

ECE 199 Exam 2 Fall 2003

Tuesday, November 4th, 2003

Name:

- **Be sure your exam booklet has 12 pages.**
- **Write your name at the top of each page.**
- **This is a closed book exam.**
- **You are allowed one 8.5 x 11 sheet of notes.**
- **Absolutely no interaction between students is allowed.**
- **Show all of your work.**
- **Challenge questions are marked with a *****
- **Don't panic, and good luck!**

Problem 1	20 points	_____
Problem 2	20 points	_____
Problem 3	20 points	_____
Problem 4	20 points	_____
Problem 5	20 points	_____
Total	100 points	_____

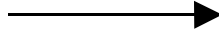
Name: _____

Problem 1 (20 points): Short Answer

Part A (4 points): What single instruction is equivalent to the following two LC-3 instructions?

```
LEA R7, #1
```

```
JMP R2
```



Part B (6 points): The following LC-3 program is supposed to ask the user for a character input and output that character in quotes. However, it has a bug. **BRIEFLY** describe what is wrong and then fix the bug by making a single change.

```
.ORIG x3000
LEA R0, PROMPT
PUTS
GETC
OUT
STI R0, DATA
LEA R0, OUTPUT
PUTS
HALT

PROMPT .STRINGZ "Enter a character: "
OUTPUT .FILL x0A ; The ASCII Line Feed character
        .FILL x22 ; The ASCII quote character (")
DATA   .FILL x22 ; Overwrite with key press
        .FILL x22
        .FILL x0
        .END
```

Name: _____

Problem 1, continued:

Parts C and D deal with the process of assembly. Using the systematic approach to assembly described in class, a two-pass LC-3 assembler detects a single error with each of the following programs. In each case, briefly state the following: (1) the cause of the error, (2) the pass in which it is detected, and (3) how the assembler detects the error.

For example, a multiply-defined label is detected in the first pass when the assembler attempts to insert the second copy of the label into the symbol table.

Part C (5 points):

```
.ORIG    x0500
MY_PUTS  LDR    R1,R0,#0      ; a PUTS with caller-saved regs
          BRnp   LOOP        ; end at NUL
          RET
LOOP     LDI    R2,DSR        ; wait for display
          BRzp   LOOP
          STI    R1,DDR        ; write a character
          ADD    R0,R0,#1      ; continue with next character
          BRnzp  PUTS_LOOP
DSR      .FILL  xFE04         ; display status register address
DDR      .FILL  xFE06         ; display data register address
.END
```

Part D (5 points):

```
.ORIG    x3000
INIT_TAB AND    R0,R0,#0      ; initialize a table
          LEA    R1,TABLE      ; point R1 to start of table
          LD     R2,TAB_SIZ    ; set R2 to the size of the table
I_LOOP   STR    R0,R1,#0      ; write a 0
          ADD    R1,R1,#1      ; point to next table entry
          ADD    R2,R2,#-1     ; continue until all entries filled
          BRp    I_LOOP
          RET                    ; all done
TABLE    .BLKW  #800          ; the table
TAB_SIZ  .FILL  #800          ; the size of the table
.END
```

Name: _____

Problem 2 (20 Points): I/O and Stacks in LC-3

Part A (10 points): Many devices require that a processor attend to their needs within a certain amount of time. A typical keyboard, for example, does not have much storage of its own and cannot keep track of sequences of typed characters, only single keystrokes. If polling I/O is used, and the processor (for whatever reason) does not check for keyboard input during a period in which two or more keys are pressed, some of the keystrokes are lost.

In this problem, you will address this issue by changing the GETC trap handler to make the monitor beep by sending an ASCII BEL character to the display whenever keystrokes are lost. To support your extension, assume that the KBSR contains an OVERFLOW bit (bit 0, with value 1) to indicate the loss of keystrokes to you. The OVERFLOW bit is set whenever the processor fails to read KBDR between two keystrokes, and is cleared after KBSR is read.

Things you may find useful: the display status register (DSR) is mapped at address xFE04; the display data register (DDR) is mapped at address xFE06; the ASCII code for BEL (bell, or beep) is 7. Keep in mind that registers other than R0 are callee-saved (*i.e.*, you must save them).

```
        .ORIG    x0600
GETC    LDI      R0,KBSR      ; wait for a keystroke; note that
        BRzp    GETC        ; overflow can't occur without one
; a. your new code goes here (write it below)
GET_KEY LDI      R0,KBDR     ; read the last keystroke
        RET      ; and return it
KBSR    .FILL   xFE00       ; keyboard status register address
KBDR    .FILL   xFE02       ; keyboard data register address
; b. also indicate any new data/storage that you need (which go here)
        .END
```

a.

b.

Name: _____

Problem 2, continued:

Part B (10 points): Use stacks to help you write a short LC-3 program to print out a list of strings in reverse. The strings are stored in memory exactly as in MP2: beginning at x5005, NUL terminated, and with an additional terminating NUL indicating the end of the list. There may be zero or more strings. You must print all strings in the reverse order that they were found; there is no error checking on the number of strings. Follow each string with a linefeed character (0x0A). You **must** make use of the three provided subroutines, whose semantics are described below. Assume they are already written for you and use the callee save convention. Also assume the stack is initially empty.

FIND_NUL – Locates the first NUL in memory, beginning from the passed in address.

Inputs: R0 – The address to begin looking for the next NUL character.

Outputs: R0 – Unchanged. R1 – The address of the character past the first NUL found.

R5 – Set to 1 if R0 pointed to a NUL character, 0 else.

POP – Pops the top of the stack and returns it.

Inputs: None

Outputs: R0 – The data from the top of the stack.

R5 – 1 if underflow occurred, 0 else.

PUSH – Pushes data onto the top of the stack.

Inputs: R0 – The data to be pushed.

Outputs: None (**You may assume overflow will not occur.**)

.ORIG x3000

Name: _____

Problem 3 (20 Points): Functions and Activation Records

Given below is part of the LC-3 translation of a C function Foo and part of the function Foo itself. Also given is the stack frame (activation record) for Foo. Remember that memory increases in the direction of the arrow. Answer the questions on the following page.

```

unsigned int Foo(unsigned int a)
{
    unsigned int i;
    for ( i = a; i > 0; i--) {

        /* Body of loop written by
         * you in Part B
         */
    }

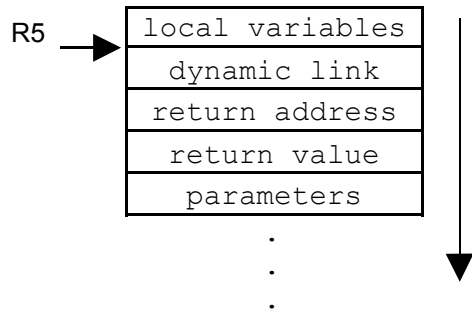
    return a;
}

```

```

.
.
.
LDR R0, R5, #4 ; 1
STR R0, R5, #0 ; 2
OUTER_LOOP
LDR R0, R5, #0 ; 3
BRz DONE ; 4
INNER_LOOP
LDR R0, R5, #4 ; 5
LDR R1, R5, #4 ; 6
AND R2, R2, #0 ; 7
LOOP
ADD R2, R2, R0 ; 8
ADD R1, R1, #-1 ; 9
BRnp LOOP ; 10
STR R2, R5, #4 ; 11
LDR R0, R5, #0 ; 12
ADD R0, R0, #-1 ; 13
STR R0, R5, #0 ; 14
BRnzp OUTER_LOOP ; 15
DONE
LDR R0, R5, #4 ; 16
STR R0, R5, #3 ; 17
.
.
.

```



Name: _____

Problem 3, continued:

Part A (12 points): Which LC-3 instructions correspond to (give the instruction numbers shown in the comments):

- a. The initialization of the `for` loop?

- b. The test (middle part) of the `for` loop?

- c. The update (re-initialization) of the `for` loop?

Part B (6 points): Using the LC-3 translation of `Foo`, write the body of the `for` loop here.

*****Part C** (2 points): What is the return value of `a` relative to the value of `a` passed into `Foo`?

Name: _____

Problem 4 (20 Points): C Control Flow

Part A (8 points): Fill in the blank below so that the program executes the code in the `if` statement as specified.

```
if ( _____ ) {  
    /* Execute this block of code if the  
    * integer variable x is a multiple  
    * of 5 whose absolute value is  
    * greater than 100  
    */  
  
    /* Code */  
}
```

Part B (12 points): Rewrite the below function to use exactly one `return` statement. You do not have to rewrite the comments.

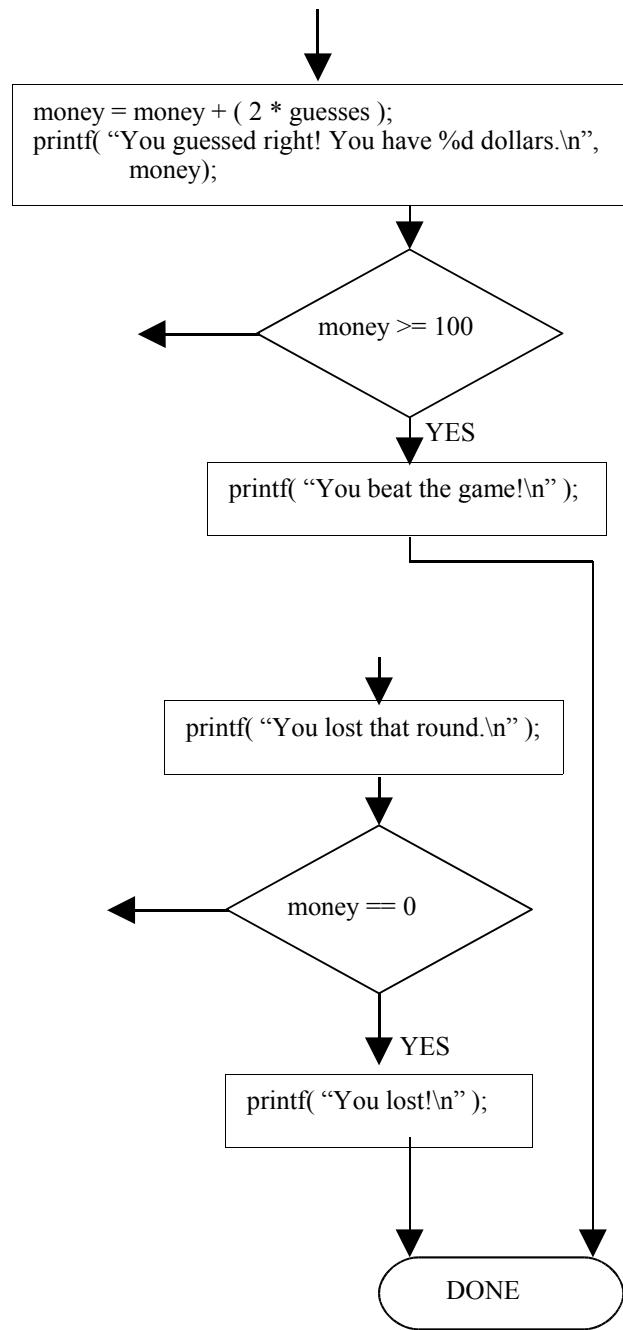
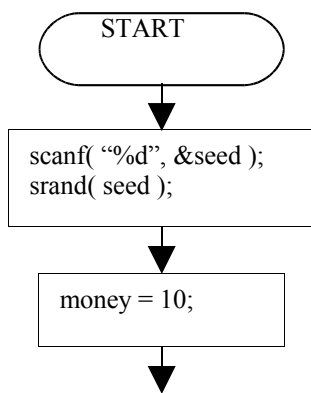
```
int PosDiv( int a, int b ) {  
    /* 'a' and 'b' are some arbitrary user  
    supplied integers. */  
    if( b == 0 ) {  
        return -1; /* Error 1: Don't divide by zero */  
    }  
    if( ( a < 0 ) || ( b < 0 ) ) {  
        return -2; /* Error 2: Positive numbers only */  
    }  
    return a / b;  
}
```

Name: _____

Problem 5 (20 Points): Systematic Decomposition

Part A (12 points): Complete the diagram below according to the following program description. A few entries have already been provided, and must be used.

The program is a number guessing game. The user initially starts with 10 dollars. To begin each round, one dollar is taken from the user. During each round, the user tries to guess a new random number between 0 and 99. The user is given a maximum of 5 guesses per round to guess the secret number. If the user guesses right, the user wins 2 times the number of guesses he/she had left when entering the current round (for example, the user would win 10 dollars for guessing correctly on the first guess). This process repeats with a new random number each round, until the user ends a round with 0 dollars, or has 100 or more dollars. Be sure to prompt the user for a guess, and tell the user if they guessed wrong.



Name: _____

Part B (8 points): Now assume the program is written with your work in **Part A** being placed into a function called `PlayRound`. This function will play only a single round of the game. Give the declaration of `PlayRound` and show how it would be called. Explain your reasoning for this declaration. Note: Assume no variables are global.

Name: _____

Scratch Paper for Calculations

Name: _____

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD*	0001			DR			SR1			0	00		SR2			
ADD*	0001			DR			SR1			1	imm5					
AND*	0101			DR			SR1			0	00		SR2			
AND*	0101			DR			SR1			1	imm5					
BR	0000			n	z	p	PCoffset9									
JMP	1100			000			BaseR			000000						
JSR	0100			1	PCoffset11											
JSRR	0100			0	00		BaseR			000000						
LD*	0010			DR			PCoffset9									
LDI*	1010			DR			PCoffset9									
LDR*	0110			DR			BaseR			offset6						
LEA*	1110			DR			PCoffset9									
NOT*	1001			DR			SR			111111						
RET	1100			000			111			000000						
RTI	1000			000000000000												
ST	0011			SR			PCoffset9									
STI	1011			SR			PCoffset9									
STR	0111			SR			BaseR			offset6						
TRAP	1111			0000			trapvect8									
reserved	1101															