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 %10110010, what is the corresponding decimal value of this signed integer?

(5) **Question 2.** What type of memory on the 9S12 is nonvolatile? 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
   adda #140
```

(5) **Question 4.** What will be the value of the overflow (V) bit after executing the following?

```
   ldab #-90
   subb #-100
```

(5) **Question 5.** Consider the result of executing the following two 9S12 assembly instructions.

```
   ldaa #$A5
   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 $6000$. Show the assembly code that establishes the reset vector.

(10) **Question 7.** Assume PC is $4120$, and the SP is initially $3FF4$. Show the simplified bus cycles occurring when the `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. *Just show the one instruction.*

```
$4120 07F0           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 PT4 an output and PT1 an input. You may use these definitions.

```
PTT   equ $0240
DDRT  equ $0242
```

Part a) Write assembly code that makes PT4 an output and PT1 an input. Comments are required.

Part b) Write assembly code that sets PT4 to 1 if PT1 is 0, and does not change PT4 if PT1 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 ceiling operation, such that if an unsigned overflow occurs, set the output to the maximum value, \( \text{RegA} = 255 \).

```assembly
;*****Add1 subroutine****************
;Inputs:  RegA is the first number
         RegB is the second number
;Outputs: RegA is the sum of the first+second numbers
         RegA is returned as 255 if an unsigned overflow occurs
Add1
```

(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.

```assembly
org $0800
Input rmb 2 ;signed 16-bit integer
Output rmb 2 ;signed 16-bit integer
org $4000
```
aba 8-bit add RegA=RegA+RegB
dey 16-bit decrement RegY
abx unsigned add RegX=RegX+RegB
div RegY=(Y:D)/RegX, unsigned divide
aby unsigned add RegY=RegY+RegB
divs RegY=(Y:D)/RegX, signed divide
adca 8-bit add with carry to RegA
eamcs 16 by 16 signed mult, 32-bit add
addch 8-bit add with carry to RegB
emaxd 16-bit unsigned maximum in RegD
addda 8-bit add to RegA
emaxm 16-bit unsigned maximum in memory
adddb 8-bit add to RegB
emind 16-bit unsigned minimum in RegD
adddd 16-bit add to RegD
emul RegY=RegY*RegD unsigned mult
anda 8-bit logical and to RegA
eumuls RegY=RegY*RegD signed mult
andb 8-bit logical and to RegB
andcc 8-bit logical and to RegCC
addc 8-bit add to RegA
aslb/lslb 8-bit arith left shift RegB
addcb 8-bit add with carry to RegB
asla/lsla 8-bit left shift RegA
adcb 8-bit add with carry to RegB
asl/lsl 8-bit left shift Memory
dada 8-bit arith right shift Memory
asrb/lsrb 8-bit arith right shift to RegB
asr 8-bit arith right shift to RegA
asrd 8-bit arith right shift RegD
bclr PTT,#$01
bclr PTT,#$01
bcs branch if carry set
bcs branch if carry set
bclr bit clear in memory
bclr PTT,#$01
bclr PTT,#$01
bclf branch if carry clear
bclf branch if carry clear
bclr branch if carry clear
bclr branch if carry clear
bcs branch if carry set
bcs branch if carry set
bce branch if result is zero (Z=1)
beq branch if result is zero
bge branch if signed ≥
bge branch if signed ≥
bgt branch if result >=
bgt branch if result >
bl branch if unsigned ≥
bl branch if unsigned ≥
bhi branch if unsigned >
blhi branch if unsigned >
bls branch if unsigned ≥
bls branch if unsigned ≥
bh branch if unsigned ≥
bs branch if unsigned ≥
blt branch if signed <
blt branch if signed <
bmi branch if result is negative (N=1)
bmi branch if result is negative (N=1)
bl branch if result is nonzero (Z=0)
bl branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
bpl branch if result is positive (N=0)
bra branch always
bra branch always
bs branch if bits are set
bs branch if bits are set
bset branch if bits are set
bset branch if bits are set
bclr branch if bits are clear
bclr branch if bits are clear
br set branch if bits are set
br set branch if bits are set
brnz branch never
brcr branch to subroutine
brnz branch never
bvc branch if overflow clear
bvc branch if overflow clear
bvs branch if overflow set
bvs branch if overflow set
call subroutine in expanded memory
call subroutine in expanded memory
cbra 8-bit compare RegA with RegB
cb relieved from subroutine
clc clear carry bit, C=0
cli clear T=0, enable interrupts
clr 8-bit memory clear
clreb RegA clear
clrnb RegB clear
clcv clear overflow bit, V=0
cmpa 8-bit compare RegA with memory
cmpb 8-bit compare RegB with memory
com 8-bit logical complement to memory
comb 8-bit logical complement to RegA
comb 8-bit logical complement to RegA
cpd 16-bit compare RegD with memory
cppx 16-bit compare RegX with memory
cpy 16-bit compare RegY with memory
Cpy 16-bit compare RegY with memory
daa 8-bit decimal adjust accumulator
dbec decrements and branch if result=0
dbec decrements and branch if result=0
dbeq Y,loop
dbeq Y,loop
dbne decrement and branch if result≠0
dbne decrement and branch if result≠0
dbne A,loop
dbne A,loop
de 8-bit decrement memory
dec 8-bit decrement memory
deca 8-bit decrement RegA
decb 8-bit decrement RegB
decch 8-bit decrement RegB
des 16-bit decrement RegSP
des 16-bit decrement RegSP
DEX 16-bit decrement RegX

de neg 8-bit 2's complement negate memory
nd neg 8-bit 2's complement negate RegA
negb 8-bit 2's complement negate RegB
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 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  
rora 8-bit roll shift right RegA  
rorb 8-bit roll shift right RegB  
rtc return sub in expanded memory  
rti return from interrupt  
rtu 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  
styl 8-bit look up and interpolation  
tbeq test and branch if result=0  
tbeq Y,loop  
tbl test and branch if result≠0  
tbeq A,loop  
tfr transfer register to register  
tpa transfer CC to A  
tpc transfer CC to C  
tq transfer Q to X  
tx transfer X to S  
txs transfer X to S  
tys transfer Y to S  
tst 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  
txa transfer A to X  
txa exchange RegA with RegB  
tx exchange RegX with RegY  
txb exchange RegB with RegY  
swi software interrupt, trap  
swx weighted Fuzzy logic average  
xgdx exchange RegD with RegX  
xgdy exchange RegD with RegY  

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

Freescale 6812 addressing modes r is X, Y, SP, or PC

Pseudo op meaning

org Specific absolute address to put subsequent object code
set Define 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
Allocate 32-bit words of storage without initialization

**BSR**

**Branch to Subroutine**

**Operation:**

\[(SP) - 0002 \Rightarrow SP\]
\[RTNH : RTNL \Rightarrow M(SP) : M(SP+1)\]
\[(PC) + \text{Rel} \Rightarrow PC\]

**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. Decrement 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>

**ABA**

**Add Accumulator B to Accumulator A**

**Operation:**

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

**Description:** Adds the content of accumulator B to the content of accumulator A and places the result in A. The content of B is not changed. This instruction affects the H status bit so it is suitable for use in BCD arithmetic operations. See DAA instruction for additional information.

**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>Δ</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

**H:**

\[A3 \cdot B3 \cdot R3 + R3 \cdot A3\]
Set if there was a carry from bit 3; cleared otherwise

**N:**

Set if MSB of result is set; cleared otherwise

**Z:**

Set if result is $00$; cleared otherwise

**V:**

\[A7 \cdot B7 \cdot R7 + \bar{A7} \cdot B7 \cdot R7\]
Set if a two's complement overflow resulted from the operation; cleared otherwise

**C:**

\[A7 \cdot B7 \cdot R7 + \bar{R7} \cdot R7 \cdot A7\]
Set if there was a carry from the MSB of the result; cleared otherwise