

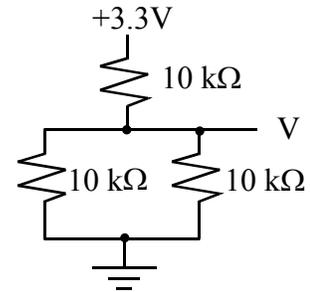
First: _____ Last: _____

This is a closed book exam. You must put your answers in the space provided. You have 3 hours, so allocate your time accordingly. *Please read the entire exam before starting.*

Please read and affirm our honor code:

“The core values of The University of Texas at Austin are learning, discovery, freedom, leadership, individual opportunity, and responsibility. Each member of the university is expected to uphold these values through integrity, honesty, trust, fairness, and respect toward peers and community.”

(5) **Question 1.** What is the voltage at V? Show your work.



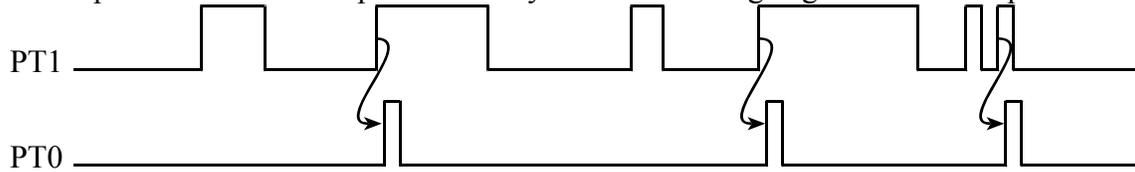
(5) **Question 2.** The PLL is not active, and the E clock frequency is 8 MHz. The **TCNT** timer is active with **TSCR2** equal to 2. The ADC is active with **ATDOCTR4=\$03**, so that the ADC clock is 1 MHz. The SCI1 baud rate is 9600 bits/sec. Output compare 7 is used to sample the 10-bit ADC once every interrupt. The data is stored in a buffer. The following code occurs in the output compare 7 ISR

```
TC7 = TC7 + 100;
```

(3) **Part a)** What is the time period between output compare 7 interrupts?

(2) **Part b)** What is the largest frequency component faithfully represented in the data in the buffer?

(10) Question 3. Initialize PTT so that PT1 is an input and PT0 is an output. Write a C program that issues a short pulse on the PT0 output after every other each rising edge of the PT1 input.



Include *all the software* for this system. You may use the standard port names, such as **PTT**. Interrupts and the timer are not needed. It does not matter how long the pulse width is on PT0.

(20) Question 4. You can solve this problem in assembly or in C, your choice. This is a simplex communication channel; there is SCI1 output, but no SCI1 input. In C, the user will call your function by reference. In assembly, the user calls your function by reference using Reg D

<pre>void user(void) { SCI1_Output("Hello"); }</pre>	<pre>Msg fcb "Hello",0 User ldd #Msg jsr SCI1_Output rts</pre>
--	--

Your **SCI1_Output** includes SCI1 initialization and output. The data is null-terminated. You may assume the E clock is 8 MHz. The desired baud rate is 5000 Hz. You are allowed to add additional RAM-based variables. Write the **SCI1_Output** function, the SCI1 interrupt service routine, and the code to set the SCI1 interrupt vector. Disarm SCI1 interrupts after the last character is sent. You do not have to transmit the null character. For full credit you must implement the SCI1 output using interrupt synchronization. For a possible 5 points out of 20, you solve this problem with busy-wait.

(6) Question 5. Your embedded system uses a 12-bit ADC to sample the fluid flow through an oil pipe. The system will display flow rate on an LCD. The flow can vary from -20 to +20 L/min. The appropriate transducer and analog circuit maps in a linear fashion the full scale flow rate into the 0 to +5V full scale range of the ADC. I.e., -20 L/min maps to 0V and +20 L/min maps to +5V. The ADC uses straight binary: 0V maps to 0 and +5V maps to the largest digital output.

(2) Part a) What ADC value do you get if the flow rate is -10 L/min.

(2) Part b) What resolution will you use for fixed-point number system to store the oil flow data?

(2) Part c) What precision will you use for fixed-point number system to store the oil flow data?

(4) Question 6. Consider the result of executing the following two 9S12 assembly instructions.

```
ldaa #1
adda #-1
```

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

What will be the value of the zero (Z) bit?

(5) **Question 7.** Give the simplified memory cycles produced when the following one instruction is executed. Initially, PC is \$4065, **DDRT** is \$FF, and **PTT** is \$08. Just show R/W=Read or Write, Address, and Data for each cycle. You may need more or less entries than the 5 boxes given.

```
$4065 1C024001      bset PTT,#$01
```

R/W	Addr	Data

(5) **Question 8.** For each activity choose the debugging term that *best* matches. Choose answers from the word bank. Not all words in the bank will be used. No word is used twice.

Word bank: heartbeat, breakpoint, drop out, scanpoint, friendly, profile, minimally intrusive, highly intrusive, bandwidth, monitor, real-time, stabilization, volatile

Activity	Debugging term
Measuring where and when software executes	
Recording data during execution without pausing	
Debugging with a small but inconsequential effect on the system itself	
Adding a LCD to display important variables during execution; the LCD is not part of the necessary components of the system.	
Flashing an LED letting the user know the software is running	

(15) Question 9. Design and implement a FIFO that can hold up to 4 elements. Each element is 3 bytes. There will be three subroutines: **Initialization**, **Put** one element into FIFO and **Get** one element from the FIFO.

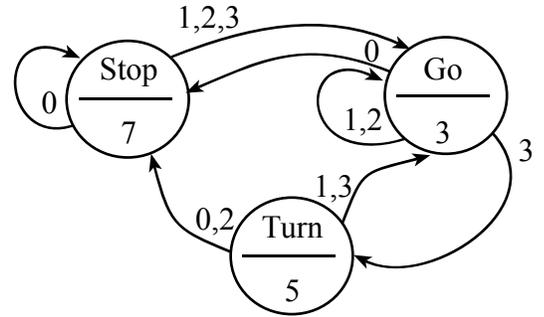
(4) Part a) Show the RAM-based variables are available, and NO additional storage may be allocated

(4) Part b) Write an assembly function that initializes the FIFO.

(4) Part c) Write an assembly function that puts one 3-byte element into the FIFO. The input parameter is call by reference using Register Y. The output parameter is returned in Register D, 0 for failure because the FIFO was full, and 1 for success because the data was properly stored.

(3) Part d) Write an assembly function that gets one 3-byte element from the FIFO. When your Get function is called, Register Y points to an empty 3-byte RAM buffer. The output parameters are return by value in Register D and return by reference through Register Y. If data can be removed, three bytes are copied into the buffer pointed to by Register Y and Register D is returned as 1. If no data can be removed, because the FIFO was empty when the function was called, return Register D equal to 0.

(5) **Question 10.** Assume the initial state is Stop and the input is a constant value of 3. What will be the sequence of outputs?



(10) **Question 11.** Consider output compare 7 interrupts. Assume the name of the interrupt service routine is **TC7Handler**.

(3) **Part a)** What three events in general need to be true for any interrupt to occur? Furthermore, give those three events specifically for output compare 7.

(4) **Part b)** List the events that occur as the computer switches from running in the foreground to running an output compare 7 interrupt in the background? E.g., the 9S12 is running the main program, *stuff happens*, the 9S12 is running **TC7Handler**. List the events in *stuff happens*.

(3) **Part c)** Write assembly code to acknowledge an output compare 7 interrupt.

(10) Question 12. In this question, you will translate the C code into 9S12 assembly. You must use stack frame *binding* using Register Y for the **c** and **d** parameters within the **main** program. I.e., both local variables must be allocated on the stack. You may assume **PTT** is an input port. Use **org** statements to place the program at an appropriate place within the 9S12, and include the reset vector.

```
void main(void) {
unsigned short c;
unsigned char d;
    c = 0;
    for(;;) {
        d = PTT;
        if(d&0x01) {
            c = c+d;
        }
    }
}
```

aba	8-bit add RegA=RegA+RegB	des	16-bit decrement RegSP
abx	unsigned add RegX=RegX+RegB	dex	16-bit decrement RegX
aby	unsigned add RegY=RegY+RegB	dey	16-bit decrement RegY
adca	8-bit add with carry to RegA	ediv	RegY=(Y:D)/RegX, unsigned divide
adcb	8-bit add with carry to RegB	edivs	RegY=(Y:D)/RegX, signed divide
adda	8-bit add to RegA	emacs	16 by 16 signed multiply, 32-bit add
addb	8-bit add to RegB	emaxd	16-bit unsigned maximum in RegD
addd	16-bit add to RegD	emaxm	16-bit unsigned maximum in memory
anda	8-bit logical and to RegA	emind	16-bit unsigned minimum in RegD
andb	8-bit logical and to RegB	eminm	16-bit unsigned minimum in memory
andcc	8-bit logical and to RegCC	emul	RegY:D=RegY*RegD unsigned multiply
asl/lsl	8-bit left shift Memory	emuls	RegY:D=RegY*RegD signed multiply
asla/lsla	8-bit left shift RegA	eora	8-bit logical exclusive or to RegA
aslb/lslb	8-bit arith left shift RegB	eorb	8-bit logical exclusive or to RegB
asld/lsl d	16-bit left shift RegD	etbl	16-bit look up and interpolation
asr	8-bit arith right shift Memory	exg	exchange register contents exg X,Y
asra	8-bit arith right shift to RegA	fdiv	unsigned fract div, X=(65536*D)/X
asrb	8-bit arith right shift to RegB	ibeq	increment and branch if result=0 ibeq Y,loop
bcc	branch if carry clear	ibne	increment and branch if result≠0 ibne A,loop
bclr	bit clear in memory bclr PTT,#\$01	idiv	16-bit unsigned div, X=D/X, D=remainder
bcs	branch if carry set	idivs	16-bit signed divide, X=D/X, D= remainder
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 unsigned ≤	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	lbl	long branch if signed ≤
brclr	branch if bits are clear brclr PTT,#\$01,loop	lblo	long branch if unsigned <
brn	branch never	lbls	long branch if unsigned ≤
brset	branch if bits are set brset PTT,#\$01,loop	lblt	long branch if signed <
bset	bit set in memory bset PTT,#\$04	lbmi	long branch if result is negative
bsr	branch to subroutine	lbne	long branch if result is nonzero
bvc	branch if overflow clear	lbp1	long branch if result is positive
bvs	branch if overflow set	lbra	long branch always
call	subroutine in expanded memory	lbrn	long branch never
cba	8-bit compare RegA with RegB, RegA-RegB	lbvc	long branch if overflow clear
clc	clear carry bit, C=0	lbvs	long branch if overflow set
cli	clear I=0, enable interrupts	ldaa	8-bit load memory into RegA
clr	8-bit memory clear	ldab	8-bit load memory into RegB
clra	RegA clear	ladd	16-bit load memory into RegD
clrb	RegB clear	lds	16-bit load memory into RegSP
clv	clear overflow bit, V=0	ldx	16-bit load memory into RegX
cmpa	8-bit compare RegA with memory	ldy	16-bit load memory into RegY
cmpb	8-bit compare RegB with memory	leas	16-bit load effective addr to SP leas 2,sp
com	8-bit logical complement to memory	leax	16-bit load effective addr to X leax 2,x
coma	8-bit logical complement to RegA	leay	16-bit load effective addr to Y leay 2,y
comb	8-bit logical complement to RegB	lsr	8-bit logical right shift memory
cpd	16-bit compare RegD with memory	lsra	8-bit logical right shift RegA
cpx	16-bit compare RegX with memory	lsrb	8-bit logical right shift RegB
cpy	16-bit compare RegY with memory	lsrd	16-bit logical right shift RegD
daa	8-bit decimal adjust accumulator	maxa	8-bit unsigned maximum in RegA
dbeq	decrement and branch if result=0 dbeq Y,loop	maxm	8-bit unsigned maximum in memory
dbne	decrement and branch if result≠0 dbne A,loop	mem	determine the Fuzzy logic membership grade
dec	8-bit decrement memory	mina	8-bit unsigned minimum in RegA
deca	8-bit decrement RegA	minm	8-bit unsigned minimum in memory
decb	8-bit decrement RegB	movb	8-bit move memory to memory movb #100,PTT

movw 16-bit move memory to memory movw #13,SCIBD
mul unsigned 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=RegA-RegB
sbca 8-bit sub with carry from RegA
sbc b 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	Mode	Effective Address
ldaa #u	immediate	No EA
ldaa u	direct	EA is 8-bit address
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-incr	r=r+v, EA=r (1 to 8)
ldaa v,-r	pre-dec	r=r-v, EA=r (1 to 8)
ldaa v,r+	post-inc	EA=r, r=r+v (1 to 8)
ldaa v,r-	post-dec	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
ldaa W,r	16-bit index	EA=r+W
ldaa [D,r]	D indirect	EA={r+D}
ldaa [W,r]	indirect	EA={r+W}

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

Pseudo op	Meaning
org	Where to put subsequent code
= equ set	Define a constant symbol
dc.b db fcb .byte	Allocate byte(s) with values
fcc	Create an ASCII string
dc.w dw fdb .word	Allocate word(s) with values
dc.l dl .long	Allocate 32-bit with values
ds ds.b rmb .blkb	Allocate bytes without init
ds.w .blkw	Allocate word(s) without init

n is Metrowerks number

Vector	n	Interrupt Source	Arm
\$FFFE		Reset	None
\$FFF8	3	Trap	None
\$FFF6	4	SWI	None
\$FFF0	7	Real time interrupt	CRGINT.RTIE
\$FFE0	8	Timer channel 0	TIE.C0I
\$FFEC	9	Timer channel 1	TIE.C1I
\$FFEA	10	Timer channel 2	TIE.C2I
\$FFE8	11	Timer channel 3	TIE.C3I
\$FFE6	12	Timer channel 4	TIE.C4I
\$FFE4	13	Timer channel 5	TIE.C5I
\$FFE2	14	Timer channel 6	TIE.C6I
\$FFE0	15	Timer channel 7	TIE.C7I
\$FFDE	16	Timer overflow	TSCR2.TOI
\$FFD6	20	SCIO TDRE, RDRF	SCIOCR2.TIE,RIE
\$FFD4	21	SCII TDRE, RDRF	SCII CR2.TIE,RIE
\$FFCE	24	Key Wakeup J	PIEJ.[7,6,1,0]
\$FFCC	25	Key Wakeup H	PIEH.[7:0]
\$FF8E	56	Key Wakeup P	PIEP.[7:0]

Interrupt Vectors and interrupt number.

Address	Bit 7	6	5	4	3	2	1	Bit 0	Name
\$0040	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0	TIOS
\$0044-5	Bit 15	14	13	12	11	10		Bit 0	TCNT
\$0046	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0	TSCR1
\$004C	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I	TIE
\$004D	TOI	0	PUPT	RDPT	TCRE	PR2	PR1	PR0	TSCR2
\$004E	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F	TFLG1
\$004F	TOF	0	0	0	0	0	0	0	TFLG2
\$0050-1	Bit 15	14	13	12	11	10		Bit 0	TC0
\$0052-3	Bit 15	14	13	12	11	10		Bit 0	TC1
\$0054-5	Bit 15	14	13	12	11	10		Bit 0	TC2
\$0056-7	Bit 15	14	13	12	11	10		Bit 0	TC3
\$0058-9	Bit 15	14	13	12	11	10		Bit 0	TC4
\$005A-B	Bit 15	14	13	12	11	10		Bit 0	TC5
\$005C-D	Bit 15	14	13	12	11	10		Bit 0	TC6
\$005E-F	Bit 15	14	13	12	11	10		Bit 0	TC7
\$0082	ADPU	AFFC	ASWAI	ETRIGLE	ETRIGP	ETRIG	ASCIE	ASCIF	ATD0CTL2
\$0083	0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0	ATD0CTL3
\$0084	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0	ATD0CTL4
\$0085	DJM	DSGN	SCAN	MULT	0	CC	CB	CA	ATD0CTL5
\$0086	SCF	0	ETORF	FIFOR	0	CC2	CC1	CC0	ATD0STAT0
\$008B	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0	ATD0STAT1
\$008D	Bit 7	6	5	4	3	2	1	Bit 0	ATD0DIEN
\$008F	PAD07	PAD06	PAD05	PAD04	PAD03	PAD02	PAD01	PAD00	PORTAD0
\$0090-1	Bit 15	14	13	12	11	10		Bit 0	ATD0DR0
\$0092-3	Bit 15	14	13	12	11	10		Bit 0	ATD0DR1
\$0094-5	Bit 15	14	13	12	11	10		Bit 0	ATD0DR2
\$0096-7	Bit 15	14	13	12	11	10		Bit 0	ATD0DR3
\$0098-9	Bit 15	14	13	12	11	10		Bit 0	ATD0DR4
\$009A-B	Bit 15	14	13	12	11	10		Bit 0	ATD0DR5
\$009C-D	Bit 15	14	13	12	11	10		Bit 0	ATD0DR6
\$009E-F	Bit 15	14	13	12	11	10		Bit 0	ATD0DR7
\$00C9	0	0	0	SBR12	SBR11	SBR10		SBR0	SCIOBD
\$00CA	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT	SCIOCR1
\$00CB	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCIOCR2
\$00CC	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	SCIOSR1
\$00CD	0	0	0	0	0	BRK13	TXDIR	RAF	SCIOSR2
\$00CF	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0	SCIODRL
\$00D0-1	0	0	0	SBR12	SBR11	SBR10		SBR0	SCII1BD
\$00D2	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT	SCII1CR1
\$00D3	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCII1CR2
\$00D4	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	SCII1SR1
\$00D5	0	0	0	0	0	BRK13	TXDIR	RAF	SCII1SR2
\$00D7	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0	SCII1DRL
\$0240	PT7	PT6	PT5	PT4	PT3	PT2	PT1	PT0	PTT
\$0242	DDRT7	DDRT6	DDRT5	DDRT4	DDRT3	DDRT2	DDRT1	DDRT0	DDRT
\$0248	PS7	PS6	PS5	PS4	PS3	PS2	PS1	PS0	PTS
\$024A	DDRS7	DDRS6	DDRS5	DDRS4	DDRS3	DDRS2	DDRS1	DDRS0	DDRS
\$0250	PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0	PTM
\$0252	DDRM7	DDRM6	DDRM5	DDRM4	DDRM3	DDRM2	DDRM1	DDRM0	DDRM
\$0258	PP7	PP6	PP5	PP4	PP3	PP2	PP1	PP0	PTP
\$025A	DDRP7	DDRP6	DDRP5	DDRP4	DDRP3	DDRP2	DDRP1	DDRP0	DDRP
\$0260	PH7	PH6	PH5	PH4	PH3	PH2	PH1	PH0	PTH
\$0262	DDRH7	DDRH6	DDRH5	DDRH4	DDRH3	DDRH2	DDRH1	DDRH0	DDRH
\$0268	PJ7	PJ6	0	0	0	0	PJ1	PJ0	PTJ
\$026A	DDRJ7	DDRJ6	0	0	0	0	DDRJ1	DDRJ0	DDRJ

TSCR1 is the first 8-bit timer control register

bit 7 **TEN**, 1 allows the timer to function normally, 0 means disable timer including **TCNT**

TIOS is the 8-bit output compare select register, one bit for each channel (1 = output compare, 0 = input capture)

TIE is the 8-bit output compare arm register, one bit for each channel (1 = armed, 0 = disarmed)

TSCR2 is the second 8-bit timer control register

bits 2,1,0 are **PR2, PR1, PR0**, which select the rate, let **n** be the 3-bit number formed by **PR2, PR1, PR0** without PLL **TCNT** is $8\text{MHz}/2^n$, with PLL **TCNT** is $24\text{MHz}/2^n$, **n** ranges from 0 to 7

PR2	PR1	PR0	Divide by	E = 8 MHz		E = 24 MHz	
				TCNT period	TCNT frequency	TCNT period	TCNT frequency
0	0	0	1	125 ns	8 MHz	41.7 ns	24 MHz
0	0	1	2	250 ns	4 MHz	83.3 ns	12 MHz
0	1	0	4	500 ns	2 MHz	167 ns	6 MHz
0	1	1	8	1 μs	1 MHz	333 ns	3 MHz
1	0	0	16	2 μs	500 kHz	667 ns	1.5 MHz
1	0	1	32	4 μs	250 kHz	1.33 μs	667 kHz
1	1	0	64	8 μs	125 kHz	2.67 μs	333 kHz
1	1	1	128	16 μs	62.5 kHz	5.33 μs	167 kHz

SCI1DRL 8-bit SCI1 data register

SCI1BD is 16-bit SCI1 baud rate register, let **n** be the 13-bit number Baud rate is $\text{EClk}/n/16$

SCI1CR1 is 8-bit SCI1 control register

bit 4 M, Mode, 0 = One start, eight data, one stop bit, 1 = One start, eight data, ninth data, one stop bit

SCI1CR2 is 8-bit SCI1 control register

bit 7 TIE, Transmit Interrupt Enable, 0 = TDRE interrupts disabled, 1 = interrupt whenever TDRE set

bit 5 RIE, Receiver Interrupt Enable, 0 = RDRF interrupts disabled, 1 = interrupt whenever RDRF set

bit 3 TE, Transmitter Enable, 0 = Transmitter disabled, 1 = SCI transmit logic is enabled

bit 2 RE, Receiver Enable, 0 = Receiver disabled, 1 = Enables the SCI receive circuitry.

SCI1SR1 is 8-bit SCI1 status register

bit 7 TDRE, Transmit Data Register Empty Flag

Set if transmit data can be written to **SCI1DRL**

Cleared by **SCI1SR1** read with TDRE set followed by **SCI1DRL** write

bit 5 RDRF, Receive Data Register Full

set if a received character is ready to be read from **SCI1DRL**

Clear the RDRF flag by reading **SCI1SR1** with RDRF set and then reading **SCI1DRL**

BSET

Set Bit(s) in Memory

Operation: $(M) + (\text{Mask}) \Rightarrow M$

Source Form	Address Mode ⁽¹⁾	Object Code	HCS12
BSET <i>opr8a, msk8</i>	DIR	4C dd mm	rPwO
BSET <i>opr16a, msk8</i>	EXT	1C hh ll mm	rPwP
BSET <i>opr0_xy, msk8</i>	IDX	0C xb mm	rPwO
BSET <i>opr9_xy, msk8</i>	IDX1	0C xb ff mm	rPwP
BSET <i>opr16_xy, msk8</i>	IDX2	0C xb ee ff mm	frPwPO

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

Z: Set if result is \$00; cleared otherwise

V: $A7 \cdot M7 \cdot \overline{R7} + \overline{A7} \cdot \overline{M7} \cdot R7$

Set if two's complement overflow resulted from the operation; cleared otherwise

C: $A7 \cdot M7 + M7 \cdot \overline{R7} + \overline{R7} \cdot A7$

Set if there was a carry from the MSB of the result; cleared otherwise

ADDA

Add without Carry to A

Operation: $(A) + (M) \Rightarrow A$