Homework 2 Solution

The purpose of this homework is to gain familiarity with the TMS320C54x assembly code syntax, the software environment used to develop and test executable code on this DSKplus board. Please refer to the Reference Manuals, the DSKplus User’s Guide and the lecture notes to answer the questions posed.

Attach separate pages of your work to this handout if necessary.

1. **Program Development.** Please complete the following table. State the programs you use and its purpose in this lab.

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editor</td>
<td>Use the text editor to create your .asm files for use with the assembler (dskplasm.exe).</td>
</tr>
<tr>
<td>Debugger</td>
<td>It is used to load and execute the object code on the DSP. It also provides many features, including register and memory read-outs, graphical view of memory locations, tracing, breakpoints, and more.</td>
</tr>
</tbody>
</table>

2. **Labels.** Cross out illegal labels in the following list.

- $this
- 23that (illegal label)
- add
- a1_$_2 (illegal label)
- abcd$ef

3. **Number Representations.** Please complete the following table in the appropriate number units.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hexadecimal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100100B</td>
<td>0x64</td>
<td>100</td>
</tr>
</tbody>
</table>

4. **Computer Arithmetic.** Please complete operations in the following table.
<table>
<thead>
<tr>
<th>Name</th>
<th>Operator(s)</th>
<th>Expression</th>
<th>Evaluates to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>+</td>
<td>~ 35</td>
<td>-36</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>~ (one’s complement)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arithmetic</td>
<td>+</td>
<td>8 % 5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left-shift</td>
<td>*</td>
<td>1 &lt;&lt; 7</td>
<td>128</td>
</tr>
<tr>
<td>Right-shift</td>
<td>&gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relational</td>
<td>&lt;</td>
<td>5 &gt;= 5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt;=</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>=</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>!=</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bitwise AND</td>
<td>&amp;</td>
<td>0x00FF &amp; 0x1234</td>
<td>0x0034</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bitwise OR</td>
<td></td>
<td>0x00FF</td>
<td>0x1234</td>
</tr>
<tr>
<td>Bitwise XOR</td>
<td>∧</td>
<td>0x00FF ∧ 0x1234</td>
<td>0x12CB</td>
</tr>
</tbody>
</table>

5. **Assembler Directives.** Assembler directives do not compile into executable code. They are compile-time commands for the compiler/assembler. Please complete the following table.

<table>
<thead>
<tr>
<th>Description</th>
<th>Assembler Directive</th>
<th>Usage Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin assembling source code into the <code>.data</code> section</td>
<td><code>.data</code></td>
<td><code>.data</code> .setsect &quot;.data&quot;, 0x800,1 .data .word 1,2,3</td>
</tr>
<tr>
<td>Begin assembling source code into the <code>.text</code> section (which usually contains executable code)</td>
<td><code>.text</code></td>
<td><code>.text</code> .setsect &quot;.text&quot;,0x600,0 .text label a = #5</td>
</tr>
<tr>
<td>Read source statement from a different file</td>
<td><code>.include</code></td>
<td>`.include &quot;myfile.asm&quot;</td>
</tr>
<tr>
<td></td>
<td><code>.copy</code></td>
<td>`.copy &quot;myfile.asm&quot;</td>
</tr>
</tbody>
</table>

6. **A Code Segment.** Assume all registers and memory locations are initialized to zeros, except for the data memory block from 0x800 to 0x809, which is the following:
After the following code segment is executed, what are the values in the memory block 0x800 to 0x809? Fill in these final values in the above table. Show all your work.

\[
\begin{align*}
AR1 &= \#0x800 \\
AR2 &= \#0x801 \\
AR3 &= \#7 \\
\text{loop A} &= \{AR1 + \}
\text{A} &= A + \{AR2 + \}
\{AR2 = A \}
\text{if( }\{AR3 - !\text{=} 0\} \text{ goto loop}
\end{align*}
\]

**Solution**: The code calculates Fibonacci numbers: \( f(n) = f(n - 1) + f(n - 2) \) with \( f(0) = 0 \) and \( f(1) = 1 \).

7. **DSKPlus Software.** How does one set breakpoints in the Code Explorer software? Which windows are shown in the monitor display when Code Explorer is running and can their attributes be changed (i.e. do you have access, through Code Explorer, to change register values, memory values, etc.?).

**Solution**: To set a breakpoint, double-click the line of code in the Dis-Assembly window. Double-click again to remove it. Program code, registers, and memory values can be displayed using Code Explorer. You can change values by double clicking the appropriate entry in the windows.

8. **Programming Exercise.** In Experiment A, the program sorts a list of numbers in descending order. Now write a program that sorts a list of numbers in ascending order. It is probably easier to modify Experiment A code than to write one from scratch. In the industry, it is a common practice to re-use code. Thus, try to re-use as much code as you can. There are multiple solutions; some short, others longer. Your program must compile without errors and execute correctly. Make sure there are comments next to new code or changes. Print out your complete program and attach it to this handout.
Solution Description

Recall that the algorithm of the program takes values from 'array'
array one-by-one and insert it at the appropriate place in the
'output' array. Because it inserts each value to 'output' array
starting from top-down (resulting in a descending sort), modifying
the program to start from down-top will result in an ascending
sort. The changes are highlighted by '*****' in the comments.

Alternate Solution (Discussion only)

There are many solutions to this assignment. Another way is to
treat the given code as a black box and write a "wrapper" around it
so that the resulting new box returns the desired result. One such
wrapper is to create another array and copy the elements of 'array'
(which is already sorted) from bottom up. This new array becomes
ascending because 'array' was descending.

*******************************************************************
.setsect *.text", 0x500,0 ;Executable code in *.text" 
;section will begin at 0x500 
;in program memory

.setsect *.data", 0x800,1 ;Numbers to be sorted will 
;begin at 0x800 in data memory

.data ;Data section begins

array .word 4,5,1,2,7,3,6,9 
output .word 0,0,0,0,0,0,0,0
output .word 0,0,0,0,0,0,0,0 ;***** 7 storage locations here
outlast .word 0 ;***** last storage here

.array' specifies the location of an array of numbers in data memory
space, where the first item is located at address 'array'. Only eight
numbers are sorted in this example, and the output area, symbolically
specified with label 'output' is initialized to all zero.

.text ;Executable code section begins.

count .set 7 ;Eight data points so seven plus one for a
;total of eight execution steps per loop.

AR0 = #count ;Registers AR0 and AR1 are being used as
;two counters for the two nested loops
AR2 = #array ;AR2 points to the input data location

loop1 AR1 = #count ;AR1 is reset at the beginning of the outer loop.

AR3 = #outlast ;***** AR3 points to last output location
A = *AR2+ << 16 ;Accumulator A loads first data point

A = A + *AR3 ;Bits 0-15 (LSB) of accumulator A is loaded with value from output list to compare and sort.

loop2 cmps(A, *AR3-) ;Main compare, select and store command

if (TC) goto low ;If LSB > MSB then keep LSB, shift it into MSB and load new LSB,
goto noshift ;else discard LSB, and reload new LSB

low A = A >> 16
noshift AR4 = A ;AR4 is temporary register during shifting of accumulator A contents

A = @AR4 << 16
A = A + *AR3
if (*AR1- != 0) goto loop2 ;check to see if loops have
if (*AR0- != 0) goto loop1 ;are complete and then halt.

stop nop
goto stop
.end