Team #:____ Name:_____
Absent: 8/21/07

Below is the description of the hardware features of a desktop PC:

- Intel® Core™ 2 Duo Processor E6300 (2MB L2 Cache,1.86GHz,1066)
- Genuine Windows Vista™ Home Premium
- 2 GB Dual-Channel DDR2 SDRAM (667MHz), expandable to 4 GB
- 500 GB Serial ATA Hard Drive
- DVD+RW/CD-RW Drive
- 3.5" Floppy Drive and 13-in-1 Media Reader
- Graphics card: 512MB NVIDIA GeForce 7600 GS
- Sound Blaster X-Fi™ XtremeMusic with Dolby 5.1
- Video: 1 DVI, VGA and 1 S-Video (with add-in PCI-Express video card)
- 9 USB 2.0 ports and 2 IEEE 1394 (FireWire) ports
- Integrated (10/100/1000) Gigabit Ethernet
- Expansion Slots: 3 PCI Slots, 1 PCIe x1 Slot, 1 PCIe x16 (Graphics) Slots, 1 PCIe x4/x8 Slot

1) What does the processor do?

2) What is stored in main memory (RAM)?

3) What is stored on the hard disk?

4) What is the purpose of cache memory?

5) What terms relate to interconnection of internal PC components?

6) What terms relate to interconnection of external PC components?

7) What is a KB, MB, GB, Gigabit, MHz, GHz?

8) What is the role of the operating system?

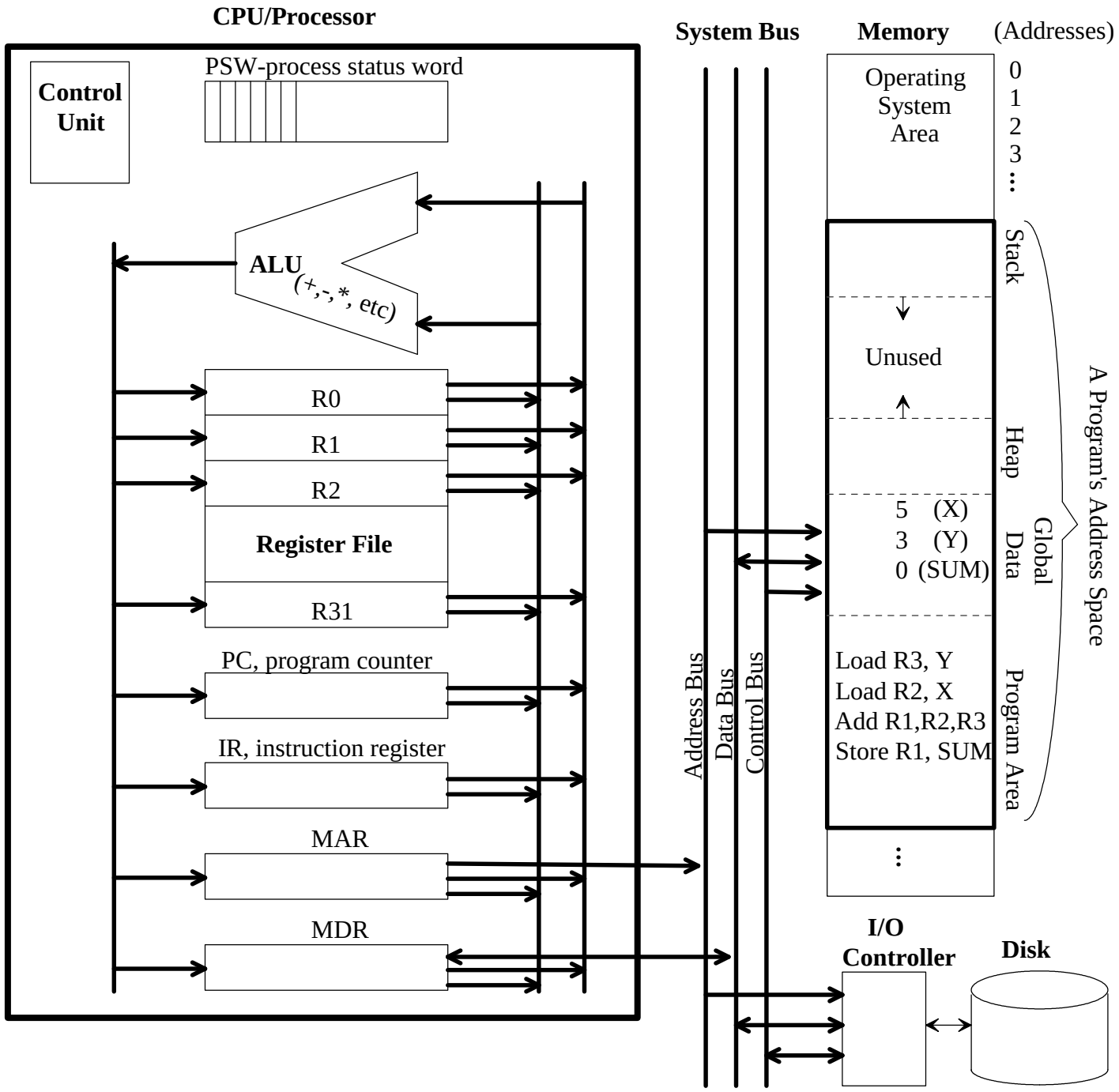
9) What is the role of a compiler?

10) What advantages do high-level languages (Ada, C, C++, Java, COBOL, etc.) have over assembly language?

11) Why do people program in assembly language (AL)?

Team #:____ Name:_____

Absent: 8/21/07



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 #:____ Name:_____
Absent: 8/21/07

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 n^{th} 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
```

HLL variables	Trace of Program (time →)							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	$1

A complete assembly language MIPS program to calculate the n^{th} fibonacci number.

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

                      .text
                      .globl main
main:       li    $2, 0           # $2 holds temp2
            li    $3, 1           # $3 holds temp3
for_init:   li    $6, 2           # initialize i ($6) to 2
            lw    $5, n           # load "n" into $5
for_loop:   bgt   $6, $5, end_for
            add   $4, $2, $3      # $4 holds temp4
            move  $2, $3          # shift temp3 to temp2
            move  $3, $4          # shift temp4 to temp3
            addi  $6, $6, 1       # increment i ($6)
            j     for_loop
end_for:
            sw    $4, result      # store the result to memory
            li    $v0, 10         # system code for exit
            syscall               # call the operating system
```