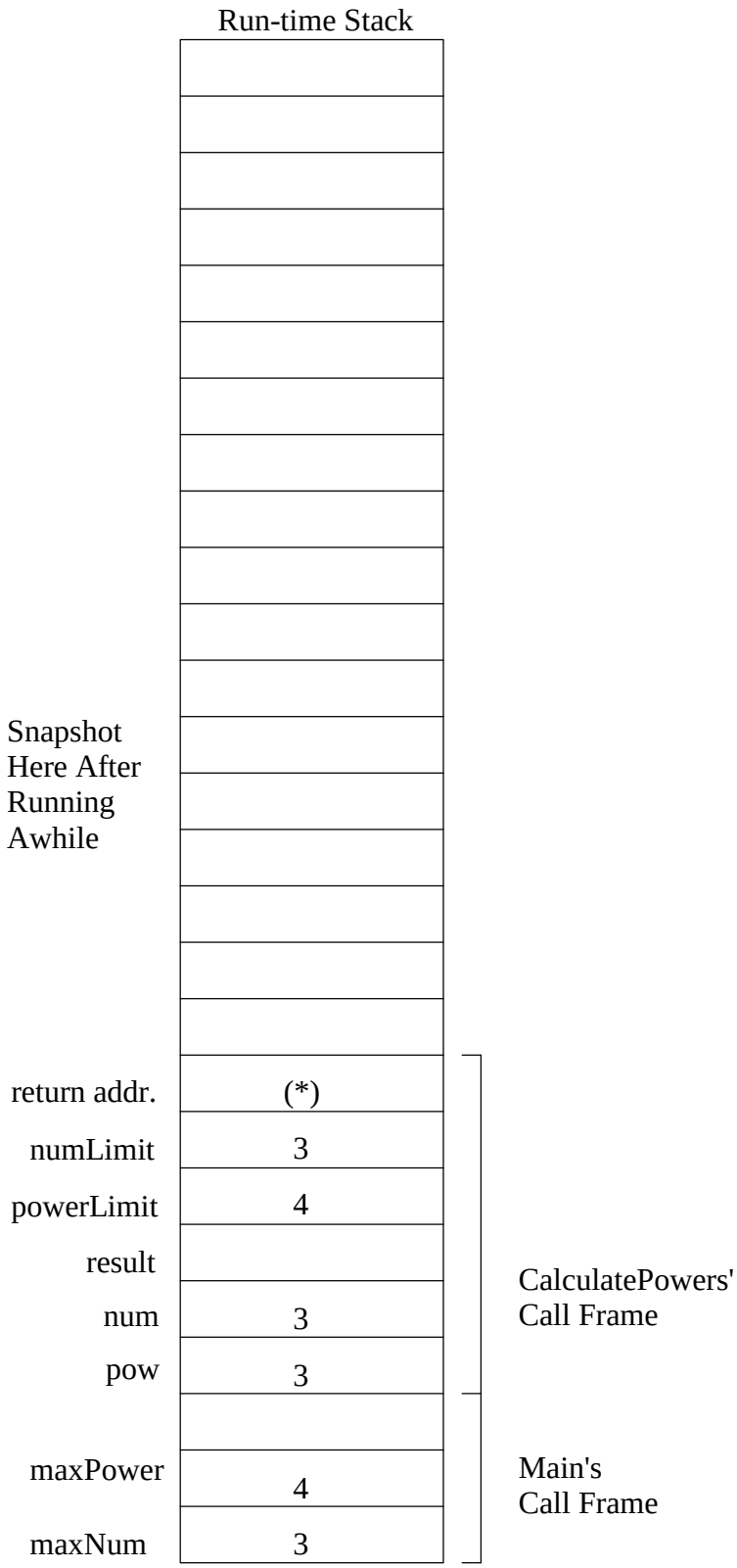
Name:_____

High-level Language Programmer's View

```
main:
maxNum = 3
maxPower = 4

CalculatePowers(maxNum, maxPower)
(*)
. . .
end main


CalculatePowers(In: integer numLimit,
                    integer powerLimit)
integer num, pow, result

for num := 1 to numLimit do
    for pow := 1 to powerLimit do
        Power(num, pow, result)
(**)    print num " raised to " pow " power is "
                result
    end for pow
end for num

end CalculatePowers


Power( In:  integer n, integer e, Out: result)

if e = 0 then
    result = 1
else if e = 1 then
    result = n
else
    Power(n, e - 1, result)
    result = result * n          (***)
end if

end Power
```

Snapshot Here After Running Awhile



1) Trace the execution of the recursive function Power by showing the run-time stack.

2) What is the most number of call frames on the stack for the **whole** program?

Name:_____

Assembly-language Programmer's View

3) Trace the hypothetical assembly language program and indicate the resulting value of the registers Reg1, Reg2, Reg3, and Reg4.

```
        .data
X:      .WORD     2                                  ; variable X initialized at assembly time to 2
Y:      .WORD     3                                  ; variable Y initialized at assembly time to 3
Z:      .WORD     0                                  ; variable Z initialized at assembly time to 0

        .program
Begin:
        LOAD Reg1, X                                 ; loads X's value into register Reg1
        LOAD Reg2, Y
        ZERO  Reg3                                   ; sets Reg3's value to 0
        MOVE Reg4, Reg2                              ; Reg4 := Reg2
Loop:
        ADD   Reg3, Reg3, Reg1                       ; Reg3 := Reg3 + Reg1
        SUB_IMMEDIATE  Reg4, Reg4, #1                ; Reg4 := Reg4 - 1
        BRANCH_GREATER_THAN_ZERO Reg4, Loop          ; if Reg4 > 0 then goto Loop label
        STORE  Reg3, Z                               ; store Reg3's value into variable Z
End:
```

	Reg1	Reg2	Reg3	Reg4
Resulting register values				

a) What is the resulting value in Z?

b) What calculation does this code perform?

4) During the execution of the above assembly language code: (Assuming no cache)

a) How many memory reads were performed? (state any assumptions)

data reads =

instruction reads (assume one read per instruction-fetch) =

b) How many memory writes were performed? (state any assumptions)

5) List (in decreasing order of importance) why somebody would write assembly language code.

(top reason) a)

b)

c)

Name:_____

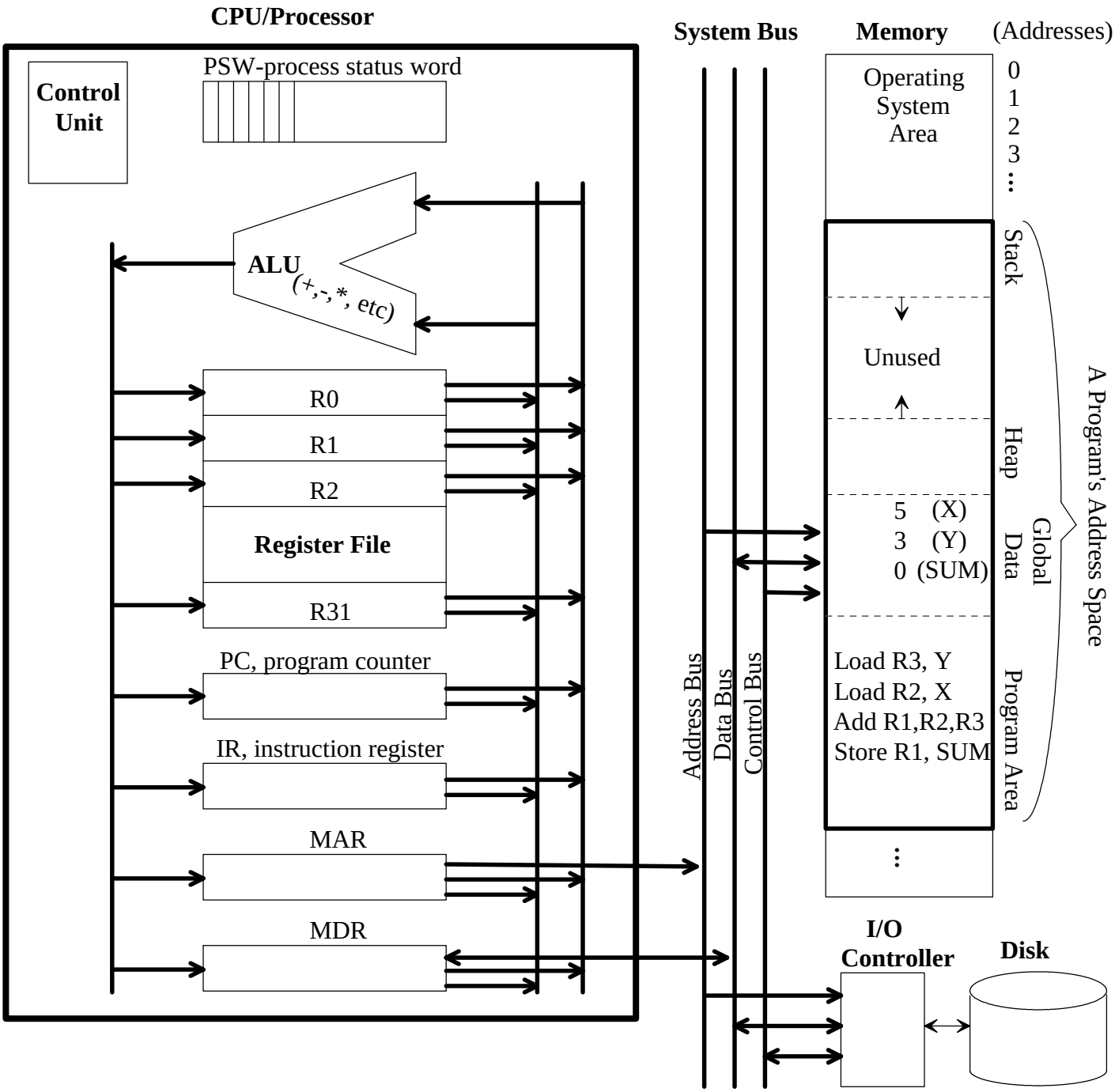
6) “Moore’s law” (Gordon Moore - cofounder of Intel) - predicts that the number of transistors that could be put on a single chip would double every year (later changed to 18 months). What kind of curve (# transistors vs. time) does Moore’s law predict? (linear, quadratic, exponential, etc.)

7) What impact does Moore’s law have on Computer Architectures?

8) In the "Machine-Instruction Weighted" columns in Table 13.2 (Stalling) why does a subroutine call require many assembly-language instructions?

9) In the "Memory-Reference Weighted" columns in Table 13.2 (Stallings) why does the subroutine call take even longer when memory accesses are considered?

Name:_____



Instruction/Machine Cycle of stored-program computer - repeat all day

1. Fetch Instruction - read instruction pointed at by the program counter (PC) from memory into Instr. Reg. (IR)

2. Decode Instruction - figure out what kind of instruction was read
3. Fetch Operands - get operand values from the memory or registers
4. Execute Instruction - do some operation with the operands to get some result
5. Write Result - put the result into a register or in a memory location

(Note: Sometime during the above steps, the PC is updated to point to the next instruction.)

Name:_____

Type of Instruction	**MIPS Assembly Language**	**Register Transfer Language Description**
Memory Access (Load and Store)	lw $4, Mem	$4← [Mem]
	sw $4, Mem	Mem←$4
	lw $4, 16($3)	$4← [Mem at address in $3 + 16]
	sw $4, Mem	[Mem at address in $3 + 16]← $4
Move	move $4, $2	$4← $2
	li $4, 100	$4← 100
Load Address	la $5, mem	$4← load address of mem
Arithmetic Instruction (reg. operands only)	add $4, $2, $3	$4← $2 + $3
	mul $10, $12, $8	$10← $12 * $8 (32-bit product)
	sub $4, $2, $3	$4← $2 - $3
Arithmetic with Immediates (last operand must be an integer)	addi $4, $2, 100	$4← $2 + 100
	mul $4, $2, 100	$4← $2 * 100 (32-bit product)
Conditional Branch	bgt $4, $2, LABEL (bge, blt, ble, beq, bne)	Branch to LABEL if $4 > $2
Unconditional Branch	j LABEL	Always Branch to LABEL

Fibonacci Sequence:	0	1	1	2	3	5	8	13	21
Position in Sequence:	0	1	2	3	4	5	6	7	8

A high-level language program to calculate the nth fibonacci number would be:

```
temp2 = 0
temp3 = 1

for i = 2 to n do
    temp4 = temp2 + temp3
    temp2 = temp3
    temp3 = temp4
end for

result = temp4
```

Trace of Program (time ⟶)

HLL variables								MIPS registers
temp2	0	1	1	2	3	5	8	$2
temp3	1	1	2	3	5	8	13	$3
temp4	1	2	3	5	8	13	21	$4
i	2	3	4	5	6	7	8	$6

A complete assembly language MIPS program to calculate the nth fibonacci number.

```
                .data
n:              .word  8          # variable in memory
result:         .word  0          # variable in memory

                      .text
                      .globl main
main:       li    $2, 0
            li    $3, 1
            lw    $5, n # load "n" into $5
            li    $6,
for_loop:   bgt   $6, $5, end_for
            add   $4, $2, $3
            move  $2, $3
            move  $3, $4
            addi  $6, $6, 1
            j     for_loop
end_for:
            sw    $4, result

            li    $v0, 10         # system code for exit
            syscall
```