Final Exam CSCI 255 Spring 2001 **Solution**

7 May, 2001

Name: _____

This is a closed book exam. Use of calculators is also not allowed. Be sure to show your work in order to get full credit for the problem. When possible place your answers in the provided boxes. There are 11 questions for a total of 200 points on this quiz.

This exam is to be turned in by 5:45 pm.

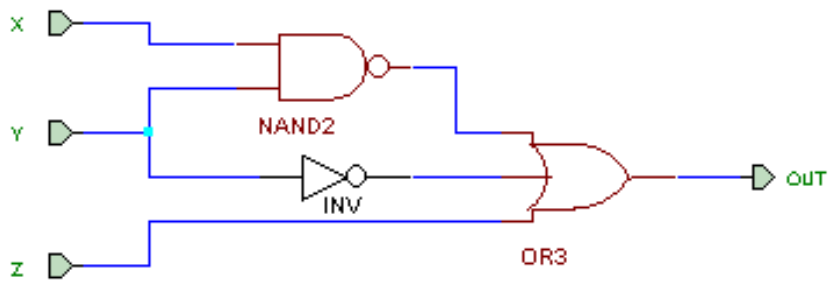
Problem 1 (10 points):

Convert the following numbers from eight-bit twos-complement notation into decimal notation.

00001111	11110000
15	**-16**

Problem 2 (10 points):

Fill in the truth table on the right to reflect the output of the circuit on the left.



x	*y*	*z*	Out
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**0**
1	1	1	**1**

Problem 3 (10 points):

Complete the following truth table for the given Boolean equation:

x	*y*	$(x + y')'+y$
0	0	0
0	1	1
1	0	0
1	1	1

Problem 4 (15 points):
Translate into LC-2 machine language (binary) the LC-2 assembly language program shown below:

```
      .ORIG     x3000
      LEA       R0,X            E009
      AND       R1,R0,#15       522F
      LDR       R2,R0,#2        6402
      ADD       R3,R0,R2        1602
      LDR       R4,R3,#1        68C1
      LDI       R5,Y            AA0E
      ADD       R6,R0,#-4       1C3C
      LDR       R7,R6,#5        6F85
      HALT                      F025
X     .FILL     x1              0001
      .FILL     x2              0002
      .FILL     x3              0003
      .FILL     x4              0004
      .FILL     x5              0005
Y     .FILL     x3004           3004
      .END
```

Problem 5 (15 points):
What are the values of registers R0 to R7 when the LC-2 assembly language program in Problem 4 is executed and reaches the HALT trap?

R0 = **x3009**	R4 = **x0005**
R1 = **x0009**	R5 = **x68C1**
R2 = **x0003**	R6 = **x3005**
R3 = **x300C**	R7 = **x0002**

Problem 6 (40 points):
In this problem you are asked to write **five** *independent* sections of LC-2 assembly code to set registers `R0` or `R1` or LC-2 memory locations based on constants, the present values of `R3` and `R4`, or LC-2 memory locations. You may use registers `R6` or `R7` as "scratch" registers but should not modify any other registers. You must assume that your code will be located somewhere between memory locations `x3000` and `x30FF`. You may use `.fill`'s when needed to initialize memory locations. You should assume that these `.fill`'s would also be stored in memory locations `x3000` to `x30FF`.

In these subproblems, the code to implement is given in the psuedo-C notation used in class lectures. `R`*n* will be used as a reference to LC-2 register *n*. `M[`*n*`]` will be used as a reference to LC-2 memory location *n*.

`R0 ← R4 | R5 ;`		**NOT**	**R6, R4**
		NOT	**R7, R5**
		AND	**R0, R6, R7**
		NOT	**R0, R0**
`R0 ← 3*R0 + 1 ;`		**ADD**	**R6, R0, R0**
		ADD	**R0, R6, R0**
		ADD	**R0, R0, #1**
`if (R0 > 30)`		**LD**	**R6, M30**
`  R0 ← R0 - 30 ;`		**ADD**	**R6, R6, M30**
		BRzp	**NS**
		ADD	**R0, R6, #0**
	NS	**…**	
		…	
	M30	**.FILL**	**#-30**
`while (R0 >= 0)`		**BR**	**MLP**
`  R0 ← R0 + R0 ;`	**LP**	**ADD**	**R0, R0, R0**
	MLP	**BRzp**	**LP**
`M[x4100] ← M[x4100] + 5 ;`		**LDI**	**R6, MX**
		ADD	**R6, R6, #5**
		STI	**R6, MX**
		…	
	MX	**.FILL**	**x4100**

Problem 7 (20 points):
Suppose A is "declared" as an array of 100 uninitialized LC-2 integers with:

```
A     .blkw     100
```

Write LC-2 code to set elements A[2], A[98], and A[R5], where R5 refers to the contents of register R5, to the value 3. You may assume that the array A resides in the same page as your code, and you may use registers R0, R1, and R2 as "scratch" registers.

```
A[2] = 3 ;      LEA     R0, A           ; R0 <- A
A[98] = 3 ;     AND     R1, R1, #0
A[R5] = 3 ;     ADD     R1, R1, #3      ; R1 <- 3
                STR     R1, R0, #2      ; A[2] = 3
                LD      R2, C98
                ADD     R2, R0, R2
                STR     R1, R2, #0      ; A[98] = 3
                ADD     R2, R0, R5
                STR     R1, R2, #0      ; A[R5] = 3
                …
    C98         .FILL   #98
```

Problem 8 (20 points):
The VAX computer has an instruction called BIC (Bit Clear) that performs the logical operation $\alpha\ \beta'$. Write an LC-2 subroutine called BIC in assembly language that performs this operation on two arguments X and Y. You should assume that BIC receives and returns its arguments on a standard LC-2 stack frame. In other words, implement the C function shown below in LC/2 assembler using the stack frame format of chapter 14.

```
int BIC(int x, int y) {
    return x & ~y;
}
```

```
    BIC     STR     R7, R6, #1
            LDR     R0, R6, #3
            LDR     R1, R6, #4
            NOT     R1, R1
            AND     R0, R0, R1
            STR     R0, R6, #0
            LDR     R7, R6, #1
            LDR     R6, R6, #2
            RET
```

Problem 9 (20 points):
Show how to call the LC-2 BIC subroutine of Problem 8. The two arguments passed to BIC are stored in registers R4 and R5 and the result should be stored in R3. Assume the size of the activation record of the calling routine is six words. That is, do:
That is, do:

```
R3 = BIC(R4, R5) ;
```

```
        STR     R4, R6, #9
        STR     R5, R6, #10
        ADD     R6, R6, #6
        JSR     BIC
        LDR     R3, R6, #6
```

Problem 10 (20 points):
Translate the following worthless function from C to LC-2 assembler. Use Chapter 14 style activation records to transmit parameters.

```
int dmbprg(int X, int *C, int *A) {
  int T ;
  if (X < 0)
    T = *C + 1
  else
    T = A[X] + 1 ;
  return T ;
}
```

```
DMBPRG  STR     R7, R6, #1
        LDR     R0, R6, #3      ; R0 <- X
        BRzp    DMBELS
        LDR     R1, R6, #4      ; R1 <- C
        BR      DMBALL
DMBELS  LDR     R1, R6, #5      ; R1 <- A
        ADD     R1, R6, R1      ; R1 <- &A[X]
DMBALL  LDR     R1, R1, #0      ; R1 <- *R1
        ADD     R1, R1, #1
        STR     R1, R6, #0
        LDR     R7, R6, #1
        LDR     R6, R6, #2
        RET
```

Problem 11 (20 points):
How have the following four concepts, programs, or standards been used in CSCI 255:

combinational circuit **A circuit where the present output depends solely on the present input. That is, there is no dependence on past inputs.**
IA-32 **Stands for Intel Architecture-32, the instruction set architecture for the Intel chips used in most of today's personal computers.**
`make` **A Unix command to control the compilation of programs from source files. Used in a CSCI 255 lab.**
memory-mapped I/O **A way of performing I/O by having device interfaces mimic memory addresses. Initiating and testing I/O devices is done by reading and writing to device *registers*.**