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 a 12-bit ADC to measure a distance. The measurement system range is -2 to +2 m. The frequency components of the distance signal can vary from DC (0 Hz) up to 400 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.

\[
\begin{align*}
\text{movb} & \#05, \text{PTT} \\
\text{movb} & \#06, \text{PTT} \\
\text{movb} & \#0A, \text{PTT} \\
\text{movb} & \#09, \text{PTT}
\end{align*}
\]

There are 200 steps per rotation of the motor. What will happen?
A) The motor will not spin at all
B) The motor will spin at 20 rps = (4 step/ms)\((1000\text{ms/s})\)*(1 rot/200 steps)
C) The motor will spin at 1000 rps = (1 rot/ms)\((1000\text{ms/s})\)
D) The motor will spin at 4000 rps = (4 rot/ms)\((1000\text{ms/s})\)
E) The motor will spin at 5 rps = (1 step/ms)\((1000\text{ms/s})\)*(1 rot/200 steps)

(4) Question 3. Write a subroutine to sample ADC channel 4 of the 9S12DP512. Assume the ADC initialized for a 10-bit sample, sequence length is 1, and the ADC clock is 2 MHz. Implement right-justified conversions, and return the result in RegX.
(8) **Question 4.** Assume you have an 11-bit ADC with a range of 0 to +10 V (not the 9S12). Write a subroutine that converts the ADC sample into a fixed-point number with a resolution of 0.01 V. The input parameter is call by value in RegD containing the right-justified ADC sample (0 to 2047). 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 7.5 V then RegY is returned as 750.

(4) **Question 5.** Assume RegA = $55, RegX=$1234 and RegY = $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) 1, 6, 3, 2, 5, 4, 7
C) 5, 1, 3, 4, 2, 6, 7
D) 1, 2, 5, 3, 4, 6, 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 1 msec using output compare 2. Show the complete main program, the OC2 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 1 msec
  org $4000
```
(5) **Question 8.** Consider a serial port operating with a baud rate of 1000 bits per second. Draw the waveform occurring at the PS1 output (voltage levels are +5 and 0) when the ASCII ‘T’ ($54$) is transmitted on SCI0. The protocol is 1 start, 8 data and 1 stop bit. The SCI0 is initially idle, and the software writes the $54$ to SCI0DRL at time=0. Show the PS1 line before and after the frame, assuming the channel is idle before and after the frame.

<table>
<thead>
<tr>
<th>PS1</th>
<th>time</th>
<th>0</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>5000</th>
<th>6000</th>
<th>7000</th>
<th>8000</th>
<th>9000</th>
<th>10000</th>
<th>11000</th>
<th>12000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-1</td>
<td>0</td>
<td>1000</td>
<td>2000</td>
<td>3000</td>
<td>4000</td>
<td>5000</td>
<td>6000</td>
<td>7000</td>
<td>8000</td>
<td>9000</td>
<td>10000</td>
<td>11000</td>
<td>12000</td>
</tr>
</tbody>
</table>

(5) **Question 9.** Consider a computer network where two 9S12s are connected via their SCI0 ports, using a 3-wire cable similar to 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 SCI0 ports are idle. The baud rate on both computers is initialized to 1000 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 SCI0SR1 then writes to SCI0DRL. The RDRF ISR in computer 2 will read its SCI1DRL then write to its SCI1DRL (echo the data back). 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.

<table>
<thead>
<tr>
<th>PT2</th>
<th>PT1</th>
<th>PT0</th>
<th>Vout (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

---

Jonathan W. Valvano

December 12, 2008
(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.

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>$3FFC</td>
</tr>
<tr>
<td>$02</td>
<td>$3FFD</td>
</tr>
<tr>
<td>$3D</td>
<td>$3FFE</td>
</tr>
<tr>
<td>$04</td>
<td>$3FFF</td>
</tr>
<tr>
<td>$CF</td>
<td>$4000</td>
</tr>
<tr>
<td>$3F</td>
<td>$4001</td>
</tr>
<tr>
<td>$FE</td>
<td>$4002</td>
</tr>
<tr>
<td>$07</td>
<td>$4003</td>
</tr>
<tr>
<td>$02</td>
<td>$4004</td>
</tr>
<tr>
<td>$20</td>
<td>$4005</td>
</tr>
<tr>
<td>$FC</td>
<td>$4006</td>
</tr>
<tr>
<td>$3B</td>
<td>$4007</td>
</tr>
</tbody>
</table>

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 (2 V, 10 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 PP0 and a positive logic LED is connected to PT0. 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 touched 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 1 ms. 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.

---

![ FSM Graph Diagram ]

---

PP0 (input)  

PT0 (output)
(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 100 ;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 RegY 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
```
leas –1,sp ;allocate L1
pshy ;save old stack frame pointer
tsy ;establish new stack frame pointer
"---------start of body-------------------
   ldaa ????? ;Reg A = value of the parameter
   staa L1,y ;save parameter into local L1
"---------end of body---------------------
puly
leas 1,sp ;deallocate
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 Y 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]
```

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

```
;input: RegB, Output: RegA=sucess
Fifo_Put
```
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  
andccc 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 result clear  
bclr bit clear in memory  
bclr PTT,#$01  
bcsl branch if carry set  
beq branch if result is zero (Z=1)  
bge branch if signed ≥  
bgt enter background debug mode  
bgt branch if signed >  
bi branch if unsigned >  
bis branch if unsigned ≥  
bita 8-bit and with RegA, sets CCR  
bitt 8-bit and with RegB, sets CCR  
ble branch if signed ≤  
blb branch if unsigned <  
bls branch if unsigned ≤  
bit branch if signed <  
blmi branch if negative (N=1)  
bleq branch if result is zero (Z=0)  
bn branch if result is nonzero ≥  
blp 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 clear in memory  
bset PTT,#$004  
bsr branch to subroutine  
bvs branch if overflow set  
call subroutine in expanded memory  
cba 8-bit compare RegA with RegB  
cic clear carry bit, C=0  
cli clear I=0, enable interrupts  
clr 8-bit memory complement  
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  
Ƌbeq decrement and branch if result=0  
Ƌbeq Y,loop  
Ƌbne decrement and branch if result≠0  
Ƌbne A,loop  
Ƌdec 8-bit decrement memory  
Ƌdecb 8-bit decrement RegB  
Ƌdecx 8-bit decrement RegX  
Ƌdey 8-bit decrement RegY  
Ƌdiv RegY=(Y:D)/RegX, unsigned divide  
Ƌdivs RegY=(Y:D)/RegX, signed divide  
Ƌemcs 16 by 16 signed multiply, 32-bit add  
Ƌemcx 16-bit unsigned maximum in RegD  
Ƌemxm 16-bit unsigned maximum in memory  
Ƌemxd 16-bit unsigned minimum in RegD  
Ƌemxm 16-bit unsigned minimum in memory  
Ƌemus RegY:D=RegY*RegD signed multiply  
Ƌemuls RegY:D=RegY*RegD unsigned multiply  
Ƌeora 8-bit logical exclusive or to RegA  
Ƌeorb 8-bit logical exclusive or to RegB  
Ƌetbl 16-bit look up and interpolation  
Ƌexg exchange register contents  
Ƌfdiv unsigned fract div, X=(65536*D)/X  
Ƌibeq increment and branch if result=0  
Ƌibeq Y,loop  
Ƌibne increment and branch if result≠0  
Ƌibne A,loop  
Ƌidiv 16-bit signed div, X=D/Y, D=remainder  
Ƌidivs 16-bit signed divide, X=D/Y, D=remainder  
Ƌincl 8-bit increment memory  
Ƌincb 8-bit increment RegB  
Ƌincb 8-bit increment RegB  
Ƌins 16-bit increment RegSP  
Ƌinx 16-bit increment RegX  
Ƌiny 16-bit increment RegY  
Ƌjmp jump always  
Ƌjar jump to subroutine  
Ƌlbcu long branch if carry clear  
Ƌlbcu long branch if carry set  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
Ƌlbq long branch if result is zero  
(428,790),(462,825)
movw 16-bit move memory to memory
mul RegD=RegA*RegB
neg 8-bit 2's complement negate memory
negb 8-bit 2's complement negate RegB
ora 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 logic 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
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
sbcb 8-bit sub with carry from RegB
sec set carry bit, C=1
sei set F=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
styx 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
tbl 8-bit look up and interpolation
tbne test and branch if result≠0
tfr transfer register to register
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

Example | Mode | Effective Address
--- | --- | ---
idaa #u | immediate | No EA
idaa u | direct | EA is 8-bit address
idaa U | extended | EA is a 16-bit address
idaa m,r | 5-bit index | EA=r+m (-16 to 15)
idaa v,r+ | pre-incr | r=r+v, EA=r (1 to 8)
idaa v,r- | pre-dec | r=r-v, EA=r (1 to 8)
idaa v,r+ | post-inc | EA=r, r=r+v (1 to 8)
idaa v,r- | post-dec | EA=r, r=r-v (1 to 8)
idaa A,r | Reg A offset | EA=r+A, zero padded
idaa B,r | Reg B offset | EA=r+B, zero padded
idaa D,r | Reg D offset | EA=r+D
idaa q,r | 9-bit index | EA=r+q
idaa W,r | 16-bit index | EA=r+W
idaa [D,r] | D indirect | EA=[r+D]
idaa [W,r] | indirect | EA=[r+W]

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

Pseudo op | Meaning
--- | ---
equ | Where to put subsequent code
= | Define a constant symbol
dc.b db fcb .byte | Allocate byte(s) with values
fcb | 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 .blkw | Allocate bytes without init
ds ds.w .blkw | Allocate word(s) without init

Vector Interrupt Source
Arm
Reset
None
Trap
None
SWI
None
Real time interrupt
crgint.rtie
Timer channel
tie.0i
tie.1i
tie.2i
tie.3i
tie.4i
tie.5i
tie.6i
tie.7i
Timer overflow
tscr2.toi
SCI0 TDRD, RDRF
scicr2.tie,rif
SCI1 TDRD, RDRF
scicr2.tie,rif
Key Wakeup J
piej.[7,6,1,0]
Key Wakeup H
pieh.[7:0]
Key Wakeup K
piek.[7:0]

Interrupt Vectors.

<table>
<thead>
<tr>
<th>Address</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0040$</td>
<td>IOS7</td>
<td>IOS6</td>
<td>IOS5</td>
<td>IOS4</td>
<td>IOS3</td>
<td>IOS2</td>
<td>IOS1</td>
<td>IOS0</td>
<td>TIOS</td>
</tr>
</tbody>
</table>

Jonathan W. Valvano
December 12, 2008
$0044-5$ Bit 15 14 13 12 11 10 00 00 00 00 00

$0046$ TEN TSWAI TSRZ TFFCA 0 0 0 0 00 00

$004C$ C7I C6I C5I C4I C3I C2I C1I C0I 0 0 0 0 00 00

$0052$ Bit 15 14 13 12 11 10 00 00 00 00 00

$0054$ Bit 15 14 13 12 11 10 00 00 00 00 00

$0056$ Bit 15 14 13 12 11 10 00 00 00 00 00

$0058$ Bit 15 14 13 12 11 10 00 00 00 00 00

$005A$ Bit 15 14 13 12 11 10 00 00 00 00 00

$005C$ Bit 15 14 13 12 11 10 00 00 00 00 00

$005E$ Bit 15 14 13 12 11 10 00 00 00 00 00

$0082$ ADPU AFFC ASWAI ETRIGLE ETRIGP ETRIG ASCIE ASCIF 00 00 00 00 00

$0083$ 0 S8C S4C S2C S1C S0C FIFO FRZ1 FRZ0 00 00 00 00

$0084$ SRES8 SMP1 SMP0 SPR4 SPR3 SPR2 SPR1 SPR0 00 00 00 00

$0085$ DJM DSMN SCAN MULT 0 CC C9 CA 00 00 00 00

$0086$ SCF 0 TOERF FIFOR 0 CC2 CC1 CC0 00 00 00 00

$0088$ CCF7 CCF6 CCF5 CCF4 CCF3 CCF2 CCF1 CCF0 00 00 00 00

$008A$ Bit 7 6 5 4 3 2 1 00 00 00 00 00

$0088$ Bit 7 6 5 4 3 2 1 00 00 00 00 00

$008A$ TSCR1 is the first 8-bit timer control register

TSCR1 is the first 8-bit timer control register

$008F$ PAD07 PAD06 PAD05 PAD04 PAD03 PAD02 PAD01 PAD00 PORTAD0 00 00 00 00 00

$0090$ Bit 15 14 13 12 11 10 00 00 00 00 00

$0092$ Bit 15 14 13 12 11 10 00 00 00 00 00

$0094$ Bit 15 14 13 12 11 10 00 00 00 00 00

$0096$ Bit 15 14 13 12 11 10 00 00 00 00 00

$0098$ Bit 15 14 13 12 11 10 00 00 00 00 00

$009A$ Bit 15 14 13 12 11 10 00 00 00 00 00

$009C$ Bit 15 14 13 12 11 10 00 00 00 00 00

$00A3$ SBR12 SBR11 SBR10 00 00 00 00 00

$00AC$ LOOPS SCIISWA1 RSRC M WAKE ILT PE PT 00 00 00 00

$00CB$ TIE TCIE RIE ILIE TE RE RWU SBK 00 00 00 00

$00CD$ TDRE TC RDRF IDLE OR NF FE PF 00 00 00 00

$00CE$ R7/T7 R6/T6 R5/T5 R4/T4 R3/T3 R2/T2 R1/T1 R0/T0 00 00 00 00

$00D0$ BRK13 TXDIR RAF 00 00 00 00

$00D2$ LOOP0 SCIISWA1 RSRC M WAKE ILT PE PT 00 00 00 00

$00D3$ TIE TCIE RIE ILIE TE RE RWU SBK 00 00 00 00

$00D5$ TDRE TC RDRF IDLE OR NF FE PF 00 00 00 00

$00D7$ R7/T7 R6/T6 R5/T5 R4/T4 R3/T3 R2/T2 R1/T1 R0/T0 00 00 00 00

$00D3$ PT7 PT6 PT5 PT4 PT3 PT2 PT1 PT0 00 00 00 00

$00D4$ DDR7 DDR6 DDR5 DDR4 DDR3 DDR2 DDR1 DDR0 00 00 00 00

$00D5$ PS7 PS6 PS5 PS4 PS3 PS2 PS1 PS0 00 00 00 00

$00D6$ DDR7 DDR6 DDR5 DDR4 DDR3 DDR2 DDR1 DDR0 00 00 00 00

$00D7$ PM7 PM6 PM5 PM4 PM3 PM2 PM1 PM0 00 00 00 00

$00D8$ DDR7 DDR6 DDR5 DDR4 DDR3 DDR2 DDR1 DDR0 00 00 00 00

$00D9$ P7 P6 P5 P4 P3 P2 P1 P0 00 00 00 00

$00DA$ DDR7 DDR6 DDR5 DDR4 DDR3 DDR2 DDR1 DDR0 00 00 00 00

$00DB$ TIE is the 8-bit output compare select register, one bit for each channel (1 = output compare, 0 = input capture)

$00DC$ TIE is the 8-bit output compare select register, one bit for each channel (1 = output compare, 0 = input capture)

$00DE$ SCIxBD is 16-bit SCI baud rate register, let n be the 16-bit number Baud rate is 250 kHz/n

$00DF$ SCIxBD is 16-bit SCI baud rate register, let n be the 16-bit number Baud rate is 250 kHz/n

Jonathan W. Valvano December 12, 2008
**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**.

### LDA\(A\)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LDA\#&lt;expr&gt;</code></td>
<td>INM</td>
<td><code>00 11</code></td>
</tr>
<tr>
<td><code>LDA\,a</code></td>
<td>DIR</td>
<td><code>00 dd</code></td>
</tr>
<tr>
<td><code>LDA\,r,#&lt;expr&gt;</code></td>
<td>EXT</td>
<td><code>66 hh ll</code></td>
</tr>
<tr>
<td><code>LDA\,[&lt;expr&gt;,&lt;expr&gt;]</code></td>
<td>IDX</td>
<td><code>A6 xb</code></td>
</tr>
<tr>
<td><code>LDA\,[&lt;expr&gt;,&lt;expr&gt;]</code></td>
<td>IDX1</td>
<td><code>A6 xb ff</code></td>
</tr>
<tr>
<td><code>LDA\,[&lt;expr&gt;,&lt;expr&gt;]</code></td>
<td>IDX2</td>
<td><code>A6 xb ee ff</code></td>
</tr>
</tbody>
</table>

### EMUL

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

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EMUL</code></td>
<td>INH</td>
<td><code>10</code></td>
</tr>
</tbody>
</table>

### EDIV

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

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EDIV</code></td>
<td>INH</td>
<td><code>11</code></td>
</tr>
</tbody>
</table>

### BSR

Branch to Subroutine

<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><code>07 xx</code></td>
</tr>
</tbody>
</table>

### RTS

Return from Subroutine

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTS</td>
<td>INH</td>
<td><code>3D</code></td>
</tr>
</tbody>
</table>