Department of Electrical and Computer Engineering
The University of Texas at Austin

EE 306, Fall, 2006
Yale Patt, Instructor
TAs: Aseem Bathla, Cameron Davison, Lisa de la Fuente, Phillip Duran, Jose Joao, Jasveen Kaur, Rustam Miftakhutdinov, Veynu Narasiman, Nady Obeid, Poorna Samanta.
Final Exam, December 13, 2006

Name:_____

Problem 1 (15 points):_____

Problem 2 (10 points):_____

Problem 3 (10 points):_____

Problem 4 (10 points):_____

Problem 5 (10 points):_____

Problem 6 (10 points):_____

Problem 7 (10 points):_____

Problem 8 (10 points):_____

Problem 9 (15 points):_____

Total (100 points):_____

Note: Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Note: Please be sure your name is written legibly on each sheet of the exam.

I will not cheat on this exam.

_____
Signature

GOOD LUCK!

Name:_____

Problem 1 (15 points)

Part a (5 points): Here is a list of the 16 opcodes. Circle the ones that write to a general purpose register (R0 to R7) at some point during the instruction cycle.

ADD	AND	BR	JMP	JSR	LD	LEA	LDI
LDR	NOT	RTI	ST	STI	STR	TRAP	reserved

Part b (5 points): Program A running at priority level 0 starts running on the LC-3 at the start of time unit 1. Program A requires 8 time units to complete. The table below shows information for three interrupting events (x, y, z): when each occurs, the priority level of each, and the number of time units needed by the corresponding interrupt service routine (X, Y, Z) to complete its task.

```
x:  interrupt occurs at end of time unit 6.  Priority 4.  X needs 3 time units.
y:  interrupt occurs at end of time unit 8.  Priority 8.  Y needs 5 time units.
z:  interrupt occurs at end of time unit 9.  Priority 5.  Z needs 2 time units.
```

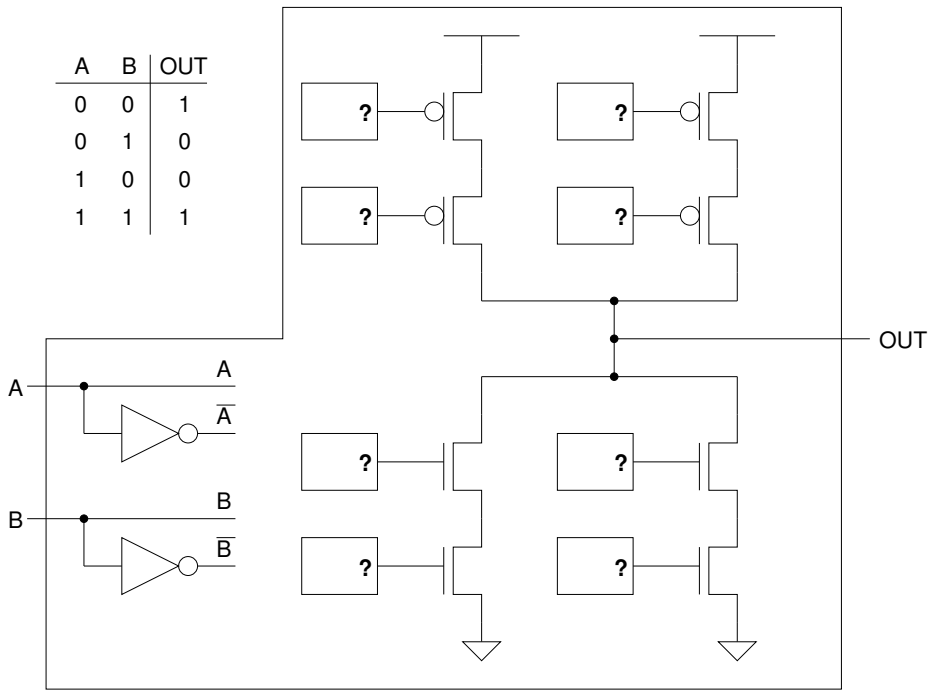
In the boxes below, write the letter (A, X, Y, or Z) of the program or service routine running during that time unit.

Note: We have inserted the answer for time units 1,2, and 3.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	A	A															

Part c (5 points): Label the gate inputs to each of the 8 transistors in the figure below so that the operation of the transistor circuit behaves as specified in the truth table below. You can label each gate input as A, $\bar{A}$, B, or $\bar{B}$.

A	B	OUT
0	0	1
0	1	0
1	0	0
1	1	1



Name:_____

Problem 2 (10 points)

A subroutine TEST_FOR_NEG can be used to examine a value and return a 1 in R5 if it is negative, and a 2 if it is positive or zero. The routine is shown below:

```
TEST_FOR_NEG    BRn   SKIP
                AND   R5, R5, #0
                ADD   R5, R5, #2
                BRnzp DONE
SKIP            AND   R5, R5, #0
                ADD   R5, R5, #1
DONE            RET
```

Suppose we are writing a program that needs this subroutine. For example, we need to test the value contained at location A. We did the following and everything worked fine.

```
LD    R0, A
JSR   TEST_FOR_NEG
```

An A&M student decided that the JSR opcode is unnecessary, and replaced it in our main program with two instructions, as follows:

```
LD    R0, A
LEA   R7, #1
BRnzp TEST_FOR_NEG
```

Part a (5 points): After making this change, the program would not assemble. What was the assemble-time error? Answer in the box provided below in fewer than 20 words. Note: More than 20 words will get a 0.

Answer:

Part b (5 points): With the help of an OU student, they fixed the assembly-time error. When they tried to execute the program, they got a run-time error. What was the run-time error? Answer in the box provided below in fewer than 20 words. Note: More than 20 words will get a 0.

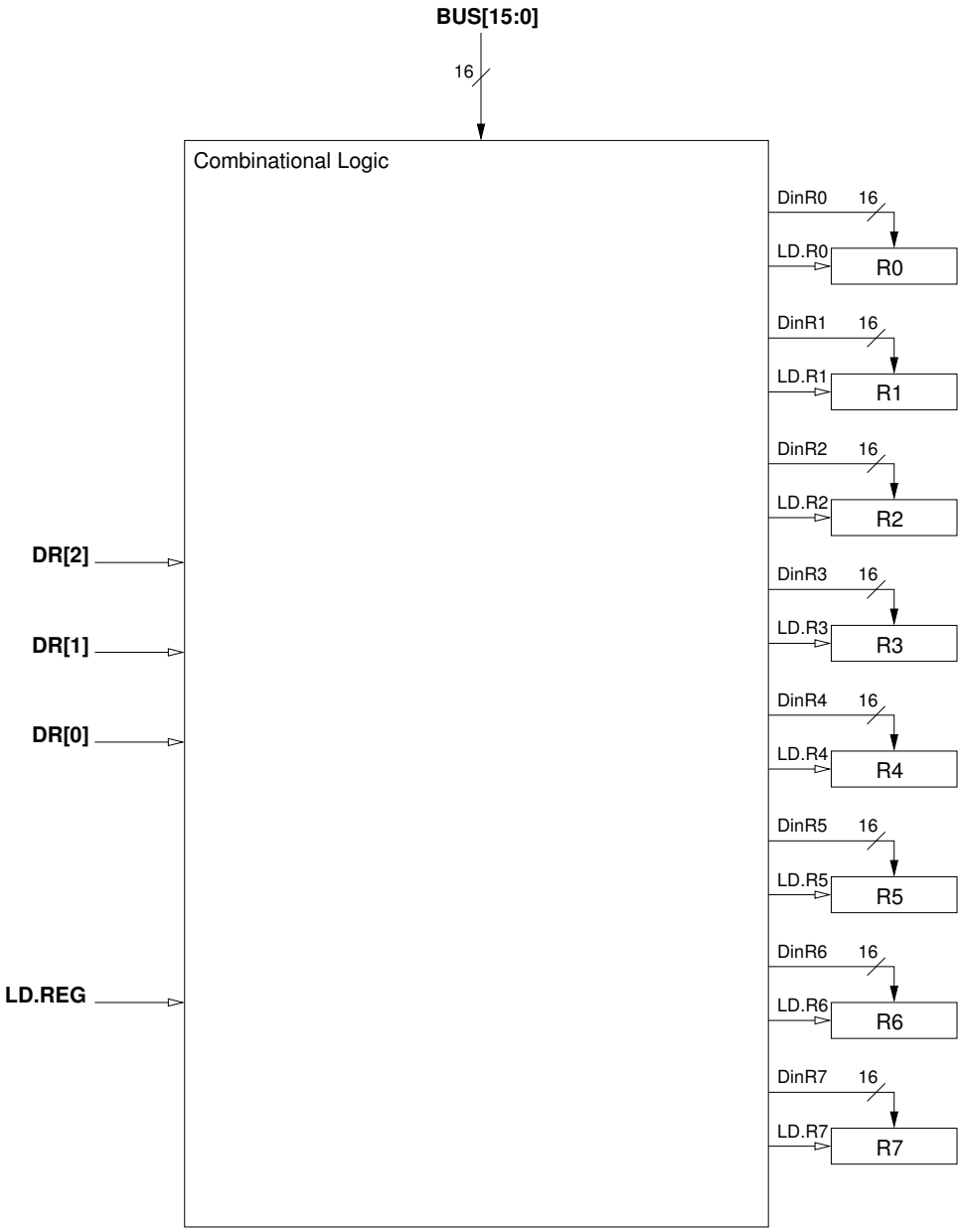
Answer:

Name:_____

Problem 3 (10 points)

The 8 general purpose registers of the LC-3 (R0 to R7) make up the Register File. To write a value to a register, the LC-3 control unit must supply 16 bits of data (BUS[15:0]), a destination register (DR[2:0]), and a write enable signal (LD.REG) to load a register. The Combinational Logic Block below shows inputs BUS[15:0], DR[2:0], and LD.REG and outputs DinR0[15:0], DinR1[15:0], DinR2[15:0], ... DinR7[15:0], LD.R0, LD.R1, LD.R2, ... LD.R7.

Your job: Add wires, logic gates, and standard logic blocks as necessary to complete the Combinational Logic Block. Note: If you use a standard logic block, it is not necessary to show the individual gates. However, it is necessary to identify the logic block specifically (e.g., "16-to-1 mux"), along with labels for each relevant input or output, according to its function.



Name:_____

Problem 4 (10 points)

The table below shows part of the state of the LC-3 (the contents of the eight registers, the PC, the condition codes, and memory locations x2000 through x2007) at a particular instant in time (the column labeled **Before**). Then the LC-3 was single-stepped twice (i.e., two LC-3 instructions were executed), producing the state of the machine indicated by the column labeled **After**.

Your job: Fill in the five missing entries in the table indicated by question marks (?). Note: We have supplied the high order hex digit for the contents of memory location x2006.

	Before	After
R0	x0505	x0505
R1	x1FFF	x1FFF
R2	x3006	x3006
R3	x2100	x2101
R4	x4567	x4567
R5	xFFFF	xFFFF
R6	x2002	x2004
R7	x3010	x3010
PC	x2006	x3007
N	0	?
Z	0	?
P	1	?
M[x2000]	x20A0	
M[x2001]	x0702	
M[x2002]	x3007	
M[x2003]	x8004	
M[x2004]	x601A	
M[x2005]	x0501	
M[x2006]	? x1 _ _ _	
M[x2007]	? x	

Name:_____

Problem 5 (10 points)

Part a (3 points): Generate the symbol table for the program below. You may not need all of the spaces provided. Note: One of the symbols, PUSH, is global and has already been done for you.

```
        .EXTERNAL PUSH

        .ORIG  x3000
        LEA    R6, BOTTOM
        LEA    R1, MESSAGE
        AND    R0, R0, #0
        JSR    PUSH
AGAIN   LDR    R0, R1, #0
        BRz    DONE
        JSR    PUSH
        ADD    R1, R1, #1
        BRnzp  AGAIN
DONE    ADD    R0, R6, #0
        PUTS
        HALT
MESSAGE .STRINGZ "STACKS ARE COOL"
        .BLKW #20
BOTTOM  .BLKW #1
        .END
```

Symbol	Address
PUSH	GLOBAL

Part b (7 points): What does this program output to the console? Assume that the `PUSH` and `POP` subroutines are already written for you and work EXACTLY as described in class. Answer in the box provided, please.

Name:_____

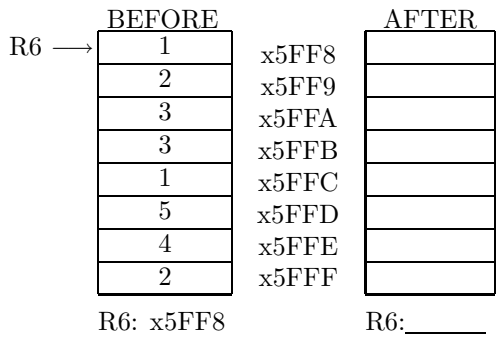
Problem 6 (10 points)

A program (part of which is shown below) uses a stack to store values that the program operates on. The stack is implemented in memory locations x5FF8 (MAX) to x5FFF (BASE). At the time the instruction stored at the label `AGAIN` is executed, the Stack Pointer (R6) contains x5FF8.

Your job is to show the contents of the stack after the program has executed. Also, show the final contents of the Stack Pointer.

You can assume that the `PUSH` and `POP` routines have been written for you and that they behave EXACTLY as described in class. Recall that success (0) and failure (1) information is passed back to the calling program through R5, and if the stack operation failed, R0 and R6 remained unchanged.

```
AGAIN   AND   R0, R0, #0
        JSR   POP
        ADD   R1, R0, #0
        AND   R0, R0, #0
        JSR   POP
        ADD   R2, R0, #0
        AND   R0, R0, #0
        JSR   POP
        ADD   R0, R0, R1
        ADD   R0, R0, R2
        ADD   R5, R5, #0
        BRp   DONE
        JSR   PUSH
        BRnzp AGAIN
DONE    JSR   PUSH
        HALT
```

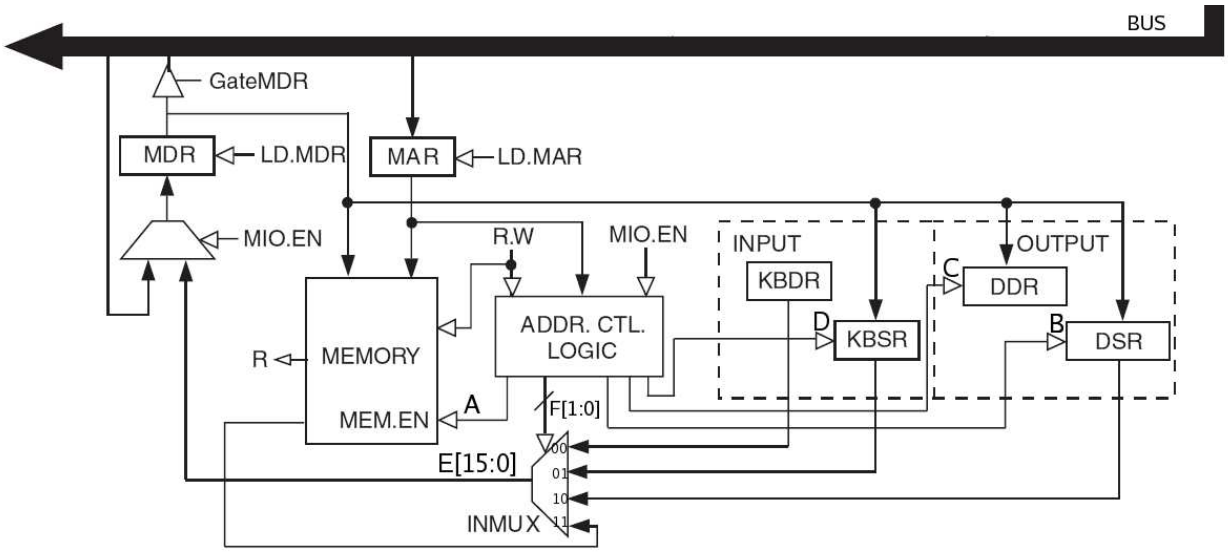


Name:_____

Problem 7 (10 points)

The statement of this problem is identical to problem 9 on Problem Set 6; only the values and the instruction you have to work out are different.

The figure below shows the part of the LC-3 data path that deals with memory and I/O. Note the signals labeled A through F. A is the memory enable signal; if it is 1, memory is enabled, if it is 0, memory is disabled. B, C, and D are the load enable signals for the Device Registers that can be written to. If the load enable signal is 1, the corresponding register is loaded with a value, otherwise it is not. E[15:0] is the 16-bit output of INMUX. F[1:0] are the two select lines for INMUX.



The initial values of some of the processor registers and the I/O registers, and some memory locations are as follows:

R0 = xFE00	KBSR = x8000	M[x3009] = xFE00
R1 = x0039	KBDR = x0061	M[x300A] = xFE02
PC = x3000	DSR = x8000	M[x300B] = xFE04
	DDR = x0031	M[x300C] = xFE06

As you know, during one complete instruction cycle, memory-I/O is accessed either one, two, or three times. Your job in this problem is to specify all the memory-I/O accesses associated with processing the instruction STI R1,#11.

Complete the table with the values of each control signal and register at the end of each cycle in which a memory-I/O access occurs. If the value of a signal does not matter in a cycle, put an X in the corresponding entry in that row. If the instruction does not require three memory-I/O accesses, draw a line through each unnecessary row in the table.

PC	Instruction	Access	MAR	A	B	C	D	E[15:0]	F[1]	F[0]	MDR
x3000	STI R1,#11	1									
		2									
		3									

Name:_____

Problem 8 (10 points)

The program below inputs a single digit from the keyboard (1-9) and then echoes it to the console. Let's call the number the user typed N. The program displays to the console every Nth letter of the alphabet, starting with the letter A. Some of the instructions are missing. Fill in the blanks to make the program work as specified. Note: Each blank corressponds to one instruction. Some example output is provided below:

Example 1:

```
1ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Example 2:

```
2ACEGIKMOQSUWY
```

Example 3:

```
5AFKPUZ
```

```
            .ORIG  x3000
            TRAP   x20
            TRAP   x21
            [                          ]   ; R1 = N, converted from ascii
            LEA    R2, ALPHA
LOOP        [                          ]
            BRz    END
            TRAP   x21
            [                          ]
            BRnzp  LOOP
END         HALT
ALPHA       .STRINGZ "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
            .FILL   #0
            .FILL   #0
            .FILL   #0
            .FILL   #0
            .FILL   #0
            .FILL   #0
            .FILL   #0
            .FILL   #0
            .FILL   #0
            .END
```

Name:_____

Problem 9 (15 points)

A program encounters a breakpoint and halts. The computer operator does not change the state of the computer in any way, but immediately presses the **run** button to resume execution.

The table below shows the contents of MAR and MDR for the first nine memory accesses that the LC-3 performs after resuming execution.

Your job: Fill in the missing entries.

	MAR	MDR
1st:		x5020
2nd:		xF0F0
3rd:		
4th:	x2000	x020A
5th:		x040A
6th:		x61FE
7th:		
8th:		xC1C0
9th:	x4002	xF025

Some information you might find useful...

The Stack Protocol

```
01  ; Subroutine for carrying out the PUSH and POP functions.  This
02  ; program works with a stack consisting of memory locations BASE
03  ; through MAX. R6 is the stack pointer.  R0 contains the data.
04  ;
05  PUSH          ST        R1,SaveR1        ; R1 is needed by PUSH routine
06                LD        R1,FULL          ; FULL contains -(MAX)
07                ADD       R1,R1,R6
08                BRz       Fail_exit        ; SP=MAX (no room on the stack)
09  ;
0A                ADD       R6,R6,#-1
0B                STR       R0,R6,#0         ; R0 gets PUSHed onto the stack
0C                BR        Success_exit
0D  ;
0E  POP           ST        R1,SaveR1        ; R1 is needed by POP routine
0F                LD        R1,EMPTY         ; EMPTY contains -(BASE+1)
10                ADD       R1,R1,R6
11                BRz       Fail_exit        ; SP=(BASE+1) (nothing on the stack)
12  ;
13                LDR       R0,R6,#0         ; Top of stack gets POPped to R0
14                ADD       R6,R6,#1
15  ;
16  Success_exit  AND       R5,R5,#0         ; Success code: R5=0
17                LD        R1,SaveR1
18                RET
19  Fail_exit     AND       R5,R5,#0
1A                ADD       R5,R5,#1         ; Fail code: R5=1
1B                LD        R1,SaveR1
1C                RET
1D  FULL          .FILL   -MAX
1E  EMPTY         .FILL   -(BASE+1)
1F  SaveR1        .BLKW   1
```

The PSR register