Team #:_____ 8/22/06
Absent:

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

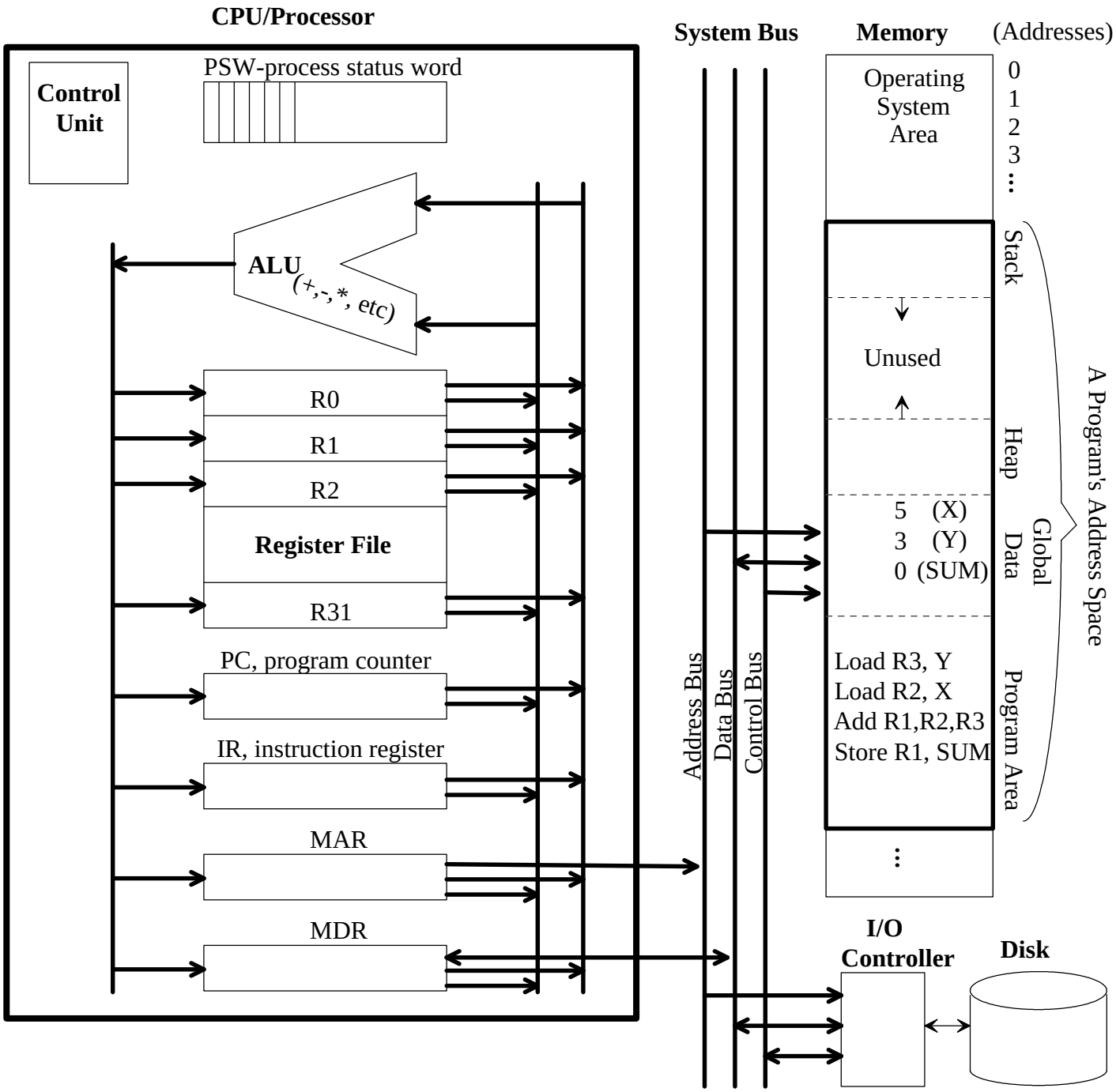
Run-time Stack

return addr.	(*)	CalculatePowers' Call Frame
numLimit	3	
powerLimit	4	
result		
num	3	
pow	3	
		Main's Call Frame
maxPower	4	
maxNum	3	

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

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

Team #:_____ 8/22/06
Absent:



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.)

Team #:_____ 8/22/06
Absent:

RISC 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                                          ; Variables setup in MEMORY before execution
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)

6) Is the operating system hardware or software?

Team #:_____ 8/22/06
Absent:

7) How is a computer system protected from a user program that goes into an infinite loop?

8) Modern CPU's have *dual-modes of operation* where there are two (or more) modes of operation: user mode and system(/supervisor/monitor) mode. A mode-bit(s) within the CPU's *processor-status-word* (PSW) register is used to indicate whether the CPU is executing in user or system mode. The set of all machine-language instructions are divided into:

a) privileged instructions that can only be executed in system mode, and

b) non-privileged instructions that can be executed in any mode of operation.

Every time an instruction is executed by the CPU, the hardware checks to see if the instruction is privileged and whether the mode is user. Whenever this case is detected, an exception (internal interrupt) is generated that turns CPU control back over to the operating system.

Can you think of some privileged instructions related to the answer to question (7) above?

9). Suppose we had a block transfer from an I/O device to memory. The block consists of 1024 words and one word can be transferred at a time. For each of the following, indicate the number of interrupts needed to transfer a block:

a) programmed-I/O

b) interrupt-driven I/O

c) DMA (direct-memory access)

8) Assume special I/O instructions are used to fill I/O-controller registers. Why can't a user program use these instructions to communicate with the I/O device directly and "by-pass" the operating system's protection checking?

9) Assume that memory-mapped I/O is used. Since Load and Store instructions are used to communicate with the I/O-controller registers, why can't a user program communicate with the I/O device directly and "by-pass" the operating system's protection checking?