A blue and gold logo

AI-generated content may be incorrect.

**San Francisco Bay University**

**EE488 - Computer Architecture**

**Homework Assignment #5**

**Due day: 4/4/2025**

**Instruction:**

1. **The homework answer sheet should contain the original questions and corresponding answers.**
2. **The answer sheet must be in MS-Word file format with Github links for the programming questions. As follows is the answer sheet name format.**

***<course\_id>\_week<week\_number>\_StudentID\_FirstName\_LastName.pdf***

1. **The program name in Github must follow the format like   
   *<course\_id>\_week<week\_number>\_q<question\_number>\_StudentID\_FirstName\_LastName***
2. **Show screenshot of all running results, including the system date/time.**
3. **The calculation process must be typed if needed, handwriting can’t be accepted.**
4. **Only accept homework submission uploaded via Canvas.**
5. **Overdue homework submission can’t be accepted.**
6. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**
7. Implement a subprogram that prompt the user for *3* numbers, finds the median (middle value) of the *3*, and returns that value to the calling program.

.data

prompt1: .asciiz "Enter first number: "

prompt2: .asciiz "Enter second number: "

prompt3: .asciiz "Enter third number: "

result: .asciiz "Median is: "

newline: .asciiz "\n"

.text

.globl main

main:

# --- Read a, b, c into $s0, $s1, $s2 ---

li $v0, 4

la $a0, prompt1

syscall

li $v0, 5

syscall

move $s0, $v0 # s0 = a

li $v0, 4

la $a0, prompt2

syscall

li $v0, 5

syscall

move $s1, $v0 # s1 = b

li $v0, 4

la $a0, prompt3

syscall

li $v0, 5

syscall

move $s2, $v0 # s2 = c

# --- Compute sum = a + b + c in $t4 ---

add $t4, $s0, $s1

add $t4, $t4, $s2

# --- Find max(a,b) into $t0 ---

slt $t1, $s0, $s1 # t1 = 1 if a < b

bne $t1, $zero, L1 # if a<b, go to L1 (b is larger)

move $t0, $s0 # else a ≥ b → t0 = a

j L2

L1: move $t0, $s1 # t0 = b

L2:

# t0 now holds max(a,b)

slt $t1, $t0, $s2 # t1 = 1 if max(a,b) < c

bne $t1, $zero, L3 # if so, c is the overall max

# else max(a,b) ≥ c

# t0 already max(a,b)

j GotMax

L3: move $t0, $s2 # c is the overall max

GotMax:

# --- Now t0 = M = max(a,b,c) ---

# --- Find min(a,b) into $t1 ---

slt $t2, $s1, $s0 # t2 = 1 if b < a

bne $t2, $zero, L4 # if b<a, go to L4

move $t1, $s0 # else a ≤ b → t1 = a

j L5

L4: move $t1, $s1 # t1 = b

L5:

# t1 now holds min(a,b)

slt $t2, $s2, $t1 # t2 = 1 if c < min(a,b)

bne $t2, $zero, L6 # if so, c is the overall min

# else min(a,b) ≤ c

j GotMin

L6: move $t1, $s2 # c is the overall min

GotMin:

# --- Now t1 = m = min(a,b,c) ---

# --- median = sum − max − min → $t2 ---

sub $t2, $t4, $t0 # t2 = sum - max

sub $t2, $t2, $t1 # t2 = (a+b+c) - max - min

# --- Print ---

li $v0, 4

la $a0, result

syscall

move $a0, $t2

li $v0, 1

syscall

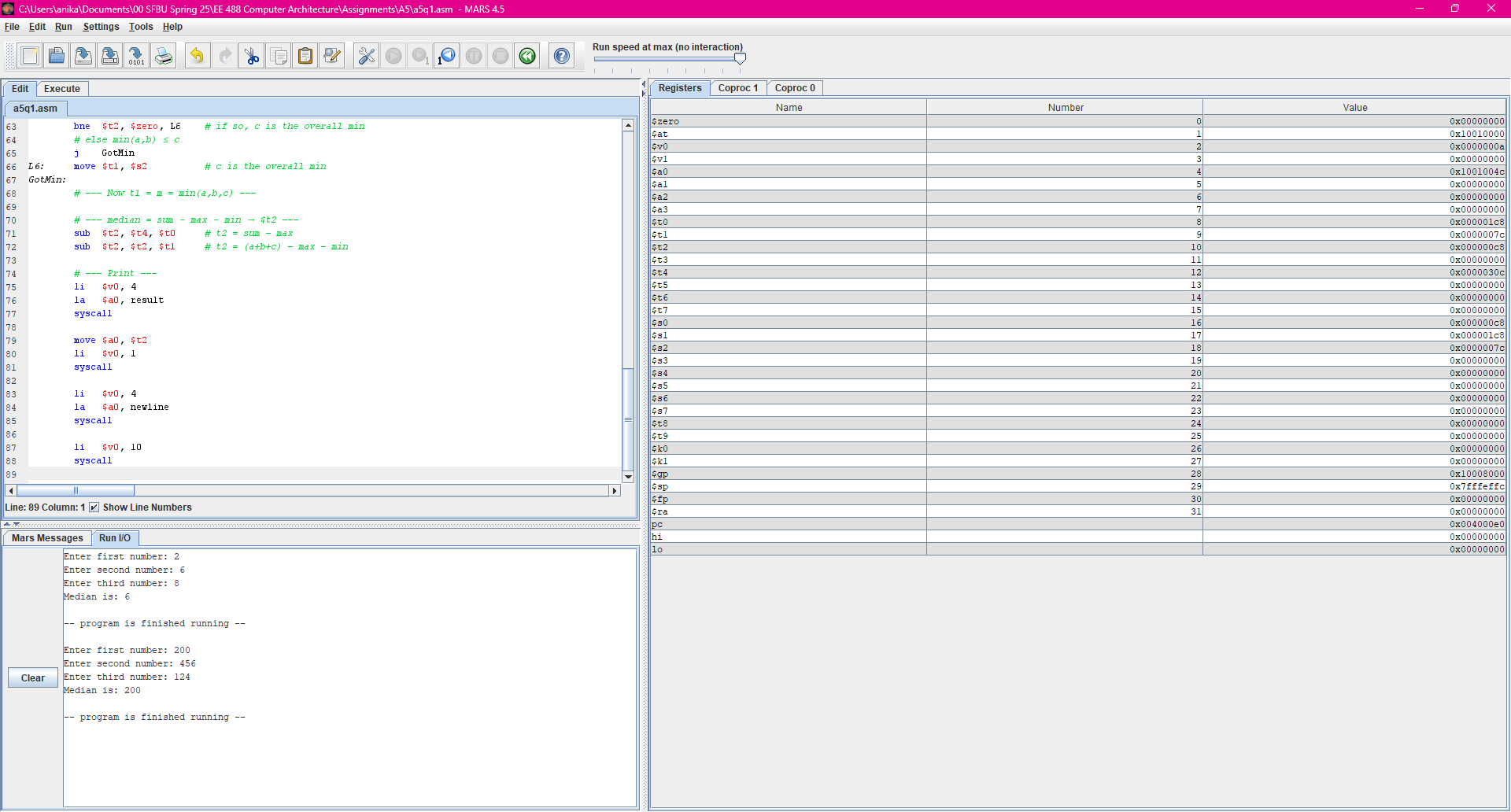
li $v0, 4

la $a0, newline

syscall

li $v0, 10

syscall



1. Implement a recursive program that takes in a number and finds the square of that number through addition. For example if the number *3* is entered, you would add *3+3+3=9*. If *4* is entered, you would add *4+4+4+4=16*. This program must be implemented using recursion to add the numbers together.

.data

prompt: .asciiz "Enter n: "

result\_msg: .asciiz "Square is: "

newline: .asciiz "\n"

.text

.globl main

main:

# --- Prompt & read n into $v0 ---

li $v0, 4

la $a0, prompt

syscall

li $v0, 5

syscall

move $s1, $v0 # $s1 = original n (saved)

move $a0, $v0 # $a0 = countdown = n

# --- Compute n\*n via recursion; result in $v0 ---

jal square\_rec

# --- Preserve the true result before printing the label ---

move $t0, $v0 # t0 = n\*n

# --- Print "Square is: " ---

li $v0, 4

la $a0, result\_msg

syscall

# --- Print the numeric result from $t0 ---

move $a0, $t0

li $v0, 1

syscall

# --- Newline & exit ---

li $v0, 4

la $a0, newline

syscall

li $v0, 10

syscall

# square\_rec:

# in: $a0 = how many copies of n to add

# $s1 = original n (never changed)

# out: $v0 = original\_n \* (initial\_count)

square\_rec:

addiu $sp, $sp, -4

sw $ra, 0($sp)

blez $a0, base\_zero # if count ≤ 0 → square = 0

addiu $a0, $a0, -1 # decrement count

jal square\_rec # recurse

addu $v0, $v0, $s1 # add one copy of n

j unwind

base\_zero:

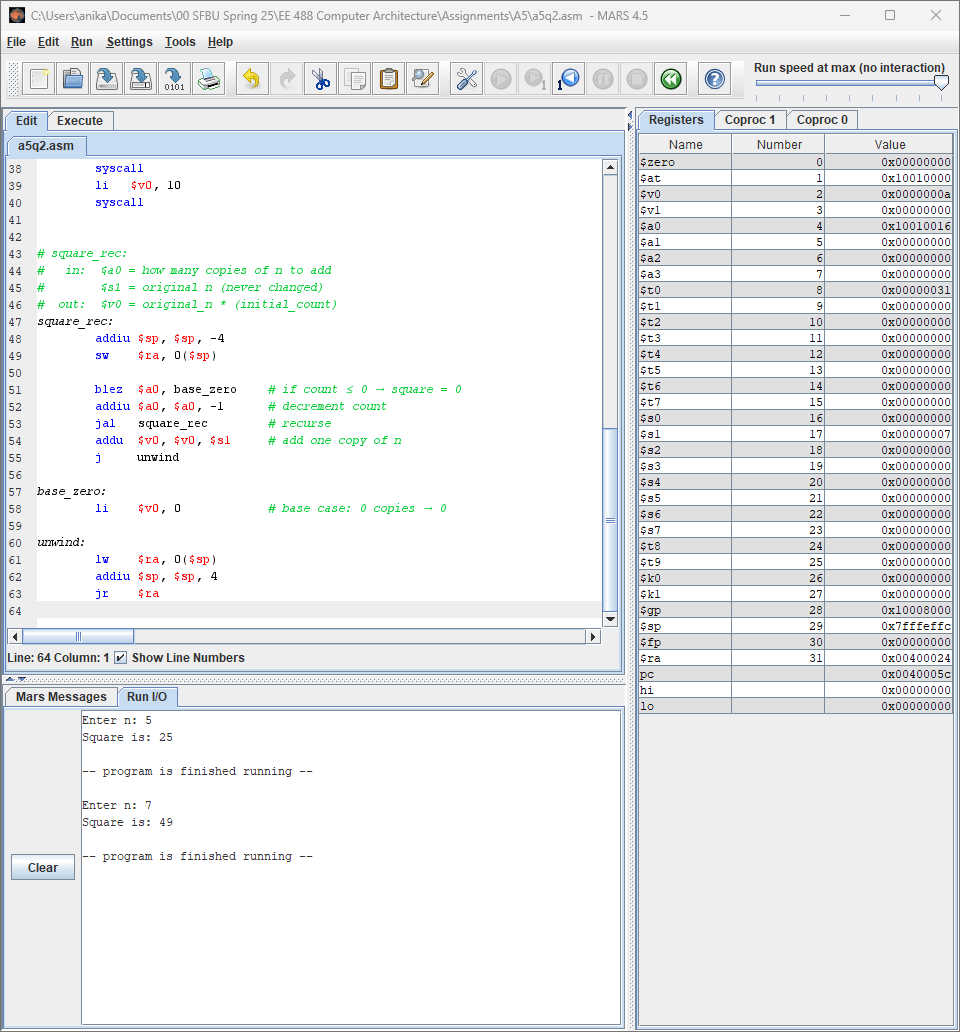
li $v0, 0 # base case: 0 copies → 0

unwind:

lw $ra, 0($sp)

addiu $sp, $sp, 4

jr $ra



1. Write a recursive program to calculate factorial numbers. Use the definition of factorial as *F(n) = n \* F(n-1)*

.data

prompt: .asciiz "Enter n: "

result\_msg: .asciiz "Factorial is: "

newline: .asciiz "\n"

.text

.globl main

main:

# --- Prompt & read n into $a0 ---

li $v0, 4

la $a0, prompt

syscall

li $v0, 5

syscall

move $a0, $v0 # $a0 = n

# --- Compute factorial(n) → $v0 ---

jal factorial

# --- Preserve result before printing label ---

move $t0, $v0 # t0 = n!

# --- Print "Factorial is: " ---

li $v0, 4

la $a0, result\_msg

syscall

# --- Print the numeric result from $t0 ---

move $a0, $t0

li $v0, 1

syscall

# --- Newline & exit ---

li $v0, 4

la $a0, newline

syscall

li $v0, 10

syscall

# factorial:

# in: $a0 = n

# out: $v0 = n!

factorial:

addiu $sp, $sp, -8

sw $ra, 4($sp)

sw $a0, 0($sp) # save original n

li $t1, 1

ble $a0, $t1, fact\_base # if n ≤ 1, return 1

# recursive case: compute fact(n-1)

addiu $a0, $a0, -1

jal factorial

lw $t2, 0($sp) # reload original n

mul $v0, $v0, $t2 # v0 = (n-1)! \* n

j fact\_done

fact\_base:

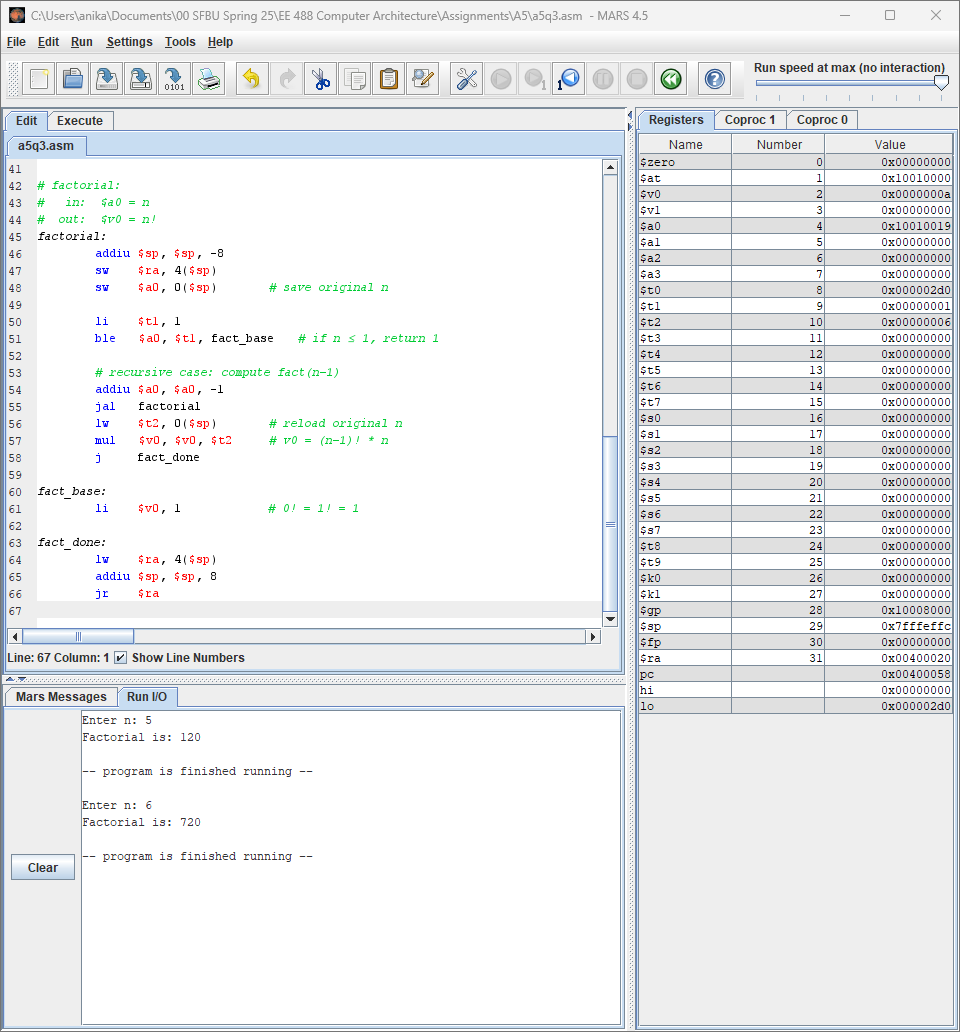
li $v0, 1 # 0! = 1! = 1

fact\_done:

lw $ra, 4($sp)

addiu $sp, $sp, 8

jr $ra



1. The following pseudo code converts an input value of a single decimal number from

into a single hexadecimal digit. Translate this pseudo code into MIPS assembly.

*main{*

*String a[16]*

*a[0] = "0x0"*

*a[1] = "0x1"*

*a[2] = "0x2"*

*a[3] = "0x3"*

*a[4] = "0x4"*

*a[5] = "0x5"*

*a[6] = "0x6"*

*a[7] = "0x7"*

*a[8] = "0x8"*

*a[9] = "0x9"*

*a[10] = "0xa"*

*a[11] = "0xb"*

*a[12] = "0xc"*

*a[13] = "0xd"*

*a[14] = "0xe"*

*a[15] = "0xf"*

*int i = prompt("Enter a number from 0 to 15 ")*

*print("your number is " + a[i])*

*}*

.data

hex0: .asciiz "0x0"

hex1: .asciiz "0x1"

hex2: .asciiz "0x2"

hex3: .asciiz "0x3"

hex4: .asciiz "0x4"

hex5: .asciiz "0x5"

hex6: .asciiz "0x6"

hex7: .asciiz "0x7"

hex8: .asciiz "0x8"

hex9: .asciiz "0x9"

hexa: .asciiz "0xa"

hexb: .asciiz "0xb"

hexc: .asciiz "0xc"

hexd: .asciiz "0xd"

hexe: .asciiz "0xe"

hexf: .asciiz "0xf"

table: .word hex0,hex1,hex2,hex3,hex4,hex5,hex6,hex7

.word hex8,hex9,hexa,hexb,hexc,hexd,hexe,hexf

newline: .asciiz "\n"

prompt: .asciiz "Enter a value 0-15: "

.text

.globl main

main:

# --- prompt for digit ---

li $v0, 4

la $a0, prompt

syscall

# --- read integer into t0 ---

li $v0, 5

syscall

move $t0, $v0 # t0 = user input (0–15)

# --- compute address: table + t0\*4 ---

la $t1, table

sll $t2, $t0, 2 # byte offset = input\*4

add $t1, $t1, $t2

# --- load the address of the right string into a0 ---

lw $a0, 0($t1)

# --- print the hex string ---

li $v0, 4

syscall

# --- print newline ---

li $v0, 4

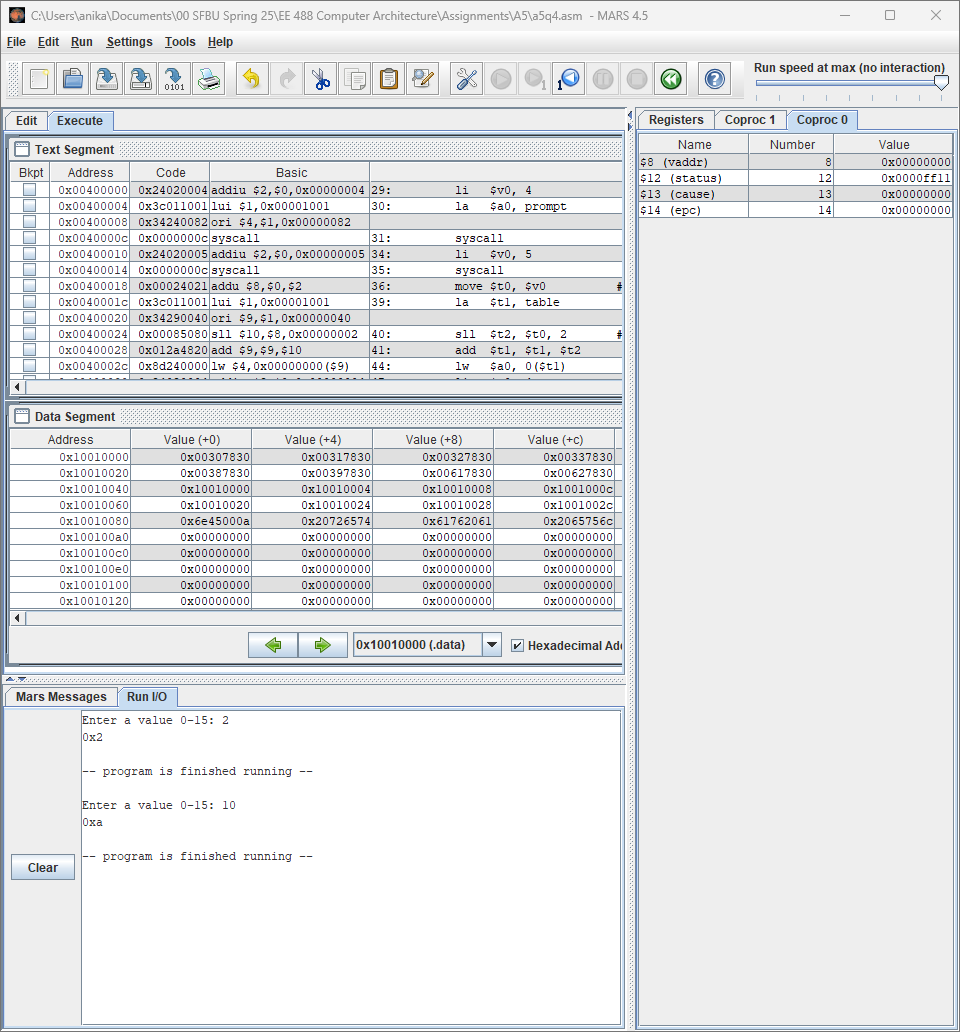
la $a0, newline

syscall

# --- exit ---

li $v0, 10

syscall



1. The following pseudo code program calculates the Fibonacci numbers from *1…n*, and stores them in an array. Translate this pseudo code into MIPS assembly, and use the PrintIntArray subprogram to print the results.

*main{*

*int size = PromptInt(“Enter a max Fibonacci number to calc: “)*

*int Fibonacci[size]*

*Fibonacci[0] = 0*

*Fibonacci[1] = 1*

*for (int i = 2; i < size; i++){*

*Fibonacci[i] = Fibonacci[i-1] + Fibonacci[i-2]*

*}*

*PrintIntArray(Fibonacci, size)*

*}*

.data

prompt: .asciiz "Enter max Fibonacci index: "

space: .asciiz " "

newline: .asciiz "\n"

.text

.globl main

main:

# --- Prompt & read n into $s0 ---

li $v0,4

la $a0,prompt

syscall

li $v0,5

syscall

move $s0,$v0 # length = n

# --- Allocate n words on stack for Fib[0..n-1] ---

sll $t0,$s0,2 # t0 = n\*4

subu $sp,$sp,$t0

move $s1,$sp # s1 = base of Fib[]

# --- Base cases ---

sw $zero,0($s1) # Fib[0] = 0

li $t1,1

li $t2,2

blt $s0,$t2,skip1 # if n<2 skip Fib[1]

sw $t1,4($s1) # Fib[1] = 1

skip1:

# --- Fill Fib[2..n-1] ---

li $t3,2 # i = 2

fill\_loop:

bge $t3,$s0,done\_fill

sll $t4,$t3,2

add $t5,$s1,$t4 # &Fib[i]

# load Fib[i-1]

addi $t6,$t3,-1

sll $t6,$t6,2

add $t6,$s1,$t6

lw $t7,0($t6)

# load Fib[i-2]

addi $t8,$t3,-2

sll $t8,$t8,2

add $t8,$s1,$t8

lw $t9,0($t8)

add $t7,$t7,$t9 # Fib[i] = Fib[i-1] + Fib[i-2]

sw $t7,0($t5)

addi $t3,$t3,1

j fill\_loop

done\_fill:

# --- Print the array ---

move $a0,$s1 # base pointer

move $a1,$s0 # length

jal PrintIntArray

# --- Restore stack & exit ---

move $sp,$s1

li $v0,10

syscall

#-------------------------------------------------------

# PrintIntArray: prints each element of the int array on one line.

# In: $a0 = base address, $a1 = length

PrintIntArray:

addiu $sp,$sp,-12

sw $ra,8($sp)

sw $s0,4($sp)

sw $s1,0($sp)

move $s0,$a1 # counter = length

move $s2,$a0 # save base pointer in s2

li $t0,0 # index i = 0

print\_loop:

beq $t0,$s0,end\_print

sll $t1,$t0,2

add $t2,$s2,$t1 # t2 = address of Fib[i]

lw $a0,0($t2) # load Fib[i]

li $v0,1

syscall # print integer

addi $t0,$t0,1

blt $t0,$s0,do\_space

j print\_loop

do\_space:

li $v0,4

la $a0,space

syscall # print a space

j print\_loop

end\_print:

li $v0,4

la $a0,newline

syscall # final newline

lw $s1,0($sp)

lw $s0,4($sp)

lw $ra,8($sp)

addiu $sp,$sp,12

jr $ra

