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."

(4) Question 1. You are working for an engineering firm and have been given the task to design a new DAC. After completing the design and construction of an initial prototype, you now have to test it. List four experimental parameters you will measure to evaluate the quality of your new DAC. These parameters will be measurable with test equipment, like you used in lab or like the professor/TAs demonstrated to you in this class. For each parameter, give both the name of the parameter and the definition of the parameter. You do not need to give the experimental procedure, rather simply list four names and four definitions of the parameters. I do not mean physical parameters like size, weight, or color. Furthermore, I do not mean economic factors like production costs, profit, mean time between failures, or maintenance costs.

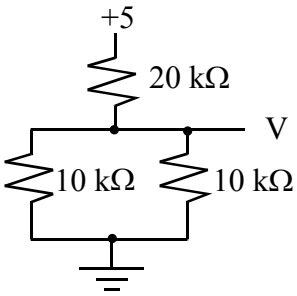
Part a) Give a parameter and its definition

Part b) Give a parameter and its definition

Part c) Give a parameter and its definition

Part d) Give a parameter and its definition

(6) Question 2. What is the voltage V?



_____

(10) Question 3. Write a C function to sample ADC channels 3 and 4 of the 9S12DP512. Assume the ADC initialized for a 10-bit sample, sequence length is 2, and the ADC clock is 1 MHz. Implement right-justified conversions, and return by value the larger of the two ADC samples.

(15) Question 4. Write three C functions that operate the SCI0 module.
(5) Part a) Write an initialization function that turns on both the transmitter and receiver for the SCI0. Set the baud rate equal to 38400 bits/sec. You may assume the E clock is 8 MHz.

(5) Part b) Write a C function that outputs one ASCII character to SCI0 using busy-wait synchronization and call by value parameter passing.

(5) Part c) Write a C function that outputs an ASCII string to the SCI0. You may assume the variable length string is null-terminated. Use call by reference parameter passing.

(5) Question 5. Write assembly code that implements the unsigned operation RegD = 0.3456*RegX, eliminating overflow and minimizing dropout errors.

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

```
ldaa #156
suba #-50
```

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

_____

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

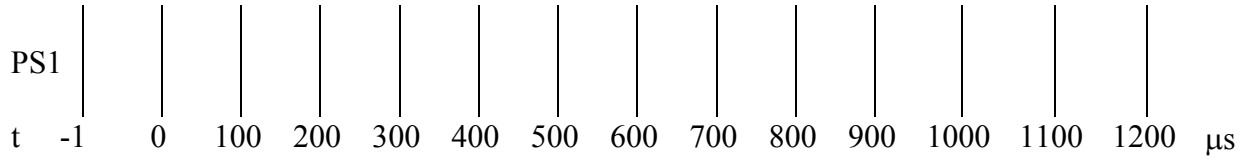
_____

(5) Question 7. Give the simplified memory cycles produced when the following one instruction is executed. Assume the PC contains $4200, and the SP equals $3FF2. Just show R/W=Read or Write, Address, and Data for each cycle. You may not need all 5 entries in the solution box.

```
$4200 070E   bsr $4210
```

R/W	Addr	Data

(5) Question 8. Consider a serial port operating with a baud rate of 10,000 bits per second. Draw the waveform occurring at the PS1 output (voltage levels are +5 and 0) when the ASCII 'a' ($61) is transmitted on SCI0. The protocol is 1 start, 8 data and 1 stop bit. SCI0 is initially idle, and the software writes the $61 to SCI0DRL at time=0. Show the PS1 line before and after the frame, assuming the channel is idle before and after the frame.

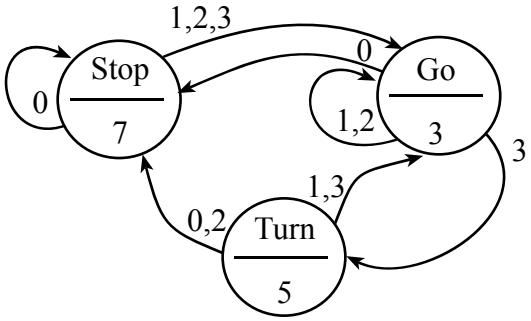


(5) Question 9. For each application choose the data structure that best matches. Choose answers from the word bank.

Word bank: first in first out queue, buffer, last in first out stack, linked list, table

Application	**Data structure**
A data structure used to implement buffered I/O.	
A data structure used to store calibration data	
A data structure used to implement a local variables	
A data structure used to implement a debugging dump	
A data structure used to implement a Mealy machine	

(20) Question 10. Write assembly code to implement the following FSM using output compare interrupts. After initialization, the entire FSM will run within the output compare 0 ISR. The command sequence will be output, wait 1ms, input, then branch to next state. There can be NO backward jumps or conditional branches within the ISR. The 2-bit input is on Port H (PH1 and PH0) and the 3-bit output is on Port T (PT2, PT1, PT0). Assume the E clock is 8 MHz. You may use a 16-bit pointer named **`Pt`**, located in RAM.



(5) Part a) Show the assembly code defining the FSM graph in ROM.

(7) Part b) Show the assembly subroutine to initialize the FSM controller, arm OC0, and enable interrupts. Please initialize the state pointer **`Pt`** and perform the first output.

(8) Part c) Show the assembly language for the output compare 0 ISR that runs the FSM including interrupt vector.

(10) Question 11. In this question, you will translate the C code line by line into 9S12 assembly. The assembly code for the global variables and main are given. The two parameters are passed into **Add** using call by reference, and both MUST be pushed on the stack. You must use SP-relative binding for the **p1** and **p2** parameters within the **Add** function.

`short D1;`
`short D2;`
`void main(void){`
`  D1 = -1000;`
`  D2 = +500;`
`  for(;;){`
`  Run();`
`  }`
`}`
	`  org  $0800`
	`D1  rmb  2`
	`D2  rmb  2`
	`  org  $4000`
	`main  lds  #$4000`
	`  movw  #-1000,D1`
	`  movw  #500,D2`
	`loop  bsr  Run`
	`  bra  loop`

`void Run(void){`	
`  Add(&D1,&D2);`	
`}`	

`void Add(short *p1,`
`  short *p2){`
	`p1  set  _____  ;SP relative binding`
	`p2  set  _____  ;SP relative binding`
	`Add`

`  *p2 = *p2 + *p1;`	
`}`	

(10) Question 12. For each definition, fill in the term that best matches. Choose answers from the word bank. Some words will not be used.

Word bank: accuracy, arm, baud rate, acknowledge , bandwidth, big endian, ceiling, disarm, drop out, enable, friendly, intrusive, little endian, nonintrusive, noninvasive, nonvolatile, open collector, overflow, real-time, resolution, stabilization, thread, tristate, volatile

Definition	**Term**
Activate an individual trigger so that interrupts are requested when that trigger flag is set.	
The information transfer rate, the amount of real data transferred per second.	
A measure of how close our instrument measures the desired parameter referred to the NIST.	
A debugging process that fixes all the inputs to a system	
Mechanism for storing multiple byte numbers such that the most significant byte exists first	
Establishing an upper bound on the result of an operation.	
Clearing the interrupt trigger flag that requested the interrupt.	
An error that occurs after a right shift or a divide, loosing its ability to represent all of the values.	
A characteristic when the presence of the collection of information itself does not affect the parameters being measured.	
A digital logic output that has two states low and HiZ.	

`aba` 8-bit add RegA=RegA+RegB
`abx` unsigned add RegX=RegX+RegB
`aby` unsigned add RegY=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 to RegA
`asrb` 8-bit arith right shift to RegB
`bcc` branch if carry clear
`bclr` bit clear in memory `bclr PTT,#$01`
`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 `brclr PTT,#$01,loop`
`brn` branch never
`brset` branch if bits are set `brset PTT,#$01,loop`
`bset` bit set in memory `bset PTT,#$04`
`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, RegA-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 `dbeq Y,loop`
`dbne` decrement and branch if result≠0 `dbne A,loop`
`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 multiply, 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 multiply
`emuls` RegY:D=RegY*RegD signed multiply
`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 `exg X,Y`
`fdiv` unsigned fract div, X=(65536*D)/X
`ibeq` increment and branch if result=0 `ibeq Y,loop`
`ibne` increment and branch if result≠0 `ibne A,loop`
`idiv` 16-bit unsigned div, X=D/X, D=remainder
`idivs` 16-bit signed divide, X=D/X, D= remainder
`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 `leas 2,sp`
`leax` 16-bit load effective addr to X `leax 2,x`
`leay` 16-bit load effective addr to Y `leay 2,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 Fuzzy logic membership grade
`mina` 8-bit unsigned minimum in RegA
`minm` 8-bit unsigned minimum in memory
`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
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   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
$FFEE	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	**SCI0 TDRE, RDRF**	SCI0CR2.TIE,RIE
$FFD4	21	**SCI1 TDRE, RDRF**	SCI1CR2.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	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**

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 8MHz/2^{n}, with PLL **TCNT** is 24MHz/2^{n}, **n** ranges from 0 to 7

				E = 8 MHz		E = 24 MHz	
PR2	PR1	PR0	Divide by	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

SCI0DRL 8-bit SCI0 data register

SCI0BD is 16-bit SCI0 baud rate register, let **n** be the 13-bit number Baud rate is EClk/**n**/16

SCI0CR1 is 8-bit SCI0 control register

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

SCI0CR2 is 8-bit SCI0 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.

SCI0SR1 is 8-bit SCI0 status register

bit 7 TDRE, Transmit Data Register Empty Flag
Set if transmit data can be written to **SCI0DRL**
Cleared by **SCI0SR1** read with TDRE set followed by **SCI0DRL** write

bit 5 RDRF, Receive Data Register Full
set if a received character is ready to be read from **SCI0DRL**
Clear the RDRF flag by reading **SCI0SR1** with RDRF set and then reading **SCI0DRL**

EMUL

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

Operation: (D) × (Y) ⇒ Y : D

Source Form	Address Mode	Object Code
EMUL	INH	13

EDIV

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

Operation: (Y : D) ÷ (X) ⇒ Y; Remainder ⇒ D

Source Form	Address Mode	Object Code
EDIV	INH	11

BSR

Branch to Subroutine

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

Source Form	Address Mode	Object Code
RTS	INH	3D