

ECE 199 Exam II Spring 2004

Tuesday, April 6th, 2004

Name:

- **Be sure that your exam booklet has 13 pages.**
- **Write your name at the top of each page.**
- **This is a closed book exam.**
- **You are allowed one handwritten 8.5" x 11" sheet 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!**

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

Total 100 points

Problem 1 (20 points): Short Answer

Part A (5 points): Give one reason for using local variables instead of global variables.

Part B (5 points): Describe one advantage of using a debugger.

Part C (5 points): Write the output of the following C code.

```
for( i = 0 ; i < 4 ; i++) {  
    if( i == 2 )  
        continue;  
    for( j = 0 ; j < 4 ; j++){  
        if( j == 2 )  
            break;  
        printf( "%d %d\n" , i , j );  
    }  
}
```

Part D (5 points): You and a friend are charged with writing a function that reorganizes an array of integers by discarding any negative elements and adjusting the count of valid elements. Your friend proposes the following declaration:

```
void keep_positive (int* array, int num_things);
```

Explain why this declaration cannot serve the intended purpose.

Problem 2 (20 Points): Stacks and Traps in LC-3

Parts A and B of this problem both make use of the `CHECK_X` function described below.

CHECK_X Description	Inputs	Outputs
Checks if there are at least X elements on the stack, where X is a non-negative integer.	R5 – X R6 – stack pointer	Returns 1 in R1 if there are at least X items on the stack, 0 otherwise.

Part A (6 points): Fill in the chart below describing the subroutine `FOO`. Use the description of the function `CHECK_X` to help you.

```

FOO          ST    R7, SAVE_R7
             JSR   CHECK_X
             ADD   R1, R1, #0
             BRZ   RESTORE
             ADD   R6, R6, R5
RESTORE      LD   R7, SAVE_R7
             RET
SAVE_R7      .BLKW 1
    
```

FOO Description	Inputs	Outputs

Parts A and B of this problem both make use of the `CHECK_X` function described below.

CHECK_X Description	Inputs	Outputs
Checks if there are at least X elements on the stack, where X is a non-negative integer.	R5 – X R6 – stack pointer	Returns 1 in R1 if there are at least X items on the stack, 0 otherwise.

Part B (6 points): Write the subroutine `IS_FULL` described below using the `CHECK_X` subroutine described above (repeated on this page for convenience).

IS_FULL Description	Inputs	Outputs
Checks if a stack of a given size is full (i.e., nothing else can be pushed onto the stack). Your subroutine must preserve the values of all registers other than R1.	R6 – stack pointer	Returns 1 in R1 if the stack is full, 0 otherwise.

```

STACK_SIZE .FILL ??? ; size of stack
                ; (you do not need to know the value)

IS_FULL
; your code goes here
    
```

Part C (8 points): The following code asks the user to input a single digit and prints out that number of letters from the beginning of the alphabet. For example, if the user enters “4”, “abcd” is printed to the screen, followed by a line feed. Complete the code.

```
.ORIG      x3000
LEA       R0, PROMPT
PUTS
GETC
OUT
LD        R1, NEG_ASC_0
ADD       R2, R0, R1
LD        R0, LINEFEED
OUT
; Insert your code here.  You may not change anything else.
```

```
DONE      HALT
PROMPT    .STRINGZ  "Enter a single digit: "
LETTER_A  .FILL     x0061 ; ASCII 'a'
LINEFEED  .FILL     x000A ; ASCII LF
NEG_ASC_0 .FILL     xFFD0 ; negative ASCII '0'
.END
```

Problem 3 (20 Points): Operators and Scope in C

Part A (10 points): Write the result of evaluating of the following expressions. The following declarations are used for each expression.

```
int i = 5, j = 12; /* signed 32-bit integers */
```

Evaluate each expression independently, using the same initial values of *i* and *j*.

a) $i \& j$

b) $++i \mid j$

c) $i = 7$

d) $!((i \mid j) \& 2)$

e) $7 + ((i++ - 6) \mid \mid (++j - 13))$

Part B (5 points): Your friend has written a program to convert temperatures from Fahrenheit to Celsius and says that the following code doesn't work:

```
temp_c = (5/9) * (temp_f - 32);
```

Explain the bug and fix the code.

Part C (5 points): Inside the function `foo()`, cross out any statements that cause compilation errors (i.e., are not legal in C).

```
void foo(int p);

int main() {
    int x = 10, y = 42;
    foo(y);
    return 0;
}

void foo(int p) {

    p = 0;

    x++;

    y++;

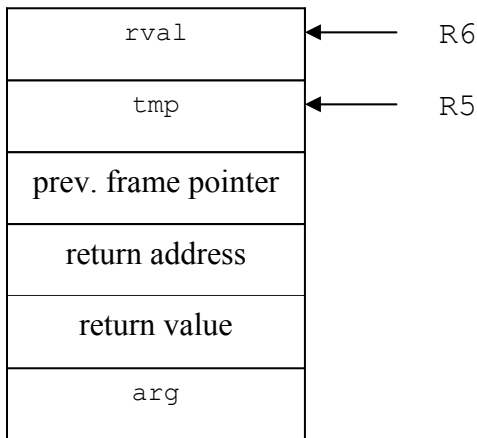
    p++;

}
```


Part B (10 points): Translate the C function below to LC-3 assembly instructions. The diagram of the stack frame for the function call has been provided for you.

Translate each of the **three statements** in the function body **independently, with no register values shared between statements**. The stack frame management and register save/restore has been done for you.

```
int is_power_of_two (int arg)
{
    int tmp = arg - 1;
    int rval = ((tmp & arg) == 0);
    return rval;
}
```



stack frame for is_power_of_two

```
;create stack frame and
;save registers
...
```

```
;translation for
;int tmp = arg - 1;
```

```
;translation for
;int rval = ((tmp & arg) == 0);
```

```
;translation for
;return rval;
```

```
;restore registers and
;tear down stack frame
...
```

Part C (4 points): The following is a code fragment from a C source file.

```
int a;  
  
int foo()  
{  
    int* b = &a;  
}
```

In what part of the program memory (e.g., system memory, code, etc.) are the following values stored?

i) a

ii) b

iii) &b

Problem 5 (20 Points): Programming in C

Part A (15 points): Write a function that returns a pointer to the **last** occurrence of the character `c` in the string `s`. If `s` does not contain `c`, your function should return `NULL`.

```
char* last_occurrence (char c, char* s)
{
```

```
}
```

*****Part B (5 points):** Write a function that returns a pointer to the **last** occurrence of the **string** `sub` in the string `s`. If `s` does not contain `sub`, your function should return `NULL`.

```
char* last_substring (char* sub, char* s)
{
```

```
}
```

Page 12 Name: _____

Use this page for scratch work.

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