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.** Give the hex value……

(5) **Question 2.** Specify 0 or 1 ………

(5) **Question 3.** Specify 0 or 1 ………

(5) **Question 4.** Specify A-H ………

(5) **Question 5.** Show the equation….  

(5) **Question 6.** Show the machine code….  

(5) **Question 7.** How many binary bits?……

(5) **Question 8.** Specify values…………

RegA =

RegX =

(10) **Question 9.** Simplified memory cycles (you may or may not need all 5 entries)

<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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(25) Part 10a) Write the main program (figure shows the result after running your program)

```assembly
DDRM equ $0252  ; Port M Direction
DDRT equ $0242  ; Port T Direction
PTM  equ $0250  ; Port M I/O Register
PTT  equ $0240  ; Port T I/O Register
org $4000
main
```

(25) Part 10b) Write the assembly language subroutine.

```assembly
;*******Check***********
; if PT6 is 1, then set PT5=1
; if PT6 is 0, then return without modifying PT5
Check
org $FFFF
fdb main
```
aba 8-bit add RegA+RegB
abx unsigned add RegX+RegB
aby unsigned add RegY+RegB
adca 8-bit add with carry to RegA
adcb 8-bit add with carry to RegB
add 8-bit add to RegA
addb 8-bit add to RegB
addw 16-bit add to RegD
anda 8-bit logical and to RegA
andb 8-bit logical and to RegB
andcc 8-bit logical and to RegCC
asl/asla 8-bit left shift
aslb/lslb 8-bit arith left shift RegA
asld/lsld 16-bit left shift RegD
asr 8-bit arith right shift Memory
asrb 8-bit arith right shift to RegB
asra 8-bit arith right shift
asr 8-bit arith right shift to RegA
bls branch if unsigned
blo branch if unsigned <
ble branch if signed ≤
bhi branch if signed >
bhhi branch if signed ≥
bhs branch if unsigned ≥
bhs branch if unsigned >
biv branch if signed ≥
ble branch if signed ≤
bit 8-bit and with RegA, sets CCR
bitt 8-bit and with RegB, sets CCR
biv branch if signed ≥
blo branch if unsigned <
bit branch if signed ≤
bit branch if signed ＜
bit branch if signed ＜
bmi branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
br branch always
brcl branch if bits are clear,
brn branch never
brset branch if bits are set
bsr branch to subroutine
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
cbe branch if signed ≥
cbgd enter background debug mode
cbt branch if signed ＞
cl branch if signed ＞
cl branch if signed ≥
cli clear I=0, enable interrupts
clr 8-bit and with RegB, sets CCR
clcb 8-bit and with RegB, sets CCR
clra RegA clear
clrb RegB clear
civ clear overflow bit, V=0
cmp 8-bit compare RegA with memory
cmpb 8-bit compare RegB with memory
cmpcc 8-bit compare RegA with RegB with memory
cmpsb 8-bit compare RegB with RegB with memory
com 8-bit logical complement to Memory
com 8-bit logical complement to RegA
comb 8-bit logical complement to RegB
cpx 16-bit compare RegX with memory
cpy 16-bit compare RegY with memory
daa 8-bit decimal adjust accumulator
dbeq decrement and branch if result=0
dbeq decrement and branch if result=0
dbne decrement and branch if result≠0
dec 8-bit decrement memory
deca 8-bit decrement RegA
decb 8-bit decrement RegB
des 16-bit decrement RegSP
dex 16-bit decrement RegX
dey 16-bit decrement RegY
div RegY=RegX, unsigned divide
divs RegY=RegX, signed divide
divreg RegY=RegX, signed divide
emacs 16 by 16 signed mult, 32-bit add
emad 16-bit unsigned maximum in RegA
emaxd 16-bit unsigned maximum in memory
emaxm 16-bit unsigned maximum in memory
pulc pop 8 bits off stack into RegCC  stya 16-bit store memory from RegY
puld pop 16 bits off stack into RegD  suba 8-bit sub from RegA
pulx pop 16 bits off stack into RegX  subb 8-bit sub from RegB
puly pop 16 bits off stack into RegY  subd 16-bit sub from RegD
rev Fuzzy logic rule evaluation  revw weighted Fuzzy rule evaluation
rol 8-bit roll shift left Memory  tap transfer A to C
rola 8-bit roll shift left RegA  tba transfer B to A
rolb 8-bit roll shift left RegB  tbeq test and branch if result=0
ror 8-bit roll shift right Memory  tbl 8-bit look up and interpolation
rorax 8-bit roll shift right RegAx  tbeq test and branch if result=0
rorb 8-bit roll shift right RegB  tfn transfer register to register
rtc return sub in expanded memory  tpa transfer CC to A
rti return from interrupt  trap illegal instruction interrupt
rts return from subroutine  trap illegal op code, or software trap
sba 8-bit subtract RegA-RegB  tsta 8-bit compare RegA with zero
sbca 8-bit sub with carry from RegA  tstb 8-bit compare RegB with zero
sbcb 8-bit sub with carry from RegB  sec set carry bit, C=1
sec set carry bit, C=1  sei set I=1, disable interrupts
sev set overflow bit, V=1  sev set overflow bit, V=1
sex sign extend 8-bit to 16-bit reg  sx transfer X-1 to S
staa 8-bit store memory from RegA  staa transfer carry bit, C=1
stax 16-bit store memory from SP  staa transfer carry bit, C=1
std 16-bit store memory from RegD  stax transfer X-1 to S
stx 16-bit store memory from RegX  stsy transfer S+1 to Y
sty 16-bit store memory from RegY  tsta transfer carry bit, C=1
suba 8-bit sub from RegA  tst transfer S+1 to Y
subb 8-bit sub from RegB  txs transfer X-1 to S
subd 16-bit sub from RegD  wai wait for interrupt
tfr transfer register to register  wav weighted Fuzzy logic average
tab transfer A to B  xgdex exchange RegD with RegX
tap transfer CC to A  xgdyy exchange RegD with RegY
tbeq test and branch if result=0  xgdx exchange RegD with RegX
tbeq test and branch if result=0  xgdy exchange RegD with RegY

INC

Increment Memory

Operation: \((M) + \$01 \rightarrow M\)

<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>INC opr16a</td>
<td>EXT</td>
<td>72 hh ll</td>
<td>rPwO</td>
</tr>
<tr>
<td>INC oprx0_xyssp</td>
<td>IDX</td>
<td>62 xb</td>
<td>rPw</td>
</tr>
<tr>
<td>INC oprx9,xyssp</td>
<td>IDX1</td>
<td>62 xb ff</td>
<td>rPwO</td>
</tr>
<tr>
<td>INC oprx16,xyssp</td>
<td>IDX2</td>
<td>62 xb ee ff</td>
<td>frPwP</td>
</tr>
<tr>
<td>INC [D,xyssp]</td>
<td>[D,IDX]</td>
<td>62 xb</td>
<td>fIFrPw</td>
</tr>
<tr>
<td>INC [oprx16,xyssp]</td>
<td>[IDX2]</td>
<td>62 xb ee ff</td>
<td>fIFrPw</td>
</tr>
</tbody>
</table>

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+W (-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
(5) Question 1. What is the 8-bit hexadecimal representation ($00$ to $\$FF$) for decimal value $-90$?

(5) Question 2. What will be the value of the carry (C) bit after executing the following?
\[
\begin{align*}
&\text{ldab } \#50 \\
&\text{subb } \#210 \\
\end{align*}
\]

(5) Question 3. What will be the value of the overflow (V) bit after executing the following?
\[
\begin{align*}
&\text{ldaa } \#50 \\
&\text{adda } \#-60 \\
\end{align*}
\]

(5) Question 4. A software variable can take on the following specific values $-5.0$, $-4.9$, $-4.8$, ..., $4.8$, $4.9$, $5.0$. Which fixed-point format should be used for this variable? If more than one format could be used to solve the problem, choose the most space-efficient format. Enter the correct letter A-H.

\[
\begin{align*}
&\text{A) 8-bit signed fixed-point, } \Delta = 0.1 \\
&\text{B) 16-bit signed fixed-point, } \Delta = 0.01 \\
&\text{C) 8-bit signed fixed-point, } \Delta = 0.01 \\
&\text{D) 16-bit unsigned fixed-point, } \Delta = 0.001 \\
&\text{E) 16-bit signed fixed-point, } \Delta = 0.1 \\
&\text{F) 8-bit unsigned fixed-point, } \Delta = 0.1 \\
&\text{G) 32-bit floating point} \\
&\text{H) fixed-point is not needed} \\
\end{align*}
\]

(5) Question 5. Rewrite the following equation using fixed-point math, assuming $X, Y, Z$ are integers. Basically, your equation should perform this operation using integer add, subtract, multiply and divide, minimizing the error caused by dropout. Don’t worry about overflow.
\[
Z = 0.123*X - 0.66*Y + 0.5
\]

(5) Question 6. Show the machine code generated by the instruction
\[
\text{inc 1, y+}
\]

(5) Question 7. You are asked to measure a parameter to $3\frac{1}{2}$ decimal digits. What is the approximate precision in binary bits?

(5) Question 8. Assume the SP has been properly initialized. What will be the contents of RegA and RegX after the following six instructions are executed? Hint: draw stack pictures.
\[
\begin{align*}
&\text{ldaa } \#12 \\
&\text{ldx } \#3456 \\
&\text{pshx} \\
&\text{psha} \\
&\text{pulx} \\
&\text{pula} \\
\end{align*}
\]

(10) Question 9. Assume RegX is $3800$, RegY is $3810$, the PC is $4123$, and Ram locations $3800$ to $38FF$ are initially $00$, $01$, ..., $FF$ respectively. E.g., location $3856$ contains $56$. Show the simplified bus cycles occurring when the 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.
\[
\$4123 \ 620A \ \ \text{inc 10, x}
\]

Question 10. Write a main program and a subroutine Check that inputs from Port T bit 6 and outputs to Port T bit 5. The software system will continuously check the status of the input signal, setting the output high if the input ever becomes high. The system never sets the output low. Full credit will be given to the code that is friendly.

(25) Part a) Write the assembly language main program that first initializes the stack. Next it should make Port T bit 6 an input and Port T bit 5 an output. The body of the main program should call the part b) subroutine, Check, over and over without stopping.

(25) Part b) Write the assembly language subroutine called Check, which first reads Port T bit 6. If Port T bit 6 is 1, then set Port T bit 5 equal to 1. If Port T bit 6 is 0, then Port T bit 5 is not changed.