ICS 233 - Computer Architecture & Assembly Language

Exam I – Fall 2007

Saturday, November 3, 2007
7:00 – 9:00 pm

Computer Engineering Department
College of Computer Sciences & Engineering
King Fahd University of Petroleum & Minerals

Student Name: **SOLUTION**

Student ID: ____________________

Q1	/ 15	Q2	/ 15
Q3	/ 15	Q4	/ 10
Q5	/ 10	Q6	/ 15
Q7	/ 20		
Total	/ 100		

Important Reminder on Academic Honesty

Using unauthorized information or notes on an exam, peeking at others work, or altering graded exams to claim more credit are severe violations of academic honesty. Detected cases will receive a failing grade in the course.

Q1. (15 pts) Find the word or phrase that best matches the following descriptions:

a) Program that manages the resources of a computer for the benefit of the programs that run on that machine.

Operating System

b) Program that translates from a high-level notation to assembly language.

Compiler

c) Component of the processor that tells what to do according to the instructions.

Control Unit

e) Interface that the hardware provides to the software.

Instruction Set Architecture

d) Microscopic flaw in a wafer.

Defect

f) Rectangular component that results from dicing a wafer.

Die

g) Computer inside another device used for running one predetermined application or collection of software.

Embedded System

h) (3 pts) In a magnetic disk, the disks containing the data are constantly rotating. On average, it should take half a rotation for the desired data on the disk to spin under the read/write head. Assuming that the disk is rotating at 10000 RPM (Rotations Per Minute), what is the average time for the data to rotate under the disk head?

Average rotational latency = 1/2 * 60 * 1000 (msec /min) / 10000 = 3 milliseconds

i) (5 pts) Assume you are in a company that will market a certain IC chip. The cost per wafer is $5000, and each wafer can be diced into 1200 dies. The die yield is 40%. Finally, the dies are packaged and tested, with a cost of $9 per chip. The test yield is 80%; only those that pass the test will be sold to customers. If the retail price is 50% more than the cost, what is the selling price per chip?

Number of working dies per wafer = 1200 * 0.4 = 480

Packaging cost = 480 * $9 = $4320

Number of working chips that will be sold to customers = 480 * 0.8 = 384

Cost per chip = ($5000 + $4320) / 384 = $24.27

Selling price per chip = $24.27 * 1.5 = $36.4

Q2. (15 pts) Consider the following data definitions:

```
.data
var1:    .byte    3, -2, 'A'
var2:    .half    1, 256, 0xffff
var3:    .word    0x3de1c74, 0xff
.align 3
str1:    .asciiz  "ICS233"
```

a) Show the content of each byte of the allocated memory, **in hexadecimal** for the above data definitions. The **Little Endian** byte ordering is used to order the bytes within words and halfwords. Fill the symbol table showing **all labels** and their **starting address**. The ASCII code of character 'A' is 0x41, and '0' is 0x30. Indicate which bytes are skipped or unused in the data segment.

Data Segment

Address	Byte 0	Byte 1	Byte 2	Byte 3
0x10010000	0x03	0xfe	0x41	--
0x10010004	0x01	0x00	0x00	0x01
0x10010008	0xff	0xff	--	--
0x1001000C	0x74	0x1c	0xde	0x03
0x10010010	0xff	0x00	0x00	0x00
0x10010014	--	--	--	--
0x10010018	0x49	0x43	0x53	0x32
0x1001001C	0x33	0x33	0x00	
0x10010020				
0x10010024				
0x10010028				
0x1001002C				

← **Unused**

Symbol Table

Label	Address
var1	0x10010000
var2	0x10010004
var3	0x1001000C
str1	0x10010018

b) How many bytes are allocated in the data segment including the skipped bytes?

31 Bytes including the skipped ones

Q3. (15 pts) For each of the following pseudo-instructions, produce a **minimal** sequence of real MIPS instructions to accomplish the same thing. You may use the **`$at`** register only as a temporary register.

a) **`abs   $s1, $s2`**

```
addu $s1, $zero, $s2
bgez $s2, next
subu $s1, $zero, $s2
next:
```

b) **`addiu $s1, $s2, imm32   # imm32 is a 32-bit immediate`**

```
lui  $at, upper16
ori  $at, $at, lower16
addu $s1, $s2, $at
```

c) **`bleu $s1, $s2, Label    # branch less than or equal unsigned`**

```
sltu $at, $s2, $s1
beq  $at, $zero, Label
```

d) **`bge $s1, imm32, Label   # imm32 is a 32-bit immediate`**

```
lui $at, upper16
ori $at, $at, lower16
slt $at, $s1, $at
beq $at, $zero, Label
```

e) **`rol $s1, $s2, 5         # rol = rotate left $s2 by 5 bits`**


```
srl $at, $s2, 27
sll $s1, $s2, 5
or  $s1, $s1, $at
```

Q4. (10 pts) Translate the following loop into assembly language where **a** and **b** are integer arrays whose base addresses are in **$a0** and **$a1** respectively. The value of **n** is in **$a2**.

```
for (i=0; i<n; i++) {
  if (i > 2) {
    a[i] = a[i-2] + a[i-1] + b[i];
  }
  else {
    a[i] = b[i]
  }
}
```

```
        li    $t0, 0         # $t0 = i = 0
        beq   $a2, $0, skip  # skip loop if n is zero
loop:   lw    $t1, 0($a1)    # $t1 = b[i]
        bgt   $t0, 2, else   # if (i>2) goto else
        lw    $t2, -8($a0)   # $t2 = a[i-2]
        lw    $t3, -4($a0)   # $t3 = a[i-1]
        addu  $t2, $t2, $t3  # $t2 = a[i-2]+a[i-1]
        addu  $t1, $t2, $t1  # $t1 = a[i-2]+a[i-1]+b[i]
else:   sw    $t1, 0($a0)    # a[i] = $t1
        addiu $a0, $a0, 4    # advance array a pointer
        addiu $a1, $a1, 4    # advance array b pointer
        addiu $t0, $t0, 1    # i++
        bne   $t0, $a2, loop
skip:
```

Q5. (10 pts) Translate the following **if-else** statement into assembly language:

```
if (($t0 >= '0') && ($t0 <= '9')) {$t1 = $t0 – '0';}
else if (($t0 >= 'A') && ($t0 <= 'F')) {$t1 = $t0+10-'A';}
else if (($t0 >= 'a') && ($t0 <= 'f')) {$t1 = $t0+10-'a';}
```

```
    blt   $t0, '0', else1
    bgt   $t0, '9', else1
    addiu $t1, $t0, -48      # '0' = 48
    j     next
else1:
    blt   $t0, 'A', else2
    bgt   $t0, 'F', else2
    addiu $t1, $t0, -55      # 10-'A' = 10-65=-55
    j     next
else2:
    blt   $t0, 'a', next
    bgt   $t0, 'f', next
    addiu $t1, $t0, -87      # 10-'a' = 10-97=-87
next:
```

Q6. The following code fragment processes two arrays and produces an important result in register **$v0**. Assume that each array consists of **N** words, the base addresses of the arrays **A** and **B** are stored in **$a0** and **$a1** respectively, and their sizes are stored in **$a2** and **$a3**, respectively.

```
        sll    $a2, $a2, 2
        sll    $a3, $a3, 2
        addu   $v0, $zero, $zero
        addu   $t0, $zero, $zero
outer:  addu   $t4, $a0, $t0
        lw     $t4, 0($t4)
        addu   $t1, $zero, $zero
inner:  addu   $t3, $a1, $t1
        lw     $t3, 0($t3)
        bne    $t3, $t4, skip
        addiu  $v0, $v0, 1
skip:   addiu  $t1, $t1, 4
        bne    $t1, $a3, inner
        addiu  $t0, $t0, 4
        bne    $t0, $a2, outer
```

a) (5 pts) Describe what the above code does and what will be returned in register **$v0**.

This code compares every element in the first array against all elements of the second array. It counts the number of matching elements between the two arrays.

$v0 will contain the count of the number of matching elements between the two arrays.

b) (10 pts) Write a loop that calculates the first *N* numbers in the Fibonacci sequence (1, 1, 2, 3, 5, 8, 13, …), where *N* is stored in register **$a0**. Each element in the sequence is the sum of the previous two. Declare an array of words and store the generated elements of the Fibonacci sequence in the array.

```
.data
  fibs: .space 200    # space for 50 integers
.text
.globl main
main:
    # you can read N from the input
    la     $t0, fibs
    li     $t1, 1
    li     $t2, 1
L1:
    sw     $t1, 0($t0)
    addu   $t3, $t1, $t2
    move   $t1, $t2
    move   $t2, $t3
    addiu  $t0, $t0, 4
    addiu  $a0, $a0, -1
    bne    $a0, $zero, L1
```

Q7. (20 Pts) Write MIPS assembly code for the procedure **BinarySearch** to search an array which has been previously sorted. Each element in the array is a 32-bit signed integer. The procedure receives three parameters: register **$a0** = **address of array** to be searched, **$a1** = **size** (number of elements) in the array, and **$a2** = **item** to be searched. If found then **BinarySearch** returns in register **$v0** = **address** of the array element where **item** is found. Otherwise, **$v0** = **0**.

```
BinarySearch ($a0=array, $a1=size, $a2=item) {
  lower = 0;
  upper = size-1;
  while (lower <= upper) {
    middle = (lower + upper)/2;
    if (item == array[middle])
      return $v0 = ADDRESS OF array[middle];
    else if (item < array[middle])
      upper = middle–1;
    else
      lower = middle+1;
  }
  return $v0=0;
}
```

```
BinarySearch:
    li      $t0, 0            # $t0 = lower index
    addiu   $t1, $a1, -1      # $t1 = upper index
loop:
    bgt     $t0, $t1, ret
    addu    $t2, $t0, $t1     # $t2 = lower+upper
    srl     $t2, $t2, 1       # $t2 = (lower+upper)/2
    sll     $v0, $t2, 2       # $v0 = middle*4
    addu    $v0, $a0, $v0     # $v0 = address array[middle]
    lw      $t3, 0($v0)       # $t3 = value array[middle]
    bne     $a2, $t3, else1   # (item == array[middle])?
    jr      $ra               # return
else1:
    bgt     $a2, $t3, else2   # (item < array[middle])?
    addiu   $t1, $t2, -1      # upper = middle-1
    j       loop
else2:
    addiu   $t0, $t2, 1       # lower = middle+1
    j       loop
ret:
    andi    $v0, $v0, 0       # $v0 = 0
jr     $ra  # return
```