Name: **ANSWERS**

1. ____/17 2. ____/8 3. ____/8 4. ____/20 5. ____/16 6. ____/17 7. ____/10 8. ____/14

**CSI 333 Midterm Examination**

Professors Chaiken and Rosenkrantz

Fall 1999

October 1999

This is a closed book exam with one sheet of notes permitted. There are 8 parts and 11 printed pages. After that, find 14 pages of reference material which includes **Hex** and **powers of 2** tables.

**PART 1 (17 points)**

(a. 4 points) Show how the sum of the two given numbers represented in 8-bit binary is calculated. Write the answer bits below and all the carry bits above.

\[
\begin{aligned}
\text{carries} & \quad 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
X & = & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
Y & = & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
\text{sum} & = & (1, \text{opt}) & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1
\end{aligned}
\]

(b. 4 points) Express **X** and **Y** above in (2 digit) hexadecimal: **X** [**6E**] **Y** [**AD**]

(c. 4 points) Write the **unsigned binary** interpretations of bit strings **X**, **Y** and their sum in decimal:

\[
X \begin{array}{c}
110
\end{array} + Y \begin{array}{c}
173
\end{array} = \text{sum} \begin{array}{c}
27
\end{array}
\]

(d. 2 points) Write the **2-s complement signed binary** interpretations of bit strings **X**, **Y** and their sum in decimal:

\[
X \begin{array}{c}
110
\end{array} + Y \begin{array}{c}
-83
\end{array} = \text{sum} \begin{array}{c}
27
\end{array}
\]

(e. 3 points) Create ONE example of two 8-bit integers (by writing your two integers and their sum in binary) whose sum DOES OVERFLOW when the hardware’s calculation of sum is interpreted as 2-s complement SIGNED 8 bit binary but the same calculation does NOT OVERFLOW when the interpretation is UNSIGNED.

Any correct addition example for which the top bits are as follows:

\[
\begin{array}{c}
\text{1st integer} \\
\text{2nd integer} \\
\text{sum}
\end{array}
\]

1
PART 2 (8 points) What does the MAL program below print, and how many lines does it print? Write the exact messages in the order they are printed. Hint: There’s a bug.

start: Hello.
One: Hello.
Two: Hello.
Two: Goodbye.
One: Goodbye. (infinite loop reprints this message indefinitely)

.globl __start
__start:
la $a0, startEnter
li $v0, 4
syscall
jal One
la $a0, startReturn
li $v0, 4
syscall
li $v0, 10
syscall
jr $ra

One:
la $a0, OneEnter
li $v0, 4
syscall
jal Two
la $a0, OneReturn
li $v0, 4
syscall
jr $ra

Two:
la $a0, TwoEnter
li $v0, 4
syscall
la $a0, TwoReturn
li $v0, 4
syscall
jr $ra

.data
startEnter: .asciiz "start: Hello.\n"
startReturn: .asciiz "start: Goodbye.\n"
OneEnter: .asciiz "One: Hello.\n"
OneReturn: .asciiz "One: Goodbye.\n"
TwoEnter: .asciiz "Two: Hello.\n"
TwoReturn: .asciiz "Two: Goodbye.\n"
PART 3 (8 points) A MAL program begins:

```assembly
.text
.globl __start
__start:ori $s0, $zero, 0
    addiu $s1, $s0, 9
e<
```

An xspim screen image shows the .text segment contents:

```
[0x00400000] 0x34100000 ori $16, $0, 0 ; 3: ori $s0, $zero, 0
[0x00400004] 0x26110009 addiu $17, $16, 9 ; 4: addiu $s1, $s0, 9
[0x00400008] 0x2632000a addiu $18, $17, 10 ; 5: addiu $s2, $s1, 10
[0x0040000c] 0xc3100040 lui $19, 64 [__start] ; 6: la $s3, __start
[0x00400010] 0x233a020 add $20, $17, $19 ; 7: add $s4, $s1, $s3
[0x00400014] 0x8e750000 lw $21, 0($19) ; 8: lw $s5, 0($s3)
[0x00400018] 0x26b60007 addiu $22, $21, 7 ; 9: addiu $s6, $s5, 7
```

Here are some contents of the register display window before running:

```
R16 (s0) = 00000000
R17 (s1) = 00000000
R18 (s2) = 00000000
R19 (s3) = 00000000
R20 (s4) = 00000000
R21 (s5) = 00000000
R22 (s6) = 00000000
R23 (s7) = 00000000
```

Write what’s in the register display just after the last addiu executes (remember it’s in Hex):

(BE VERY CAREFUL WITH INSTRUCTIONS LIKE 1A SINCE DEREFERENCING THE WRONG ADDRESS CAUSES SEVERAL ERRORS LATER.)

```
R16 (s0) = 00000000
R17 (s1) = 00000009
R18 (s2) = 00000013
R19 (s3) = 00400000
R20 (s4) = 00400009
R21 (s5) = 34100000
R22 (s6) = 34100007
R23 (s7) = 00000000
```
PART 4 (20 points) The following assembly language fragment

.data
CARDS: .ascii "MDSWIAHUKQNFJRYPJBEQZTGL" #26 ASCII Letters
PLRONE: .ascii " " #13 ASCII Space chars
PLRTWO: .ascii " " #13 ASCII Space chars

assembles into the 26 byte array beginning at the address symbolized by CARDS the ASCII codes for a shuffled sequence of letters. At addresses symbolized by PLRONE and PLRTWO are 13 byte arrays.

Your job is to write a MAL program (.text only) that will copy, in increasing address sequence, the 26 character codes in the CARDS array one-by-one alternately into arrays beginning at PLRONE and PLRTWO. For example, after your program runs, the first few characters at PLRONE should be MSIHK, and at PLRTWO there should be DWAUX. More precisely, implement the pseudo-code:

for (i = 0, i < 13; i++)
{   PLRONE[ i ] = CARDS[ 2*i ];   PLRTWO[ i ] = CARDS[ 2*i + 1 ]; }

- For any credit, it must be done with a loop, almost every MAL instruction must have a comment that explains its purpose clearly and correctly, and comments must explain the usage of each relevant register.

- For full credit, the purpose of every instruction must be explained by a correct invariant: A comment about relationships among data that is ALWAYS TRUE when control flow reaches that comment. See the next sheet for 2 examples of invariant comments, also pages 9-10.

- Use the space below and/or the next sheet.

   .text
   .globl __start
__start:
(continuing)

```assembly
.text
.globl __start
__start: la $s0, CARDS #$s0 = source addr for next card
    li $s1, 13 #$s1 = number of deals remaining, 13-i
    la $s2, PLRONE #$s2 = dest for next PLRONE card
    la $s3, PLRTWO #$s3 = dest for next PLRTWO card
LOOP: beqz $s1, DONE #$s1 != 0, more to do
    lbu $t0, 0($s0) #$t0 = CARDS[2*i]
    sb $t0, 0($s2) #PLRONE[i] = CARDS[2*i]
    addi $s2, $s2, 1 #$s2 = addr PLRONE[i+1]
    lbu $t0, 1($s0) #$t0 = CARDS[2*i + 1]
    sb $t0, 0($s3) #PLRTWO[i] = CARDS[2*i + 1]
    addi $s3, $s3, 1 #$s3 = addr PLRTWO[i+1]
    addi $s0, $s0, 2 #$s0 = addr CARDS[2*i + 2]
    addi $s1, $s1, -1 #increment i, now $s1 = 13 - i
    j LOOP #not reached
DONE:
```

```
li $v0, 10  # $v0 = the (x)sim exit syscall code
syscall    # FALSE ... Control never reaches here
```
PART 5 (16 points) **Show the calculations that lead to each result.**
(a. 4 points) Express $29.875_{\text{TEN}}$ as a binary radix “real” numeral.

$$29 = 16 + 8 + 4 + 1 = 11101_{\text{TWO}}, \quad 0.875 = \frac{7}{8} = 0.111_{\text{TWO}}, \quad \text{so} \quad 29.875_{\text{TEN}} = 11101.111_{\text{TWO}}$$

(b. 4 points) Express $-19.5_{\text{TEN}}$ as a normalized binary radix “real” numeral. (Normalized means mantissa $m$ satisfies $1.0 \leq |m| < 2.0$)

$$19 = 16 + 2 + 1 = 10011_{\text{TWO}}, \quad 0.5 = \frac{1}{2} = 0.1_{\text{TWO}}, \quad \text{so} \quad -19.5 = 10011.1_{\text{TWO}} = -1.00111 \times 2^4$$

(c. 4 points) Express $-13.75_{\text{TEN}}$ as an IEEE 754 single precision floating point numeral with the bits written out. Recall single precision IEEE floating point numerals have a 1-8-23 bit format whose fields are sign ($S$), biased exponent ($E$) and significand ($V$). The formula for the value is

$$(-1)^S \times (1.0 + V/2^{23}) \times 2^{(E-127)}$$

(so $E$ is obtained by adding 127 to the mathematically true exponent.)

$$13.75 = 1101.11_{\text{TWO}} = 1.10111 \times 2^3.$$  

$$127 + 3 = 130 = 128 + 2 = 10000010_{\text{TWO}}.$$  

Answer: $1\text{-}10000010\text{-}101110000000000000000000$

Write answer here: 

(d. 4 points) Explain briefly but precisely why every fraction with a terminating binary radix expression will have a terminating decimal expression. Hint:

$$\text{value} = \sum_{i=1}^{L} b_i (1/2)^i = \frac{1}{2} b_1 + \frac{1}{4} b_2 + \frac{1}{8} b_3 + \cdots + \frac{1}{2^L} b_L$$

If a fraction has a terminating binary radix expression, then this formula applies and so it is a finite sum of powers of $\frac{1}{2}$. Each power of $\frac{1}{2}$ has a terminating decimal expression. (All remainders when dividing by 2 are either 0 or 1, and $\frac{1}{2} = 0.5_{\text{TEN}}$.) Finite sums of fractions with terminating radix expressions will have terminating radix expressions (because the number of non-zero digits right of the radix point is at most the maximum of this number for each of the fractions in the sum).
PART 6 (7 points)

Draw a labelled box and pointer diagram like what you did in Lab Exercise 2 to picture the data structures created by the MAL directives below. (Use the Lab notation of the pointer arrows and characters written into array boxes. To show an integer value in a storage cell, write the value in decimal inside the box. Each box will picture a memory cell; it must be labelled with the label that symbolizes the address of that memory cell. Do NOT write address values because they are unpredictable from the information supplied.)

```assembly
.data
PCOW: .word STR
NUM: .word 6
PNUM: .word NUM
STR: .asciiz "Now see a cow"
AA: .word BB
BB: .word CC
CC: .word AA
```

![Diagram of box and pointer structure]

```
PCOW:               STR: Now see a cow\0
NUM:  6
PNUM:               
AA: BB
BB: CC
CC: AA
```
PART 7 (10) MIPS immediate and memory access type instructions have an “opcode” in their
high order 6 bits which identifies the exact kind of instruction. We said such instructions have
6-5-5-16 format. MIPS instructions that operate on 3 registers have 0 in the 6 bit opcode field; the
particular kind of instruction is identified by the value in the low order 6 bits. Those instructions
have 6-5-5-5-5-6 format. The rightmost, low order 6 bit field is called the “funccode,” for function
code.

Write a MAL fragment with specification:

1. It will be run with a MIPS instruction in register $a0.

2. It can mess up any of the registers: Don’t bother saving/restoring anything. You will have to
use some other registers besides $a0

3. Extract the opcode into register $a0. If it is non-zero, print it as an integer and go to the
label REGIMM

4. If the opcode is zero (so it’s a 3 register instruction), extract the funccode into register $a0.
Print it as an integer and go to the label SPECIAL

For any credit, comments must be supplied according to the rules given for the previous MAL
coding problem.

    move    $t0,$a0    #$t0 = input MIPS instruction (to save it)
    srl     $a0,$a0,26  #$a0 = opcode (use shift to extract it)
    beq     $a0,0,doSpecial   #opcode != 0 (go to doSpecial if opcode=0)
    li      $v0,1    #$v0 = code for print int syscall
    syscall   #opcode in $a0 printed.
    j       REGIMM

doSpecial:
    andi    $a0,$t0,0x3F   #opcode=0, get the 6 bit funccode in $a0 now
    li      $v0,1    #$v0 = code for print int syscall
    syscall   #funccode in $a0 printed.
    j       SPECIAL
PART 8 (14 points) This MAL program does procedure calls and returns with the jal and jr $ra instructions, and saves/restore $ra and other registers on the stack using $sp in the standard MIPS way.

    Assume $sp=0x7FFFFFFC initially.

(a. 9 points) Show on page 11 the contents of the stack memory region and the contents of registers $ra, $sp, $s0 and $s1 after the program reaches the infinite loop at label sta1: Include data “left in the stack,” at the shown addresses which are < the value in $sp.

(b. 5 points) Assume the value assembled into memory at address COUNT is changed to 2 (the COUNT: .word 1 directive is replaced by COUNT: .word 2). When (x)spim is reinitialized, the edited .s file is loaded and run, it will go into an infinite loop at label BINF: Show on page 11 the same kind of information after it reaches this infinite loop.

Runtime information you will need (return addresses and address symbolized by COUNT) is given by comments in the code.

.data
COUNT: .word 1    # COUNT symbolizes the address of the counter variable.
    # It will be decremented by procedure B. Then, its new value will control
    # what procedure B does. The counter is initially 1

.text
.globl __start
__start:
    li $s0, 7    #s0 = 7
    jal A
#sta1 symbolizes 0x00400008
sta1:  j  sta1  #INFINITE LOOP: REPORT STATE FOR QUESTION (a.)
A:       addi $sp, $sp, -8    # beginning of A's prologue
        sw $s0, 0($sp)
        sw $ra, 4($sp)    # A's prologue done
        li $s0, 9    # Body of A: $s0 = 9
        li $s1, 3    # $s1 = 3 just for fun
        jal B       # If counter=1 return, loop infinitely if > 1.
#AEPI symbolizes 0x00400024

AEPI:    lw $s0, 0($sp)    # begin A's epilogue
        lw $ra, 4($sp)
        addi $sp, $sp, 8
        jr $ra    # return
##########################################################
#
# The text of procedure B is printed on the next page
### Procedure B

- (1) Decrement the global counter (subtract 1).
- (2) If the new counter value is 0, return.
- (3) Otherwise, call B recursively.
- (4) Loop infinitely if the new counter value (computed at step 2) is 1.
- (5) Otherwise, return.

```assembly
B:    addi $sp, $sp, -12  # beginning prolog, allocate frame
     sw $s0, 0($sp)
     sw $s1, 4($sp)
     sw $ra, 8($sp)  # end of prologue

1a $s0, COUNT  # $s0 = address of counter [IT WILL BE 0x10010000]
lw $s1, 0($s0)  # $s1 = counter value
     addi $s1, $s1, -1  # $s1 = counter value - 1
     sw $s1, 0($s0)  # $s1 = new counter value = old count value - 1
     beqz $s1, BRETURN  # return if new counter value = 0

# counter not zero, call B (recursively)
    jal B
#BINF symbolizes 0x0040005C

# If the counter equals 1, then loop infinitely. SHOW THE ENTIRE STACK SEGMENT
    # CONTENTS IN YOUR ANSWER FOR PART (b.)

BINF:    beq $s1, 1, BINF

# The counter is neither 0 nor 1
# procedure B will return, after its epilogue

BRETURN: lw $s0, 0($sp)  # begin epilogue
    lw $s1, 4($sp)
    lw $ra, 8($sp)
    addi $sp, $sp, 12
    jr $ra  # return
### a. `COUNT:.word 1`

**Write final values:**

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7FFEFD8</td>
<td>00000000</td>
</tr>
<tr>
<td>0x7FFEFCDC</td>
<td>00000003</td>
</tr>
<tr>
<td>0x7FFEFE00</td>
<td>00400024</td>
</tr>
<tr>
<td>0x7FFEFE44</td>
<td>00000007</td>
</tr>
<tr>
<td>0x7FFEFE88</td>
<td>00400008</td>
</tr>
<tr>
<td>0x7FFEFFC</td>
<td>00000001</td>
</tr>
</tbody>
</table>

---

### b. `COUNT:.word 2`

**Write final values:**

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7FFEFD8</td>
<td>10010000</td>
</tr>
<tr>
<td>0x7FFEFCDC</td>
<td>00000001</td>
</tr>
<tr>
<td>0x7FFEFE00</td>
<td>0040005C</td>
</tr>
<tr>
<td>0x7FFEFE88</td>
<td>00000009</td>
</tr>
<tr>
<td>0x7FFEFC</td>
<td>00000003</td>
</tr>
<tr>
<td>0x7FFEFF0</td>
<td>00400024</td>
</tr>
<tr>
<td>0x7FFEFF4</td>
<td>00000007</td>
</tr>
<tr>
<td>0x7FFEFF8</td>
<td>00400008</td>
</tr>
<tr>
<td>0x7FFEFFC</td>
<td>00000001</td>
</tr>
</tbody>
</table>