CS/ECE 252: INTRODUCTION TO COMPUTER ENGINEERING

UNIVERSITY OF WISCONSIN—MADISON

Prof. Gurindar Sohi

TAs: Junaid Khalid and Pradip Vallathol

Examination 4

In Class (50 minutes)

Wednesday, December 12, 2012

Weight: 17.5%

NO: BOOK(S), NOTE(S), OR CALCULATORS OF ANY SORT.

The exam has nine pages. **Circle your final answers**. Plan your time carefully since some problems are longer than others. You **must turn in the pages 1-7**.

LAST NAME: ____

FIRST NAME: ____

ID# ____

Problem	Maximum Points	Points Earned
1	6	
2	6	
3	6	
4	3	
5	3	
6	6	
Total	30	

Problem 1: Assembly Language

(a) Briefly explain the four assembly errors in the following LC-3 program.

(4 Points)

```
            .ORIG x3000
            LD  R2, INPUT
            AND R0, R0, #0
            ADD R1, R0, #1
            BR NEXT

LOOP        AND R4, R2, R1
            BRz SKIP
            NOR R0, R0, #1

NEXT        ADD R1, R1, R1
            ADD R3, R3, x2A
            LD  R6, SKIP
            NOT R6, R6
            BRzp LOOP

INPUT       .FILL x1997
NEXT        .FILL x1998
    .END
```

i. Label SKIP is not declared.

ii. Duplicate label NEXT

iii. x2A cannot be represented as a signed number in 5 bits

iv. NOR is an undefined instruction

(b) Which of the following (if any) of the following pseudo-ops can be used multiple times in a single assembly file. Circle all options that apply. **(2 Points)**

i. .FILL

ii. .ORIG

iii. .BLKW

iv. .END

v. .STRINGZ

Problem 2: Two pass Assembly Process

An LC-3 assembly language program in given below:

```
          .ORIG x3000
          AND R3, R3, #0
          LD  R0, M0
          LD  R1, M1
          LD  R2, M2

          ADD R3, R3, R2
LOOP      ADD R3, R3, #1
          ADD R0, R0, #-1
          BRn LOOP

DONE      ST  R3, RESULT
          HALT
RESULT    .FILL x0000
M0        .BLKW #7
M1        .STRINGZ "Section-3"
M2        .FILL x0009
          .END
```

(a) A symbol table is created during the first pass by the assembler. Fill in the following symbol table for the above program: **(4 Points)**

Symbol	Address
LOOP	x3005
DONE	x3008
RESULT	x300A
M0	x300B
M1	x3012
M2	x301C

(b) The assembly program is converted into a binary file during the second pass by the assembler. Fill in the binary instructions at the following memory locations: **(2 Points)**

Address	**Instructions**
x3007	0000 1001 1111 1101
x3008	0011 0110 0000 0001

Problem 3: Traps and Subroutines **(6 Points)**

The following LC-3 assembly program takes a single character as input from the user. If the input character is a digit (0-9), it prints the message "Is a digit" on the display. This process is continued until the user enters the termination character '#', and the program halts. Fill in the missing parts of the program indicated by ________.

```
            .ORIG x3000

GETINPUT    TRAP x20            ; Input a character from the user
                                ; (Do not echo it on the display)

            LD  R1, TERMCHAR    ; termination check
            ADD R1, R0, R1
            BRz END             ; Branch to END on '#'

            JSR CHECKINPUT      ; Call CHECKINPUT subroutine
            BR  GETINPUT

END         HALT

CHECKINPUT
            ST R7, SAVELOC      ; Save something here
            LD  R2, DIGIT0
            ADD R2, R0, R2
            BRn RELOAD

            LD  R2, DIGIT9
            ADD R2, R0, R2
            BRp RELOAD

DISP_IS     LEA R0, STR_IS      ; print a string
            TRAP x22            ; to the display

RELOAD      LD R7, SAVELOC      ; Load something here

            RET

; Data
SAVELOC     .BLKW       #1
STR_IS      .STRINGZ    "Is a digit\n"
STR_NOT     .STRINGZ    "Not a Digit\n"
TERMCHAR    .FILL       0xFFDD      ; negative ASCII value of '#'
DIGIT0      .FILL       0xFFD0      ; negative ASCII value of '0'
DIGIT9      .FILL       0xFFC7      ; negative ASCII value of '9'

            .END
```

Problem 4: I/O

a) Briefly explain the difference between *interrupt-driven I/O* and *polling based I/O*? **(2 Points)**

Polling: CPU keeps checking status register until new data arrives or device ready for new data.

Interrupt: Device sends a special signal to CPU when new data arrives or device ready for next data.

b) What is the main reason to prefer asynchronous I/O over synchronous I/O in recent microprocessor designs? **(1 Point)**

I/O devices usually operate at speeds very different from that of a microprocessor. The rate at which data is provided or consumed is not predictable and usually not in lockstep with the processor clock.

Problem 5: Trap Handling **(3 Points)**

List the main steps of the TRAP mechanism involved in executing the instruction **`TRAP x86`**.

a. Lookup the starting address of the service routine to execute in the Trap Vector table at location 0x86.

b. Transfer control to service routine (Set PC to contents of the memory location 0x86). Save return address in R7.

c. Return from service routine (JMP R7).

Problem 6: Short Answer Questions

Answer the flowing questions briefly.

a) How many trap service routines can be defined in LC-3? **(1 Point)**

256

b) Explain briefly the problem that the *callee-save* and the *caller-save* approaches are trying to solve. **(2 Point)**

If a register value is "destroyed" by actions of a subroutine or service routine, the value has to be saved before it is modified, and reloaded before it is used again.

c) What important feature does the instruction **JSRR** provide that **JSR** does not? **(1 Point)**

JSRR uses the contents a register as the address to jump to (16 bits), while JSR instruction provides an 11 bit offset to PC. Thus the range of addresses to which a JSRR instruction can jump to is larger than that of the JSR instruction.

d) What happens during the **linking** phase of an assembly program? **(1 Point)**

Linking is the process of resolving symbols between independent object files. The linker will search symbol tables of other modules to resolve symbols and complete code generation before loading.

e) What is the use of *Comments* in an assembly program? **(1 Point)**

Comments are useful to humans to document or understand programs. They are ignored by the assembler.

Extra page for hand written work, if needed. This page is not required and will NOT affect your grade. You don't even need to hand this page in.

ASCII Table

<table>
<tr><th>Character</th><th>Hex</th><th>Character</th><th>Hex</th><th>Character</th><th>Hex</th><th>Character</th><th>Hex</th></tr>
<tr><td>nul</td><td>00</td><td>sp</td><td>20</td><td>@</td><td>40</td><td>`</td><td>60</td></tr>
<tr><td>soh</td><td>01</td><td>!</td><td>21</td><td>A</td><td>41</td><td>a</td><td>61</td></tr>
<tr><td>stx</td><td>02</td><td>“</td><td>22</td><td>B</td><td>42</td><td>b</td><td>62</td></tr>
<tr><td>etx</td><td>03</td><td>#</td><td>23</td><td>C</td><td>43</td><td>c</td><td>63</td></tr>
<tr><td>eot</td><td>04</td><td>$</td><td>24</td><td>D</td><td>44</td><td>d</td><td>64</td></tr>
<tr><td>enq</td><td>05</td><td>%</td><td>25</td><td>E</td><td>45</td><td>e</td><td>65</td></tr>
<tr><td>ack</td><td>06</td><td>&</td><td>26</td><td>F</td><td>46</td><td>f</td><td>66</td></tr>
<tr><td>bel</td><td>07</td><td>‘ (Apostr.)</td><td>27</td><td>G</td><td>47</td><td>g</td><td>67</td></tr>
<tr><td>bs</td><td>08</td><td>(</td><td>28</td><td>H</td><td>48</td><td>h</td><td>68</td></tr>
<tr><td>ht</td><td>09</td><td>)</td><td>29</td><td>I</td><td>49</td><td>i</td><td>69</td></tr>
<tr><td>lf</td><td>0A</td><td>*</td><td>2A</td><td>J</td><td>4A</td><td>j</td><td>6A</td></tr>
<tr><td>vt</td><td>0B</td><td>+</td><td>2B</td><td>K</td><td>4B</td><td>k</td><td>6B</td></tr>
<tr><td>ff</td><td>0C</td><td>, (Comma)</td><td>2C</td><td>L</td><td>4C</td><td>l</td><td>6C</td></tr>
<tr><td>cr</td><td>0D</td><td>-</td><td>2D</td><td>M</td><td>4D</td><td>m</td><td>6D</td></tr>
<tr><td>so</td><td>0E</td><td>. (Period)</td><td>2E</td><td>N</td><td>4E</td><td>n</td><td>6E</td></tr>
<tr><td>si</td><td>0F</td><td>/</td><td>2F</td><td>O</td><td>4F</td><td>o</td><td>6F</td></tr>
<tr><td>dle</td><td>10</td><td>0</td><td>30</td><td>P</td><td>50</td><td>p</td><td>70</td></tr>
<tr><td>dc1</td><td>11</td><td>1</td><td>31</td><td>Q</td><td>51</td><td>q</td><td>71</td></tr>
<tr><td>dc2</td><td>12</td><td>2</td><td>32</td><td>R</td><td>52</td><td>r</td><td>72</td></tr>
<tr><td>dc3</td><td>13</td><td>3</td><td>33</td><td>S</td><td>53</td><td>s</td><td>73</td></tr>
<tr><td>dc4</td><td>14</td><td>4</td><td>34</td><td>T</td><td>54</td><td>t</td><td>74</td></tr>
<tr><td>nak</td><td>15</td><td>5</td><td>35</td><td>U</td><td>55</td><td>u</td><td>75</td></tr>
<tr><td>syn</td><td>16</td><td>6</td><td>36</td><td>V</td><td>56</td><td>v</td><td>76</td></tr>
<tr><td>etb</td><td>17</td><td>7</td><td>37</td><td>W</td><td>57</td><td>w</td><td>77</td></tr>
<tr><td>can</td><td>18</td><td>8</td><td>38</td><td>X</td><td>58</td><td>x</td><td>78</td></tr>
<tr><td>em</td><td>19</td><td>9</td><td>39</td><td>Y</td><td>59</td><td>y</td><td>79</td></tr>
<tr><td>sub</td><td>1A</td><td>:</td><td>3A</td><td>Z</td><td>5A</td><td>z</td><td>7A</td></tr>
<tr><td>esc</td><td>1B</td><td>;</td><td>3B</td><td>[</td><td>5B</td><td>{</td><td>7B</td></tr>
<tr><td>fs</td><td>1C</td><td><</td><td>3C</td><td>\</td><td>5C</td><td>|</td><td>7C</td></tr>
<tr><td>gs</td><td>1D</td><td>=</td><td>3D</td><td>]</td><td>5D</td><td>}</td><td>7D</td></tr>
<tr><td>rs</td><td>1E</td><td>></td><td>3E</td><td>^</td><td>5E</td><td>~</td><td>7E</td></tr>
<tr><td>us</td><td>1F</td><td>?</td><td>3F</td><td>_ (Undrscre)</td><td>5F</td><td>del</td><td>7F</td></tr>
</table>

LC-3 Instruction Set (Entered by Mark D. Hill on 03/14/2007; last update 03/15/2007)

PC′: incremented PC. setcc(): set condition codes N, Z, and P. mem[A]:memory contents at address A.
SEXT(immediate): sign-extend immediate to 16 bits. ZEXT(immediate): zero-extend immediate to 16 bits.

```
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ADD DR, SR1, SR2 ; Addition
| 0   0   0   1 |    DR     |    SR1    | 0 | 0   0 |    SR2    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← SR1 + SR2 also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ADD DR, SR1, imm5 ; Addition with Immediate
| 0   0   0   1 |    DR     |    SR1    | 1 |       imm5        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← SR1 + SEXT(imm5) also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ AND  DR, SR1, SR2 ; Bit-wise AND
| 0   1   0   1 |    DR     |    SR1    | 0 | 0   0 |    SR2    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← SR1 AND SR2 also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ AND DR,SR1,imm5 ; Bit-wise AND with Immediate
| 0   1   0   1 |    DR     |    SR1    | 1 |       imm5        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← SR1 AND SEXT(imm5) also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ BRx,label (where x={n,z,p,zp,np,nz,nzp}); Branch
| 0   0   0   0 | n | z | p |             PCoffset9             | GO ← ((n and N) OR (z AND Z) OR (p AND P))
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ if(GO is true) then PC←PC′+ SEXT(PCoffset9)

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ JMP BaseR ; Jump
| 1   1   0   0 | 0   0   0 |   BaseR   | 0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ PC ← BaseR

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ JSR label ; Jump to Subroutine
| 0   1   0   0 | 1 |                PCoffset11                 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ R7 ← PC′, PC ← PC′ + SEXT(PCoffset11)

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ JSRR BaseR ; Jump to Subroutine in Register
| 0   1   0   0 | 0 | 0   0 |   BaseR   | 0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ temp ← PC′, PC ← BaseR, R7 ← temp

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ LD DR, label ; Load PC-Relative
| 0   0   1   0 |    DR     |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← mem[PC′ + SEXT(PCoffset9)] also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ LDI DR, label ; Load Indirect
| 1   0   1   0 |    DR     |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR←mem[mem[PC′+SEXT(PCoffset9)]] also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ LDR DR, BaseR, offset6 ; Load Base+Offset
| 0   1   1   0 |    DR     |   BaseR   |        offset6        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← mem[BaseR + SEXT(offset6)] also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ LEA, DR, label ; Load Effective Address
| 1   1   1   0 |    DR     |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← PC′ + SEXT(PCoffset9) also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ NOT DR, SR ; Bit-wise Complement
| 1   0   0   1 |    DR     |    SR     | 1 | 1   1   1   1   1 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← NOT(SR) also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ RET ; Return from Subroutine
| 1   1   0   0 | 0   0   0 | 1   1   1 | 0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ PC ← R7

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ RTI ; Return from Interrupt
| 1   0   0   0 | 0   0   0   0   0   0   0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ See textbook (2nd Ed. page 537).

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ST SR, label ; Store PC-Relative
| 0   0   1   1 |    SR     |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ mem[PC′ + SEXT(PCoffset9)] ← SR

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ STI, SR, label ; Store Indirect
| 1   0   1   1 |    SR     |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ mem[mem[PC′ + SEXT(PCoffset9)]] ← SR

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ STR SR, BaseR, offset6 ; Store Base+Offset
| 0   1   1   1 |    SR     |   BaseR   |        offset6        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ mem[BaseR + SEXT(offset6)] ← SR

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ TRAP ; System Call
| 1   1   1   1 | 0   0   0   0 |           trapvect8           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ R7 ← PC′, PC ← mem[ZEXT(trapvect8)]

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ; Unused Opcode
| 1   1   0   1 |                                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ Initiate illegal opcode exception
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
```

TRAP CODES

Code	*Equivalent*	*Description*
HALT	TRAP x25	Halt execution and print message to console.
IN	TRAP x23	Print prompt on console, read (and echo) one character from keybd. Character stored in R0[7:0].
OUT	TRAP x21	Write one character (in R0[7:0]) to console.
GETC	TRAP x20	Read one character from keyboard. Character stored in R0[7:0].
PUTS	TRAP x22	Write null-terminated string to console. Address of string is in R0.