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

EE 306, Fall 2015
Yale Patt, Instructor
Stephen Pruett, Siavash Zangeneh, Kamyar Mirzazad, Esha Choukse, Ali Fakhrzadegan, Zheng Zhao, Steven Flolid, Nico Garofano, Sabee Grewal, William Hoenig, Adeesh Jain, Matthew Normyle
Exam 2, November 11, 2015

Name: Solution

Problem 1 (20 points):______

Problem 2 (15 points):______

Problem 3 (15 points):______

Problem 4 (25 points):______

Problem 5 (25 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 recorded on each sheet of the exam.

I will not cheat on this exam.

____
Signature

GOOD LUCK!

Name: Solution

Problem 1. (20 points):

Part a. (5 points): Part of the state of the computer is as follows:

R3: x3000	Mem[x4000]: x1234
R4: x4000	Mem[x4001]: x2345
R5: x5000	Mem[x4002]: x3456
	Mem[x4003]: x4567

Then, LDR R5,R4,#2 is executed.

After this instruction is executed, R5 contains $\boxed{\text{x3456}}$

Part b. (5 points): The program below adds the absolute value of the integer in A to the absolute value of the integer in B, and stores the sum in C. We decide to use the subroutine ABS to take as input the contents of R0, and return its absolute value in R0.

```
        .ORIG x3000
        LD R0, A
        JSR ABS
        ADD R4, R0, #0
        LD R0, B
        JSR ABS
        ADD R0, R4, R0
        ST R0, C
        HALT
  A     .BLKW 1
  B     .BLKW 1
  C     .BLKW 1

 ABS    ADD R4, R0, #0
        BRzp DONE
SKIP    NOT R4, R4
        ADD R0, R4, #1
DONE    RET
        .END
```

Why will the above program not work correctly? Please answer in 20 words or fewer.

The subroutine modifies R4 without saving/restoring it R4 is used by the callee.

Name: Solution

Part c. (5 points): The following program is assembled, loaded into LC-3 memory, and executed.

```
       .ORIG x3000
       LD R0, A            ; R0 ← xF000
       LD R1, B            ; R1 ← x0025
       ADD R0, R1, R0      ; R0 ← xF025
       ST R0, B
A      .STRINGZ "%"
B      .FILL xF000  → x25
       .END
```

A { % → x0025 (NOP)
  { NULL → x0000 (NOP)

Does the program halt? If yes, explain what causes the program to halt. If no, explain why the program doesn't halt. Please answer in 20 words or fewer.

Halt instruction xF025 gets stored at label B which is executed since characters in A are "not taken" branches.

Part d. (5 points): Create the Symbol Table for this piece of code that an Aggie wrote one night when he was drunk.

```
        .ORIG x4000
        LEA   R1, X
AGAIN   ADD   R2, R1, R1
        ST    R2, X
        ADD   R2, R1, R1
        ST    R2, Y
        BRz   AGAIN
        HALT
PROMPT  .STRINGZ "EE306 ROCKS!"
     X  .BLKW 10
     Y  .BLKW 1
     Z  .FILL xAE00
        .END
```

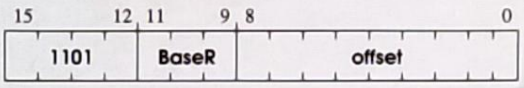
NULL Character

string length = 12 + 1 = 13

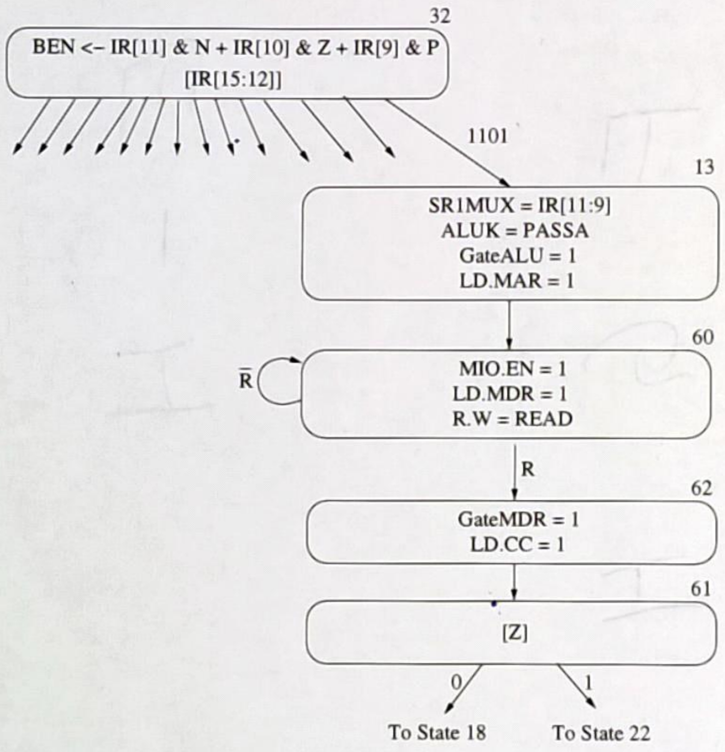
Symbol	Address
AGAIN	x4001
PROMPT	x4007
X	x4014
Y	x401E
Z	x401F

Name: Solution

Problem 2. (15 points): We want to add a new instruction to the LC-3, using the unused opcode 1101. It will have the following format:



To implement this instruction we add four new states, shown below.



We show in each state the control signals that are needed to implement the processing for that clock cycle. All control signals not shown in a state are assumed to be 0.

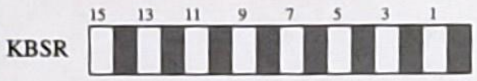
Note that from state 61, we branch either to state 18 or state 22.

What does this new instruction do? Be concise, but complete in your answer.

It branches to the instruction at PC (incremented) + 9-bit offset if the content of the memory location pointed by the base register is zero.

Name: Solution

Problem 3. (15 points): We want to support 8 input keyboards instead of 1. To do this we need 8 ready bits in KBSR, and 8 separate KBDRs. We will use the 8 odd-numbered bits in the KBSR as ready bits for the 8 keyboards, as shown below. We will set the other 8 bits in the KBSR to 0.



The 8 memory-mapped keyboard data registers and their corresponding ready bits are as follows:

FE04:	KBSR	
FE06:	KBDR1,	Ready bit is KBSR[1]
FE08:	KBDR2,	Ready bit is KBSR[3]
FE0A:	KBDR3,	Ready bit is KBSR[5]
FE0C:	KBDR4,	Ready bit is KBSR[7]
FE0E:	KBDR5,	Ready bit is KBSR[9]
FE10:	KBDR6,	Ready bit is KBSR[11]
FE12:	KBDR7,	Ready bit is KBSR[13]
FE14:	KBDR8,	Ready bit is KBSR[15]

We wish to write a program that polls the keyboards and loads the ASCII code typed by the highest priority keyboard into R0. That is, if someone had previously typed a key on keyboard 1, we want to load the ASCII code in KBDR1 into R0. If no key was typed on keyboard 1, but a key had been typed on keyboard 2, we want to load the ASCII code in KBDR2 into R0. ...and so on. That is, KB1 has higher priority than KB2, which has higher priority than KB3, which has higher priority than KB4, etc. KB8 has the lowest priority.

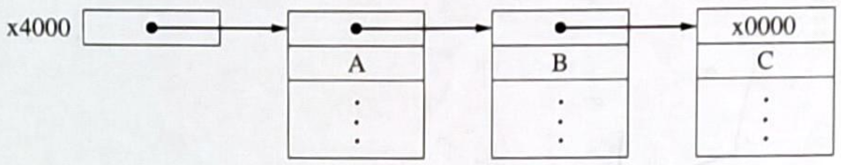
The following program will do the job AFTER you fill in the missing instructions:

```
        .ORIG X3000
        LD    R0, KBDR1
POLL    LDI   R1, KBSR
        BRz   POLL
        AND   R2, R2, #0
        ADD   R2, R2, #2
AGAIN   AND R3,R1,R2            } mask a bit
        BRnp     FOUND
        ADD      R0, R0, #2
        ADD R2,R2,R2            } shift R2 to left twice
        ADD R2,R2,R2
        BRnp     AGAIN
        HALT
FOUND   LDR R0,R0,#0            } load KBDR which is ready
        HALT
KBSR    .FILL    xFE04
KBDR1   .FILL    xFE06
        .END
```

Your job: fill in the missing instructions.

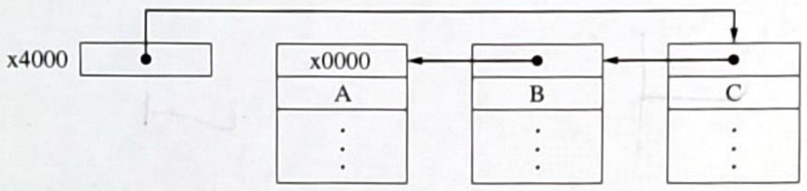
Name: Solution

Problem 4. (25 points):
You are given a linked list, consisting of at most 20 elements, as shown below.



Note the listhead is at location x4000.

We want to reverse the nodes of the linked list. For the above linked list, the result would be:



The program on the following page (with missing instructions filled in) does the job, using subroutines PUSH and POP.

Your job: fill in the missing instructions.

Name: Solution

```
        .ORIG X3000
        LEA   R6, BASE
        LD    R0, START

PHASE1  LDR   R0, R0, #0
        BRz PHASE2                 Branch out after reaching
                                   to pointer x0000
        JSR   PUSH
        BRnzp PHASE1

PHASE2  LD    R1, START

AGAIN   JSR   POP

        ADD R5, R5, #0             check success

        BRnp  DONE

add a node →     STR R0, R1, #0
go to next node → ADD R1, R0, #0   or LDR R1, R1, #0

        BRnzp AGAIN

DONE    AND   R0, R0, #0

        STR R0, R1, #0             ← last node should
                                     have pointer of x0000
        HALT

START   .FILL x4000
STACK   .BLKW #20

BASE .FILL -BASE     or X-3025    or x CFDB

PUSH    ADD R6, R6, #-1

        STR   R0, R6, #0
        RET

POP     AND   R5, R5, #0
        LD    R0, BASE
        ADD   R0, R0, R6
        BRz   EMPTY
        LDR   R0, R6, #0
        ADD   R6, R6, #1
        RET
EMPTY   ADD   R5, R5, #1
        RET

        .END
```

Name: Solution

Problem 5. (25 points): Consider the following program:

```
         .ORIG x3000
         LD  R0, A
         LD  R1, B
AGAIN    BRz DONE  ←  not taken on 1st iteration, taken on 2nd
         ADD R0,R0,R0     ; R0 ← x1800 [ x1800 = x0C00 + x0C00]
         ADD R1, R1, #-1  ; needs to result in 0
         BRnzp AGAIN
DONE     ST  R0, A
         HALT
A .FILL x0C00
~~A .FILL x0~~
   B     .FILL x0001
         .END
```

The program uses only R0 and R1. Note the boxes to indicate two missing instructions. Note also that one of the instructions in the program must be labeled AGAIN and that label is missing.

After execution of the program, the contents of A is x1800.

PROBLEM IS CONTINUED ON THE NEXT PAGE!!!

Name:____

During execution, we examined the computer during each clock cycle, and recorded some information for certain clock cycles, producing the table shown below. The table is ordered by the cycle number in which the information was collected. Note that each memory access takes 5 clock cycles.

number of cycles to execute two LDs

$30+9=39$

$57-39=18$
$=2\times 9$
Each blank instruction takes 9 cycles to execute

BR opcode ⇒ must correspond to BRz instruction

Cycle Number	State Number	Control Signals
1	18	LD.MAR: 1 · LD.REG: 0 · GateMDR: 0 · LD.PC: 1 · PCMUX: PC+1 · GatePC: 1
39	0	LD.MAR: 0 · LD.REG: 0 · BEN 0 · LD.PC: 0 · LD.CC: 0
48	1	LD.REG: 1 · DR: 000 · GateMDR: 0 · GateALU: 1 · GateMARMUX: 0
57	1	LD.MAR: 0 · ALUK: ADD · GateALU: 1 · LD.REG: 1 · DR: 001 · GatePC: 0
77	22	ADDR1MUX: PC · ADDR2MUX: PCoffset9 · LD.PC: 1 · LD.MAR 0 · PCMUX: ADDER
101	15	

ADD opcode
⇒ Second blank instruction must be ADD

Branch taken

Part a: Fill in the missing instructions in the program, and complete the program by labeling the appropriate instruction AGAIN. Also, fill in the missing information in the table.

Part b: Given values for A and B, what does the program do?

Left shifts A, B times

Execution of "BRnzp AGAIN" starts at Cycle 58 and ends at Cycle 67
Since another branch is taken at cycle 77, label AGAIN must correspond to "BRz DONE". The condition codes for these branches are set by the second blank instruction.