This is a closed book exam. You must put your answers pages 1,2,3 only. You have 50 minutes, so allocate your time accordingly. Show your work, and put your answers in the boxes. **Please read the entire quiz before starting.**

(5) **Question 1.** Consider the hexadecimal number $A5$. If the format is 8-bit signed, what decimal number does this represent?

(5) **Question 2.** Consider the following display of a digital voltmeter.

![Digital Voltmeter Display](image)

Part a) The voltmeter reads from 0.0000 to 3.9999V. How many decimal digits is the voltmeter?

Part b) If you were asked to implement this voltmeter using a microcontroller, what is the minimum number of ADC bits you could use?

(5) **Question 3.** Consider the following two instructions

```
ldab #1
addb #1
```

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

What will be the value of the carry (C) bit?

What will be the value of the zero (Z) bit?
(5) **Question 4.** What resistor value do you need to interface this LED to the 9S12? The desired operating point is 1.9V at 1 mA. At 1mA you can assume the \( V_{OH} \) of the 9S12 will be 4.9 V.

\[
\text{PP0} \quad \text{R}
\]

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

```assembly
$4035 \ CF4000 main lds #$4000
$4038 \ 860A ldaa #10
$403A \ 0703 loop bsr Output
$403C \ 0430FB dbne A,loop
$403F \ 36 Output psha
```

Hand execute this program up to and including the first execution of `psha`

Fill in specific hexadecimal bytes that are pushed on the stack.
Using an arrow label to which box the SP points.
Not all the boxes will be used.

<table>
<thead>
<tr>
<th>R/W</th>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(10) **Question 6.** Assume PC is $5000, and Register D is initially $2233. You may assume location $0812 and $0813 are initially 0. Show the simplified bus cycles occurring when the `subd` 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 \ B30812 subd $0812
```

<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>

(10) **Question 7.** Draw the circuit diagram interfacing this switch using negative logic. I.e., if the switch is not pressed the PT1 input should be high, and if the switch is pressed, the input should be low. Please specify the resistance values for the resistors and the chip numbers (e.g., 7406) for any digital logic you use.

\[
\text{PT1} \quad \text{9S12}
\]
For questions 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

\[
\begin{align*}
\text{PTT} &\equiv \$0240 \\
\text{DDRT} &\equiv \$0242 \\
\text{PTP} &\equiv \$0258 \\
\text{DDRP} &\equiv \$025A
\end{align*}
\]

**(20) Question 8.** Assume $S_g$ is a signed 16-bit global variable. Transfer the following C-code into assembly language. I.e., clear Port T bits 1 and 0 if $S_g$ is greater than or equal to 100. You may assume PT1 and PT0 are already outputs.

```c
if (Sg >= 100) PTT &= ~0x03;
```

**(30) Question 9.** Assume bits 3-0 of both Port T and Port P are already inputs. You will write a subroutine that reads Port T and Port P, setting RegA bits 7-4 to Port T bits 3-0 and RegA bits 3-0 to Port P bits 3-0. Hint you can use a temporary global variable, or you could use the aba instruction.

```asm
;*****ReadPorts subroutine**************************
;Inputs: none
;Outputs: RegA bits7-0 are PT3-0, RegA bits 3-0 are PP3-0
ReadPorts
```
oraa 8-bit logical or to RegA
orab 8-bit logical or to RegB
orcc 8-bit logical or to RegCC
pshb push 8-bit RegB onto stack
pshe push 8-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 Fuzzy logic rule evaluation
revw weighted Fuzzy 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
sex B,D
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
tap transfer A to B
tapb transfer A to CC
tbl 8-bit look up and interpolation
tba transfer B to A
th eq test and branch if result=0
base A,loop
tne test and branch if result≠0
rol 8-bit roll shift left Memory
tfr transfer register to register
rora 8-bit roll shift right RegA
tpa transfer CC to A
rorb 8-bit roll shift right RegB
trap illegal instruction, or software trap
rts 8-bit compare memory with zero
tsta 8-bit compare RegA with zero
tstb 8-bit compare RegB with zero
tsx transfer S to X
tay transfer S to Y
sec set carry bit, C=1
tys transfer Y to S
set set I=1, disable interrupts
wai wait for interrupt
setv set overflow bit, V=1
wavy weighted Fuzzy logic average
sex sign extend 8-bit to 16-bit reg
sex B,D
xgdx exchange RegD with RegX
xgdy exchange RegD with RegY

Example addressing modes

<table>
<thead>
<tr>
<th>addressing mode</th>
<th>Effective Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>lda a #u</td>
<td>immediate No EA</td>
</tr>
<tr>
<td>lda a u</td>
<td>direct EA is 8-bit address (0 to 255)</td>
</tr>
<tr>
<td>lda a U</td>
<td>extended EA is a 16-bit address</td>
</tr>
<tr>
<td>lda a,m,r</td>
<td>5-bit index EA=r+m (-16 to 15)</td>
</tr>
<tr>
<td>lda a,r</td>
<td>pre-increment r=r+v, EA=r (1 to 8)</td>
</tr>
<tr>
<td>lda a,-r</td>
<td>pre-decrement r=r-v, EA=r (1 to 8)</td>
</tr>
<tr>
<td>lda a,A,r</td>
<td>post-increment EA=r, r=r+v (1 to 8)</td>
</tr>
<tr>
<td>lda a,B,r</td>
<td>post-decrement EA=r, r=r-v (1 to 8)</td>
</tr>
<tr>
<td>lda a, A,r</td>
<td>Reg A offset EA=r+A, zero padded</td>
</tr>
<tr>
<td>lda a, B,r</td>
<td>Reg B offset EA=r+B, zero padded</td>
</tr>
<tr>
<td>lda a, D,r</td>
<td>Reg D offset EA=r+D</td>
</tr>
<tr>
<td>lda a,q,r</td>
<td>9-bit index EA=r+q (-256 to 255)</td>
</tr>
<tr>
<td>lda a,W,r</td>
<td>16-bit index EA=r+W (-32768 to 65535)</td>
</tr>
<tr>
<td>lda a, [D,r]</td>
<td>D indirect EA=(r+D)</td>
</tr>
<tr>
<td>lda a, [W,r]</td>
<td>indirect EA=(r+W) (-32768 to 65535)</td>
</tr>
</tbody>
</table>

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
dc.b db fcb .byte Allocate byte(s) of storage with initialized values
fcc Create an ASCII string (no termination character)
dc.w dw fdb .word Allocate word(s) of storage with initialized values
dc.l dl .long Allocate 32-bit long word(s) of storage with initialized values
ds ds.b rmb .blkb Allocate bytes of storage without initialization
ds.w .blkw Allocate bytes of storage without initialization
ds.l .blkl Allocate 32-bit words of storage without initialization
**SUBD**

**Subtract Double Accumulator**

**Operation:** \((A : B) - (M : M + 1) \Rightarrow A : B\)

**Description:** Subtracts the content of memory location \(M : M + 1\) from the content of double accumulator \(D\) and places the result in \(D\).

<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>SUBD #opr16i</td>
<td>IMM</td>
<td>83 jj kk</td>
<td>PO</td>
</tr>
<tr>
<td>SUBD opr8a</td>
<td>DIR</td>
<td>93 dd</td>
<td>RPf</td>
</tr>
<tr>
<td>SUBD opr16a</td>
<td>EXT</td>
<td>B3 hh ll</td>
<td>RPO</td>
</tr>
<tr>
<td>SUBD oprx0_ysp</td>
<td>IDX</td>
<td>A3 xb</td>
<td>RPf</td>
</tr>
<tr>
<td>SUBD oprx9,xyssp</td>
<td>IDX1</td>
<td>A3 xb ff</td>
<td>RPO</td>
</tr>
<tr>
<td>SUBD oprx16,xyssp</td>
<td>IDX2</td>
<td>A3 xb ee ff</td>
<td>fRPP</td>
</tr>
</tbody>
</table>

**BSR**

**Branch to Subroutine**

**Operation:**
- \((SP) - \$0002 \Rightarrow SP\)
- \(\text{RTNH} : \text{RTNL} \Rightarrow M(SP) : M(SP + 1)\)
- \((PC) + \text{Rel} \Rightarrow PC\)

**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</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</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTS</td>
<td>INH</td>
<td>3D</td>
<td>UFPPP</td>
</tr>
</tbody>
</table>