Quiz 1 CSCI 255 Spring 2001 **Solution**

26 February, 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.

Problem 1 (8 points): 3:15-3:21

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

-25	17
11100111	**00010001**

Problem 2 (8 points): 3:21-3:27

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

00010001	11111000
17	**-8**

Problem 3 (4 points): 3:27-3:30

Express the following decimal numbers in base 7.

50
101

Problem 4 (12 points): 3:30-3:39

Add the following two pair of eight-bit twos-complement numbers. Which, if any, of the additions results in an overflow?

10011110 + 10101000 **01000110** **overflow occurs**	11111100 + 10101000 **10100100** **no overflow**

Problem 5 (8 points): 3:39-3:45

Compute the following bit-wise logical operations on four-bit binary numbers.

NOT(1010) AND 0011 **0101 AND 0011** **0001**	NOT (1010 OR 0011) **NOT 1011** **0100**

Problem 6 (8 points): 3:45-3:51

Complete the following truth tables for the two given Boolean equations:

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

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

Problem 7 (8 points): 3:51-3:57

Translate the following truth table into a Boolean equation.

x' y z' + x' y z + x y' z'

x	*y*	*z*	out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Problem 8 (8 points): 3:57-4:03

Assume Z is a C integer variable. Write a C statement that will set bits 8 and 9 of Z to 1 and clear bits 4 and 5 to 0.

This is a question about masking bits.

```
Z = (Z | 0x300) & ~0x30 ;
```

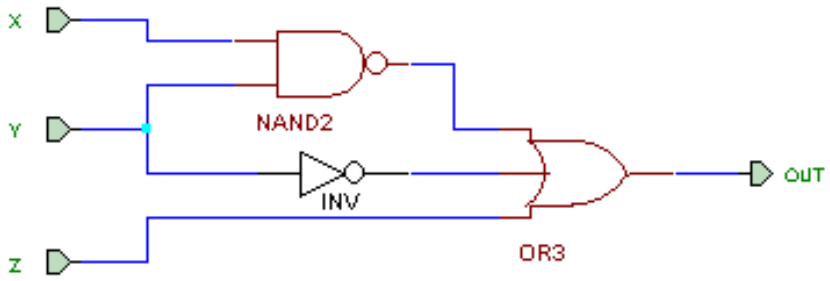
Problem 9 (8 points): 4:03-4:09

Convert the following 16-bit binary numbers to hexadecimal numbers.

1010000001111111 **A07F**	0000110010011011 **0C9B**

Problem 10 (8 points): 4:09-4:15

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 11 (20 points): 4:15-4:30

How have the following five concepts or standards been used in CSCI 255:

ASCII **Standard code for mapping letters into binary**
Combinational circuit **A circuit where the present output always depends solely on that present input.** **Even a wire or a one-gate circuit can be combinational.**
IEEE floating point format **Standard binary representation of "real" numbers expressed in scientific notation**
Multiplexer **A circuit that receives 2^m data inputs and m selector inputs and produces a single selected data output.**
P-type transistor **Used to build inverters** **Conducts when its gate is low**

Quiz 2 **Solution** CSCI 255 Spring 2001

9 April, 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 9 questions for a total of 150 points on this quiz.

Problem 1 (64 points): 3:15-3:47

In this problem you are asked to write **eight** *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 initial 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*.

There are many possible right answers. These are probably the shortest.

`R0 ← 5 * R3 ;`		**ADD**	**R0,R3,R3**
		ADD	**R0,R0,R0**
		ADD	**R0,R0,R3**
`R0 ← R3 – R4 ;`		**NOT**	**R0,R4**
		ADD	**R0,R0,#1**
		ADD	**R0,R0,R3**
`R0 ← R3 & R4 ;`		**AND**	**R0,R3,R4**

`if (R3 == 15)`
`  R0 ← R4 ;`
`else`
`  R0 ← R4 + 1 ;`
		ADD	**R0,R4,#0**

		ADD	**R6,R3,#-15**
		BRz	**DONE**
		ADD	**R0,R0,#1**
	DONE	**…**	

`R0 ← R4 ;`
`while (R0 < 107)`
`  R0 ← R0 + R0 ;`
		LD	**6,M107**

		ADD	**R0,R4,#0**
		BR	**MDLOOP**
	BGLOOP	**ADD**	**R0,R0,R0**
	MDLOOP	**ADD**	**R7,R6,R0**
		BRn	**BGLOOP**
		…	
	M107	**.FILL**	**#-107**

M[x3100] ← M[x3100] + 5 ;		LD	R6,x3010
		ADD	R6,R6,#5
		ST	R6,x3010
M[x4100] ← M[x4100] + 5 ;		LDI	R6,PTR
		ADD	R6,R6,#5
		STI	R6,PTR
		…	
	PTR	.FILL	x3010
R0 ← R3 + 1 ; R1 ← R4 + '1' ;		ADD	R0,R3,#1
		LD	R6,ASC1
		ADD	R1,R4,R6
		…	
	ASC1	.FILL	X31

Problem 2 (16 points): 3:47-3:55

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

A fairly common problem was starting the first instruction (LD) at x3001 rather than x3000.

	.ORIG	x3000	
	LD	R1,MX	0010001000000101
	LDI	R2,MX	1010010000000101
	LEA	R3,MX	1110011000000101
	LDR	R4,R1,#1	0110100001000001
	HALT		1111000000100101
MX	.FILL	0x3006	0011000000000110
MY	.FILL	0x3007	0011000000000111
MZ	.FILL	0x3008	0011000000001000
	.END		

The offsets for these three instructions (LD, LDI, and LDR) are for the LC-2. The LC-3 has a different format for instruction offsets.

Problem 3 (12 points): 3:55-4:01

What are the values of registers R1 to R4 after the LC-2 assembly language program in Problem 2 is executed?

R1 = **x3006**
R2 = **x3007**
R3 = **x3005**
R4 = **x3008**

If you (incorrectly) assumed that MX was located at x3006, then R1, R2, and R3 would be set to x3006 and R4 would be set to x3007.

Problem 4 (8 points): 4:01-4:05

What Linux command would you use to assemble the LC-2 assembly program `lab9.asm`? Give not only the name of the command, but the arguments you use with it.

```
lc2asm lab9.asm lab9
```

lc3 lab9.asm for LC-3 Linux

Problem 5 (10 points): 4:05-4:10

Write some LC-2 assembly code to write the contents of register 5 to the CRT using the CRT data and status registers?

```
OUTLOOP   LDI      R6, ACRTSR
          BRzp     OUTLOOP
          STI      R5, ACRTDR
          ……
ACRTSR    .FILL    xF3FC
ACRTDR    .FILL    xF3FF
```

These device registers are stored at different locations on the LC-3.

Problem 6 (8 points): 4:10-4:14

Write some LC-2 assembly code to write the contents of register 5 to the CRT using a LC-2 trap routine?

```
          ADD      R0, R5, #0
          OUT
```

Problem 7 (12 points): 4:14-4:20

The VAX computer has an instruction called BIC (Bit Clear) that performs the logical operation $\alpha\ \beta'$. Write a LC-2 subroutine called BIC in assembly language that performs this operation on registers R0 and R1, that is, the subroutine performs the operation:

R0 ← R0 & ~R1 ;

```
BIC       NOT      R1, R1
          AND      R0, R0, R1
          NOT      R1, R1          ; restore R1
          RET
```

Problem 8 (8 points): 4:20-4:24

Show the complete LC-2 instruction needed to call the BIC subroutine of Problem 7. You may assume that both the calling and called subroutine are on the same page.

```
          JSR      BIC
```

Problem 9 (10 points): 4:24-4:30

Translate the following two LC-2 binary instructions into LC-2 assembly code.

0001011011111110	ADD R3, R3, #-2
0101011011000111	AND R3, R3, R7

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.

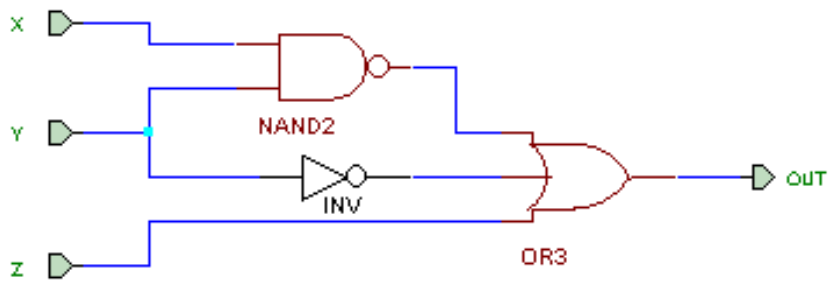
Problem 1 (10 points):

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**

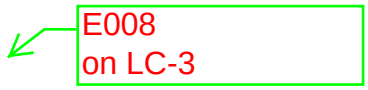
Problem 3 (10 points):

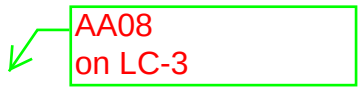
Complete the following truth table for the given Boolean equation:

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**

The questions on this page are not particularly relevant to the Spring 2008 ECE 109/CSCI 255

This is from a much later chapter of the book which covers arrays.

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;
}
```

A different stack frame format is used on the LC-3.

Very little about this answer would be correct now.

```
    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
```

The questions on this page are not particularly relevant to the Spring 2008 ECE 109/CSCI 255

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) ;
```

Again, with the present LC/3 stack format, this would not work.

```
        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 ;
}
```

Once again, this would not work for the LC/3.

```
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*.**