

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. Show your work, and put your answers in the boxes.

**Please read the entire quiz before starting.**

**(5) Question 1.** Assume an 8-bit signed integer format. Give the 8-bit binary representation of the value -88.

**(5) Question 2.** When you add two 8-bit unsigned numbers an overflow error can occur. Which of the following techniques can be used to handle the problem of overflow? If there is more than one answer, give all answers that could work.

- A) Write software that is friendly.
- B) Write software using structured programming.
- C) Write software so there is drop out.
- D) Implement a ceiling and floor.
- E) Write software to mask the two input data
- F) Promote the numbers and perform the addition with this new precision.
- G) Demote the numbers and perform the addition with this new precision.
- H) Convert the numbers to signed and perform the addition with signed math.

**(5) Question 3.** Consider the following two instructions

```
ldab #-10
subb #200
```

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

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

What will be the value of the negative (N) bit?

(5) **Question 4.** Consider the result of executing the following three 9S12 assembly instructions.

```
ldaa #100
ldab #3
mul
```

What is the value in Register B after these three instructions are executed?

(5) **Question 5.** We are designing an ohmmeter that measures resistance in the range of 0 to 3,999Ω with a resolution of 1 Ω. What is the precision of this system in **decimal digits**?

(5) **Question 6.** How many bus cycles does it take to execute **bsr** on a real 9S12?

(10) **Question 7.** Assume PC is \$6000, and Register Y is initially \$2233. You may assume location \$0812 and \$0813 are initially 0. Show the simplified bus cycles occurring when the **sty** 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.*

```
$6000 7D0812          sty $0812
```

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

(15) **Question 8.** Draw the circuit diagram interfacing two positive logic switches to PP7 and PP6. Please specify the resistance values for the resistors and the chip numbers (e.g., 7406) for any digital logic you use. You may assume PP7 and PP6 are inputs.



For questions 9 and 10, you will write three subroutines. Don't worry about establishing the reset vector, creating a main program, calling the subroutines or initializing the stack pointer.

(25) **Question 9.** You may use these definitions. **Data** is a signed 8 bit variable.

```
PTT    equ    $0240
DDRT   equ    $0242
        org    $0800    ;RAM
Data   rmb    1        ;signed 8-bit
        org    $4000    ;ROM
```

Part a) Write an assembly subroutine that makes PT7 an output and **Data** equal to 100. Comments are required. Please make the software friendly.

Part b) Write an assembly subroutine that sets PT7 to 1 if **Data**>25, and does not change PT7 if **Data**≤25. Comments are required. Please make the software friendly.

(20) **Question 10.** 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**. Find the maximum of these two unsigned numbers and return the result in **RegA**. (Do not try to use the **maxa** instruction.)

```
;*****Max subroutine*****
;Inputs:  RegA is the first number, RegB is the second number
;Outputs: RegA is the maximum of first and second
Max
```

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 ;clears PT0=0	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	blt	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 in memory	lbpl	long branch if result is positive
	bset PTT,\$04 ;sets PT2=1	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 (A-B)	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=0 clear	ldy	16-bit load memory into RegY
clrb	RegB=0 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 ;moves 100 into PTT
	dbne A,loop	movw	16-bit move memory to memory
dec	8-bit decrement memory		movw #13,SCIBD ;moves 13 into 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	staa	8-bit store memory from RegA
orab	8-bit logical or to RegB	stab	8-bit store memory from RegB
orcc	8-bit logical or to RegCC	std	16-bit store memory from RegD
psha	push 8-bit RegA onto stack	sts	16-bit store memory from SP
pshb	push 8-bit RegB onto stack	stx	16-bit store memory from RegX
pshc	push 8-bit RegCC onto stack	sty	16-bit store memory from RegY
pshd	push 16-bit RegD onto stack	suba	8-bit sub from RegA
pshx	push 16-bit RegX onto stack	subb	8-bit sub from RegB
pshy	push 16-bit RegY onto stack	subd	16-bit sub from RegD
pula	pop 8 bits off stack into RegA	swi	software interrupt, trap
pulb	pop 8 bits off stack into RegB	tab	transfer A to B
pulc	pop 8 bits off stack into RegCC	tap	transfer A to CC
puld	pop 16 bits off stack into RegD	tba	transfer B to A
pulx	pop 16 bits off stack into RegX	tbeq	test and branch if result=0 tbeq Y,loop
puly	pop 16 bits off stack into RegY	tbl	8-bit look up and interpolation
rev	Fuzzy logic rule evaluation	tbne	test and branch if result#0 tbne A,loop
revw	weighted Fuzzy rule evaluation	tfr	transfer register to register tfr X,Y ;transfers X to Y
rol	8-bit roll shift left Memory	tpa	transfer CC to A
rola	8-bit roll shift left RegA	trap	illegal instruction interrupt
rolb	8-bit roll shift left RegB	trap	illegal op code, or software trap
ror	8-bit roll shift right Memory	tst	8-bit compare memory with zero
rora	8-bit roll shift right RegA	tsta	8-bit compare RegA with zero
rorb	8-bit roll shift right RegB	tstb	8-bit compare RegB with zero
rtc	return sub in expanded memory	tsx	transfer S to X
rti	return from interrupt	tsy	transfer S to Y
rts	return from subroutine	txs	transfer X to S
sba	8-bit subtract RegA=RegA-RegB	tys	transfer Y to S
sbca	8-bit sub with carry from RegA	wai	wait for interrupt
sbc	8-bit sub with carry from RegB	wav	weighted Fuzzy logic average
sec	set carry bit, C=1	xgdx	exchange RegD with RegX
sei	set I=1, disable interrupts	xgdy	exchange RegD with RegY
sev	set overflow bit, V=1		
sex	sign extend 8-bit to 16-bit reg sex B,D ;extend B into D		

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.l .blkl	Allocate 32-bit words of storage without initialization

# STY

## Store Index Register Y

# STY

**Operation:**  $(YH : YL) \Rightarrow M : M + 1$

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is \$0000; cleared otherwise

V: 0; cleared

**Description:** Stores the content of index register Y in memory. The most significant byte of Y is stored at the specified address, and the least significant byte of Y is stored at the next higher byte address (the specified address plus one).

Source Form	Address Mode	Object Code	Access Detail HCS12
STY opr16a	EXT	7D hh ll	PWO

# 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

# RTS

## Return from Subroutine

# RTS

**Operation:**  $(M(SP) : M(SP+1)) \Rightarrow PCH : PCL; (SP) + \$0002 \Rightarrow SP$

**Description:** Restores context at the end of a subroutine. Loads the program counter with a 16-bit value pulled from the stack and increments the stack pointer by two. Program execution continues at the address restored from the stack.

Source Form	Address Mode	Object Code	Access Detail HCS12
RTS	INH	3D	UfPPP