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.*

**Question 1.** An embedded system will use a 12-bit ADC to measure a parameter. The measurement system range is 0 to 10 cm. What is the precision in decimal digits?

_________

**Question 2.** An 11-bit ADC (not the 9S12) has an input range of 0 to +10 volts and an output range of 0 to 2047. What digital value will be returned when an input of +2.5 volts is sampled?

_________

**Question 3.** Consider the result of executing the following two 9S12 assembly instructions.

```
ldaa #160
suba #150
```

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

_________

**Question 4.** Consider the result of executing the following two 9S12 assembly instructions.

```
ldaa #-30
adda #-90
```

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

_________
(4) Question 5. Assume all 8 bits of PTT are output. Write software to clear PT5 (make it 0).

(4) Question 6. What is the bug in the following initialized global variable on the 9S12?

```
org $3900
Slope fdb 50
```

A) $3900 is not RAM  
B) RAM is volatile  
C) RAM is nonvolatile  
D) 50 is an 8-bit number, and fdb defines a 16-bit number  
E) Using global variables is poor style and should never be used.  
F) No error, this definition is acceptable.

(4) Question 7. These seven events all occur during each output compare 7 interrupt.

1) The TCNT equals TC7 and the hardware sets the flag bit (e.g., C7F=1)  
2) The output compare 7 vector address is loaded into the PC  
3) The I bit in the CCR is set by hardware  
4) The software executes `movb #$80,TFLG1`  
5) The CCR, A, B, X, Y, PC are pushed on the stack  
6) The software executes something like

```
ldd TC7  
addd #1000  
std TC7
```
  
7) The software executes `rti`

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

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

(4) Question 8. Assume the E clock is 4 MHz (250 ns) and TSCR2 = 1. At what interrupt period will the output compare 7 interrupt described in Question 7 occur? GIVE UNITS

__________
(4) **Question 9.** What is the machine code for the following instruction?
   \[
   \text{sty } 2, \text{sp} +
   \]
   __________

(4) **Question 10.** Is this a legal stack operation? Answer yes or no
   \[
   \text{sty } 2, \text{sp} +
   \]
   __________

(4) **Question 11.** Assume \textit{Height} is the integer part of an 8-bit unsigned fixed-point variable with a resolution of 0.1 cm. The goal is to add 0.5 cm to the value of the variable. Will the following software always operate properly?
   \[
   \begin{align*}
   \text{ldaa} & \quad \text{Height} \\
   \text{sex} & \quad A, D \ ; \text{promote to 16 bits} \\
   \text{addd} & \quad #5 \ ; \text{perform the addition in 16-bit mode} \\
   \text{tfr} & \quad D, A \ ; \text{demote back to 8 bits} \\
   \text{staa} & \quad \text{Height}
   \end{align*}
   \]
   A) Yes, the program has no errors.
   B) No, overflow can occur.
   C) No, dropout can occur. __________
   D) No, the carry bit could be set
   E) No, one needs to divide by 10 to get the correct result.
   F) No, the \textit{addd} instruction should have been \textit{addd} \ #0.5

(5) **Question 12.** Give the simplified memory cycles produced when the following one instruction is executed. Assume the PC contains $4200$, and the SP equals $3FF0$. Just show R/W=Read or Write, Address, and Data for each cycle. You may not need all 5 entries in the solution box.
   \[
   \begin{align*}
   $4200 & \quad 070E \quad \text{bsr } \quad $4210
   \end{align*}
   \]

<table>
<thead>
<tr>
<th>R/W</th>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(5) **Question 13.** You are given an LED with a 3V 10mA operating point. Interface this LED to the 9S12 using a 7406, such that the LED is on when PM1 is high (5V) and the LED is off when PM1 is low (0V). Label all resistor values. No software is required.

(5) **Question 14.** You are given a *double-pole* switch that has three pins. The figure shows the switch in the position that occurs when the switch is pressed. If the switch is pressed, pins 1 and 2 are connected (0 resistance) and pins 2 and 3 are not connected (infinite resistance). If the switch is not pressed, pins 2 and 3 are connected (0 resistance) and pins 1 and 2 are not connected (infinite resistance). Pins 1 and 3 are never connected (it is a *break-before-make* switch). Interface this switch to the 9S12, such that PM0 is high (5V) if the switch is pressed and PM0 is low (0V) if the switch is not pressed. You do not need to debounce the switch. Label all chip numbers and resistor values. No software is required.
Question 15. In this problem you will implement three unsigned 8-bit local variables on the stack using Reg X stack frame addressing and symbolic binding. The variables are called front, center, and back. The code in this question is part of a subroutine, which ends in rts.

Part a) Show the assembly code that (in this order) saves Register X, establishes the Register X stack frame, and allocates the three 8-bit local variables.

Part b) Assume the stack pointer is equal to $3F0A just before jsr instruction is executed that calls this subroutine. Draw a stack picture showing the return address, the three variables, Register X, and the stack pointer SP. Cross-out the SP arrow and move it to its new location.

Part c) Show the symbolic binding for front, center, and back.

Part d) Show code that implements center=100; using Reg X stack frame addressing.

Part e) Show the assembly code that deallocates the local variables, and restores Reg X.
(10) **Question 16.** Write an assembly subroutine that starts the ADC to sample channel 2, waits for ADC to finish, then reads one 10-bit conversion from the ADC. You may assume the ADC interface is already initialized to sample one channel in 10-bit mode. Use busy-wait synchronization and return the result by value in Register X. The result should vary from 0 to 1023.

(10) **Question 17.** Write an assembly subroutine that waits for new input, then reads one 8-bit character from the SCI serial port. You may assume the serial port is already initialized to 1 start bit, 8 data bits, and 1 stop bit, running at 9600 bits/sec. Use busy-wait synchronization and return the result by value in Register B.
(15) Question 18. Write an assembly main program that implements this Mealy finite state machine. The FSM data structure, shown below, is given and cannot be changed. The next state links are defined as 16-bit pointers. Each state has 8 outputs and 8 next-state links. The input is on Port T bits 2,1,0 and the output is on Port M bits 5,4,3,2,1,0. There are three states (S0,S1,S2), and initial state is S0. Show all assembly software required to execute this machine including the reset vector. You need not be friendly, but do initialize the direction registers. The repeating execution sequence is input, output (depends on input and current state), next (depends on input and current state).

```
or  $4000 ; EPROM

* Finite State Machine
S0  fcb  0,0,5,6,3,9,3,0  ; Outputs for inputs 0 to 7
    fdb  S0,S0,S1,S1,S1,S2,S2,S2  ; Next states for inputs 0 to 7
S1  fcb  1,2,3,9,6,5,3,3  ; Outputs for inputs 0 to 7
    fdb  S2,S0,S0,S0,S2,S2,S2,S1  ; Next states for inputs 0 to 7
S2  fcb  1,2,3,9,6,5,3,3  ; Outputs for inputs 0 to 7
    fdb  S2,S2,S2,S2,S0,S0,S2,S1  ; Next states for inputs 0 to 7
```
dey 16-bit decrement RegY
dex 16-bit decrement RegX
des 16-bit decrement RegSP
decb 8-bit decrement RegB
deca 8-bit decrement RegA
dec 8-bit decrement memory
dbne A,loop
bls branch if unsigned
blo branch if unsigned <
bhs branch if unsigned ≥
bhi branch if unsigned >
bgt branch if result >
bgnd enter background debug mode
bh branch if unsigned >
bs branch if unsigned ≥
bits 8-bit and with RegA, sets CCR
bitb 8-bit and with RegB, sets CCR
ble branch if signed ≤
bl branch if unsigned <
bls branch if unsigned ≤
bit 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 always
brcr branch if bits are clear
bclr PTT,#$01,loop
bc<byte> branch if carry set
bge branch if signed ≥
bqn branch if signed ≥
bgt branch if signed >
bhi branch if unsigned >
bhs branch if unsigned ≥
blo branch if unsigned <
bis branch if unsigned <
bis branch if unsigned ≤
bint branch if overflow set
inca 8-bit increment RegA
incb 8-bit increment RegB
inc 8-bit increment memory
bcs branch if carry set
bgeq branch if result is zero (Z=1)
bgeq branch if result=0
bicm branch if overflow set
bicp branch if signed ≥
bvc branch if overflow clear
bvs branch if overflow set
bcv branch if overflow clear
bvc branch if overflow clear
call subroutine in expanded memory
cla clear carry bit, C=0
cle clear I=0, enable interrupts
clib 8-bit memory clear
crla RegA clear
cr cb RegB clear
cle 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
decm 8-bit decrement RegA
decb 8-bit decrement RegB
dex 16-bit decrement RegX
dey 16-bit decrement RegY
orcc  8-bit logical or to RegCC
psra push 8-bit RegA onto stack
psrb push 8-bit RegB onto stack
psrc push 8-bit RegCC onto stack
psrd push 16-bit RegD onto stack
psrx push 16-bit RegX onto stack
pulc pop 8 bits off stack into RegC
pulm pop 16 bits off stack into RegM
push 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
puly pop 16 bits off stack from RegY
pulm pop 8 bits off stack into RegM

orcc  8-bit logical or to RegCC
psha push 8-bit RegA onto stack
psrb push 8-bit RegB onto stack
psrc push 8-bit RegCC onto stack
psrd push 16-bit RegD onto stack
psrx push 16-bit RegX onto stack
pulc pop 8 bits off stack into RegC
pulm pop 16 bits off stack into RegM
push 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
puly pop 16 bits off stack from RegY
pulm pop 8 bits off stack into RegM

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
ror a 8-bit roll shift right RegA
rorb 8-bit roll shift right RegB
rtc return 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
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
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
tbx transfer B to A
tbq test and branch if result=0

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
ror a 8-bit roll shift right RegA
rorb 8-bit roll shift right RegB
rtc return 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
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
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
tbx transfer B to A
tbq test and branch if result=0

Freescale 6812 addressing modes

```
<table>
<thead>
<tr>
<th>example</th>
<th>addressing mode</th>
<th>Effective Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld da #u immediate</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>ld da u direct</td>
<td>EA is 8-bit address (0 to 255)</td>
<td></td>
</tr>
<tr>
<td>ld da U extended</td>
<td>EA is a 16-bit address</td>
<td></td>
</tr>
<tr>
<td>ld da m,r 5-bit index</td>
<td>EA=m+EA (0 to 255)</td>
<td></td>
</tr>
<tr>
<td>ld da v,r pre-increment</td>
<td>r=v, EA=r (1 to 8)</td>
<td></td>
</tr>
<tr>
<td>ld da v,r+ post-increment</td>
<td>r=v+r, EA=r (1 to 8)</td>
<td></td>
</tr>
<tr>
<td>ld da v,r- post-decrement</td>
<td>r=v, EA=r (1 to 8)</td>
<td></td>
</tr>
<tr>
<td>ld da A,r Reg A offset</td>
<td>EA=A, zero padded</td>
<td></td>
</tr>
<tr>
<td>ld da B,r Reg B offset</td>
<td>EA=B, zero padded</td>
<td></td>
</tr>
<tr>
<td>ld da D,r Reg D offset</td>
<td>EA=D</td>
<td></td>
</tr>
<tr>
<td>ld da q,r 9-bit index</td>
<td>EA=q (256 to 555)</td>
<td></td>
</tr>
<tr>
<td>ld da W,r 16-bit index</td>
<td>EA=W (-32768 to 65535)</td>
<td></td>
</tr>
<tr>
<td>ld da [D,r] D indirect</td>
<td>EA=r+D</td>
<td></td>
</tr>
<tr>
<td>ld da [W,r] indirect</td>
<td>EA=r+W (-32768 to 65535)</td>
<td></td>
</tr>
</tbody>
</table>
```

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 word(s) of storage with initialized values
- **ds ds.b rmb .blkb** Allocate 32-bit long word(s) of storage with initialized values
- **ds .w .blkw** Allocate bytes of storage without initialization
- **ds .l .blkl** Allocate 32-bit words of storage without initialization
### Address Bit 7 6 5 4 3 2 1 Bit 0 Name

<table>
<thead>
<tr>
<th>Address</th>
<th>0082</th>
<th>0083</th>
<th>0084</th>
<th>0085</th>
<th>0086</th>
<th>008B</th>
<th>008D</th>
<th>0270</th>
<th>0272</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADPU</td>
<td>AFFC</td>
<td>AWAI</td>
<td>ETRIGLE</td>
<td>ETRIGP</td>
<td>ETRIG</td>
<td>ASCIE</td>
<td>ASCIF</td>
<td>ATDCTL2</td>
</tr>
<tr>
<td></td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
<td></td>
<td>ATDCTL3</td>
</tr>
<tr>
<td></td>
<td>SRES8</td>
<td>SMP1</td>
<td>SMP0</td>
<td>PRS4</td>
<td>PRS3</td>
<td>PRS2</td>
<td>PRS1</td>
<td>PRS0</td>
<td>ATDCTL4</td>
</tr>
<tr>
<td></td>
<td>DJM</td>
<td>DSGN</td>
<td>SCAN</td>
<td>MULT</td>
<td>0</td>
<td>CC</td>
<td>CB</td>
<td>CA</td>
<td>ATDCTL5</td>
</tr>
<tr>
<td></td>
<td>SCF</td>
<td>ETORF</td>
<td>FIFOR</td>
<td>0</td>
<td>CC2</td>
<td>CC1</td>
<td>CC0</td>
<td></td>
<td>ATDSTAT0</td>
</tr>
<tr>
<td></td>
<td>$0083</td>
<td>0</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
</tr>
<tr>
<td></td>
<td>$0084</td>
<td>0</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
</tr>
<tr>
<td></td>
<td>$0085</td>
<td>0</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
</tr>
<tr>
<td></td>
<td>$0086</td>
<td>0</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
</tr>
<tr>
<td></td>
<td>$0087</td>
<td>0</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
</tr>
<tr>
<td></td>
<td>$0088</td>
<td>0</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
</tr>
<tr>
<td></td>
<td>$0089</td>
<td>0</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
</tr>
<tr>
<td></td>
<td>$008A</td>
<td>0</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
</tr>
<tr>
<td></td>
<td>$008B</td>
<td>0</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
</tr>
<tr>
<td></td>
<td>$008C</td>
<td>0</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
</tr>
<tr>
<td></td>
<td>$008D</td>
<td>0</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
</tr>
<tr>
<td></td>
<td>$008E</td>
<td>0</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZI</td>
<td>FRZO</td>
</tr>
</tbody>
</table>

### Address Bit 7 6 5 4 3 2 1 Bit 0 Name

<table>
<thead>
<tr>
<th>Address</th>
<th>0090</th>
<th>0092</th>
<th>0094</th>
<th>0096</th>
<th>0098</th>
<th>009A</th>
<th>009C</th>
<th>009E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0090</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0092</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0094</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0096</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0098</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$009A</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$009C</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$009E</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Address Bit 7 6 5 4 3 2 1 Bit 0 Name

<table>
<thead>
<tr>
<th>Address</th>
<th>0090</th>
<th>0092</th>
<th>0094</th>
<th>0096</th>
<th>0098</th>
<th>009A</th>
<th>009C</th>
<th>009E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0090</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0092</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0094</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0096</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0098</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$009A</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$009C</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$009E</td>
<td>15 14 13 12</td>
<td>11 10 9 8 7</td>
<td>6 5 4 3</td>
<td>2 1 0</td>
<td>ATDDRO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Vector Address Interrupt Source or Trigger flag Enable Local Arm

<table>
<thead>
<tr>
<th>Address</th>
<th>0046</th>
<th>004D</th>
<th>0040</th>
<th>004C</th>
<th>004E</th>
<th>004F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TEN</td>
<td>TSWAI</td>
<td>TSBCK</td>
<td>TFFCA</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>TOI</td>
<td>0</td>
<td>0</td>
<td>TCRI</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>IOS7</td>
<td>IOS6</td>
<td>IOS5</td>
<td>IOS4</td>
<td>IOS3</td>
<td>IOS2</td>
</tr>
<tr>
<td></td>
<td>C7I</td>
<td>C6I</td>
<td>C5I</td>
<td>C4I</td>
<td>C3I</td>
<td>C2I</td>
</tr>
<tr>
<td></td>
<td>C7F</td>
<td>C6F</td>
<td>C5F</td>
<td>C4F</td>
<td>C3F</td>
<td>C2F</td>
</tr>
<tr>
<td></td>
<td>TOF</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**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 $4MHz/2^n$, with PLL TCNT is $24MHz/2^n$, $n$ 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)
### SCIBD

SCIBD is 16-bit SCI baud rate register, let \( n \) be the 16-bit number. Baud rate is \( 250 \text{ kHz} / n \)

### SCICR2

SCICR2 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.

### SCISR1

SCISR1 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 SCISR1 read with TDRE set followed by SCIDRL write.
- **bit 5** RDRF, Receive Data Register Full
  - Set if a received character is ready to be read from SCIDRL
  - Clear the RDRF flag by reading SCISR1 with RDRF set and then reading SCIDRL.

### STY

**Operation:** \[ (Y_n, Y_{n+1}) = M \cdot M + 1 \]

**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 1).

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>STY opr6a</td>
<td>DIR</td>
<td>5D dd</td>
</tr>
<tr>
<td>STY opr16a</td>
<td>EXT</td>
<td>7D hh 11</td>
</tr>
<tr>
<td>STY oprx0.0yxp</td>
<td>IDX</td>
<td>6D xh</td>
</tr>
<tr>
<td>STY oprx0.0xyxp</td>
<td>IDX1</td>
<td>6D xh ff</td>
</tr>
<tr>
<td>STY oprx1.0yxp</td>
<td>IDX2</td>
<td>6E xh ee ff</td>
</tr>
<tr>
<td>STY [D.0yxp]</td>
<td>[D,IDX]</td>
<td>6D xh</td>
</tr>
<tr>
<td>STY [oprx1.0xyxp]</td>
<td>[IDX2]</td>
<td>6D xh ee ff</td>
</tr>
</tbody>
</table>

### BSR

**Operation:** \( (\text{SP}) - $0002 = \text{SP} \)

RTN:\( H, L = M(\text{SP}) : M(\text{SP+1}) \)

(\( PC \) + \( \text{Rel} \) = \( PC \))

**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.

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSR rel8</td>
<td>REL</td>
<td>07 rr</td>
</tr>
</tbody>
</table>