

ECE 109 Sections 602 to 605

Answer to *some* problems

Final exam Fall 2007

13 December, 2007

Problem 1 (6 points) Memories

a) How many **bits** are in a memory with 2K words and a word size of 32?

2*32K or 64K

b) How many **bits** are required to address a memory with 64K words?

16 since $2^{16} = 64K = 64 * 1024 = 2^6 * 2^{10}$

c) How many 16-bit words can be stored in a 2 KB memory?

1K or 2KB*8/16

Problem 2 (2 points) Overflow

Add the following pairs of six-bit two's complement numbers **and indicate which additions result in an overflow.**

$\begin{array}{r} 110111 \\ + 010010 \\ \hline 001001 \\ \text{no overflow} \end{array}$	$\begin{array}{r} 010101 \\ + 010101 \\ \hline 101010 \\ \text{overflow} \end{array}$
--	---

Problem 3 (2 points) Twos complement representation

Translate the following two decimal numbers into five-bit two's complement numbers. Show your work in the space below!

$\begin{array}{r} 12 \\ 01100 \end{array}$	$\begin{array}{r} -11 \\ 10101 \end{array}$
--	---

Problem 4 (4 points) Fixed point numbers

Express 21.75 as a fixed point binary number. (Remember, 11.1 binary represents 3.5 decimal.)

10101.11

Problem 5 (4 points) Logical foundations

In Chapter 2, we learned about deMorgan's law and twos-complement representation. How are these fundamental concepts used to perform the bitwise OR and arithmetic subtract operations in the LC/3?

deMorgan's law allows us to perform an OR of registers R1 and R2 as the following

NOT R1,R1

NOT R1,R2

AND R3,R1,R2

NOT R3,R3

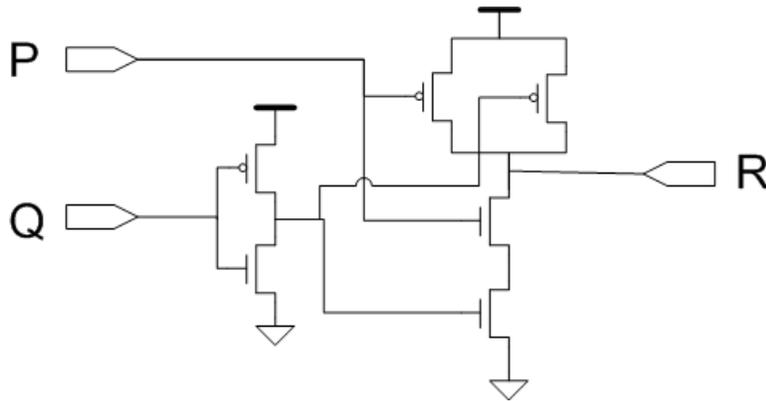
;; can also NOT R1 and R2 back to their original values

Twos-complement allows us to perform a subtraction of R2 from R1 as

NOT R3, R2
 ADD R3, R3, #1
 ADD R3, R1, R3

Problem 6 (2 points) MOS to truth

Fill in the truth table on the right to represent the MOS circuit that is on the left. (This should be familiar.)

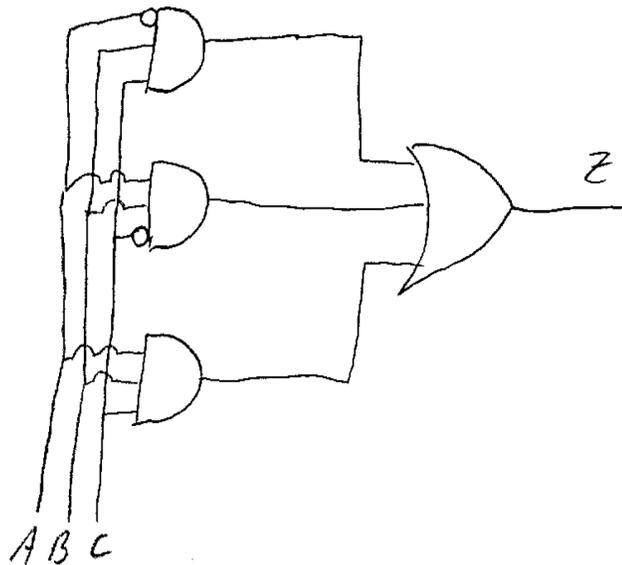


P	Q	R
0	0	1
0	1	1
1	0	0
1	1	1

Problem 7 (4 points) Truth to gates

Draw a circuit, at the gate level, that will implement the truth table, where Z is the single output variable, shown on the right below.

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Problem 8 (4 points) Gates to MOS

Draw the MOS circuit that implements a 2-input OR gate.

Look in the textbook!

Problem 10 (12 points) Vocabulary

Compare and contrast the following related pairs of terms. You may cross out one to indicate you don't want it graded. Otherwise I'll grade the first six you answer.

Look in the textbook!

Assembly <i>versus</i> machine language
Combinational logic <i>versus</i> sequential logic
Conditional <i>versus</i> iterative execution
Interrupt-driven I/O <i>versus</i> pooling
Interrupt service routine <i>versus</i> system call
MAR <i>versus</i> MDR
Sign-extension <i>versus</i> sign-magnitude

Problem 10 (6 points) Assembling

Using the symbol table shown below

CASH	x4023
LYNN	x4057
NELSON	x40A3
PARTON	x4123

write the appropriate 16-bit LC-3 machine language word, in binary or hex, for each assembly language statement shown in the left column. Assume that the instruction is located at address x3400 in all cases. If the assembly language statement is illegal, state the reason why this is so.

AND	R0, R5, #3	0101000101100011
AND	R7, NELSON	illegal – needs three register operands
BRnzp	LYNN	illegal – LYNN past 9 bit offset
BRp	PARTON	illegal – PARTON past 9 bit offset
JSR	R3	illegal – needs address for operand
JSR	PARTON	illegal – PARTON past 11 bit offset
JSRR	R5	0100000101000000
LEA	R2, CASH	illegal – CASH past 9 bit offset
RET		1100000111000000
SUB	R3, R3, R3	illegal – no SUB opcode
STR	R7, R3	illegal – needs three operands
STR	R3, R6, #-3	0111011110111101
TRAP	x15	1111000000010101

Problem 11 (6 points) Reverse engineering

The binary program shown in the left column below is loaded into memory at location x3000. In the right column, write the LC/3 assembly instructions or appropriate psuedo-ops corresponding to this program. Be sure to include appropriate labels and `.ORIG` and `.END` statements.

Binary	Assembly
	<code>.ORIG x3000</code>
0101100000100000	<code>AND R4, R0, #0</code>
0001100100100001	<code>Label01 ADD R4, R4, #1</code>
0010011000000100	<code>LD R3, Label07</code>
1000000011000000	<code>RTI</code>
0101101000111111	<code>AND R5, R0, #-1</code>
0000011111111011	<code>BRzp Label01</code>
1111000000100101	<code>HALT</code>
0101000000000000	<code>Label07 .FILL x5000</code>
	<code>.END</code>

Problem 13 (12 points) Being JSR'ed around

The following program *really* has been successfully assembled and simulated on the LC/3 software.

```

        .ORIG    x3000                ; instruction location
START   AND     R1, R1, #0            ; x3000
        LEA     R5, START            ; x3001
        LDI     R3, ADD1             ; x3002
        JSRR   R3                    ; x3003
        ADD     R1, R1, x9           ; x3004
        JSR    LABZ                  ; x3005
LABX    ADD     R1, R1, x3           ; x3006
LABZ    AND     R3, R1, x4           ; x3007
        BRp    STOP                 ; x3008
        RET    RET                  ; x3009
STOP    STR     R1, R5, #12          ; x300A
        HALT   HALT                 ; x300B
ADD1    .FILL   ADD2                ; x300C
ADD2    .FILL   LABX                ; x300D
        .END

```

The last two `.FILL`'s really are legal LC/3 pseudo-ops. If it makes you more comfortable, you can replace them with the following more obscure, though equivalent, pseudo-ops.

```

ADD1    .FILL   x300D
ADD2    .FILL   x3006

```

Your job is to describe what the LC/3 will do while executing this code until either (1) you have filled the following table or (2) the LC/3 executes the HALT trap. Put the address of each instruction in the left column and its action in the right column. I've filled in one for you.

x3000	Sets R1 to 0
x3001	Sets R5 to x3000
x3002	Sets R3 to x3006
x3003	Sets PC to x3006 and sets R7 to x3004
x3006	Sets R1 to 3
x3007	Sets R3 to 0
x3008	Doesn't branch as Z bit was set on last instruction
x3009	Sets PC to x3004, the value stored in R7
x3004	Sets R1 to 12 (x0C)
x3005	Sets PC to x3007 and sets R7 to x3006
x3007	Sets R3 to 4
x3008	Sets PC to x300A as P bit is set
x300A	Stores 12 (R1) in memory location x300C
x300B	Halts

Problem 13 (16 points) LC/3 programming

In this long question of many parts, write little (many only two or three instructions long) LC/3 programs to solve the following small problems stated in a C-like syntax. Answers that are unnecessary long or complicated may not receive full credit.

R3 = R5 + 18 ; ADD R3, R5, #9 ADD R3, R3, #9
R5 = R7 - R3 ; NOT R5, R3 ADD R5, R5, #1 ADD R5, R7, R5
R4 = R3 ; ADD R4, R3, #0

```

while (R1<93) {
    R1 = 3*R1 ;
}

    LD      R2 ,M93
LOOPADD  R3 ,R1 ,R2
    BRnz   EOPR
    ADD    R3 ,R1 ,R1
    ADD    R1 ,R3 ,R1
    BR     LOOP
EOPR . . . .

```

```

M93 .FILL #-93

```

```

if (R4 > 0 && R3>0) {
    R5 = 15 ;
}

    ADD    R4 ,R4 ,#0
    BRnz   EOPR
    ADD    R5 ,R6 ,#0
    BRnz   EOPR
    AND    R5 ,R5 ,#0
    ADD    R5 ,R5 ,#15
EOPR

```

```

if (R5 == 'n') {
    R5 = 'N' ;
}

    LD      R4 ,NCHn
    ADD    R4 ,R5 ,R4
    BRnp   EOPR
    LD      R5 ,PCHN
EOPR . . . .

```

```

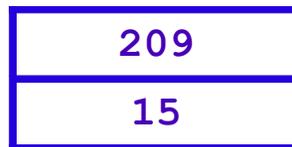
NCHn .FILL #-110
PCHN .FILL #78

```

Problem 14 (2 point) POP and PUSH

Draw the stack the results from the following sequence of pop and push operations and write the value retrieved on the two POP operations.

```
PUSH 15 ;  
PUSH 109 ;  
POP ;           --> 109 retrieved here  
PUSH 209 ;  
PUSH 15 ;  
POP ;           --> 15 retrieved here
```



Problem 15 (4 points) Device registers

Using the keyboard status and data registers (KBSR and KBDR) and none of the LC/3 trap routines, write a silly LC/3 trap routine that reads one character from the keyboard and returns in R0, 1 if the character was 'y' or 'Y', and 0, otherwise. Except for R0, your routine should restore all registers that it modifies.

P.S. Your code would look the same if it was a subroutine rather than a trap routine.

```
PR15    ST    R1 , SVR1  
        ST    R2 , SVR2  
        AND   R0 , R0 , #0  
SPOL    LDI   R1 , KBSR  
        BRzp  SPOL  
        LDI   R1 , KBDR  
        LD    R2 , MCHy  
        ADD   R2 , R2 , R1  
        BRz   SRSTR  
        LD    R2 , MCHY  
        ADD   R2 , R2 , R1  
        BRz   SRSTR  
        ADD   R1 , R1 , #1  
SRSTR   LD    R1 , SVR1  
        LD    R2 , SVR2  
        RET  
KBSR    .FILL xFE00  
KDSR    .FILL xFE02  
MCHy    .FILL #-121  
MCHY    .FILL #-89
```

Problem 16 (4 points) LC/3 subroutine

Write a LC/3 subroutine that receives in register R0 an LC/3 address. Your subroutine should then store zeros in the 15 LC/3 memory locations starting at the address stored in R0. To get full credit for this problem, you must use a loop.

```

PR16    ST    R1 , SVR1
        ST    R2 , SVR2
        AND   R1 , R1 , #0
        ADD   R1 , R1 , #15
        AND   R2 , R2 , #0
LP      STR   R2 , R0 , #0
        ADD   R0 , R0 , #1
        ADD   R1 , R1 , #-1
        BRp  LP
        ADD   R0 , R0 , #-15
        LD    R1 , SVR1
        LD    R2 , SVR2
        RET

```

Problem 17 (2 points) Invoking the subroutine

Show how to call the LC/3 subroutine you wrote in Problem 16 to set the 15 memory locations starting with x5555 to 0.

```

        LD    R0 , K5555
        JSR   PR17
        . . . .
K5555   .FILL x5555

```

Problem 18 (2 points) C starter

What is the purpose of the following line often seen at the beginning of a C program?

```
#include <stdio.h>
```

Includes the standard I/O package, so that printf, scanf, etc., can be used.

Problem 19 (2 points) Tracing C

What does the following C code print when it is executed?

```
#include <stdio.h>
main() {
    int i = 3 ;
    int j = 8 ;
    while (i<j) {
        printf("%d\n", i) ;
        i = i + 2 ;
    }
    printf("%d\n", i+20) ;
}
```

- 3**
- 5**
- 7**
- 29**

Problem 20 (4 points) Writing C

Write a complete C program that reads in two numbers, multiplies them together, and writes out both numbers along with their product.

```
#include <stdio.h>
main() {
    double n1, n2 ;
    scanf("%lf %lf", &n1, &n2) ;
    printf("%lf, %lf, %lf\n", n1, n2, n1*n2) ;
}
```

Problem 21 (2 points) C I/O

Assuming that the following variable declarations have been made in C:

```
int i ;
float x ;
```

which of the following eight C statements are bad. (Write *good* or *bad* in the box.)

printf("%d", i) ; GOOD	scanf("%d", i) ; BAD: Need & before i
printf("%f", x) ; GOOD	scanf("%d", &i) ; GOOD
printf("i = %d", i) ; GOOD	printf("%f %d", x) ; BAD: Needs two data values
printf(this can't be right); BAD: in so many ways	printf("%d", x) ; BAD: need %f to print x