

First: _____ Middle Initial: _____ 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.”

Signed: _____

December 12, 2008

(4) Question 1. An embedded system will use an 11-bit ADC to measure a distance. The measurement system range is -5 to +5 m. The frequency components of the distance signal can vary from DC (0 Hz) up to 200 Hz. You will use a periodic output compare interrupt to sample the ADC. What rate (in Hz) should you activate an output compare interrupt? Give a brief explanation.

(4) Question 2. A 1-ms periodic output compare interrupt is used to spin a stepper motor. During each ISR the four-bit motor output is set to 5, 6, 10, then 9. The stepper is interfaced to PT3-0, and the following four instructions occur during each ISR, without any delay between these instructions.

```
movb #$05,PTT
movb #$06,PTT
movb #$0A,PTT
movb #$09,PTT
```

There are 200 steps per rotation of the motor. What will happen?

- A) The motor will spin at 5 rps = $(1 \text{ step/ms}) * (1000\text{ms/s}) * (1 \text{ rot}/200\text{steps})$
- B) The motor will spin at 20 rps = $(4 \text{ step/ms}) * (1000\text{ms/s}) * (1 \text{ rot}/200\text{steps})$
- C) The motor will spin at 1000 rps = $(1 \text{ rot/ms}) * (1000\text{ms/s})$
- D) The motor will spin at 4000 rps = $(4 \text{ rot/ms}) * (1000\text{ms/s})$
- E) The motor will not spin at all

(4) Question 3. Write a subroutine to sample ADC channel 5 of the 9S12DP512. Assume the ADC initialized for a 10-bit sample, sequence length is 1, and the ADC clock is 1 MHz. Implement right-justified conversions, and return the result in RegY.

(8) Question 4. Assume you have a 12-bit ADC with a range of 0 to +4 V (not the 9S12). Write a subroutine that converts the ADC sample into a fixed-point number with a resolution of 0.001 V. The input parameter is call by value in RegD containing the right-justified ADC sample (0 to 4095). Minimize errors due to dropout and overflow. Return by value the integer part of the fixed-point number in RegY. E.g., if the input voltage is 1.25 V then RegY is returned as 1250.

(4) Question 5. Assume RegA = \$55, RegY=\$1234 and RegX = \$5678. What is the value in RegX after executing these instructions?

```

psha
stx 2,-sp
sty 2,sp-
leas 2,sp
pula
pulx

```

(4) Question 6. These seven events all occur during each RDRF interrupt.

- 1) There is data in the receive data register and the hardware sets the flag bit (e.g., RDRF=1)
- 2) The SCI vector address is loaded into the PC
- 3) The I bit in the CCR is set by hardware
- 4) The software reads **SCI1DRL**
- 5) The software reads **SCI1SR1**
- 6) The CCR, A, B, X, Y, PC are pushed on the stack
- 7) The software executes **rti**

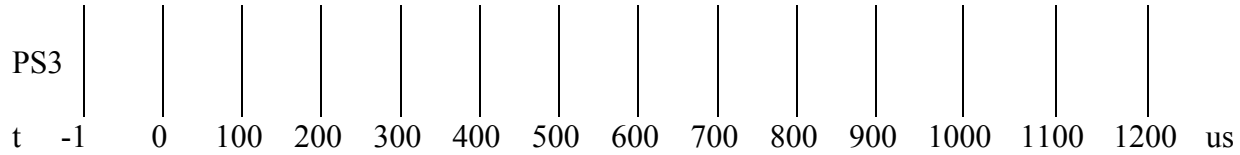
Which of the following sequences could be possible? Pick one answer A-F (only one is correct)

- A) 1,3,6,2,4,5,7
- B) 5,1,3,4,2,6,7
- C) 1,2,5,3,4,6,7
- D) 1,6,3,2,5,4,7
- E) 1,6,3,2,4,5,7
- F) None of the above sequences are possible

(10) Question 7. Write software that increments a 16-bit global variable every 2 msec using output compare 3. Show the complete main program, the OC3 ISR, the interrupt vector, and the reset vector. After initialization the main program executes a do-nothing loop. Write code as friendly as possible. Assume the E clock is 8 MHz. To make it easier for me to grade, leave TSCR2 equal to 0.

```
org $0800
Count rmb 2 ;incremented every 2 msec
org $4000
```

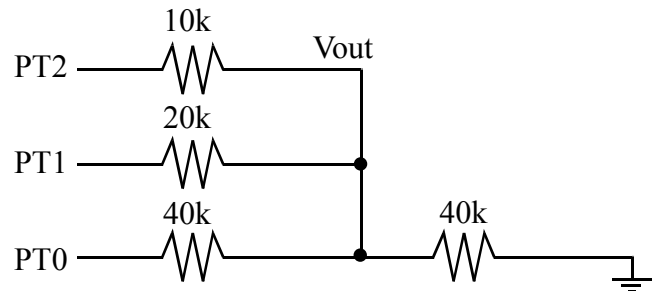
(5) Question 8. Consider a serial port operating with a baud rate of 10,000 bits per second. Draw the waveform occurring at the PS3 output (voltage levels are +5 and 0) when the ASCII 'S' (53) is transmitted on SCI1. The protocol is 1 start, 8 data and 1 stop bit. SCI1 is initially idle, and the software writes the 53 to SCI1DRL at time=0. Show the PS3 line before and after the frame, assuming the channel is idle before and after the frame.



(5) Question 9. Consider a computer network where two 9S12s are connected via their SCI1 ports, using the 3-wire cable like in Lab 7. The transmitter of computer 1 is connected to the receiver of computer 2, and the transmitter of computer 2 is connected to the receiver of computer 1. Initially, both SCI1 ports are idle. The baud rate on both computers is initialized to 10000 bits/sec, with 1 start, 8 data and 1 stop bit. Both computers have their RDRF flags armed and enabled. The transmitters are active, but not armed for interrupts. The I bit is clear in both computers. At time 0, computer 1 reads SCI1SR1 then writes to SCI1DRL. The RDRF ISR in computer 1 will read its SCI1SR1 then write to its SCI1DRL. Approximately how long after computer 1 writes to SCI1DRL will an RDRF interrupt occur back in computer 1? Assume the software execution time is fast compared to the I/O transmission time.

(4) Question 10. Consider the following three-bit DAC connected to Port T. Fill in the expected response table assuming V_{OH} is 5 V and V_{OL} is 0 V.

PT2	PT1	PT0	Vout (V)
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

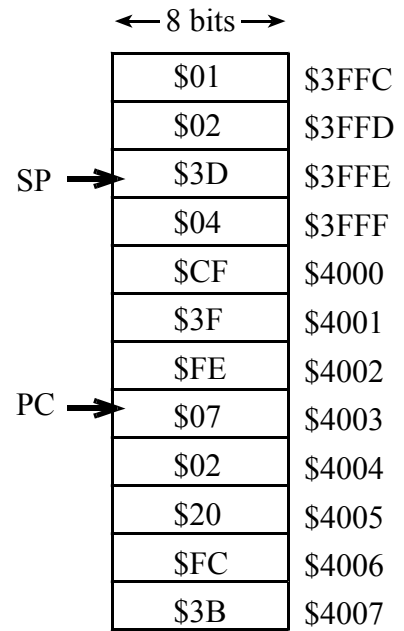


(8) Question 11. Assume the PC equals \$4003, and the SP equals \$3FFE. Initially, memory contains data as shown in the figure. You will be executing one instruction and answering questions about executing that one instruction.

Part a) Given the initial conditions in this figure, what instruction will be executed next?

Part b) As you execute that one instruction, two bytes are stored into memory? Give the addresses and the 8-bit data values that are stored.

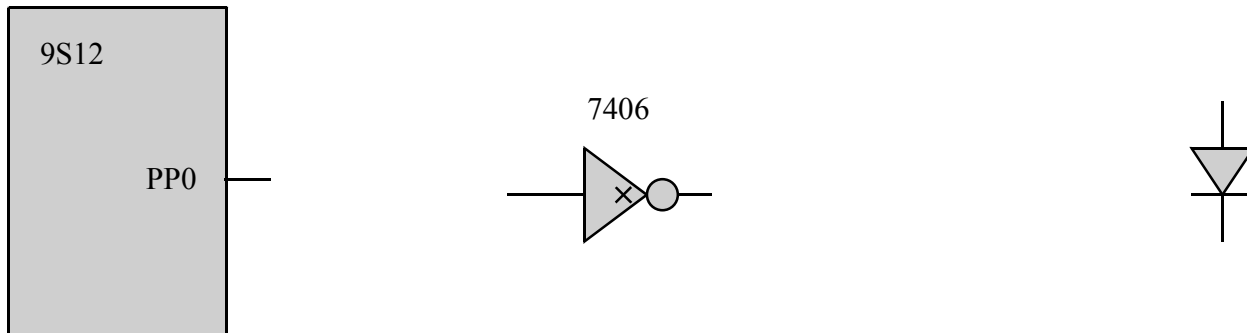
Address	Data



Part c) What is the SP after the one instruction is executed?

Part d) What is the PC after the one instruction is executed?

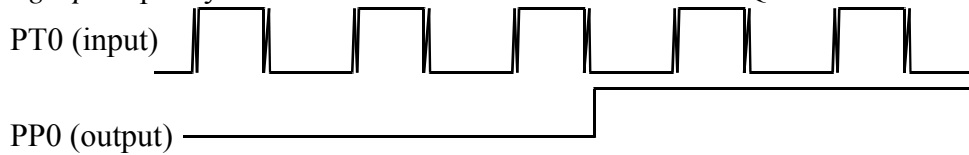
(5) Question 12. You are given an LED with a (1.5 V, 30 mA) operating point. Interface this LED to the 9S12 using a 7406, such that the LED is on when PP0 is high and the LED is off when PP0 is low. The V_{OL} of the 7406 is 0.5 V. Label all resistor values. No software is required.



(10) Question 13. A positive logic switch is connected to PT0 and a positive logic LED is connected to PP0. Design a Moore finite state machine that counts the number of times the switch is pressed and released, so that the LED is turned on if the switch is pressed 3 or more times. The LED should come on after the switch is released the third time. Switch bounce causes the input to toggle low/high/low/high every time the switch is touched, and to toggle high/low/high/low every time the switch is released. This bounce is typically less than 1ms. You may assume the switch input is high for at least 100 ms when touched and low for at least 100 ms when released. In other words, the maximum rate at which the operator will push the switch is 5 times/sec. To eliminate switch bounce, you will read the input at a rate slower than every 10 ms, but faster than every 100 ms. The FSM controller will repeat this sequence in the foreground over and over

- 1) Output to the LED, as defined by the state
- 2) Wait a prescribed amount of time, as defined by the state
- 3) Input from the switch
- 4) Go to the next state, as defined by the state and by the input

Draw the FSM graph. Specify the initial state. NO SOFTWARE IS REQUIRED.



(10) Question 14. In this question, the subroutine implements a call by reference parameter passed on the stack. There are no return parameters. Call by reference means an address to the data is pushed on the stack. A typical calling sequence is

```

    org    $4000
Data fcb  200          ;8-bit information
Main lds  #$4000
    movw  #Data,2,-sp ;pointer to the Data is pushed
    jsr   Subroutine
    leas  2,sp        ;discard parameter

```

The subroutine allocates one 8-bit local variable, **L1**, and uses **RegX** frame pointer addressing to access the local variable and parameter. The binding for these three are

```

Pt set   ??? ;16-bit pointer to 8-bit data
L1 set   ??? ;8-bit local variable

```

```

Subroutine
    pshx          ;save old stack frame pointer
    tsx           ;establish new stack frame pointer
    leas -1,sp    ;allocate L1
;-----start of body-----
    ldaa ??????  ;Reg A = value of the parameter
    staa L1,x    ;save parameter into local L1
;-----end of body-----
    leas 1,sp    ;deallocate
    pulx
    rts

```

Part a) Show the binding for the ??? parameters in the above program.

Pt set _____

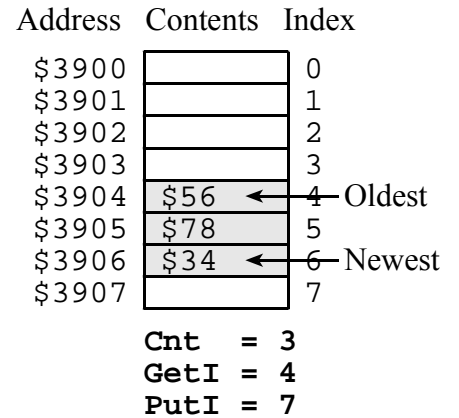
L1 set _____

Part b) Show the operand for the ?????? in the above program. In particular, you must use Register X stack frame addressing, **Pt** binding, and bring the value of the parameter into Register A. It can be done in one instruction, but for partial credit you can use two instructions.

(15) Question 15. This FIFO queue has 8 allocated locations and can hold up to eight 8-bit data values. The picture shows it currently holding three values (shaded). The FIFO and its three variables are defined in RAM. When the counter is zero the FIFO is empty.

```

    org $3900
    Fifo rmb 8 ;allocates 8 bytes
    GetI rmb 1 ;index where to find oldest data
    PutI rmb 1 ;index where to put next data
    Cnt rmb 1 ;number of elements stored in fifo
    This function initializes the FIFO
    Fifo_Init clr Cnt ;no data in Fifo
              clr GetI ;Get next from Fifo[GetI]
              clr PutI ;Put next into Fifo[PutI]
              rts
    
```



Write an assembly subroutine, **Fifo_Put**, that implements the put operation. The input parameter contains the data to put as call by value in RegA, and a result code is returned in RegB. If RegB=1, then the input data was successfully stored. If RegB=0, the data could not be saved in the FIFO because it was previously full at the time of the call.

```

;input: RegA,    Output: RegB=success
Fifo_Put
    
```

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	emacsb	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/lsld	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 clear 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	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 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 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
pulg 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	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

Vector	Interrupt Source	Arm
\$FFFE	Reset	None
\$FFF8	Trap	None
\$FFF6	SWI	None
\$FFF0	Real time interrupt	CRGINT.RTIE
\$FFEE	Timer channel 0	TIE.C0I
\$FFEC	Timer channel 1	TIE.C1I
\$FFEA	Timer channel 2	TIE.C2I
\$FFE8	Timer channel 3	TIE.C3I
\$FFE6	Timer channel 4	TIE.C4I
\$FFE4	Timer channel 5	TIE.C5I
\$FFE2	Timer channel 6	TIE.C6I
\$FFE0	Timer channel 7	TIE.C7I
\$FFDE	Timer overflow	TSCR2.TOI
\$FFD6	SCI0 TDRE, RDRF	SCI0CR2.TIE,RIE
\$FFD4	SCI1 TDRE, RDRF	SCI1CR2.TIE,RIE
\$FFCE	Key Wakeup J	PIEJ.[7,6,1,0]
\$FFCC	Key Wakeup H	PIEH.[7:0]
\$FF8E	Key Wakeup P	PIEP.[7:0]

Interrupt Vectors.

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	SCI0BD
\$00CA	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT	SCI0CR1
\$00CB	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCI0CR2
\$00CC	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	SCI0SR1
\$00CD	0	0	0	0	0	BRK13	TXDIR	RAF	SCI0SR2
\$00CF	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0	SCI0DRL
\$00D0-1	0	0	0	SBR12	SBR11	SBR10		SBR0	SCI1BD
\$00D2	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT	SCI1CR1
\$00D3	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCI1CR2
\$00D4	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	SCI1SR1
\$00D5	0	0	0	0	0	BRK13	TXDIR	RAF	SCI1SR2
\$00D7	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0	SCI1DRL
\$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**

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

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)

SCIxBD is 16-bit SCI baud rate register, let **n** be the 16-bit number Baud rate is $250\text{ kHz}/n$

SCIxCR2 is 8-bit SCI 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.

SCIxSR1 is 8-bit SCI status register

- bit 7 TDRE, Transmit Data Register Empty Flag
Set if transmit data can be written to SCDR
Cleared by **SCIxSR1** read with TDRE set followed by **SCIxDRL** write.
- bit 5 RDRF, Receive Data Register Full
set if a received character is ready to be read from **SCIxDRL**
Clear the RDRF flag by reading **SCIxSR1** with RDRF set and then reading **SCIxDRL**.

LDAA

Load Accumulator A

Operation: $(M) \Rightarrow A$

Description: Loads the content of memory location M into accumulator A. The condition codes are set according to the data.

CCR Details:

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	0	-

N: Set if MSB of result is set; cleared otherwise
Z: Set if result is \$00; cleared otherwise
V: 0; cleared

LDAA

Source Form	Address Mode	Object Code
LDAA #opr8i	IMM	86 ii
LDAA opr8a	DIR	96 dd
LDAA opr16a	EXT	B6 hh ll
LDAA oprx0_xysp	IDX	A6 xb
LDAA oprx9_xysp	IDX1	A6 xb ff
LDAA oprx16_xysp	IDX2	A6 xb ee ff
LDAA [D,xysp]	[D,IDX]	A6 xb
LDAA [opr16,xysp]	[IDX2]	A6 xb ee ff

EMUL

Extended Multiply
16-Bit by 16-Bit (Unsigned)

Operation: $(D) \times (Y) \Rightarrow Y : D$

Source Form	Address Mode	Object Code
EMUL	INH	13

EDIV

Extended Divide 32-Bit by 16-Bit
(Unsigned)

Operation: $(Y : D) \div (X) \Rightarrow Y; \text{Remainder} \Rightarrow D$

Source Form	Address Mode	Object Code
EDIV	INH	11

BSR

Branch to Subroutine

Operation: $(SP) - \$0002 \Rightarrow SP$
 $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$
 $(PC) + \text{Rel} \Rightarrow PC$

Source Form	Address Mode	Object Code
BSR rel8	REL	07 rr

RTS

Return from Subroutine

Operation: $M_{(SP)} : M_{(SP+1)} \Rightarrow PC_H : PC_L; (SP) + \$0002 \Rightarrow SP$

Source Form	Address Mode	Object Code
RTS	INH	3D