

ECE 190 Exam II Fall 2005

Tuesday, November 1st, 2005

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 may not use a calculator.**
- **You are allowed TWO handwritten 8.5 x 11” sheets of notes.**
- **Absolutely no interaction between students is allowed.**
- **Show all of your work.**
- **Be sure to clearly indicate any assumptions that you make.**
- **More challenging questions are marked with a ***.**
- **Don’t panic, and good luck!**

“No one can know how glad I am to find
On any sheet the least display of mind.”
– Robert Frost

Problem 1	20 points	_____
Problem 2	20 points	_____
Problem 3	20 points	_____
Problem 4	20 points	_____
Problem 5	20 points	_____
<hr/>		
Total	100 points	

Problem 1 (20 points): Short Answer

Part A (5 points): Suppose an I/O event (e.g., a keystroke) occurs infrequently, and at irregularly distributed times. Would a polled or interrupt-driven approach to processing the event be a better design choice? Give two reasons why the approach that you chose is the better one.

Part B (5 points): A certain C function `bar ()` accepts a variable number of non-negative parameters/arguments of type `int`. Explain two ways to design `bar ()` so that the callee can determine the actual number of parameters passed.

Problem 1, continued

Part C (5 points): What is wrong with the following function?

```
int* sum_of_int (int x, int y)
{
    int sum = x + y;
    return &sum;
}
```

Part D (5 points): Complete the output from the program fragments below.

```
int x = 0;
int i = 4;
for ( i = 0; 10 > i; i++ ) {
    i++;
    x++;
}
printf ("x: %d\ni: %d\n", x, i);
```

Output:

x: _____

i: _____

```
int x = 0;
int i = 4;
while ( 10 > i )
{
    if ( 3 < x ) { break; }
    x++;
    i++;
}
printf ("x: %d\ni: %d\n", x, i);
```

Output:

x: _____

i: _____

Problem 2 (20 points): The Assembly Process

```

1  .ORIG x2000
2      LD R1, COUNT
3      NOT R1, R1
4      ADD R1, R1, #1
5      AND R2, R2, #0
6      LEA R3, DATA
7  LOOP
8      STR R2, R3, #0
9      ADD R3, R3, #1
10     ADD R2, R2, #1
11     ADD R4, R2, R1
12     BRn DO_LOOP
13     LEA R0, DONE_STRING
14     JSR PRINT_STRING
15     TRAP xF025
16 PRINT_STRING
17     ST R7, SAVE_R7
18     PUTS
19     RET
20 SAVE_R7
21     .BLKW #1
22 COUNT
23     .FILL x200
24 DATA
25     .BLKW x200
26 DONE_STRING
27     .STRINGZ "Finished!"
28 CHALLENGE
29     .END

```

Label	Address
LOOP	
PRINT_STRING	
SAVE_R7	
COUNT	
DATA	
DONE_STRING	
CHALLENGE	

All parts of this problem refer to the LC-3 assembly program shown above. The line numbers are **not** part of the program, but are used for reference in the problem.

Part A (8 points): Fill in the addresses for the symbol table above as they would be generated by the assembler.

Part B (2 points): Write the binary word that would be generated by the assembler for **line 2** (the first instruction) of the program.

Part C (10 points): Assuming that both passes of the assembler were to execute, indicate which line numbers result in errors reported by the assembler, specify in which pass each error occurs, and *briefly* explain why each is an error.

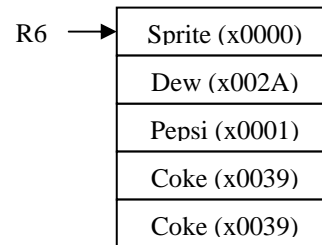
Problem 3 (20 points): Assembly Programming using Subroutines

You are given a vending machine filled with four different types of sodas. The vending machine behaves like a stack: when you buy a drink, it is “popped” off the stack and given to you. Unfortunately, you cannot open the machine, so the only way to remove the sodas is to “pop” them off, and the only way to put sodas back in the machine is to “push” them in. Your job is to inventory the sodas, restore the machine to its initial state, and then print out the total number of each soda.

You are provided with 2 stacks. One stack is a representation of the soda dispenser (*Stack 1*), and the other stack is empty (*Stack 2*). The stack pointer for *Stack 1* is R6, and the stack pointer for *Stack 2* is R5. *Stack 1* (the soda dispenser) has an arbitrary number of sodas in it. Each of the four types of drinks is represented on the stack by a unique value.

Example of Stack 1

Type of Soda	Encoded Value
Coke	x0039
Pepsi	x0001
Sprite	x0000
Mountain Dew	x002A



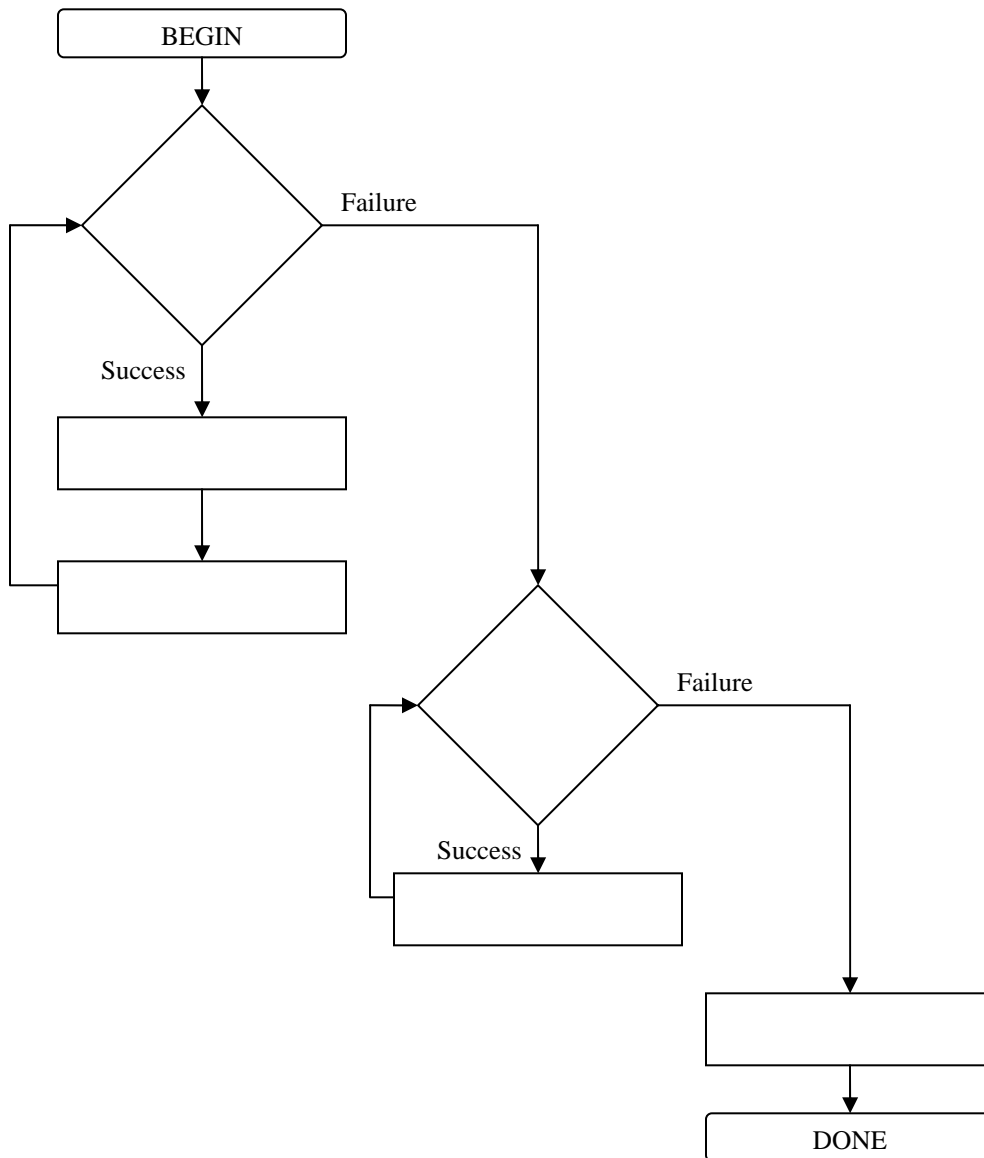
You are provided with the following subroutines:

PUSH1 and PUSH2	
<i>Description</i>	Pushes the value in R0 onto <i>Stack 1</i> (PUSH1) or <i>Stack 2</i> (PUSH2)
<i>Input</i>	R0 (the value to be pushed)
<i>Returns</i>	Success: R0 contains 0 Failure: R0 contains -1 (the stack is full)
POP1 and POP2	
<i>Description</i>	Pops a value off the stack into R0 (POP1 for <i>Stack 1</i> ; POP2 for <i>Stack 2</i>)
<i>Input</i>	None
<i>Returns</i>	Success: R0 contains popped value Failure: R0 contains -1 (the stack is empty)
ADD_BIN	
<i>Description</i>	Adds the drink to a histogram stored at memory location x4000
<i>Input</i>	R0 contains the encoded value of the drink to add
<i>Returns</i>	Nothing
PRINT_HIST	
<i>Description</i>	Prints out the histogram created by ADD_BIN
<i>Input</i>	None
<i>Returns</i>	Nothing

Problem 3, continued

Part A (6 points): Again, your job is to inventory the sodas in *Stack 1*, restore the machine to its initial state, and then print out the total number of each soda. Complete the systematic decomposition below by writing the name of one of the subroutines provided to you in each box of the flowchart below.

You may assume that neither PUSH1 nor PUSH2 ever fails.



Problem 3, continued

Part B (8 points): Using the subroutines provided, write LC-3 assembly code for the task assigned to you: again, count the number of each type of soda on *Stack 1*, restore *Stack 1* to its initial state, and print out the total number of each type of soda. Note that

- you may assume that neither PUSH1 nor PUSH2 ever fails,
- you may not access the stacks using their stack pointers directly,
- you may assume that registers R0, R1, R2, R3, and R4 are callee saved, and that R5 and R6 are only affected as specified by the subroutine semantics,
- you may make **no other assumptions** about any subroutine's implementation,
- you do not need to include .ORIG, HALT, or .END, and
- if you write more than fifteen or twenty lines, you're doing it wrong.

Problem 3, continued

Part C (6 points): Write the subroutine PRINT_HIST, which prints out the data stored in the histogram (the total number of each type of drink) located at memory locations x4000 to x4003. You may use TRAP x26 to print out decimal numbers; note that the version given to you **also prints a line feed** after the number. *You may use R0 through R3 without saving them for this part; other register values must be preserved.*

Example output for PRINT_HIST:

0
4
3
1

TRAP x26	
<i>Description</i>	Prints R0 to the screen as a decimal number followed by a linefeed
<i>Input</i>	R0 contains decimal number to be printed
<i>Returns</i>	Nothing

Problem 4 (20 points): LC-3 and C

Part A (6 points): Explain why a compiler that does not make use of library subroutines requires more LC-3 instructions to implement the statement below on the left than the statement below on the right. The variable `i` is an `int`.

```
i = (i >> 1);
```

```
i = (i << 1);
```

Part B (6 points): Two global variables, `int* op1_ptr` and `int* op2_ptr`, reside at offsets 6 and 10, respectively, from the global data pointer (R4). Write a short sequence of LC-3 assembly code that adds the two integers pointed to by these variables and puts their sum in R3. You may assume that both pointers are valid (*i.e.*, non-NULL).

Part C (6 points): Write a C function corresponding to the LC-3 code on the left. The code shown implements only the body of the function; setting up and tearing down the stack frame are omitted. Remember that parameters are found at offsets of 4 or more from the frame pointer (R5) and that local variables are found at offsets of 0 or less from the frame pointer.

```

                LDR    R0,R5,#4      void func (char*
                STR    R0,R5,#0      ) {
AGAIN          LDR    R0,R5,#0
                LDR    R0,R0,#0
                BRz   DONE
                LDR    R0,R5,#0
                ADD    R0,R0,#1
                STR    R0,R5,#0
                BRnzp AGAIN
DONE          LDR    R0,R5,#0
                LDR    R1,R5,#5
                STR    R0,R1,#0     }

```

*****Part D** (2 points): Briefly explain the purpose of the function from Part C.

Problem 5 (20 points): C Programming

```
0 int foo (int x, int y) {
1     int z, val = x;
2
3     if (0 > x || 0 > y) {
4         return -1;
5     }
6
7     if (x > y) {
8         z = x / y;
9         val = x - y * z;
10    }
11    return val;
12 }
```

Most parts of this problem pertain to the C function shown above. The line numbers are only for reference and are not part of the program.

Part A (3 points): What is the return value of `foo (10, 3)` ? _____

Part B (3 points): What is the return value of `foo (452, 500)` ? _____

Part C (4 points): Using no multiplication operators, fill in a simple C expression below such that we can replace lines 7 through 10 of the function with the statement below without affecting the results returned from the function.

`val = _____;`

Part D (5 points): List the five types of information typically stored in a C function's stack frame (also called an activation record in the textbook)?

Part E (5 points): How many LC-3 memory locations does the stack frame for function `foo()` require? Assume that each `int` occupies one memory location. Justify your answer.

Page 11 Name: _____

Use this page for scratchwork.

A.3 The Instruction Set

	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															

Figure A.2 Format of the entire LC-3 instruction set. Note: + indicates instructions that modify condition codes