ECE199JL Final Exam, Fall 2012
Tuesday 18 December

Name and UIUC Net ID:

SOLUTION

- Be sure that your exam booklet has 13 pages.
- Write your name at the top of each page.
- This is a closed book exam.
- We have included a scratch sheet and two LC-3 reference pages.
- Appendix A of the textbook is available to you on request.
- You are allowed FOUR 8.5 × 11" sheets of notes.
- Absolutely no interaction between students is allowed.
- Show all of your work.
- Challenge questions are marked with ***.
- Don’t panic, and good luck!

“I think there is a world market for maybe five computers.”
—Thomas Watson (Chairman of IBM), 1943

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>10 points</td>
<td>____________</td>
</tr>
<tr>
<td>Problem 2</td>
<td>15 points</td>
<td>____________</td>
</tr>
<tr>
<td>Problem 3</td>
<td>15 points</td>
<td>____________</td>
</tr>
<tr>
<td>Problem 4</td>
<td>15 points</td>
<td>____________</td>
</tr>
<tr>
<td>Problem 5</td>
<td>25 points</td>
<td>____________</td>
</tr>
<tr>
<td>Problem 6</td>
<td>10 points</td>
<td>____________</td>
</tr>
<tr>
<td>Problem 7</td>
<td>10 points</td>
<td>____________</td>
</tr>
<tr>
<td>Total</td>
<td>100 points</td>
<td>____________</td>
</tr>
</tbody>
</table>
Problem 1 (10 points): Representations

Part A (3 points): Explain why an $N$-bit signed magnitude representation allows you to represent only $2^N - 1$ different numbers.

The number 0 has two representations, sign bit 0 or 1, magnitude 0, so one fewer than $2^N$ numbers are represented.

Part B (4 points): Two $N$-bit 2’s complement numbers, $A$ and $B$, are added to find their sum $S$, as shown to the right.

Write a Boolean expression for the overflow condition for the addition in terms of the variables shown.

$$A_{N-1}A_{N-2}...A_2A_1A_0 + B_{N-1}B_{N-2}...B_2B_1B_0 = S_{N-1}S_{N-2}...S_2S_1S_0$$

$$A_{N-1}B_{N-1}S_{N-1} + A_{N-1}B_{N-1}S_{N-1}$$

Part C (3 points): As you know, addition of two IEEE single-precision floating-point numbers is not associative. In other words, for some values of $A$, $B$, and $C$,

$$(A + B) + C \neq A + (B + C)$$

Give an example of values for $A$, $B$, and $C$ for which this lack of associativity holds (write decimal numbers or scientific notation—you need not translate to the binary representation for IEEE floating-point!).

A
B 1.2e-20
C 1x10^-40
Problem 2 (15 points): Logic

The block diagram below illustrates a specialized $N$-bit unsigned comparator. The comparator operates on two unsigned numbers, $A$ and $B$, to produce outputs $P_N$ and $Q_N$ with meanings defined in the table to the right.

<table>
<thead>
<tr>
<th>$P_N$</th>
<th>$Q_N$</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$A &lt; B$ and both $A$ and $B$ are odd</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$A \geq B$ and both $A$ and $B$ are odd</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$A &lt; B$ and $(A$ and $B$ are not both odd)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$A \geq B$ and $(A$ and $B$ are not both odd)</td>
</tr>
</tbody>
</table>

Part A (5 points): Design the even/odd detector. Show all work, including drawing a gate-level diagram implementing outputs $P$ and $Q$ in terms of inputs $A$ and $B$.

Part B (10 points): Design the general bit slice for this comparator. Show all work, including drawing a gate-level diagram implementing outputs $P$ and $Q$ in terms of inputs $A$, $B$, $R$, and $S$. 
Problem 3 (15 points): Finite State Machines

Part A (5 points): Professor Lumetta promised Professor Cangellaris to design a holiday light display for the new ECE building, but Lumetta has been too busy writing exam problems!

The design to the right shows what he needs: combinational logic that translates the output of a binary counter (counts upward) into RGB signals according to the following repeating sequence in the table below.

The RST input to the counter forces it back to 000 in the following cycle.

Design the logic needed to compute the RGB signals given the state $S_2S_1S_0$ of the counter. Use a few gates along with the decoder shown to the right to implement the functions $R$, $G$, and $B$ as described by the table above.

<table>
<thead>
<tr>
<th>color</th>
<th>RGB</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED</td>
<td>100</td>
</tr>
<tr>
<td>YELLOW</td>
<td>110</td>
</tr>
<tr>
<td>GREEN</td>
<td>010</td>
</tr>
<tr>
<td>BLUE</td>
<td>001</td>
</tr>
<tr>
<td>PURPLE</td>
<td>101</td>
</tr>
</tbody>
</table>

Part B (10 points): Draw an abstract transition diagram for a sequence recognizer that identifies the following sequences: 110, 0110, and 1100. In particular, the output $R$ of the sequencer should be 1 whenever the input $B$ has seen any of those three sequences in the last cycles.

Use as few states as possible, explain the meaning of your states, and be sure to specify the starting state.

Note that your diagram states should be labeled with names and output bit, but not with internal state bits (you do not need to pick a representation), but the arcs should be labeled with input combinations.

0110 implies 110, so only need to consider 110 and 1100
Problem 4*** (15 points): Machine Code Analysis

An LC-3 program is located in memory location x3000 to x3007.

The program starts executing at x3000. If we keep track of all values loaded into the MAR as the program executes, we obtain the sequence shown to the right. Such a sequence of values is referred to as a trace.

Fill in the table below with the bits stored in locations x3000 to x3007, then translate the bits to assembly code (fill in the blanks at the bottom of the page).

Some of the bits in the table have been filled in already—use these to deduce the values of the others such that the resulting program leads to the MAR trace shown to the right.

You will need some additional information:
- All registers contain x0000 when the program starts.
- Data stored in location x4FF8 and x5000 are x2012.
- HALT is TRAP x25.

<table>
<thead>
<tr>
<th>MAR trace</th>
<th>first value in MAR</th>
<th>second value in MAR</th>
<th>third value in MAR</th>
<th>fourth value in MAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>x3000</td>
<td>x3000</td>
<td>x3007</td>
<td>x3001</td>
<td>...</td>
</tr>
<tr>
<td>x3002</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x3003</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x3004</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x3005</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x3006</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x3007</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Translate the bits to LC-3 assembly code (not RTL) with numeric operands. Do not use labels.

- x3000: \( \text{LD R0, #6} \)
- x3001: \( \text{JSR } #1 \)
- x3002: \( \text{HALT} \)
- x3003: \( \text{LDI } R1, #3 \) / \( \text{SRI } R1, #3 \)
- x3004: \( \text{LOR } R2, R0, #8 \)
- x3005: \( \text{ADD } R2, R2, #1 \) / \( \text{ADD } R1 \)
- x3006: \( \text{JMP R7} / \text{RET} \)
- x3007: \( \text{FILL x5000} \) (or \( \text{AND } R0, R0, R0 \))
Problem 5 (25 points): Assembly Code

Part A (5 points): The LC-3 assembler produces the symbol table below when assembling the code shown to the right. Fill in the blank entries.

// Symbol table
// Scope level 0:
// Symbol Name       Address     
// ------------------ -------
// MAIN               x3000     
// LD1               x3001     
// ST1               x3006     
// LD2               x3007     
// ST2               x300C     
// INIT               x300F     
// FOO               x3015     
// CALC               x301B     
// NEXT               x3020     
// DONE               x3024     
// RETN               x302A     
// OP1               x302B     
// OP2               x302C     
// EXT                x302D     
// FLAG               x302E     
// NAME               x302F     
// SIGN               x3039     
// MASK               x303A

.ORIG x3000

MAIN AND R3, R3, #0
LD R1, OP1
LD R4, SIGN
AND R5, R1, R4
BRz ST1
JSR FOO
ST1 ADD R2, R1, #0
LD R1, OP2
LD R4, SIGN
AND R5, R1, R4
BRz ST2
JSR FOO
ST2 ADD R1, R1, #0
AND R3, R3, #1
ST R3, FLAG
INIT AND R4, R4, #1
AND R5, R5, #0
ADD R5, R5, #1
AND R6, R6, #0
JSR CALC
HALT

FOO ADD R3, R3, #1
LD R4, EXT
ADD R1, R1, R4
NOT R1, R1
ADD R1, R1, #1
RET

CALC ADD R3, R4, #0
BRzp DONE
AND R3, R5, R2
BRz NEXT
ADD R6, R1, R6
NEXT ADD R5, R5, R5
ADD R1, R1, R1
ADD R4, R4, #1
BRnzp CALC
DONE LD R3, FLAG
BRz RETN
NOT R6, R6
ADD R6, R6, #1
LD R4, MASK
AND R6, R6, R4
RET

OP1 .FILL x0008
OP2 .FILL x0007
EXT .FILL x0000
FLAG .BLKW #1
NAME .STRINGZ "Read this"
SIGN .FILL x0080
MASK .FILL x4FF

.END
Problem 5, continued:

**Part B (10 points):** The following LC-3 program determines whether or not two strings match (that is, whether or not they have identical contents). The first string starts at memory location x4000, and the second string starts at memory location x5000. Both strings are in the .STRINGZ format. If the two strings are the same, the program terminates with a 1 in R6. If the two strings are different, the program terminates with a 0 in R6. Write one LC-3 assembly instruction into each blank to complete the program. *You should not need to define any new labels.*

```
.ORIG x3000
LD R1, STRING1
JSR LENGTH
ADD R4, R0, #0 or R3 ; part A
LD R1, STRING2
JSR LENGTH
ADD R3, R0, #0 any matching value is ok
NOT R4, R4
ADD R4, R4, #1
ADD R4, R4, R3
Bرنp NO
LD R1, STRING1
LD R2, STRING2
LDR R3, R1, #0
LDR R4, R2, #0
Bرز YES ; part C
NOT R4, R4
ADD R4, R4, #1
ADD R4, R4, R3
Bرنp NO
ADD R1, R1, #1 ; part D
ADD R2, R2, #1
Bرنzp CONTINUE
YES AND R5, R6, #0
ADD R5, R6, #1
Bرنzp DONE
NO AND R6, R6, #0
DONE HALT

; a subroutine
LENGTH AND R0, R0, #0
COUNT LDR R5, R1, #0
Bرز RETURN
ADD R0, R0, #1
ADD R1, R1, #1
Bرنzp COUNT
RETURN RET
STRING1 .FILL x4000
STRING2 .FILL x5000
.END
```
Problem 5, continued:

Part C (10 points): Write a program in LC-3 assembly language that computes $RESULT = |A - 4|$. The $| \cdot |$ notation means “absolute value.” Your program must have the following characteristics:

- The program must start at memory address x2800.
- The values $A$ and $RESULT$ must be placed at the two memory addresses that immediately follow the last instruction in the program. These two addresses must be labeled $A$ and $RESULT$, respectively.
- The value $A$ must be initialized to 3.
- The program must produce the correct result for any initial value of $A$ in the range [-1000, 1000].
- The program must load the value of $A$, and store the correct $RESULT$, from/to the labeled memory locations.
- The program must execute HALT upon completion of this task.
- Appropriately comment your program so that the grader can understand your intent.

```
.ORIG x2800
LDR RO, A
ADD RO, RO, # -4
BREP DONE
NOT RO, RO
ADD RO, RO, # 1
DONE: ST RO, RESULT
HALT
A .FILL #3
RESULT .BLKW #1
.END
```
Problem 6 (10 points): LC-3 Implementation

Attached to the back of this exam is a copy of the LC-3 state machine (reproduced from the textbook). Tear it off for use with this problem.

Fill in the table below with the appropriate state numbers from that diagram for the ordered sequence of states that are active during the processing of an LC-3 LDR instruction. Note: you may not need all rows of the table below.

Next, for each state, indicate whether each control signal is active (1) or inactive (0). For your convenience, the LC-3 datapath is reproduced below (again from the textbook). DO NOT LEAVE BLANK ENTRIES.

<table>
<thead>
<tr>
<th>State #</th>
<th>Gate.PC</th>
<th>LD.PC</th>
<th>LD.IR</th>
<th>LD.CC</th>
<th>LD.MAR</th>
<th>LD.MDR</th>
<th>GateMDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Problem 7 (10 points): Critical Paths and Control Unit Design

Part A (3 points): Explain why the critical path through a tree-structured adder such as a Kogge-Stone adder is typically shorter than the critical path through a ripple carry adder.

Tree structure balances delay among all paths by breaking repeatedly into halves (or any size, really), so proportional to \( \log_2(\# \text{ of bits}) \).

Ripple carry delay along carry chain is proportional to \# of bits.

Part B (4 points): Explain how a memory can be used to implement \( N \) Boolean logic functions on \( M \) variables. Be specific about the size of memory needed. (*You may want to draw a picture.*)

![Diagram of memory implementation](attachment:image.png)

Part C (3 points): What is a microinstruction?

A set of control signals that implement the RTL for a particular state in a processor's state machine.
This page provided as scratch paper. If you need us to look at this page when grading, indicate this need on the page of the corresponding problem (not here!).
NOTES: RTL corresponds to execution (after fetch!); JSRR not shown

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Binary Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>0001 DR SR1 0 00 SR2</td>
<td>ADD DR, SR1, SR2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR ← SR1 + SR2, Setcc</td>
</tr>
<tr>
<td>ADD</td>
<td>0001 DR SR1 1 imm5</td>
<td>ADD DR, SR1, imm5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR ← SR1 + SEXT(imm5), Setcc</td>
</tr>
<tr>
<td>AND</td>
<td>0101 DR SR1 0 00 SR2</td>
<td>AND DR, SR1, SR2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR ← SR1 AND SR2, Setcc</td>
</tr>
<tr>
<td>AND</td>
<td>0101 DR SR1 1 imm5</td>
<td>AND DR, SR1, imm5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR ← SR1 AND SEXT(imm5), Setcc</td>
</tr>
<tr>
<td>BR</td>
<td>0000 n z p PCoffset9</td>
<td>BR{nzp} PCoffset9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>((n AND N) OR (z AND Z) OR (p AND P)): PC ← PC + SEXT(PCoffset9)</td>
</tr>
<tr>
<td>JMP</td>
<td>1100 000 BaseR 000000</td>
<td>JMP BaseR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PC ← BaseR</td>
</tr>
<tr>
<td>JSR</td>
<td>0100 1 PCoffset11</td>
<td>JSR PCoffset11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R7 ← PC, PC ← PC + SEXT(PCoffset11)</td>
</tr>
<tr>
<td>TRAP</td>
<td>1111 0000 trapvect8</td>
<td>TRAP trapvect8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R7 ← PC, PC ← M[ZEXT(trapvect8)]</td>
</tr>
<tr>
<td>LD</td>
<td>0010 DR PCoffset9</td>
<td>LD DR, PCoffset9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR ← M[PC + SEXT(PCoffset9)], Setcc</td>
</tr>
<tr>
<td>LDI</td>
<td>1010 DR PCoffset9</td>
<td>LDI DR, PCoffset9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR ← M[PC + SEXT(PCoffset9)], Setcc</td>
</tr>
<tr>
<td>LDR</td>
<td>0110 DR BaseR offset6</td>
<td>LDR DR, BaseR, offset6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR ← M[BaseR + SEXT(offset6)], Setcc</td>
</tr>
<tr>
<td>LEA</td>
<td>1110 DR PCoffset9</td>
<td>LEA DR, PCoffset9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR ← PC + SEXT(PCoffset9), Setcc</td>
</tr>
<tr>
<td>NOT</td>
<td>1001 DR SR 111111</td>
<td>NOT DR, SR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR ← NOT SR, Setcc</td>
</tr>
<tr>
<td>ST</td>
<td>0011 SR PCoffset9</td>
<td>ST SR, PCoffset9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M[PC + SEXT(PCoffset9)] ← SR</td>
</tr>
<tr>
<td>STI</td>
<td>1011 SR PCoffset9</td>
<td>STI SR, PCoffset9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M[M[PC + SEXT(PCoffset9)]] ← SR</td>
</tr>
<tr>
<td>STR</td>
<td>0111 SR BaseR offset6</td>
<td>STR SR, BaseR, offset6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M[BaseR + SEXT(offset6)] ← SR</td>
</tr>
</tbody>
</table>
For use with Problem 6.