Sample Problem Set #1 - SOLUTIONS

Notes:
These problems are typical exam problems; most are drawn from previous homeworks and exams. This exam is open book, open notes. It may help to have a calculator.
For partial credit, please show all work, reasoning, and steps leading to solutions.

1. The meaning of what is stored in memory depends on your interpretation. Assume that memory locations $800$ and $801$ contain the machine codes for the instructions “COMA” and “INCA”. Give the meaning of these values in these locations if you interpret them as:

(a) ASCII characters

The values are $41$ and $42$. If interpreted as ASCII, these are the characters “A” and “B”.

(b) Unsigned 8-bit integers (i.e., give the decimal values)

These have the decimal values 65 and 66. (It would be the same if they were two’s complement integers)

2. Give the machine code corresponding to the following HCS12 assembly language program. Indicate the contents of memory at each address after the program is loaded into memory.

```assembly
ORG   $0D00
MAIN   LDX   #MAIN
STX   $3000
CPX   $10
BHS   HERE
INX
HERE   NOP
```

Solution:
- Opcodes are loaded into memory starting at $0D00.
- The value of the label MAIN is 0D00.
- The BHS instruction uses relative addressing for the access to HERE.
- The value of the label HERE is 0D0B.
- The program counter will point to 0D0A just before BHS executes
- So we need an offset of +1 decimal

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Assembly source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0D00</td>
<td>CE 0D 00</td>
<td>ORG $0D00</td>
</tr>
<tr>
<td>$0D03</td>
<td>7E 30 00</td>
<td>MAIN LDX #MAIN</td>
</tr>
<tr>
<td>$0D06</td>
<td>9E 10</td>
<td>STX $3000</td>
</tr>
<tr>
<td>$0D08</td>
<td>24 01</td>
<td>CPX $10</td>
</tr>
<tr>
<td>$0D0A</td>
<td>08</td>
<td>BHS HERE</td>
</tr>
<tr>
<td>$0D0B</td>
<td>A7</td>
<td>INX</td>
</tr>
</tbody>
</table>

Starting from $0D00$, memory contains $CE$, $0D$, $00$, $7E$, etc as shown above.
3. Fill in the blanks below, indicating the address and the contents of memory for the corresponding assembly language source code.

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Assembly language source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0800</td>
<td>ORG $0800</td>
<td>M DS.B 1 ; reserve one byte</td>
</tr>
<tr>
<td>______</td>
<td>________</td>
<td>N DC.B 1 ; reserve one byte &amp; initialize it</td>
</tr>
<tr>
<td>OD00</td>
<td>ORG $0D00</td>
<td></td>
</tr>
<tr>
<td>OD00</td>
<td>CC_______</td>
<td>LDD #10</td>
</tr>
<tr>
<td>______</td>
<td>________</td>
<td>STD $0</td>
</tr>
<tr>
<td>______</td>
<td>B60100</td>
<td>LDAA $100</td>
</tr>
<tr>
<td>______</td>
<td>________</td>
<td>STAA M</td>
</tr>
<tr>
<td>______</td>
<td>________</td>
<td>HERE ABA</td>
</tr>
<tr>
<td>______</td>
<td>26_______</td>
<td>BNE HERE</td>
</tr>
</tbody>
</table>

Solution:

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Assembly language source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0800</td>
<td>ORG $0800</td>
<td>M DS.B 1 ; reserve one byte</td>
</tr>
<tr>
<td>0800</td>
<td>??</td>
<td>N DC.B 1 ; reserve one byte &amp; initialize it</td>
</tr>
<tr>
<td>0801</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>OD00</td>
<td>ORG $0D00</td>
<td></td>
</tr>
<tr>
<td>OD00</td>
<td>CC000A</td>
<td>LDD #10</td>
</tr>
<tr>
<td>OD03</td>
<td>5C00</td>
<td>STD $0</td>
</tr>
<tr>
<td>OD05</td>
<td>B60100</td>
<td>LDAA $100</td>
</tr>
<tr>
<td>OD0B</td>
<td>7A0800</td>
<td>STAA M</td>
</tr>
<tr>
<td>OD0D</td>
<td>1806</td>
<td>HERE ABA</td>
</tr>
<tr>
<td>OD0D</td>
<td>26FC</td>
<td>BNE HERE</td>
</tr>
</tbody>
</table>

the address of the next instruction is OD0F

4. The interface shown can be used for low current LEDs. Assume the LED voltage drop is 2 V. The resistor is 1000 Ω. When the software outputs a high, the voltage on PP0 becomes 4.9 V. When the software outputs a low, the voltage on PP0 becomes 0.5 V. What is the LED current when the LED is on?

Solution:

\[
I = \frac{(5-2-0.5V)}{1000Ω} = 2.5V/1000Ω = 2.5 \text{ mA}
\]

5. A simple security system has two TR257-1 motion sensors connected to a HCS12 microcontroller. The motion sensors have an open collector output - when a motion sensor senses motion, it outputs a logic low. If either sensor detects motion, the system illuminates a light emitting diode (LED). Draw a schematic diagram for this system, showing the connections you would make from these components to
the MCU. Don’t forget the resistors. Note – use a single input port pin for both sensors. It is not necessary to write any code for this problem.

Solution: We will use PT1 for output and PT0 for input (you can use any of the digital I/O pins). We can use a single input pin for both sensors, because the sensors have open collector outputs. Their outputs are connected together in a “wired AND” configuration. If either sensor detects motion it pulls the output low. If neither sensor detects motion, then the output is high.

6. Write HCS12 C code that sequentially illuminates a single LED segment in a seven-segment display, and traces a figure “8”. Specifically, it illuminates the top segment (“a”) for a short time, then illuminates “b”, “g”, “e”, “d”, “c”, “g”, and “f” in turn.

Solution:

Figure 4.17 Driving a single seven-segment display
There are eight codes we have to display in a sequence. Let’s put these in a table.

```c
char table[] = {
    0x40, // a
    0x20, // b
    0x01, // g
    0x04, // e
    0x08, // d
    0x10, // c
    0x01, // g
    0x02}; // f
```

```c
void main(void)
{
    int n;
    DDRB = 0x7f; // configure bits 0..6 for output
    while (1) {
        PTB = table[n]; // display next code
        delay(); // delay a little (use your own function)
        if (++n == 8) // increment n to point to next code
            n = 0; // reset back to 0
    }
}
```

7. What does the following HCS12 assembly language program do? Describe the result of executing the program; don’t just say what each instruction is doing.

Instructions:
- ld x ##0800
- ldaa #5
- ldab #0
- loop stab 0,x
- incb
- inx
- dbne a,loop

Solution: this program stores $00, $01, $02, $03, $04, into addresses $0800, $0801, $0802, $0803, $0804.

8. Predict the result of executing the code below. What do registers a,b,x,y contain?

```
ldaa #$aa
ldab #$bb
ld x #$1234
ld y #$5678
psha
pshb
Solution:

After pushing, the stack contains:

$56$
$78$
$12$
$34$
$bb$
$aa$

We pull off in reverse order. So

\begin{align*}
a &= \$56 \\
b &= \$78 \\
x &= \$1234 \\
y &= \$bbaa
\end{align*}

9. **Estimate the running time of the following code fragment (assume a 24 MHz clock).**

```assembly
ldab  #$10
LOOP
  deb
  bne  LOOP
```

Solution: The time in cycles for each instruction is shown:

\begin{align*}
(1) & \quad ldab  \#$10 \\
(1) & \quad LOOP  deb \\
(3/1) & \quad bne  LOOP
\end{align*}

The loop is executed 16 times. The first 15 times, each iteration takes 4 cycles. The last time, it takes 2 cycles. Plus, we have one cycle from the LDAB instruction. So the total is

\[1 + 15 \times 4 + 2 = 63 \text{ cycles}\]

Each cycle takes (1/24) usec. So the total is 63/24 usec = 2.625 usec.

10. **Give the contents of the indicated registers or memory locations after the execution of each of the following program modules. Assume that prior to the execution of each of the following parts:**

- **Memory contains**
  
  \begin{align*}
  ($0080) &= \$01 \\
  ($0081) &= \$02
  \end{align*}
($0082) = $03  
($0083) = $04

- The M68HC12 registers contain: A = $7F, X = $0080  
- The NZVC bits in the CCR are 0001

(Note: Do not treat the program modules as executing sequentially, one following another.)

Program module: After execution:

(a) ADCA $80       A =       NZVC =

(b) ADCA #$80      A =       NZVC =

(c) BHI $E000
   BRA $E010      PC =       NZVC =

(d) LDD 0,X
   ABX           A =       X =

(e) ORG $D00
    LDS #$82
    JSR $0C10     PC =       ($0081) =

Solution:

Program module: After execution:

(a) ADCA $80       A = $81       HNZVC = 1010
    this is 1 + 0111 1111 + 0000 0001 = 1000 0001

(b) ADCA #$80      A = $00       HNZVC = 0101
    this is 1 + 0111 1111 + 1000 0000 = 0000 0000

(c) BHI $E000
   BRA $E010      PC = $E010     HNZVC = 0001
   BHI will branch if C or Z=0. It is not, so we go to $E010. CCR bits are not changed.

(d) LDD 0,X
   ABX           A = $01       X = $0082
    The first instruction loads A with 01, B with 02. The 2nd instruction adds 02 to X, to get 0082.

(e) ORG $D00
    LDS #$82
    JSR $0C10     PC = $0C10     ($0081) = 06
    The LDS instruction is 3 bytes long and the JSR instruction is 3 bytes long. The return address is $0D06.

11. Assume that the stack pointer has the value $0a00. A HCS12 program calls a subroutine, and the subroutine pushes registers A, X, and Y onto the stack. What does the stack pointer contain now?

The subroutine call pushes the program counter (2 bytes) onto the stack. The A, X, and Y registers are 1, 2, and 2 bytes, respectively. A total of 2+1+2+2 = 7 bytes are pushed onto the stack. The new stack pointer is $0a00 – 7 = $09f9.
12. The C function below is called with the following input arguments: an array \( M \) containing \( N \) 8-bit numbers, and the size \( N \). Describe what the function does.

\[
\text{int func(int M[], int N)} \\
{ \begin{align*}
\text{int i;} \\
\text{int x = M[0];} \\
\text{for (i=1; i<N; i++)} \\
\text{if (M[i] < x)} \\
\quad x = M[i]; \\
\text{return x;}
\end{align*}
}\]

Solution: 
The function finds the smallest element of the given array and returns it.

13. Write C code (using a loop) to compute the sum of the squares of the first 100 odd integers.

Solution: We will declare a variable called “sum” and accumulate the squares of the first 100 odd integers, using a loop. But first, let’s estimate whether the sum will exceed what can be stored in an “int” variable. A 16-bit “int” variable can hold up to \( 2^{16} - 1 = 65535 \). The largest integer we will square is the 100th odd integer, which is 201. The square of 201 is about \( 200^2 = 40,000 \). If all 100 values were this large, the sum would be \( 100 \times 40,000 = 4,000,000 \) (but we know the sum will be less than this). 4,000,000 is too big for an “int”, but will fit in a “long int”, which can hold up to \( 2^{32} = 4294967296 \).

\[
\text{int i, j; long sq_sum; sq_sum = 0; for (i = 0; i < 100; i++) { j = 2 * i + 1; sq_sum += (long)j * (long)j; }}
\]

14. Write a C function that converts all uppercase ASCII letters in a string, to lowercase. The string is passed into the function as an input argument.

Solution:

\[
\text{void upper2lower (char *ptr)} \\
{ \begin{align*}
\text{while(*ptr++)} \\
\text{if (((*ptr <= 0x5A) && (*ptr >= 0x41))} \\
\quad *ptr = *ptr + 0x20;
\end{align*}
}\]

15. Write C code that takes an array of 10 integers called “buff”, computes the difference between the maximum and minimum values in the array, and stores it into an integer variable called “diff”. When calculating the difference, don’t worry about possible overflow:

Solution:
// assume buff is defined elsewhere
int i, max, min, diff;

max = buff[i];   // initialize max and min to 1st element
min = buff[i];
for (i=1; i<10; i++)   {
  if (buff[i] > max)  max = buff[i];
  if (buff[i] < min)  min = buff[i];
}
diff = max-min;

16. The following C program performs the “factorial” operation using “recursive” function calls. Assume that the stack pointer originally has the value 0x900 when the program starts. Assume that you put a breakpoint at the instruction marked with $$$$, and look at the stack pointer (SP) register when the program breaks there. Choose the most likely correct answer below:
(a) SP still contains 0x900.
(b) SP contains 0x8FD (i.e., slightly smaller than the original value).
(c) SP contains 0x903 (i.e., slightly larger than the original value).
(d) SP contains 0x8EF (i.e., substantially smaller than the original value).
(e) SP contains 0x914 (i.e., substantially larger than the original value).

#include <hidef.h>      /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */

unsigned char fact(unsigned char n)
{
  if (n==0)
    return 1;   // $$$$$$$$$$ //
  else
    return n*fact(n-1);
}

void main(void)
{
  unsigned char  n, result;
  EnableInterrupts;
  n = 5;
  result = fact(n);
  for(;;) ;  /* loop forever */
}

Solution: Every time the function “fact” is called, the system stores the return address (2 bytes) onto the stack. It also passes in the input parameter “n” on the stack. This is done 6 times, so you have just put 6x3 = 18 bytes onto the stack. So the SP is substantially smaller than its original value (answer d).

It is not necessary to run the program to determine the answer, but this is what I got when I did it. I stepped through the program and when it got to the instruction with the $$ $$, the stack pointer points to $08EF. Here are the contents of the stack at that point:

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>08EF</td>
<td>00</td>
<td>Function input parameter “n”</td>
</tr>
<tr>
<td>Address</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>08F0</td>
<td>C0</td>
<td>Return address</td>
</tr>
<tr>
<td>08F1</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>08F2</td>
<td>01</td>
<td>Function input parameter “n”</td>
</tr>
<tr>
<td>08F3</td>
<td>C0</td>
<td>Return address</td>
</tr>
<tr>
<td>08F4</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>08F5</td>
<td>02</td>
<td>Function input parameter “n”</td>
</tr>
<tr>
<td>08F6</td>
<td>C0</td>
<td>Return address</td>
</tr>
<tr>
<td>08F7</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>08F8</td>
<td>03</td>
<td>Function input parameter “n”</td>
</tr>
<tr>
<td>08F9</td>
<td>C0</td>
<td>Return address</td>
</tr>
<tr>
<td>08FA</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>08FB</td>
<td>05</td>
<td>Function input parameter “n”</td>
</tr>
<tr>
<td>08FC</td>
<td>C0</td>
<td>Return address</td>
</tr>
<tr>
<td>08FD</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>08FE</td>
<td>00</td>
<td>Main program local variable “result”</td>
</tr>
<tr>
<td>08FF</td>
<td>05</td>
<td>Main program local variable “n”</td>
</tr>
</tbody>
</table>