

NAME _____
First Last

Student ID# _____

STONY BROOK UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
MIDTERM #2 EXAMINATION
VERSION A

CSE 220
Fall Semester 2007
November 13, 2007

This is a closed-book exam. (80 minutes)
Use this form for your work and return it.

The exam has 9 problems.
It is crucial to show all work done on the provided paper.

#1 _____ (10 pts)	#6 _____ (5 pts)
#2 _____ (12 pts)	#7 _____ (10 pts)
#3 _____ (8 pts)	#8 _____ (20 pts)
#4 _____ (10 pts)	#9 _____ (20 pts)
#5 _____ (5 pts)	

TOTAL _____ (100 max)

EXTRA CREDIT _____ (15 max)

1 [1 pts each] Multiple Choice. Write in the correct answer.

D _____ Register \$1

- (a) always contains value 1
- (b) points to the top of the stack
- (c) is reserved for the O/S
- (d) is reserved for the assembler

B _____ MIPS contains 4 registers for parameters (\$a0-\$a3), additional parameters are

- (a) not allowed
- (b) stored on stack
- (c) stored in registers \$t0-\$t4
- (d) stored on disk

C _____ The I instruction format does not contain which of the following fields:

- (a) Target register
- (b) Source register
- (c) Function
- (d) Opcode
- (e) Immediate

D _____ The jump instruction j

- (a) can jump to 2^{32} addresses
- (b) uses the immediate (I) format
- (c) uses a signed 26-bit address
- (d) none of the above

A _____ The jump register instruction ('jr') uses which format:

- (a) Register (R)
- (b) Immediate (I)
- (c) Jump (J)
- (d) None of the above

- D** ___ Which instruction does not use register addressing
- (a) Bitwise Or `or`
 - (b) Add `add`
 - (c) set less than `slt`
 - (d) store word `sw`
 - (e) jump register `jr`
- A** ___ Which instruction stores the offset as bytes and not as words
- (a) load word `lw`
 - (b) branch `b`
 - (c) branch not equal `bne`
 - (d) load upper immediate `lui`
- E** ___ Which of the following is not a MIPS addressing mode
- (a) Immediate Addressing
 - (b) Pseudodirect addressing
 - (c) Base Addressing
 - (d) Register Addressing
 - (e) All of them are addressing modes
- A** ___ Consider linking three modules with text segments T1, T2, and T3. The assembled start address is 0. The linker prepares an executable program with start address of 0x00400000, where T1 begins. T2 and then T3 are placed. The sizes of T1, T2 and T3 are 0x1000, 0x3000 and 0x4000 bytes respectively. What is the start address of text segment T2?
- (a) 0x00401000
 - (b) 0x00402000
 - (c) 0x00404000
 - (d) 0x00405000
 - (e) 0x00408000
- C** ___ Which of these statements is NOT TRUE for Virtual memory
- (a) Allows execution of programs larger than physical memory
 - (b) All addresses generated by a program (during execution) are virtual addresses
 - (c) Instruction cycle requires the same amount of time as before (time without virtual memory).
 - (d) Every program is executed in its own virtual space
 - (e) All are True
-

2 [12 pts] Short Answer. Fill in the blanks.

- (a) [6 pts] There are **32** (how many) MIPS registers. (**64 was acceptable if you wrote including floating point**)

How many registers are not assignable? Name them. **1, \$zero**

How many registers are reserved (not for MIPS instructions)? Name them. **3, \$at, \$k0, \$k1**

- (b) [3 pts] Assemblers operate in two passes. Label which pass each of the following steps occur in.

Pass 1 ___ Convert pseudo instructions

Pass 1 ___ Process Macro functions

Pass 2 ___ Resolve forward references

Pass 1 ___ Resolve backward references

Pass 2 ___ Writes instructions to linker file

Pass 1 ___ Create the symbol table

- (c) [3 pts] An instruction for rotate arithmetic shift doesn't make sense. Why? Give an example.

For an arithmetic instruction, the sign of the value should be preserved. Rotating, doesn't preserve sign. Ex: 1100 0000 rotated right by say 3 would be 0001 1000. The value has become positive not negative like the original value.

3 [8 pts] Suppose two 8-bit numbers $A = 0110\ 1110$ and $B = 1100\ 1100$ are in 8-bit registers, $\$R1$ and $\$R2$ respectively.

(a) [2 pts] What is the result of MIPS instruction: `and $R3 $R1 $R2`?
 $\$R3 = 01001100$

(b) [2 pts] What is the result of instruction: `ori $R3 $R1 12`?
12 is = 0000 1100. $\$R3 = 01101110$

(c) [2 pts] What instruction is `sra` & how does it work? What instruction format does it use?
Shift right arithmetic. It shifts the bits to the right, while preserving the sign bit (called "sign extension") (ie, the left most bits fill with the sign bit). Format is Register (R)

(d) [2 pts] What are the contents of R3 (value A), after: `sra $R3 $R1 7`? **$\$R3 = 0000\ 0000$**

4 [10 pts] Translate the following code into MIPS assembly:

$$A[i] = B[i-2] + B[i] + B[i+1]$$

Assume that A and B are byte arrays. Both arrays are stored in memory. Register \$s1 contains the initial address for array A; register \$s2 contains the initial address for array B; and register \$s3 contains the value of i. (Hint: use lb & sb. Also, this is not a loop, only a statement).

```
add $t7, $s2, $s3    # Calculate address of B[i]
lb $t0, -2($t7)      # Load byte of B[i-2]
lb $t1, 0($t7)       # Load byte of B[i]
lb $t2, 1($t7)       # Load byte of B[i+1]
add $t0, $t0, $t1    # add B[i-2] + B[i]
add $t0, $t0, $t2    # add B[i+1] + ( B[i-2] + B[i])
add $t6, $s1, $s3    # Calculate address of A[i]
sb $t0, ($t6)        # A[i] = B[i+1] + ( B[i-2] + B[i])
```

5 [5 pts] Some machines have increment and decrement instructions. For example,

```
inc $t0, 1          # increments $t0 by 1.
dec $t0, 1          # decrements $t0 by 1.
```

- (a) [2 pts] MIPS does not have these. Suppose we decide to add ‘inc’ as a pseudo instruction. The assembler would change it to actual MIPS instruction(s). What would be actual MIPS instructions (one or more)? (Use the fewest number of instructions). **addi \$t0, \$t0, 1**
There is no subi instruction.

- (b) [3 pts] Suppose we want to include `dec` as a true MIPS instruction, not a pseudo instruction, by assigning an opcode for `dec`. What instruction format would be appropriate for `dec`? Why?
Immediate Instruction Format. Requires a source address and an immediate value of -1.

- 6 [5 pts] Suppose we want to add a new instruction to MIPS. This is a pseudo instruction called halve (use 'hlf' for short). Its syntax is, `hlf $t1, $t0`. Here value in \$t0 divided by 2 and stored in \$t1, ignoring any overflow. An easy way to translate `hlf` is to use the following sequence of MIPS instructions.

```
li    $at, 2           # $at reserved for assembler, now stores 2.
div   $t0, $at         # Divide $t0 by $at
mflo $t1              # Copy quotient to $t1
```

- (a) [2 pts] This is really not efficient. Explain why?
Division is a slow. Shifting is faster. Inefficient in code size (3 instructions vs. 1)

- (b) [3 pts] Give a more efficient way of translating pseudo instruction `eth`, which are much better than what is given above. Give actual MIPS instruction(s).

```
sra $t1, $t0, 1
```

7 [10 pts] The `.data` section is allocated into MIPS memory sequentially as it is defined. Assume the `.data` section starts at address `0x00010000`.

- (a) Consider a structure in C defined as `struct point {int x; int y;}` and a structure `struct rectangle {struct point p1; struct point p2;}` How many bytes of MIPS memory does the structure `rectangle` require? If a struct `rectangle r` was the first element to be allocated to MIPS memory, what would be the address of `r.p2.x` be (in HEX)?

16 bytes, 0x10000008

- (b) Consider the structure in C defined as `struct StringList {char* value; struct StringList* next;}`. A `StringList` is a list data structure where the values are a address of a null terminated string and a pointer/address to the next string in the list. Futhremore, define variable `head` as the pointer/address to the first element in the `StringList`. A sample `StringList` looks like this:

Decode the following memory layout into a `StringList` similar to the one above. The address of variable `head` is `0x00001000`. The strings are coded as ASCII characters. Use the attached ASCII table. What does the final element in the string list read?

”Organization”

8 [20 pts] Pick one of the following: (You may use pseudo instructions without penalty, but points will be taken off for inefficient code.)

- (a) Write a **function** to determine if a string is a palindrome. A string is a palindrome if its reverse is the same as the original string. **Ex:** '12ABCBA21' and '229922' are palindromes. Assume the function `palin` has 2 input parameters, the address of the first character of the string and the address of the last character. The function returns 1 if the string is a palindrome, 0 otherwise.
- (b) You suspect someone is spying on you. Therefore you decide to write a MIPS text encoder. The **function** takes the address of a string as input (at most 20 characters in length), then it rotates to the right each memory word of the string by three bits. The resulting encoded string is displayed to the user as signed integer values. The function returns the address of the first integer value. **Ex:** Given input string "Stony Brook is Cool!" it encodes it (on a little endian machine) as: 776487951 -468887939 677645933 -466776595

(a) **Palindrome**

```
# assume that $a0 holds address of first char of string  [-1 if argument not in $a0/$a1]
# assume that $a1 holds address of last char of string  [-1 if argument not in $a1/$a0]
# [-3 if arguments are not in any $a0-a4 register]

# function is called palin

palin:    # [-1 if called 'main']

    #If any $s0-$s7 registers are used, then they must be saved here
    # [-1 if used $s0-$s7, -2 if used & not saved]

    # There should be no saving of the $ra on the stack
    # (leaf function, does not call another function) [-1 if saved]

    li $v0, 0                # Clear v0.
                                # Returns 0, if string is not a palindrome.

loop:    ble $a1, $a0, done    # Get out of loop if a1 <= a0
        lb $t0, ($a0)        # Load characters for comparison.  [-2 if lw]
        lb $t1, ($a1)        # Load characters for comparison.  [-2 if lw]

        bne $t0, $t1, ret
        addi $a0, 1          # Increment front pointer. [-2 if increment by 4]
        addi $a1, -1         # decrement back pointer. [-2 if increment by 4]
        j loop

# String is a palindrome, return 1. Return value should be in $v0
#[-1 if return value in $v1, -2 if not in any non-$v register]
done:    li $v0, 1

    #If any $s0-$s7 registers were used, then they must be restored here
    # [-1 if saved at top, and not restored]
```

```
#Restore $ra [-1 if saved at top, and not restored]
```

```
ret:   jr $ra      #Jump back to calling function      [-1 if used jal or syscall 10]
```

(b) Spy

```
# assume that $a0 holds address of string
# [-1 if argument not in $a0, -2 if not in an $a0-a4 register]
# function is called spy
```

```
spy:   # [-1 if called main]
```

```
# If any $s0-$s7 registers are used, then they must be saved here
# [-1 if used $s0-$s7, -2 if used & not saved]
```

```
# There should be no saving of the $ra on the stack
# (leaf function, does not call another function) [-1 if saved]
```

```
la $t1, $a0 # a0 points to the string, move it because will have to use $a0 for syscall
           # [-1 if not copying $a0 to a register]
la $t3, $a0 # a0 points to the string, move it because will have to return the value
li $t4, 4   #load $s1 with maximum number of words in string
li $t5, 0   #load $s0 with # of words for string (4)
```

```
nextWord:
```

```
lw $t0,($t1) # get word from string
ror $t0, $t0, 7 # Version A: 3, Version B: 4, Version C: 7  [-2 if not using ror]
```

```
move $a0,$t0 # system call to print      #
li $v0,1     # out integer                [-1 if use any other type of print than integer]
syscall
```

```
la $a0,endl # system call to print      # could be printing a space instead
li $v0,4 # out a newline or space
syscall
```

```
addi $t5, $t5, 1   # Inc count for the the number of words
addi $t1, $t1, 4   # move to the next word, could use
bne $t5, $t4, nextWord # branch if you havn't reached the end of the string (max 20 chars, 4 words)
```

```
move $v0, $t3      # or add $v0, $t3, $zero
```

```
#If any $s0-$s7 registers were used, then they must be restored here
#[ -1 if saved at top, and not restored]
```

```
#Restore $ra [-1 if saved at top, and not restored]
```

```
jr $ra      #Jump back to calling function      [-1 if used jal or syscall 10]
```

```
.data      #Must declare if used.
endl: .asciiz "\n" # could be a space instead
```

9

[20 pts] In mathematics, the greatest common divisor (GCD) of two non-zero integers, is the largest positive integer that divides both numbers without remainder. One way to calculate the GCD is to use Dijkstra's Algorithm.

The idea is: If $m > n$, $\text{GCD}(m,n)$ is the same as $\text{GCD}(m-n,n)$. Why? If $\frac{m}{d}$ and $\frac{n}{d}$ both leave no remainder, then $\frac{(m-n)}{d}$ leaves no remainder. Write MIPS assembly code for the following C/Java program fragment, which computes the GCD. You must follow all MIPS function call standards. You may use pseudo instructions without penalty, but points will be taken off for inefficient code. (Hint: Don't forget to use the stack.)

```
int gcd(int m, int n)
{
    int result = 0;
    if(m == n)
        result = m;
    else if (m > n)
        result = gcd(m-n, n);
    else
        result = gcd(m, n-m);
    return result;
}

# assume that $a0 holds value m [-1 if argument not in $a0/$a1]
# assume that $a1 holds value n [-1 if argument not in $a1/$a0]
# [-3 if arguments are not in any $a0-a4 register]
GCD: # [-1 if called 'main']
sub $sp,$sp,4 #Save registers on stack [-2 if use add, -1 if use words not bytes for offset]
sw $ra,0($sp) #Save $ra on the stack (calls other functions) [-1 if NOT saved]

#If any $s0-$s7 registers are used, then they must be saved here [-1 if used $s0-$s7, -2 if used & not saved]

move $v0, $a0 # $v0 = m, or add $v0, $zer0, $a0
beq $a1, $a0, ret # check if m == n if so return

ble $a0, $a1, else # branch to else if m <=n, else continue to if for m > n
sub $a0, $a0, $a1 # change $a0 argument to $m-n [-1 if move $a1 also]
jal GCD # call GCD again [-1 if don't use jal instruction, ok if done manually]
j ret # branch instruction is ok

else:
sub $a1, $a1, $a0 # change $a0 argument to $m-n [-1 if move $a1 also]
jal GCD # call GCD again [-1 if don't use jal instruction, ok if done manually]
j ret # branch instruction is ok

ret:
#If any $s0-$s7 registers were used, then they must be restored here [-1 if saved at top, and not restored]

lw $ra,0($sp) #Restore $ra [-1 if not restored, -2 if restored and NOT saved at top]
add $sp,$sp,4

jr $ra # Jump back to calling function [-1 if used jal or syscall 10]
```

EC EXTRA CREDIT [10 pts]

The following `bitcount` function counts the number of 1-bits in its integer argument:

```
int bitcount(unsigned x)
{
    int b;
    for (b = 0; x != 0; x >>= 1) {
        b += x & 01;
    }
    return b;
}
```

Write the MIPS program that computes this function.

```
#-----
# bitcount returns greatest common denominator
# Only give all points if solution is complete
#-----
# assume that $a0 holds value m [-1 if argument not in $a0/$a1]
# function is called bitcount

bitcount:  # [-1 if called 'main']
    #If any $s0-$s7 registers are used, then they must be saved here \
    #[-1 if used $s0-$s7, -2 if used & not saved]

    #There should be no saving of the $ra on the stack
    #(leaf function, does not call another function) [-1 if saved]

    li $t0, 0                # b = $t0, initialize to 0

loop: beqz $a0, ret          # branch if $a0 (x) is 0
    andi $t1, $a0, 1        # or 0x00000001
    add $t0, $t0, $t1
    srl $a0, $a0, 1
    j loop

ret:
    move $v0, $t0           # or add $v0, $t0, $zero

    #If any $s0-$s7 registers were used, then they must be restored here
    #[-1 if saved at top, and not restored]

    #Restore $ra [-1 if saved at top, and not restored]

    jr $ra                 #Jump back to calling function [-1 if used jal or syscall 10]
```

EC EXTRA CREDIT [5 pts]

Boolean Algebra. Using the postulates of Boolean algebra prove the following formula:

$$wxy + w'x(yz + yz') + x'(zw + zy') + z(x'w' + y'x) = xy + z$$

$$\begin{aligned}
 \text{lhs} &= wxy + w'xy(z + z') + x'zw + x'zy' + zx'w' + zy'x \\
 &= wxy + w'xy + x'z(w+w') + x'zy' + zy'x \\
 &= (w+w')xy + x'z + zy'(x'+x) \\
 &= xy + x'z + zy' \\
 &= xy + z(x' + y') \\
 &= xy + z(xy)' \\
 &= xy + z && \text{because } x + x'y = x + y \\
 &= \text{rhs}
 \end{aligned}$$