This is a closed book exam. You must put your answers on this piece of paper only. You have 50 minutes, so allocate your time accordingly. Please read the entire quiz before starting.

(5) Question 1. Assume an 8-bit signed integer format. If the binary is %11010100, what is the corresponding decimal value of this signed integer?

(5) Question 2. What type of memory on the 9S12 is volatile? If there is more than one type, just list one of them.

(5) Question 3. What will be the value of the carry (C) bit after executing the following?
   ldaa #110
   suba #140

(5) Question 4. What will be the value of the overflow (V) bit after executing the following?
   ldab #-90
   addb #-100

(5) Question 5. Consider the result of executing the following two 9S12 assembly instructions.
   ldaa #$C2
   asra

   Part a) What is the value in Register A after two instructions are executed? Give the answer in hexadecimal or in binary.

   Part b) What is the value of the C bit after these two instructions are executed?
(5) Question 6. Assume we wish to begin execution at $5000$. Show the assembly code that establishes the reset vector.

(10) Question 7. Assume PC is $4210$, and the SP is initially $3FF6$. Show the simplified bus cycles occurring when the \texttt{bsr} 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. \textit{Just show the one instruction.}

\texttt{$4210 \ 07E0 \ bsr \ MyFunction$}

<table>
<thead>
<tr>
<th>R/W</th>
<th>Addr</th>
<th>Data</th>
<th>Changes to A,B,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>
</tbody>
</table>

For questions 8, 9 and 10, don’t worry about initializing the variables, establishing the reset vector, creating a main program, or initializing the stack.

(20) Question 8. We wish to make PT5 an output and PT2 an input. You may use these definitions.

\texttt{PTT \ equ \ $0240$

\texttt{DDRT \ equ \ $0242$

Part a) Write assembly code that makes PT5 an output and PT2 an input. Comments are required.

Part b) Write assembly code that sets PT5 to 1 if PT2 is 0, and does not change PT5 if PT2 is 1. Comments are required. +2 point bonus if both parts of Q8 are friendly.
(20) **Question 9.** You will write a subroutine with two 8-bit unsigned inputs and one 8-bit unsigned output. The inputs are passed in using RegA and RegB. The result is returned in RegA. The subroutine implements the \( \text{RegA} = \text{RegA} - \text{RegB} \). Implement the floor operation, such that if an unsigned overflow occurs, set the output to the minimum value, \( \text{RegA}=0 \).

;*****Sub1 subroutine********************
;Inputs: RegA is the first number
;        RegB is the second number
;Outputs: RegA is the difference of the first-second numbers
;        RegA is returned as 0 if an unsigned overflow occurs
Sub1

(20) **Question 10.** There are two 16-bit signed variables, called Input and Output. Write assembly code that checks the Input, and if Input is less than -100, then the code sets the Output to 200. Conversely if Input is greater than or equal to -100, then the code sets Output to 0.

```
org $0800
Input  rmb 2    ;signed 16-bit integer
Output rmb 2    ;signed 16-bit integer
org $4000
```
ab 8-bit add RegA=RegA+RegB
ax unsigned add RegX=RegX+RegB
ay unsigned add RegY=RegY+RegB
adc 8-bit add with carry to RegA
add 8-bit add with carry to RegB
add 8-bit add to RegA
addb 8-bit add to RegB
addd 16-bit add to RegD
lda 8-bit logical and to RegA
land 8-bit logical and to RegB
andc 8-bit logical and to RegCC
asl 8-bit left shift Memory
asla/lsla 8-bit left shift RegA
asl/lsl 8-bit left shift RegA
asrb 8-bit arith right shift to RegB
asra 8-bit arith right shift to RegB
asr 8-bit arith right shift Memory
bs 8-bit arith right shift Memory
asrb 8-bit arith right shift to RegB
bs 8-bit arith right shift to RegB
brc 8-bit branch if carry clear
bcrl bit clear in memory
bcrl PTT,#$01
bcs branch if carry set
beq branch if result is zero (Z=1)
bge branch if signed ≥
bgt branch if signed >
bhi branch if unsigned >
bits 8-bit and with RegA, sets CCR
bltb 8-bit and with RegB, sets CCR
ble branch if signed ≤
blt branch if signed <
bls branch if unsigned ≤
bls branch if unsigned ≤
blt branch if signed <
ble branch if signed ≤
blti branch if result is negative (N=1)
 ble branch if result is positive (N=0)
br branch always
brcsr branch if bits are clear
brcsr PTT,#$01,loop
brn branch never
brset branch if bits are set
brset PTT,#$01,loop
bs set bit set clear in memory
bs set PTT,#$04
bsr branch to subroutine
bvc branch if overflow clear
bvs branch if overflow set
call subroutine in expanded memory
cba 8-bit compare RegA with RegB
cle clear carry bit, C=0
cle clear T=0, enable interrupts
clr 8-bit memory clear
cla RegA clear
clrb RegB clear
clv clear overflow bit, V=0
cmpa 8-bit compare RegA with memory
cmpb 8-bit compare RegB with memory
coma 8-bit logical complement to RegA
comb 8-bit logical complement to RegB
cp 16-bit compare RegD with memory
cpx 16-bit compare RegX with memory
cpy 16-bit compare RegY with memory
da 8-bit decimal adjust accumulator
dbeq decrement and branch if result=0
dbeq Y,loop
dbne decrement and branch if result≠0
dbne A,loop
dec 8-bit decrement memory
deca 8-bit decrement RegA
decb 8-bit decrement RegB
decb 8-bit decrement RegB
decb 8-bit decrement RegB
deque 16-bit decrement RepC
deque 16-bit decrement RepC
deque 16-bit decrement RegX

Jonathan W. Valvano     September 26, 2008       10:00am-10:50am
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 RegD onto stack
pshh push 16-bit RegH 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
ror a 8-bit roll shift right RegA
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
stsb 16-bit store memory from RegB
sts 16-bit store memory from SP
stx 16-bit store memory from RegX
sty 16-bit store memory from RegY
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
stsb 16-bit store memory from RegB
sts 16-bit store memory from SP
stx 16-bit store memory from RegX
sty 16-bit store memory from 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 r=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+D (-32768 to 65535)
ldaa [D,r] D indirect EA=(r+D)
ldaa [W,r] indirect EA=(r+W) (-32768 to 65535)

Freescale 6812 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
fcb .byte Allocate byte(s) of storage with initialized values
fcb Create an ASCII string (no termination character)
dw Allocate word(s) of storage with initialized values
dw .word Allocate word(s) of storage with initialized values
dl Allocate 32-bit long word(s) of storage with initialized values
dl .long Allocate 32-bit long word(s) of storage with initialized values
ds .byte Allocate bytes of storage without initialization
ds .blk Allocate bytes of storage without initialization
ds .blkw Allocate bytes of storage without initialization
ds .blkl Allocate bytes of storage without initialization
**BSR**

**Branch to Subroutine**

**Operation:**

\[
\text{(SP)} - $0002 \Rightarrow \text{SP}
\]

\[
\text{RTNH} : \text{RTNL} \Rightarrow M(\text{SP}) : M(\text{SP+1})
\]

\[
(\text{PC}) + \text{Rel} \Rightarrow \text{PC}
\]

**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>

---

**SBA**

**Subtract Accumulators**

**Operation:**

\[(A) - (B) \Rightarrow A\]

**Description:** Subtracts the content of accumulator B from the content of accumulator A and places the result in A. The content of B is not affected. For subtraction instructions, the C status bit represents a borrow.

**CCR Details:**

<table>
<thead>
<tr>
<th>S</th>
<th>X</th>
<th>H</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Δ</td>
<td>Δ</td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

N: Set if MSB of result is set; cleared otherwise
Z: Set if result is $00$; cleared otherwise
V: A7 • B7 • R7 + A7 • B7 • R7
   Set if a two’s complement overflow resulted from the operation; cleared otherwise
C: A7 • B7 + B7 • R7 + R7 • A7
   Set if the absolute value of B is larger than the absolute value of A; cleared otherwise