

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. 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**

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

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 **RegA=RegA+RegB**. Implement the ceiling operation, such that if an unsigned overflow occurs, set the output to the maximum value, **RegA=255**.

```

;*****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.

```

    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	ediv	RegY=(Y:D)/RegX, unsigned divide
aby	unsigned add RegY=RegY+RegB	edivs	RegY=(Y:D)/RegX, signed divide
adca	8-bit add with carry to RegA	emacs	16 by 16 signed mult, 32-bit add
adcb	8-bit add with carry to RegB	emaxd	16-bit unsigned maximum in RegD
adda	8-bit add to RegA	emaxm	16-bit unsigned maximum in memory
addb	8-bit add to RegB	emind	16-bit unsigned minimum in RegD
addd	16-bit add to RegD	eminm	16-bit unsigned minimum in memory
anda	8-bit logical and to RegA	emul	RegY:D=RegY*RegD unsigned mult
andb	8-bit logical and to RegB	emuls	RegY:D=RegY*RegD signed mult
andcc	8-bit logical and to RegCC	eora	8-bit logical exclusive or to RegA
asl/lsl	8-bit left shift Memory	eorb	8-bit logical exclusive or to RegB
asla/lsla	8-bit left shift RegA	etbl	16-bit look up and interpolation
aslb/lslb	8-bit arith left shift RegB	exg	exchange register contents
asld/lslld	16-bit left shift RegD		exg X,Y
asr	8-bit arith right shift Memory	fdiv	unsigned fract div, X=(65536*D)/X
asra	8-bit arith right shift to RegA	ibeq	increment and branch if result=0
asrb	8-bit arith right shift to RegB		ibeq Y,loop
bcc	branch if carry clear	ibne	increment and branch if result#0
bclr	bit clear in memory		ibne A,loop
	bclr PTT,#\$01	idiv	16-bit unsigned div, X=D/X, D=rem
bcs	branch if carry set	idivs	16-bit signed divide, X=D/X, D=rem
beq	branch if result is zero (Z=1)	inc	8-bit increment memory
bge	branch if signed >	inca	8-bit increment RegA
bgnd	enter background debug mode	incb	8-bit increment RegB
bgt	branch if signed >	ins	16-bit increment RegSP
bhi	branch if unsigned >	inx	16-bit increment RegX
bhs	branch if unsigned >=	iny	16-bit increment RegY
bita	8-bit and with RegA, sets CCR	jmp	jump always
bitb	8-bit and with RegB, sets CCR	jsr	jump to subroutine
ble	branch if signed <=	lbcc	long branch if carry clear
blo	branch if unsigned <=	lbcs	long branch if carry set
bls	branch if signed <=	lbeq	long branch if result is zero
blt	branch if signed <	lbge	long branch if signed >=
bmi	branch if result is negative (N=1)	lbgt	long branch if signed >
bne	branch if result is nonzero (Z=0)	lbhi	long branch if unsigned >
bpl	branch if result is positive (N=0)	lbhs	long branch if unsigned >=
bra	branch always	lble	long branch if signed <=
brclr	branch if bits are clear	lblo	long branch if unsigned <=
	brclr PTT,\$01,loop	lbls	long branch if signed <=
brn	branch never	lbtl	long branch if signed <
brset	branch if bits are set	lbmi	long branch if result is negative
	brset PTT,\$01,loop	lbne	long branch if result is nonzero
bset	bit set clear in memory	lbpl	long branch if result is positive
	bset PTT,\$04	lbra	long branch always
bsr	branch to subroutine	lbrn	long branch never
bvc	branch if overflow clear	lbvc	long branch if overflow clear
bvs	branch if overflow set	lbvs	long branch if overflow set
call	subroutine in expanded memory	ldaa	8-bit load memory into RegA
cba	8-bit compare RegA with RegB	ldab	8-bit load memory into RegB
clc	clear carry bit, C=0	ldd	16-bit load memory into RegD
cli	clear I=0, enable interrupts	lds	16-bit load memory into RegSP
clr	8-bit memory clear	ldx	16-bit load memory into RegX
clra	RegA clear	ldy	16-bit load memory into RegY
clrb	RegB clear	leas	16-bit load effective addr to SP
clv	clear overflow bit, V=0	leax	16-bit load effective addr to X
cmpa	8-bit compare RegA with memory	leay	16-bit load effective addr to Y
cmpb	8-bit compare RegB with memory	lsr	8-bit logical right shift memory
com	8-bit logical complement to memory	lsra	8-bit logical right shift RegA
coma	8-bit logical complement to RegA	lsrb	8-bit logical right shift RegB
comb	8-bit logical complement to RegB	lsrd	16-bit logical right shift RegD
cpd	16-bit compare RegD with memory	maxa	8-bit unsigned maximum in RegA
cpx	16-bit compare RegX with memory	maxm	8-bit unsigned maximum in memory
cpy	16-bit compare RegY with memory	mem	determine the membership grade
daa	8-bit decimal adjust accumulator	mina	8-bit unsigned minimum in RegA
dbeq	decrement and branch if result=0	minm	8-bit unsigned minimum in memory
	dbeq Y,loop	movb	8-bit move memory to memory
dbne	decrement and branch if result#0		movb #100,PTT
	dbne A,loop	movw	16-bit move memory to memory
dec	8-bit decrement memory		movw #13,SCIBD
deca	8-bit decrement RegA	mul	RegD=RegA*RegB
decb	8-bit decrement RegB	neg	8-bit 2's complement negate memory
des	16-bit decrement RegSP	nega	8-bit 2's complement negate RegA
dex	16-bit decrement RegX	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
sbc 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
tab transfer A to B
tap transfer A to CC
tba transfer B to A
tbeq test and branch if result=0
    tbeq Y,loop
tbl 8-bit look up and interpolation
tbne test and branch if result#0
    tbne A,loop
tfr transfer register to register
    tfr X,Y
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 to X
tsy transfer S to Y
txs transfer X to S
tys transfer Y to S
wai wait for interrupt
wav weighted Fuzzy logic average
xgdx exchange RegD with RegX
xgdy exchange RegD with 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+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**

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 .1` `.blk1` Allocate 32-bit words of storage without initialization

BSR

Branch to Subroutine

BSR

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

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.

Source Form	Address Mode	Object Code	Access Detail HCS12
BSR rel8	REL	07 rr	SPPP

ABA

Add Accumulator B to Accumulator A

ABA

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:

S	X	H	I	N	Z	V	C
-	-	Δ	-	Δ	Δ	Δ	Δ

H: $A3 \cdot B3 + B3 \cdot \overline{R3} + \overline{R3} \cdot A3$
 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: $A7 \cdot B7 \cdot \overline{R7} + \overline{A7} \cdot \overline{B7} \cdot R7$
 Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $A7 \cdot B7 + B7 \cdot \overline{R7} + \overline{R7} \cdot A7$
 Set if there was a carry from the MSB of the result; cleared otherwise