(5) Question 1. The format is 8-bit signed. What is the hexadecimal representation of the value -50?

(5) Question 2. When you add two 8-bit signed numbers an overflow error can occur. Which of the following techniques can be used to handle the problem of overflow? If there is more than one answer, give all answers that could work.

A) Mask the data
B) Make it friendly.
C) Use interrupts.
D) Implement ceiling and floor.
E) Add drop out.
F) Use promotion.
G) Use demotion.
H) Use unsigned math.
I) Make it nonvolatile.

(5) Question 3. Consider the following two instructions

\texttt{ldab \#-2}
\texttt{subb \#250}

What will be the value of the overflow (V) bit?

What will be the value of the carry (C) bit?
**Question 4.** Use a 7406 to interface an LED to PT5 of the 9S12. The desired operating point is 2.5V at 20 mA. At 20mA you can assume the $V_{OL}$ of the 7406 will be 0.5 V.

**Question 5.** Consider the following piece of code that starts at `main`

```assembly
$5000 08           Add1 inx
$5001 3D           rts
$5002 CF4000       main lds #$4000
$5005 CE000A       ldx #10
$5008 34           loop pshx
$5009 07F5         bsr Add1
$500B 30           pulx
$500C 0435F9       dbne x,loop
$500F 183E         stop
$FFFE               org $FFFE
$FFFE 5002         fdb main
```

Part a) Think about how this program executes up to and including the first execution of `inx`.

- Fill in specific hexadecimal bytes that are pushed on the stack.
- Using an arrow, label to which box the SP points.
- Your solution may or may not use all the boxes.

Part b) How many times is the subroutine called after reset and before stop?

**Question 6.** Assume PC is $5000$, Register D is initially $2233$, and Register X is $3000$. You may assume all RAM locations are initially 0. Show the simplified bus cycles occurring when the `std` instruction is executed. In the “changes” column, specify which registers get modified during that cycle, and the corresponding new values. Do not worry about changes to the CCR. *Just show the one instruction.*

```assembly
$5000 6C02     std 2,x
```

<table>
<thead>
<tr>
<th>R/W</th>
<th>Addr</th>
<th>Data</th>
<th>Changes to D,X,Y,S,PC,IR,EAR</th>
</tr>
</thead>
<tbody>
<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></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For questions 7, 8, and 9, don’t worry about establishing the reset vector, creating a main program, or initializing the stack pointer. You may use the following definitions

\[ \text{PTT equ $0240} \]
\[ \text{DDRT equ $0242} \]

(20) **Question 7.** Assume a positive logic switch is connected to PT6, and the direction register is properly initialized. Write an assembly code that waits until the switch is pressed.

(20) **Question 8.** Assume **Buffer** is an array of 100 16-bit numbers, located in RAM. Write assembly code that initializes all numbers to its index value. Implement the following C (you can implement the result without making it precisely match the C code)

```
for(i=0; i<100; i++)
    Buffer[i] = i;
```
(20) **Question 9.** Assume Register B contains an 8-bit signed number, which is the input parameter to the subroutine. Assume Port T bit 5 is an output to an LED. Write an assembly language subroutine that tests Reg B, if it is greater than 100, turn on the LED, otherwise do not change the LED. Full credit will be given to a friendly solution.

```assembly
;***** turn on LED if RegB>100 ***********************
;Inputs: RegB is a signed 8-bit number,
;Outputs: none
LEDout
```
oraa 8-bit logical or to RegA
orab 8-bit logical or to RegB
orcc 8-bit logical or to RegCC
psha push 8-bit RegA onto stack
pshb push 8-bit RegB onto stack
pshc push 16-bit RegCC onto stack
pshd push 16-bit RegD onto stack
pshx push 16-bit RegX onto stack
pshy push 16-bit RegY onto stack
pula pop 8 bits off stack into RegA
pulb pop 8 bits off stack into RegB
pulc pop 8 bits off stack into RegCC
puld pop 16 bits off stack into RegD
pulx pop 16 bits off stack into RegX
puly pop 16 bits off stack into RegY
rev weighted Fuzzy logic rule evaluation
rol 8-bit roll shift left Memory
rola 8-bit roll shift left RegA
rolb 8-bit roll shift left RegB
ror 8-bit roll shift right Memory
rorb 8-bit roll shift right RegB
rtc return sub in expanded memory
rti return from interrupt
rts return from subroutine
sba 8-bit subtract RegA-RegB
sbca 8-bit sub with carry from RegA
sbcb 8-bit sub with carry from RegB
sec set carry bit, C=1
sei set I=1, disable interrupts
sev set overflow bit, V=1
sex sign extend 8-bit to 16-bit reg
staa 8-bit store memory from RegA
stab 8-bit store memory from RegB
std 16-bit store memory from RegD
sts 16-bit store memory from SP
stx 16-bit store memory from RegX
sty 16-bit store memory from RegY
suba 8-bit sub from RegA
subb 8-bit sub from RegB
subd 16-bit sub from RegD
swi software interrupt, trap
tab transfer A to B
tap transfer A to CC
tba transfer B to A
tbeq test and branch if result=0
tbl 8-bit look up and interpolation
tne test and branch if result≠0
tnbe test and branch if result=0
tbne test and branch if result≠0
rol 8-bit roll shift left Memory
tfr transfer register to register
tfr X,Y
tpa transfer CC to A
trb return sub in expanded memory	tst 8-bit compare memory with zero
tsw return from interrupt
tsta 8-bit compare RegA with zero
stsb 8-bit compare RegB with zero
stx transfer S to X
txs transfer S to Y
tera 8-bit sub with carry from RegA
txsr transfer S to X
txsr transfer S to Y
sbcb 8-bit sub with carry from RegB
txsr transfer S to X
txsr transfer S to Y
sex sign extend 8-bit to 16-bit reg
sex B,D
sex sign extend 8-bit to 16-bit reg
sex B,D
xgdx exchange RegD with RegX
xgdy exchange RegD with RegY
example addressing mode Effective Address
ldaa #u immediate No EA
ldaa u direct EA is 8-bit address (0 to 255)
ldaa U extended EA is a 16-bit address
ldaa m,r 5-bit index EA=r+m (-16 to 15)
ldaa v,+r pre-increment EA=r+v, EA=r (1 to 8)
ldaa v,-r pre-decrement r=r-v, EA=r (1 to 8)
ldaa v,r+ post-increment EA=r, r=r+v (1 to 8)
ldaa v,r- post-decrement EA=r, r=r-v (1 to 8)
ldaa A,r Reg A offset EA=r+A, zero padded
ldaa B,r Reg B offset EA=r+B, zero padded
ldaa D,r Reg D offset EA=r+D
ldaa q,r 9-bit index EA=r+q (-256 to 255)
ldaa W,r 16-bit index EA=r+W (-32768 to 65535)
ldaa [D,r] D indirect EA=(r+D)
ldaa [W,r] D indirect EA=(r+W) (-32768 to 65535)

Freescale 9S12 addressing modes r is X, Y, SP, or PC

Pseudo op meaning
org Specific absolute address to put subsequent object code
equ Define a constant symbol
set Define or redefine a constant symbol
.byte Allocate byte(s) of storage with initialized values
.db Allocate word(s) of storage with initialized values
dc.b Allocate byte(s) of storage with initialized values
fcb .byte
.dw Allocate word(s) of storage with initialized values
.fcw Allocate word(s) of storage with initialized values
.dl Allocate 32-bit long word(s) of storage with initialized values
.rmb .blkb Allocate bytes of storage without initialization
.rs Allocate bytes of storage without initialization
.l word Allocate bytes of storage without initialization
.blkw Allocate 32-bit words of storage without initialization
.blkl Allocate 32-bit words of storage without initialization
STD  
Store Double Accumulator

Operation:  \((A : B) \Rightarrow M : M + 1\)
Description:  Stores the content of double accumulator D in memory location 
M : M + 1. The content of D is unchanged.

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
<th>HCS12 Access Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD opr8a</td>
<td>DIR</td>
<td>5C dd</td>
<td>PW</td>
</tr>
<tr>
<td>STD opr16a</td>
<td>EXT</td>
<td>7C hh ll</td>
<td>PWO</td>
</tr>
<tr>
<td>STD oprx0,xyssp</td>
<td>IDX</td>
<td>6C xb</td>
<td>PW</td>
</tr>
<tr>
<td>STD oprx9,xyssp</td>
<td>IDX1</td>
<td>6C xb ff</td>
<td>PWO</td>
</tr>
<tr>
<td>STD oprx16,xyssp</td>
<td>IDX2</td>
<td>6C xb ee ff</td>
<td>PWP</td>
</tr>
</tbody>
</table>

BSR  
Branch to Subroutine

Operation:  
\((SP) - $0002 \Rightarrow SP\)  
\(RTNH : RTNL \Rightarrow M(SP) : M(SP+1)\)  
\((PC) + Rel \Rightarrow PC\)
Description:  Sets up conditions to return to normal program flow, then transfers control to a subroutine. Uses the address of the instruction after the BSR as a return address. Decrements the SP by two, to allow the two bytes of the return address to be stacked. Stacks the return address (the SP points to the high-order byte of the return address). Branches to a location determined by the branch offset. Subroutines are normally terminated with an RTS instruction, which restores the return address from the stack.

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
<th>Access Detail HCS12</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSR rel8</td>
<td>REL</td>
<td>07 rr</td>
<td>SPPP</td>
</tr>
</tbody>
</table>

RTS  
Return from Subroutine

Operation:  
\((M(SP) : M(SP+1)) \Rightarrow PCH : PCL; (SP) + $0002 \Rightarrow SP\)
Description:  Restores context at the end of a subroutine. Loads the program counter with a 16-bit value pulled from the stack and increments the stack pointer by two. Program execution continues at the address restored from the stack.

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
<th>Access Detail HCS12</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTS</td>
<td>INH</td>
<td>3D</td>
<td>UfPPP</td>
</tr>
</tbody>
</table>
**SUBB**  
**Subtract B**  

**Operation:** \((B) - (M) \Rightarrow B\)

**Description:** Subtracts the content of memory location M from the content of accumulator B and places the result in B. For subtraction instructions, the C status bit represents a borrow.

N: Set if MSB of result is set; cleared otherwise  
Z: Set if result is $00$; cleared otherwise  
V: \(B_7 \cdot M_7 \cdot R_7 + B_7 \cdot M_7 \cdot R_7\)  
Set if a two's complement overflow resulted from the operation; cleared otherwise  
C: \(B_7 \cdot M_7 + M_7 \cdot R_7 + R_7 \cdot B_7\)  
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
<th>Access Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBB (#opr8i)</td>
<td>IMM</td>
<td>C0 i1</td>
<td>P</td>
</tr>
<tr>
<td>SUBB (opr8a)</td>
<td>DIR</td>
<td>D0 dd</td>
<td>rPf</td>
</tr>
<tr>
<td>SUBB (opr16a)</td>
<td>EXT</td>
<td>F0 hh ll</td>
<td>rPO</td>
</tr>
<tr>
<td>SUBB (oprX_0,xsys)</td>
<td>IDX</td>
<td>E0 xb</td>
<td>rPf</td>
</tr>
<tr>
<td>SUBB (oprX,xySp)</td>
<td>IDX1</td>
<td>E0 xb ff</td>
<td>rPO</td>
</tr>
<tr>
<td>SUBB (oprX_16,xySp)</td>
<td>IDX2</td>
<td>E0 xb ee ff</td>
<td>frPP</td>
</tr>
<tr>
<td>SUBB ([D,xySp])</td>
<td>[D,IDX]</td>
<td>E0 xb</td>
<td>f1frPf</td>
</tr>
<tr>
<td>SUBB ([oprX_16,xySp])</td>
<td>[IDX2]</td>
<td>E0 xb ee ff</td>
<td>fIPrPf</td>
</tr>
</tbody>
</table>