {(-1)^s * (1+f)*(2^E)}

```
.data
byte1: .byte 12
.text
main:
li $a1 10
li $s2 0 #and for mantissa
li $s3 23
li $t0 0 #counter to give E
slt $s0 $a1 $0 #check negative
addi $a0 $s0 0 #a0 is IEEE
sll $a0 $a0 31
li $v0 1
syscall
addi $t1 $a1 0 #temp
loop:
beq $t1 $0 equal
srl $t1 $t1 1
addi $t0 $t0 1
j loop
equal:
addi $t0 $t0 -1 #this gives E
addi $t4 $t0 127
sll $t4 $t4 23
add $a0 $a0 $t4
addi $t1 $t0 0
getMantissa:
beq $t1 $0 done
addi $t1 $t1 -1
sll $s2 $s2 1
addi $s2 1
j getMantissa
done:
addi $a3 $a1 0
and $a3 $a3 $s2
sub $s3 $s3 $t0
sll $a3 $a3 $s3
```

```
add $a0 $a0 $a3
addi $a0 $a0 0
li $v0 1
syscall
li $v0 10
syscall
```

Lab 5: Branch, jump, logical and shift operators

Refer: https://www.cs.cmu.edu/afs/cs/academic/class/15740-f97/public/doc/mips-isa.pdf

Use blt, bge and beq instructions to show relation between a0 and a1 (ie > , < ,=)

Code for a0 > a1:

```
.data
str1: .asciiz "a0 more than a1"
.text
main:
li $a0 40
li $a1 20
blt $a0 $a1 exitCall
la $a0 str1
li $v0 4
syscall
exitCall:
li $v0 10
syscall
```

For a0 < a1

```
.data
str1: .asciiz "a0 less than a1"
.text
main:
li $a0 40
li $a1 20
bge $a0 $a1 exitCall
la $a0 str1
li $v0 4
syscall
exitCall:
li $v0 10
syscall
```

For a0 == a1:

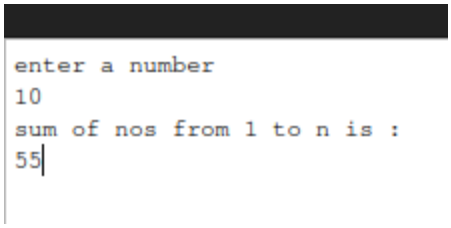
```
.data
```

```
str1: .asciiz "a0 equal to a1"
.text
main:
li $a0 40
li $a1 20
bneq $a0 $a1 exitCall
la $a0 str1
li $v0 4
syscall
exitCall:
li $v0 10
syscall
```

For a0 ≠ a1 :

```
.data
str1: .asciiz "a0 more than a1"
.text
main:
li $a0 40
li $a1 20
beq $a0 $a1 exitCall
la $a0 str1
li $v0 4
syscall
exitCall:
li $v0 10
syscall
```

Find the sum of first n natural numbers:




```
.data
str1: .asciiz "enter a number\n"
str2: .asciiz "sum of nos from 1 to n is : \n"
.text
main:
la $a0 str1
li $v0 4
syscall
li $v0 5
syscall
addi $a0 $v0 0
li $a1 1
li $a2 0
condition:blt $a0 $a1 exitCond
body:   add $a2 $a2 $a0
        addi $a0 $a0 -1
        j condition
exitCond:
la $a0 str2
li $v0 4
syscall
addi $a0 $a2 0
li $v0 1
syscall
li $v0 10
syscall
```

Conditional branching for floating point:

Write a program to compare two floating point numbers

```
.data
str1: .asciiz "f0 more than f12"
str2: .asciiz "enter f12:\n"
str3: .asciiz "enter f0:\n"
.text
main:
la $a0 str2
li $v0 4
syscall
li $v0 6
syscall
mov.s $f12 $f0
la $a0 str3
li $v0 4
syscall
li $v0 6
syscall
c.lt.s $f0 $f12  #set flag for f0 < f12
bc1t exitCall #goto label if flag is set
la $a0 str1
li $v0 4
syscall
exitCall:
li $v0 10
syscall
#c.lt.s => coprocessor, less than, single precision
# use ge, le, gt for the obvious comparison operations
# use d at the end for double precision
# use bc1f for checking if flag is set to false
```

```
enter f12:
0.12
enter f0:
0.15
f0 more than f12
```

Lab 6 Arrays

lb: load byte (same as lw just for bytes)
Characters have size of 1 byte

Write program to check if string is palindrome:

```
.data
str1: .ascii "ThsihT"
str2: .asciiz "Palindrome"
str3: .asciiz "Not palindrome"
len: .word 6
.text
main:
    la $t3 str1
    la $t4 len
    lw $t4 0($t4)
    addi $t4 $t4 -1
    add $t4 $t4 $t3
    loop:
        beq $t3 $t4 palindrome
        slt $s0 $t3 $t4
        beq $s0 $0 palindrome
        addi $s0 $0 0
        lb $s1 0($t3)
        lb $s2 0($t4)
        bne $s1 $s2 notPalindrome
        addi $t3 $t3 1
        addi $t4 $t4 -1
        j loop
    palindrome:
    la $a0 str2
    j exit
```

```
    notPalindrome:
    la $a0 str3
    exit:
    li $v0 4
    syscall
    li $v0 10
    syscall
```

Write MIPS program to find an element in an array:

```
.data
arr: .word 1 3 42 5 6 387 3 53 2
len: .word 9
foundStr: .asciiz "Found element\n"
notFoundStr: .asciiz "Didnt find element\n"
.text
main:
    la $t0 arr
    lw $t1 len
    li $s0 0
    li $a1 12
    loop:
        beq $s0 $t1 notFound
        add $s2 $s0 $s0
        add $s2 $s2 $s2
        add $s2 $s2 $t0
        lw $t3 0($s2)
        beq $a1 $t3 found
        addi $s0 $s0 1
        j loop
    notFound:
        la $a0 notFoundStr
        j exit
```

```
    found:
        la $a0 foundStr
    exit:
        li $v0 4
        syscall
        li $v0 10
        Syscall
```

Reverse a string input from user

(here you should do len-2 to account for null character otherwise console wont print anything coz major skill issue)

```
.data
str: .space 5
rev: .space 5
.text
main:
    li $a1 5
    la $a2 str
    la $a0 str
    li $v0 8
    syscall
    move $t1 $a1
    addi $t1 $t1 -2
    la $a3 rev
    li $s1 0
    loop:
        beq $s1 $a1 exit
        slt $t5 $t1 $0
        bne $t5 $0 exit
        add $s2 $t1 $a2
        lb $t4 0($s2)
        add $s2 $s1 $a3
```

```
        sb $t4 0($s2)
        addi $s1 $s1 1
        addi $t1 $t1 -1
        j loop
    exit:
        la $a0 rev
        li $v0 4
        syscall
        li $v0 10
        syscall
```

Find Min-Max in an array

```
.data
arr: .word 10 2 3 1 32 2 3 553 2 89 22 7
len: .word 12
minStr: .asciiz "\nMin = "
maxStr: .asciiz "\nMax = "
.text
main:
    la $a0 arr
    lw $a1 len
    li $s0 0
    lw $t0 0($a0) #min ele
    lw $t1 0($a0) #max ele
    loop1:
        beq $s0 $a1 next
        add $s1 $s0 $s0
        add $s1 $s1 $s1
        add $s1 $s1 $a0
        lw $s2 0($s1)
        bge $s2 $t0 geq
        move $t0 $s2
```

```
    geq: #just for jump
    addi $s0 $s0 1
    j loop1

next:
    la $a0 minStr
    li $v0 4
    syscall
    move $a0 $t0
    li $v0 1
    syscall
    li $s0 0
    la $a0 arr
    loop2:
        beq $s0 $a1 exit
        add $s1 $s0 $s0
        add $s1 $s1 $s1
        add $s1 $s1 $a0
        lw $s2 0($s1)
        blt $s2 $t1 leq
        move $t1 $s2
        leq: #just for jump
        addi $s0 $s0 1
        j loop2

exit:
    la $a0 maxStr
    li $v0 4
    syscall
    move $a0 $t1
    li $v0 1
    syscall
```

```
li $v0 10
syscall
```

Lab 7 Functions in MIPS

$a0,$a1,$a2 used for passing parameters to functions
$v0 $v1 for return values
$sp -> stack pointer
Stack grows reverse ie top is at lower address

For each call of function:
Decrement stack pointer by 4*(no of registers you want to store) {this is like storing a frame of a function call}
Save the $ra value in stack
Save any variables/parameters that maybe needed by function call ends in stack
Use a0 a1 a2 to pass parameters
FUNCTION CALL (using jal) (return from function using jr)
Retrieve all old parameters and variables using sw/sb
Increment sp to old value
Now use v0 and v1 return value from function

Q4) Find n! Recursively.

```
.data
.text
main:
addi $a0 $0 8
addi $sp $sp -4
sw $ra 0($sp)
jal factorial
lw $ra 0($sp)
addi $sp $sp 4
move $a0 $v0
li $v0 1
syscall
exit:li $v0 10
    syscall
```

```
factorial:
beq $a0 $0 return1
addi $sp $sp -8
sw $a0 0($sp)
sw $ra 4($sp)
addi $a0 $a0 -1
jal factorial
lw $a0 0($sp)
lw $ra 4($sp)
addi $sp $sp 8
mult $v0 $a0
mflo $v0
j exitFac
return1: li $v0 1
exitFac: jr $ra
```

Q5) Fibonacci Sequence recursively

```
    .data
.text
main:
li $a0 20
li $t0 1
addi $sp $sp -4
sw $ra 0($sp)
jal fibonacci
lw $ra 0($sp)
addi $sp $sp 4
move $a0 $v0
li $v0 1
syscall
li $v0 10
```

```
syscall

fibonacci:
beq $a0 $0 baseCase
beq $a0 $t0 baseCase
addi $sp $sp -8
sw $ra 0($sp)
sw $a0 4($sp)
addi $a0 $a0 -1
jal fibonacci
lw $a0 4($sp)
sw $v0 4($sp)
addi $a0 $a0 -2
jal fibonacci
lw $a0 4($sp)
lw $ra 0($sp)
add $v0 $a0 $v0
addi $sp $sp 8
j exit
baseCase: move $v0 $a0
exit:jr $ra
```

Q6) Armstrong Number recursively

```
.data
str1: .asciiz "Arms"
str2: .asciiz "No"
.text
main:
li $a0 153
li $t2 3
```

```
li $t0 10
addi $sp $sp -8
sw $a0 4($sp)
sw $ra 0($sp)
jal armstrong
lw $a0 4($sp)
lw $ra 0($sp)
beq $a0 $v0 arms
la $a0 str2
li $v0 4
syscall
j exit
arms:la $a0 str1
    li $v0 4
    syscall
exit: li $v0 10
syscall

armstrong:
beq $a0 $0 baseCase
addi $sp $sp -8
div $a0 $t0
mfhi $t1
mflo $a0
sw $ra 4($sp)
sw $t1 0($sp)
jal armstrong
lw $ra 4($sp)
lw $t1 0($sp)
```

```
li $t3 0
li $t4 1
addi $sp $sp 8
power: beq $t3 $t2 done
      mult $t1 $t4
      mflo $t4
      addi $t3 $t3 1
      j power
done: add $v0 $t4 $v0
j return
baseCase: li $v0 0
return:   jr $ra
```

Q7) Tower of Hanoi recursively

Lab 8 Exception Mechanisms

Exception handling : Coprocessor 0

Registers in Coprocessor 0 :
BadV Addr : Bad Virtual Address (Memory address where exception occurred. This address doesnt exist and thus gives error)
Status: Interrupt mask, enable bits, status when exception occurred
Cause: Type of exception and pending interrupt bits
EPC: Address of instruction that caused exception (Exception Program Counter)

Read LabSheet for this lab

Lab 9 : *read lab sheets*

Lab 10 Nani yaad aa jaegi

MIPS assembly	C equivalent

After syscall, $v0 points to 12 bytes of free memory (newly allocated)
li $a0,12 *//bytes to be allocated*
li $v0,9 *//sbrk code is 9*
syscall *//now $v0 holds the address of first byte of 12 bytes of free memory*
sw $s0, 0($v0)
sw $s1, 4($v0)
sw $s2, 8($v0)
 	
lw $s0, 0($v0)
lw $s1, 4($v0)
lw $s2, 8($v0)
	a, b, c, ptr are analogous to values of $s0, $s1, $s2, $v0 respectively.
	
	node* ptr = (node*)malloc(sizeof(node));
	
	# ptr->val = a; // $s0 has the value
	# ptr->left = b; // $s1 has left pointer #
	ptr->right = c;// $s2 has right pointer
	
	# a = ptr->val;
	# b= ptr->left;
	# c = ptr->right;

1) Binary search tree and inorder traversal

```
.data
root: .word  0 0 0
arr: .word 10 23 3 42 132 2 6 12
newline: .asciiz "\n"
over: .asciiz "Over\n"
.text
main:
la $a0 root #addr of current node
la $t0 arr #addr of element to be added
li $s0 8
li $s1 0
addi $sp $sp -12
sw $a0 0($sp)
sw $t0 4($sp)
sw $ra 8($sp)
```

```
loop:beq $s0 $s1 done
     add $s2 $s1 $s1
     add $s2 $s2 $s2
     add $a1 $t0 $s2
     lw $a1 0($a1)
     lw $a0 0($sp)
     sw $ra 8($sp)
     jal buildBST
     lw $ra 8($sp)
     addi $s1 $s1 1
     j loop

done:
sw $a1 4($sp)
sw $ra 8($sp)
la $a0 root
sw $a0 0($sp)
jal inorder
lw $a0 0($sp)
lw $a1 4($sp)
lw $ra 8($sp)

la $a0 over
li $v0 4
syscall

addi $sp $sp 12
li $v0 10
syscall

buildBST:
lw $t1 0($a0)
# move $s3 $a0
```

```
beq $t1 $0 rootNULL
move $t2 $a1
bgt $t2 $t1 rightNode
leftNode:
lw $t1 4($a0)
beq $t1 $0 nullNodeLeft
move $a0 $t1
addi $sp $sp -4
sw $ra 0($sp)
jal buildBST
lw $ra 0($sp)
addi $sp $sp 4
jr $ra
rightNode:
lw $t1 8($a0)
beq $t1 $0 nullNodeRight
move $a0 $t1
addi $sp $sp -4
sw $ra 0($sp)
jal buildBST
lw $ra 0($sp)
addi $sp $sp 4
jr $ra
rootNULL:
move $t1 $a1
sw $t1 0($a0)
jr $ra
nullNodeLeft:
move $t1 $a0
li $a0 12
li $v0 9
syscall
sw $t2 0($v0)
```

```
sw $0 4($v0)
sw $0 8($v0)
sw $v0 4($t1)
move $a0 $t1
jr $ra
nullNodeRight:
move $t1 $a0
li $a0 12
li $v0 9
syscall
sw $t2 0($v0)
sw $0 4($v0)
sw $0 8($v0)
sw $v0 8($t1)
move $a0 $t1
jr $ra

inorder:
beq $a0 $0 return
addi $sp $sp -8
sw $a0 0($sp)
sw $ra 4($sp)
lw $a0 4($a0)
jal inorder
lw $a0 0($sp)
move $t1 $a0
lw $a0 0($a0)
li $v0 1
syscall
la $a0 newline
li $v0 4
syscall
move $a0 $t1
```

```
lw $a0 8($a0)
jal inorder
lw $a0 0($sp)
lw $ra 4($sp)
addi $sp $sp 8
return:
jr $ra
```

2) Bubble Sort

```
.data
arr: .word 17 5 92 8741 10 23 55 72 36
space: .asciiz " "
.text
main:
li $t0 9 #len of arr
la $a0 arr

move $a1 $t0
addi $sp $sp -8
sw $a0 0($sp)
sw $ra 4($sp)
jal bubbleSort
lw $a0 0($sp)
lw $ra 4($sp)
addi $sp $sp 8
move $s7 $t0
la $t0 arr
print:
lw $a0,($t0) #load current word in $a0
li $v0,1
syscall #print the current word
la $a0,space
```

```
li $v0,4
syscall #print space in b/w words
addi $t0,$t0,4 #point to next word
addi $t2,$t2,1 #counter++
ble $t2,$s7,print
exit:
li $v0 10
syscall

bubbleSort:
move $t1 $a1
addi $t1 $t1 -1 # i = n-1
outer:
beqz $t1 doneI
addi $t2 $0 0  #j=0
    inner:
    beq $t2 $t1 doneJ
    addi $t3 $t2 1 #k

    add $s0 $t2 $t2
    add $s0 $s0 $s0
    add $s0 $a0 $s0
    lw $s2 0($s0) #arr[j]

    add $s1 $t3 $t3
    add $s1 $s1 $s1
    add $s1 $a0 $s1
    lw $s3 0($s1) #arr[k]

    bgt $s3 $s2 jumpinner
    #swap
    sw $s3 0($s0)
    sw $s2 0($s1)
```

```
        jumpinner:
        addi $t2 $t2 1
        j inner
        doneJ:
        addi $t1 $t1 -1
        j outer
    doneI:
    jr $ra
```

3) Insertion Sort

```
.data
arr: .word 17 5 92 8741 10 23 55 72 36
arr1: .word 9 8 7 6 5 4 3 2 1
space: .asciiz " "
.text
main:
li $t0 9
la $a0 arr
li $a1 9
addi $sp $sp -4
sw $ra 0($sp)
jal insertionSort
lw $ra 0($sp)
addi $sp $sp 4
move $s7 $t0
la $t0 arr
li $t2 0
print:
lw $a0,($t0) #load current word in $a0
li $v0,1
syscall #print the current word
```

```
la $a0,space
li $v0,4
syscall #print space in b/w words
addi $t0,$t0,4 #point to next word
addi $t2,$t2,1 #counter++
blt $t2,$s7,print
exit:
li $v0 10
syscall


insertionSort:
li $t1 1  #c=1
outer:
    beq $t1 $a1 doneC
    move $t2 $t1 #d=c
    inner:
        ble $t2 $0 doneInner
        add $s0 $t2 $t2
        add $s0 $s0 $s0
        add $s0 $a0 $s0
        lw $s1 0($s0)  #arr[d]
        addi $s0 $s0 -4
        lw $s2 0($s0) #arr[d-1]
        bgt $s1 $s2 doneInner
        sw $s1 0($s0) #arr[d-1] <- arr[d]
        addi $s0 $s0 4
        sw $s2 0($s0) #arr[d] <- old arr[d-1]
        addi $t2 $t2 -1
        j inner
    doneInner:
    addi $t1 $t1 1
    j outer
```

```
doneC:
jr $ra
```

4) Merge Sort

```
.data
arr: .word 17 5 92 8741 10 23 55 72 36
space: .asciiz " "
.text
main:
li $t0 9
li $a0 0
li $a1 8
addi $sp $sp -4
sw $ra 0($sp)
jal mergeSort
lw $ra 0($sp)
addi $sp $sp 4
move $s7 $t0
move $t0 $v0
li $t2 0
print:
lw $a0,($t0) #load current word in $a0
li $v0,1
syscall #print the current word
la $a0,space
li $v0,4
syscall #print space in b/w words
addi $t0,$t0,4 #point to next word
addi $t2,$t2,1 #counter++
blt $t2,$s7,print
exit:
li $v0 10
syscall
```

```
mergeSort:
bge $a0 $a1 baseCase
addi $sp $sp -16
sw $a0 0($sp)
sw $a1 4($sp)
sw $ra 8($sp)
add $a1 $a1 $a0
srl $a1 $a1 1
jal mergeSort
lw $a1 4($sp)
sw $v0 12($sp)
lw $a0 0($sp)
add $a0 $a1 $a0
srl $a0 $a0 1
addi $a0 $a0 1
jal mergeSort
lw $a0 0($sp)
lw $a1 4($sp)
lw $ra 8($sp)
lw $v1 12($sp)
move $s4 $v1
move $v1 $v0
move $v0 $s4
addi $sp $sp 16
move $t2 $a0
move $t3 $a1
add $a0 $t2 $t3
srl $a0 $a0 1
sub $a1 $t3 $a0
sub $a0 $a0 $t2
addi $a0 $a0 1
#merge arrays from $v0 $v1
```

```
#$a0 has length of arr in $v0 and $a1 for $v1
li $t2 0
li $t3 0
move $t4 $a0
add $a0 $a0 $a1
add $a0 $a0 $a0
add $a0 $a0 $a0
move $t5 $v0
li $v0 9
syscall
move $s3 $v0 #s3 has new allocated array
move $v0 $t5
move $a0 $t4
move $s4 $s3
merge: beq $t2 $a0 nextV1
    beq $t3 $a1 nextV0
    add $s0 $t2 $t2
    add $s0 $s0 $s0
    add $s0 $s0 $v0
    add $s1 $t3 $t3
    add $s1 $s1 $s1
    add $s1 $s1 $v1
    lw $s0 0($s0)
    lw $s1 0($s1)
    ble $s0 $s1 addV0
    sw $s1 0($s3)
    addi $s3 $s3 4
    addi $t3 $t3 1
    j merge
addV0:
    sw $s0 0($s3)
    addi $s3 $s3 4
    addi $t2 $t2 1
```

```
    j merge
nextV0:beq $t2 $a0 over
    add $s0 $t2 $t2
    add $s0 $s0 $s0
    add $s0 $s0 $v0
    lw $s0 0($s0)
    sw $s0 0($s3)
    addi $s3 $s3 4
    addi $t2 $t2 1
    j nextV0
nextV1: beq $t3 $a1 over
        add $s1 $t3 $t3
        add $s1 $s1 $s1
        add $s1 $s1 $v1
        lw $s1 0($s1)
        sw $s1 0($s3)
        addi $s3 $s3 4
        addi $t3 $t3 1
        j nextV1
over:
move $v0 $s4
jr $ra
baseCase:
la $t1 arr
add $a0 $a0 $a0
add $a0 $a0 $a0
add $a0 $t1 $a0
lw $t1 0($a0)
li $a0 4
li $v0 9
syscall
sw $t1 0($v0)
jr $ra
```

5) Heap Sort

```
.data
arr: .word 17 5 92 8741 10 23 55 72 36
space: .asciiz " "
.text
main:
li $t0 9
la $a0 arr
la $s0 arr
li $a1 9
addi $sp $sp -4
sw $ra 0($sp)
jal heapSort
lw $ra 0($sp)
addi $sp $sp 4
move $s7 $t0
la $t0 arr
li $t2 0
print:
lw $a0,($t0) #load current word in $a0
li $v0,1
syscall #print the current word
la $a0,space
li $v0,4
syscall #print space in b/w words
addi $t0,$t0,4 #point to next word
addi $t2,$t2,1 #counter++
blt $t2,$s7,print
exit:
li $v0 10
```

```
syscall

heapSort:
    srl $t1 $a1 1
    makeHeap: blt $t1 $0 sort
        move $a0 $t1  #a0 = i
        addi $sp $sp -12
        sw $a0 0($sp)
        sw $a1 4($sp)
        sw $ra 8($sp)
        jal heapify
        lw $a0 0($sp)
        lw $a1 4($sp)
        lw $ra 8($sp)
        addi $sp $sp 12
        addi $t1 $t1 -1
        j makeHeap
    sort:
    addi $a0 $a1 -1
    sortLoop:
        blt $a0 $0 done
        lw $s3 0($s0) #arr[0]
        add $s4 $a0 $a0
        add $s4 $s4 $s4
        add $s4 $s4 $s0
        lw $t4 0($s4) #arr[i]
        sw $s3 0($s4)
        sw $t4 0($s0)  #swap arr[i] arr[0]
        addi $sp $sp -12
        sw $a0 0($sp)
        sw $a1 4($sp)
        sw $ra 8($sp)
        move $a1 $a0
```

```
        li $a0 0
        jal heapify
        lw $a0 0($sp)
        lw $a1 4($sp)
        lw $ra 8($sp)
        addi $sp $sp 12
        addi $a0 $a0 -1
        j sortLoop
        done:
            jr $ra

heapify:
        move $t4 $a0
        add $t2 $a0 $a0
        addi $t2 $t2 1 #t2 = 2i + 1 -> left
        addi $t3 $t2 1 #t3 = 2i +2  -> right
        bge $t2 $a1 notLeft
        add $s2 $t2 $t2
        add $s2 $s2 $s2
        add $s2 $s2 $s0
        lw $s2 0($s2)  #arr[left]
        add $s4 $a0 $a0
        add $s4 $s4 $s4
        add $s4 $s4 $s0
        lw $s4 0($s4) #arr[largest]
        ble $s2 $s4 notLeft
        move $a0 $t2
    notLeft:
        bge $t3 $a1 notRight
        add $s3 $t3 $t3
        add $s3 $s3 $s3
        add $s3 $s3 $s0
        lw $s3 0($s3)  #arr[right]
```

```
    add $s4 $a0 $a0
    add $s4 $s4 $s4
    add $s4 $s4 $s0
    lw $s4 0($s4) #arr[largest]
    ble $s3 $s4 notRight
    move $a0 $t3
notRight:
    beq $t4 $a0 returnHeapify
    add $s3 $t4 $t4
    add $s3 $s3 $s3
    add $s3 $s3 $s0
    lw $t5 0($s3)  #arr[i]
    add $s4 $a0 $a0
    add $s4 $s4 $s4
    add $s4 $s4 $s0
    lw $t6 0($s4) #arr[largest]
    sw $t6 0($s3)
    sw $t5 0($s4)

    addi $sp $sp -12
    sw $a0 0($sp)
    sw $a1 4($sp)
    sw $ra 8($sp)
    jal heapify
    lw $a0 0($sp)
    lw $a1 4($sp)
    lw $ra 8($sp)
    addi $sp $sp 12

returnHeapify: jr $ra
```

6) Depth First Search

```
.data
space: .asciiz " "
newLine: .asciiz "\n"
startNodePrompt : .asciiz "\nEnter the starting Node:\n"
.text
main:
li $v0 5
syscall
move $a0 $v0
move $a1 $a0        #a1 = node count
addi $a0 $a0 1
mult $a0 $a0
mflo $a0
add $a0 $a0 $a0
add $a0 $a0 $a0
li $v0 9
syscall
move $t0 $v0        #base of adj matrix
addi $a0 $a1 1
li $s0 1
li $s1 1
matrixInput:
    outer:
        beq $s0 $a0 outerDone
        li $s1 1
        inner:
            beq $s1 $a0  innerDone
            mult $s0 $a1
            mflo $s2
            add $s2 $s2 $s1  #ni+j
            add $s2 $s2 $s2
            add $s2 $s2 $s2
```

```
            add $s2 $s2 $t0
            li $v0 5
            syscall
            sw $v0 0($s2)
            addi $s1 $s1 1
            j inner
        innerDone:
        addi $s0 $s0 1
        j outer
    outerDone:
li $s0 1
li $s1 1
addi $t1 $a1 1
li $s0 1
matrixPrint:
    outerP:
        beq $s0 $t1 outerDoneP
        li $s1 1
        innerP:
            beq $s1 $t1  innerDoneP
            mult $s0 $a1
            mflo $s2
            add $s2 $s2 $s1  #ni+j
            add $s2 $s2 $s2
            add $s2 $s2 $s2
            add $s2 $s2 $t0
            lw $a0 0($s2)
            li $v0 1
            syscall
            la $a0 space
            li $v0 4
            syscall
            addi $s1 $s1 1
```

```
            j innerP
        innerDoneP:
        la $a0 newLine
        li $v0 4
        syscall
        addi $s0 $s0 1
        j outerP
    outerDoneP:
la $a0 startNodePrompt
li $v0 4
syscall
li $v0 5
syscall
move $t1 $v0 #t1 has starting node
addi $a0 $a1 1
li $s5 4
mult $s5 $a0
mflo $a0
li $v0 9
syscall
move $t2 $v0  #t2 has visited array
li $s0 0
initZero:
    beq $s0 $a0 next
    add $s1 $s0 $t2
    sw $0 0($s1)
    addi $s0 $s0 4
    j initZero
next:
move $a0 $t1
addi $sp $sp -8
sw $a0 0($sp)
sw $ra 4($sp)
```

```
jal DFS
#lw $a0 0($sp)
lw $ra 4($sp)
addi $sp $sp 8

exit: li $v0 10
syscall

DFS: #a0 has starting node ,t2 has visited arr base , t0 has adj
matrix base
    add $s0 $a0 $a0
    add $s0 $s0 $s0
    add $s0 $s0 $t2
    lw $s1 0($s0)
    li $s5 1
    beq $s1 $s5 return
    li $v0 1
    syscall
    move $s1 $a0
    la $a0 space
    li $v0 4
    syscall
    move $a0 $s1
    li $s1 1
    sw $s1 0($s0)
    move $s1 $a0
    mult $a0 $a1
    mflo $s1  #ni
    li $s2 1
    loop:
        bgt $s2 $a1 return
        add $s3 $s2 $s1 #ni + j
        add $s3 $s3 $s3
```

```
        add $s3 $s3 $s3
        add $s3 $s3 $t0
        lw $s3 0($s3) #adj[i][j]
        beq $s3 $0 continue
        move $a0 $s2
        addi $sp $sp -16
        sw $a0 0($sp)
        sw $ra 4($sp)
        sw $s2 8($sp)
        sw $s1 12($sp)
        jal DFS
        lw $a0 0($sp)
        lw $ra 4($sp)
        lw $s2 8($sp)
        lw $s1 12($sp)
        addi $sp $sp 16
        continue:
        addi $s2 $s2 1
        j loop
    return: jr $ra
```

8) Breadth-First search along with queue implementation

```
.data
queue: .space 400
space: .asciiz " "
newLine: .asciiz "\n"
startNodePrompt: .asciiz "\nEnter the starting Node:\n"
noOfNodes: .asciiz "\nEnter Number of nodes:\n"
.text
main:
la $a0 noOfNodes
li $v0 4
```

```
syscall
li $v0 5
syscall
move $a0 $v0
move $a1 $a0        #a1 = node count
addi $a0 $a0 1
mult $a0 $a0
mflo $a0
add $a0 $a0 $a0
add $a0 $a0 $a0
li $v0 9
syscall
move $t0 $v0        #base of adj matrix
addi $a0 $a1 1
li $s0 1
li $s1 1
matrixInput:
    outer:
        beq $s0 $a0 outerDone
        li $s1 1
        inner:
            beq $s1 $a0  innerDone
            mult $s0 $a1
            mflo $s2
            add $s2 $s2 $s1  #ni+j
            add $s2 $s2 $s2
            add $s2 $s2 $s2
            add $s2 $s2 $t0
            li $v0 5
            syscall
            sw $v0 0($s2)
            addi $s1 $s1 1
            j inner
```

```
        innerDone:
        addi $s0 $s0 1
        j outer
    outerDone:
li $s0 1
li $s1 1
addi $t1 $a1 1
li $s0 1
matrixPrint:
    outerP:
        beq $s0 $t1 outerDoneP
        li $s1 1
        innerP:
            beq $s1 $t1  innerDoneP
            mult $s0 $a1
            mflo $s2
            add $s2 $s2 $s1  #ni+j
            add $s2 $s2 $s2
            add $s2 $s2 $s2
            add $s2 $s2 $t0
            lw $a0 0($s2)
            li $v0 1
            syscall
            la $a0 space
            li $v0 4
            syscall
            addi $s1 $s1 1
            j innerP
        innerDoneP:
        la $a0 newLine
        li $v0 4
        syscall
        addi $s0 $s0 1
```

```
        j outerP
    outerDoneP:
la $a0 startNodePrompt
li $v0 4
syscall
li $v0 5
syscall
move $t1 $v0 #t1 has starting node
addi $a0 $a1 1
li $s5 4
mult $s5 $a0
mflo $a0
li $v0 9
syscall
move $t2 $v0  #t2 has visited array
li $s0 0
initZero:
    beq $s0 $a0 next
    add $s1 $s0 $t2
    sw $0 0($s1)
    addi $s0 $s0 4
    j initZero
next:
    la $t3 queue #has queue
    li $t4 -1 #has front
    li $t5 -1 #has back
    move $a0 $t1
    addi $sp $sp -4
    sw $ra 0($sp)
    jal pushQ  #push first element
    lw $ra 0($sp)
    addi $sp $sp 4
startBFS:
```

```
beq $t5 $t4 exit
addi $sp $sp -4
sw $ra 0($sp)
jal popQ
lw $ra 0($sp)
addi $sp $sp 4
move $a0 $v0
li $v0 1
syscall
move $s0 $a0
la $a0 space
li $v0 4
syscall
move $a0 $s0
add $s0 $s0 $s0
add $s0 $s0 $s0
add $s0 $t2 $s0
li $s1 1
sw $s1 0($s0)  #vis[node] = 1
mult $a1 $a0
mflo $s0
li $s1 1
addNeighbour:
    bgt $s1 $a1 nextNode
    add $s2 $s1 $s0
    add $s2 $s2 $s2
    add $s2 $s2 $s2
    add $s2 $t0 $s2
    lw $s2 0($s2)
    beq $s2 $0 continue
    move $s4 $s1
    add $s4 $s4 $s4
    add $s4 $s4 $s4
```

```
        add $s4 $s4 $t2
        lw $s4 0($s4)
        bne $s4 $0 continue  #check for visited
        move $a0 $s1
        addi $sp $sp -4
        sw $ra 0($sp)
        jal pushQ  #push neighbour
        lw $ra 0($sp)
        addi $sp $sp 4
        continue:
            addi $s1 $s1 1
            j addNeighbour
    nextNode:
        j startBFS
exit:
    li $v0 10
    syscall

pushQ:
    li $s4 -1
    bne $t4 $s4 notemptyQPush
    li $t4 0
    li $t5 0
    notemptyQPush:
    add $s4 $t5 $t3
    sw $a0 0($s4)
    add $t5 $t5 4
    jr $ra

popQ:
    beq $t4 $t5 emptyQpop
    add $s4 $t3 $t4
    lw $v0 0($s4)
```

```
    addi $t4 $t4 4
    jr $ra
    emptyQpop:
    li $t4 -1
    li $t5 -1
    jr $ra
```

I wanna cry