First:_____ Last:_____

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 decimal value…..…	
(5) Part 2a. Specify **`0`** or **`1`** …….	
(5) Part 2b. Specify **`0`** or **`1`** …….	
(5) Question 3. Specify A-H ……..….	
(5) Question 4. Give the range….	
(5) Question 5. Show the machine code….	
(5) Question 6. Give example inputs, specify "none" if none exist.	
(5) Question 7. Give example inputs, specify "none" if none exist.	
(5) Part 8a. What value is pushed?…….	

(10) Part 8b. Simplified memory cycles (you may or may not need all 5 entries)

R/W	Addr	Data	Changes to A,B,X,Y,S,PC,IR,EAR

(15) Question 9. Write the assembly language. (not a subroutine or a main program, just instructions)

```
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
```

(30) Question 10. Write the assembly language subroutine. (Do not include the main program)

```
;*******Set5***********
; Initializes memory $3900 to 397F to value 5
; inputs: none
; outputs: none
; errors: none
Set5









     rts
```

```
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
adda   8-bit add to RegA
addb   8-bit add to RegB
addd   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/lsl   8-bit left shift Memory
asla/lsla 8-bit left shift RegA
aslb/lslb 8-bit arith left shift RegB
asld/lsld 16-bit left shift RegD
asr    8-bit arith right shift Memory
asra   8-bit arith right shift
asrb   8-bit arith right shift to RegB
bcc    branch if carry clear
bclr   clear bits in memory
bcs    branch if carry set
beq    branch if result is zero (Z=1)
bge    branch if signed =
bgnd   enter background debug mode
bgt    branch if signed >
bhi    branch if unsigned >
bhs    branch if unsigned =
bita   8-bit and with RegA, sets CCR
bitb   8-bit and with RegB, sets CCR
ble    branch if signed =
blo    branch if unsigned <
bls    branch if unsigned =
blt    branch if signed <
bmi    branch if result is negative (N=1)
bne    branch if result is nonzero (Z=0)
bpl    branch if result is positive (N=0)
bra    branch always
brclr  branch if bits are clear,
brn    branch never
brset  branch if bits are set
bset   set bits in memory
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
clc    clear carry bit, C=0
cli    clear I=0, enable interrupts
clr    8-bit Memory clear
clra   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
com    8-bit logical complement to Memory
coma   8-bit logical complement to RegA
comb   8-bit logical complement to RegB
cpd    16-bit compare RegD with memory
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
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
ediv   RegY=(Y:D)/RegX, unsigned divide
edivs  RegY=(Y:D)/RegX, signed divide
emacs  16 by 16 signed mult, 32-bit add
emaxd  16-bit unsigned maximum in RegD
emaxm  16-bit unsigned maximum in memory
emind  16-bit unsigned minimum in RegD
eminm  16-bit unsigned minimum in memory
emul   RegY:D=RegY*RegD unsigned mult
emuls  RegY:D=RegY*RegD signed mult
eora   8-bit logical exclusive or to RegA
eorb   8-bit logical exclusive or to RegB
etbl   16-bit look up and interpolation
exg    exchange register contents
fdiv   unsigned fract div, X=(65536*D)/X
ibeq   increment and branch if result=0
ibne   increment and branch if result?0
idiv   16-bit unsigned divide, X=D/X, D=rem
idivs  16-bit signed divide, X=D/X, D=rem
inc    8-bit increment memory
inca   8-bit increment RegA
incb   8-bit increment RegB
ins    16-bit increment RegSP
inx    16-bit increment RegX
iny    16-bit increment RegY
jmp    jump always
jsr    jump to subroutine
lbcc   long branch if carry clear
lbcs   long branch if carry set
lbeq   long branch if result is zero
lbge   long branch if signed =
lbgt   long branch if signed >
lbhi   long branch if unsigned >
lbhs   long branch if unsigned =
lble   long branch if signed =
lblo   long branch if unsigned <
lbls   long branch if unsigned =
lblt   long branch if signed <
lbmi   long branch if result is negative
lbne   long branch if result is nonzero
lbpl   long branch if result is positive
lbra   long branch always
lbrn   long branch never
lbvc   long branch if overflow clear
lbvs   long branch if overflow set
ldaa   8-bit load memory into RegA
ldab   8-bit load memory into RegB
ldd    16-bit load memory into RegD
lds    16-bit load memory into RegSP
ldx    16-bit load memory into RegX
ldy    16-bit load memory into RegY
leas   16-bit load effective addr to SP
leax   16-bit load effective addr to X
leay   16-bit load effective addr to Y
lsr    8-bit logical right shift memory
lsra   8-bit logical right shift RegA
lsrb   8-bit logical right shift RegB
lsrd   16-bit logical right shift RegD
maxa   8-bit unsigned maximum in RegA
maxm   8-bit unsigned maximum in memory
mem    determine the membership grade
mina   8-bit unsigned minimum in RegA
minm   8-bit unsigned minimum in memory
movb   8-bit move memory to memory
movw   16-bit move memory to memory
mul    RegD=RegA*RegB
neg    8-bit 2's complement negate memory
nega   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 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
rora   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
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
tbne   test and branch if result?0
tfr    transfer register to register
tpa    transfer CC to A
trap   illegal instruction interrupt
trap   illegal op code, or software trap
tst    8-bit compare memory with zero
tsta   8-bit compare RegA with zero
tstb   8-bit compare RegB with zero
tsx    transfer S+1 to X
tsy    transfer S+1 to Y
txs    transfer X-1 to S
tys    transfer Y-1 to S
wai    wait for interrupt
wav    weighted Fuzzy logic average
xgdx   exchange RegD with RegX
xgdy   exchange RegD with RegY
```

STX — Store Index Register X — STX

Operation: $(X_H : X_L) \Rightarrow M : M + 1$

Source Form	Address Mode	Object Code	Access Detail
STX opr8a	DIR	**5E dd**	**PW**
STX opr16a	EXT	**7E hh ll**	**PWO**
STX oprx0_xysp	IDX	**6E xb**	**PW**
STX oprx9,xysp	IDX1	**6E xb ff**	**PWO**
STX oprx16,xysp	IDX2	**6E xb ee ff**	**PWP**
STX [D ,xysp]	[D,IDX]	**6E xb**	**PIfW**
STX [oprx16,xysp]	[IDX2]	**6E xb ee ff**	**PIPW**

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)

Motorola 6812 addressing modes **r** is **X**, **Y**, **SP**, or **PC**

00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0,X	−16,X	1,+X	1,X+	0,Y	−16,Y	1,+Y	1,Y+	0,SP	−16,SP	1,+SP	1,SP+	0,PC	−16,PC	n,X	n,SP
5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	9b const	9b const
01	11	21	31	41	51	61	71	81	91	A1	B1	C1	D1	E1	F1
1,X	−15,X	2,+X	2,X+	1,Y	−15,Y	2,+Y	2,Y+	1,SP	−15,SP	2,+SP	2,SP+	1,PC	−15,PC	−n,X	−n,SP
5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	9b const	9b const
02	12	22	32	42	52	62	72	82	92	A2	B2	C2	D2	E2	F2
2,X	−14,X	3,+X	3,X+	2,Y	−14,Y	3,+Y	3,Y+	2,SP	−14,SP	3,+SP	3,SP+	2,PC	−14,PC	n,X	n,SP
5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	16b const	16b const
03	13	23	33	43	53	63	73	83	93	A3	B3	C3	D3	E3	F3
3,X	−13,X	4,+X	4,X+	3,Y	−13,Y	4,+Y	4,Y+	3,SP	−13,SP	4,+SP	4,SP+	3,PC	−13,PC	[n,X]	[n,SP]
5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	16b indr	16b indr
04	14	24	34	44	54	64	74	84	94	A4	B4	C4	D4	E4	F4
4,X	−12,X	5,+X	5,X+	4,Y	−12,Y	5,+Y	5,Y+	4,SP	−12,SP	5,+SP	5,SP+	4,PC	−12,PC	A,X	A,SP
5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	A offset	A offset
05	15	25	35	45	55	65	75	85	95	A5	B5	C5	D5	E5	F5
5,X	−11,X	6,+X	6,X+	5,Y	−11,Y	6,+Y	6,Y+	5,SP	−11,SP	6,+SP	6,SP+	5,PC	−11,PC	B,X	B,SP
5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	B offset	B offset
06	16	26	36	46	56	66	76	86	96	A6	B6	C6	D6	E6	F6
6,X	−10,X	7,+X	7,X+	6,Y	−10,Y	7,+Y	7,Y+	6,SP	−10,SP	7,+SP	7,SP+	6,PC	−10,PC	D,X	D,SP
5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	D offset	D offset
07	17	27	37	47	57	67	77	87	97	A7	B7	C7	D7	E7	F7
7,X	−9,X	8,+X	8,X+	7,Y	−9,Y	8,+Y	8,Y+	7,SP	−9,SP	8,+SP	8,SP+	7,PC	−9,PC	[D,X]	[D,SP]
5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	D indirect	D indirect
08	18	28	38	48	58	68	78	88	98	A8	B8	C8	D8	E8	F8
8,X	−8,X	8,−X	8,X−	8,Y	−8,Y	8,−Y	8,Y−	8,SP	−8,SP	8,−SP	8,SP−	8,PC	−8,PC	n,Y	n,PC
5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	9b const	9b const
09	19	29	39	49	59	69	79	89	99	A9	B9	C9	D9	E9	F9
9,X	−7,X	7,−X	7,X−	9,Y	−7,Y	7,−Y	7,Y−	9,SP	−7,SP	7,−SP	7,SP−	9,PC	−7,PC	−n,Y	−n,PC
5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	9b const	9b const
0A	1A	2A	3A	4A	5A	6A	7A	8A	9A	AA	BA	CA	DA	EA	FA
10,X	−6,X	6,−X	6,X−	10,Y	−6,Y	6,−Y	6,Y−	10,SP	−6,SP	6,−SP	6,SP−	10,PC	−6,PC	n,Y	n,PC
5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	16b const	16b const
0B	1B	2B	3B	4B	5B	6B	7B	8B	9B	AB	BB	CB	DB	EB	FB
11,X	−5,X	5,−X	5,X−	11,Y	−5,Y	5,−Y	5,Y−	11,SP	−5,SP	5,−SP	5,SP−	11,PC	−5,PC	[n,Y]	[n,PC]
5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	16b indr	16b indr
0C	1C	2C	3C	4C	5C	6C	7C	8C	9C	AC	BC	CC	DC	EC	FC
12,X	−4,X	4,−X	4,X−	12,Y	−4,Y	4,−Y	4,Y−	12,SP	−4,SP	4,−SP	4,SP−	12,PC	−4,PC	A,Y	A,PC
5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	A offset	A offset
0D	1D	2D	3D	4D	5D	6D	7D	8D	9D	AD	BD	CD	DD	ED	FD
13,X	−3,X	3,−X	3,X−	13,Y	−3,Y	3,−Y	3,Y−	13,SP	−3,SP	3,−SP	3,SP−	13,PC	−3,PC	B,Y	B,PC
5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	B offset	B offset
0E	1E	2E	3E	4E	5E	6E	7E	8E	9E	AE	BE	CE	DE	EE	FE
14,X	−2,X	2,−X	2,X−	14,Y	−2,Y	2,−Y	2,Y−	14,SP	−2,SP	2,−SP	2,SP−	14,PC	−2,PC	D,Y	D,PC
5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	D offset	D offset
0F	1F	2F	3F	4F	5F	6F	7F	8F	9F	AF	BF	CF	DF	EF	FF
15,X	−1,X	1,−X	1,X−	15,Y	−1,Y	1,−Y	1,Y−	15,SP	−1,SP	1,−SP	1,SP−	15,PC	−1,PC	[D,Y]	[D,PC]
5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	D indirect	D indirect

(5) Question 1. What is the signed integer value (in decimal) of the 8-bit hexadecimal number **$C3**?

Question 2. Consider the result of executing the following two 6812 assembly instructions.

```
ldab #100
subb #210
```

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

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

(5) Question 3. A software variable can take on the following specific values -5.00, -4.99, -4.98, ..., 4.98, 4.99, 5.00. Which number 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.

A) 8-bit signed fixed-point number with resolution of 0.1

B) 8-bit signed fixed-point number with resolution of 0.01

C) 16-bit signed fixed-point number with resolution of 0.01

D) 16-bit unsigned fixed-point number with resolution of 0.001

E) 32-bit floating point

F) 8-bit signed integer

G) 16-bit signed integer

H) 32-bit signed integer

(5) Question 4. A certain ohmmeter has a range of 0 to $\mathbf{R_{max}}$, a resolution of 0.1 Ω, and a precision of 3¾ decimal digits. What is $\mathbf{R_{max}}$?

(5) Question 5. Show the machine code generated by the instruction

```
stx -5,y
```

(5) Question 6. The **mul** instruction multiplies the unsigned value in RegA by the unsigned value in RegB and stores the product in RegD. Give example input values (if any) that cause an overflow.

(5) Question 7. The **idiv** instruction divides the unsigned value in RegD by the unsigned value in RegX and stores the quotient in RegX. Give example input values (if any) that cause an overflow.

Question 8. Consider the following program

```
$5000                                          org  $5000
$5000 CF4000          [ 2](  0){OP        }main lds  #$4000
$5003 0707            [ 4](  2){PPPS      }     bsr  Add1   ;part a)
$5005 CE1234          [ 2](  6){OP        }     ldx  #$1234
$5008 5E02            [ 2](  8){PW        }     stx  2      ;part b)
$500A 183E            [16]( 10){OOSSSfSsf+}     stop
$500C 720240          [ 4]( 26){rOPw      }Add1 inc  $0240
$500F 3D              [ 5]( 30){UfPPP     }     rts
$FFFE                                          org  $FFFE
$FFFE 5000                                     fdb  main
```

(5) Part a) What value(s) is(are) pushed on the stack when the **bsr Add1** instruction is executed?

(10) Part b) Show the simplified bus cycles occurring when the **stx 2** 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.*

(15) Question 9. Write assembly language instructions that make Port T bit 3 an input and Port T bit 2 an output. Full credit will be given to the code that modifies two bits in the direction register, without modifying the other six.

(30) Question 10. Write an assembly language subroutine, called **Set5**, which sets memory locations $3900 to $397F (128 locations) equal to the value $05. Full credit will be given to the code that implements a for-loop and uses the post-increment indexed addressing mode.