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.***

(4) 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?

_____

(4) 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?

_____

(2) 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?

_____

(2) 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?

```
sty 2,sp+
```

_____

(4) Question 10. Is this a legal stack operation? Answer yes or no

```
sty 2,sp+
```

_____

(4) Question 11. Assume **`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?

```
ldaa  Height
sex   A,D  ;promote to 16 bits
addd  #5   ;perform the addition in 16-bit mode
tfr   D,A  ;demote back to 8 bits
staa  Height
```

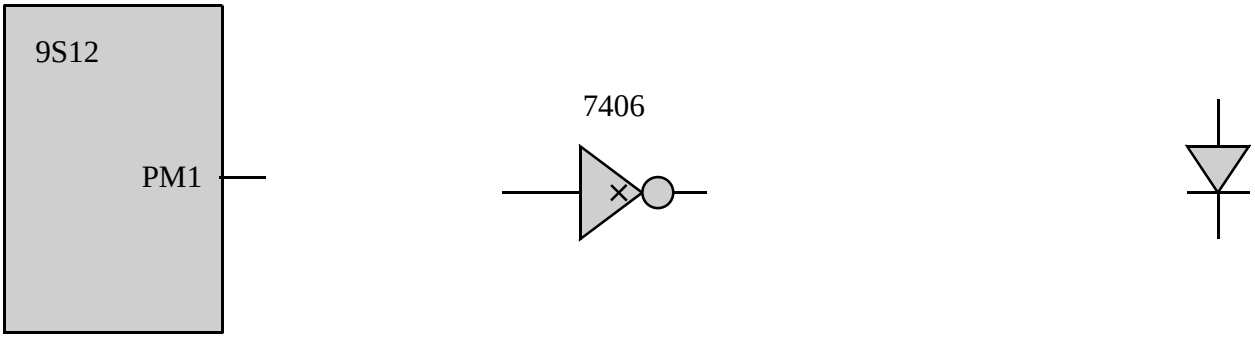
A) Yes, the program has no errors.
B) No, overflow can occur.
C) No, dropout can occur. _____
D) No, the carry bit could be set
E) No, one needs to divide by 10 to get the correct result.
F) No, the **addd** instruction should have been **`addd #0.5`**

(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.

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

R/W	Addr	Data

(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.

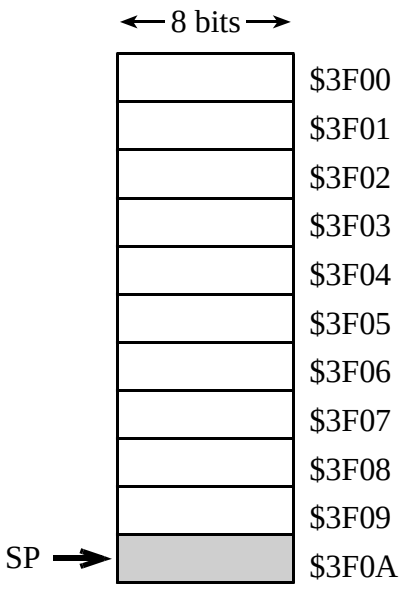


(5) Question 14. You are given a *double-pole* switch that has three pins. The figure shows the switch in the position that occurs when the switch is pressed. If the switch is pressed, pins 1 and 2 are connected (0 resistance) and pins 2 and 3 are not connected (infinite resistance). If the switch is 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.

(10) 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).

```
    org  $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
```

```
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 clear 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
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 mult, 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 mult
emuls  RegY:D=RegY*RegD signed mult
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=rem
idivs  16-bit signed divide, X=D/X, D=rem
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
leax   16-bit load effective addr to X
leay   16-bit load effective addr to 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 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    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-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	addressing mode	Effective Address
ldaa #u	immediate	none
ldaa u	direct	EA is 8-bit address (0 to 255)
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-increment	r=r+v, EA=r (1 to 8)
ldaa v,-r	pre-decrement	r=r-v, EA=r (1 to 8)
ldaa v,r+	post-increment	EA=r, r=r+v (1 to 8)
ldaa v,r-	post-decrement	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 (-256 to 255)
ldaa W,r	16-bit index	EA=r+W (-32768 to 65535)
ldaa [D,r]	D indirect	EA={r+D}
ldaa [W,r]	indirect	EA={r+W} (-32768 to 65535)

Freescale 6812 addressing modes

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 long word(s) of storage with initialized values
ds	**ds.b** **rmb**	**.blkb**	Allocate bytes of storage without initialization
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
$0082	ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIG	ASCIE	ASCIF	ATDCTL2
$0083	0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0	ATDCTL3
$0084	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0	ATDCTL4
$0085	DJM	DSGN	SCAN	MULT	0	CC	CB	CA	ATDCTL5
$0086	SCF	0	ETORF	FIFOR	0	CC2	CC1	CC0	ATDSTAT0
$008B	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0	ATDSTAT1
$008D	Bit 7	6	5	4	3	2	1	Bit 0	ATDDIEN
$0270	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0	PTAD
$0272	DDRAD7	DDRAD6	DDRAD5	DDRAD4	DDRAD3	DDRAD2	DDRAD1	DDRAD0	DDRAD

address	msb															lsb	Name
$0090	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR0
$0092	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR1
$0094	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR2
$0096	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR3
$0098	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR4
$009A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR5
$009C	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR6
$009E	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR7

address	msb															lsb	Name
$0044	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TCNT
$0050	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC0
$0052	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC1
$0054	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC2
$0056	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC3
$0058	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC4
$005A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC5
$005C	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC6
$005E	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC7

Address	Bit 7	6	5	4	3	2	1	Bit 0	Name
$0046	TEN	TSWAI	TSBCK	TFFCA	0	0	0	0	TSCR1
$004D	TOI	0	0	0	TCRE	PR2	PR1	PR0	TSCR2
$0040	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0	TIOS
$004C	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I	TIE
$004E	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F	TFLG1
$004F	TOF	0	0	0	0	0	0	0	TFLG2

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/$2^{\mathbf{n}}$, with PLL **TCNT** is 24MHz/$2^{\mathbf{n}}$, **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)

Vector Address	Interrupt Source or Trigger flag	Enable	Local Arm
$FFFE	Reset	none	none
$FFEE	Timer Channel 0, C0F	I bit	TIE.C0I
$FFEC	Timer Channel 1, C1F	I bit	TIE.C1I
$FFEA	Timer Channel 2, C2F	I bit	TIE.C2I
$FFE8	Timer Channel 3, C3F	I bit	TIE.C3I
$FFE6	Timer Channel 4, C4F	I bit	TIE.C4I
$FFE4	Timer Channel 5, C5F	I bit	TIE.C5I
$FFE2	Timer Channel 6, C6F	I bit	TIE.C6I
$FFE0	Timer Channel 7, C7F	I bit	TIE.C7I

00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0,X 5b const	−16,X 5b const	1,+X pre-inc	1,X+ post-inc	0,Y 5b const	−16,Y 5b const	1,+Y pre-inc	1,Y+ post-inc	0,SP 5b const	−16,SP 5b const	1,+SP pre-inc	1,SP+ post-inc	0,PC 5b const	−16,PC 5b const	n,X 9b const	n,SP 9b const
01	11	21	31	41	51	61	71	81	91	A1	B1	C1	D1	E1	F1
1,X 5b const	−15,X 5b const	2,+X pre-inc	2,X+ post-inc	1, Y 5b const	−15,Y 5b const	2,+Y pre-inc	2,Y+ post-inc	1,SP 5b const	−15,SP 5b const	2,+SP pre-inc	2,SP+ post-inc	1,PC 5b const	−15,PC 5b const	−n,X 9b const	−n,SP 9b const
02	12	22	32	42	52	62	72	82	92	A2	B2	C2	D2	E2	F2
2,X 5b const	−14,X 5b const	3,+X pre-inc	3,X+ post-inc	2, Y 5b const	−14,Y 5b const	3,+Y pre-inc	3,Y+ post-inc	2,SP 5b const	−14,SP 5b const	3,+SP pre-inc	3,SP+ post-inc	2,PC 5b const	−14,PC 5b const	n,X 16b const	n,SP 16b const
03	13	23	33	43	53	63	73	83	93	A3	B3	C3	D3	E3	F3
3,X 5b const	−13,X 5b const	4,+X pre-inc	4,X+ post-inc	3,Y 5b const	−13,Y 5b const	4,+Y pre-inc	4,Y+ post-inc	3,SP 5b const	−13,SP 5b const	4,+SP pre-inc	4,SP+ post-inc	3,PC 5b const	−13,PC 5b const	[n,X] 16b indr	[n,SP] 16b indr
04	14	24	34	44	54	64	74	84	94	A4	B4	C4	D4	E4	F4
4,X 5b const	−12,X 5b const	5,+X pre-inc	5,X+ post-inc	4,Y 5b const	−12,Y 5b const	5,+Y pre-inc	5,Y+ post-inc	4,SP 5b const	−12,SP 5b const	5,+SP pre-inc	5,SP+ post-inc	4,PC 5b const	−12,PC 5b const	A,X A offset	A,SP A offset
05	15	25	35	45	55	65	75	85	95	A5	B5	C5	D5	E5	F5
5,X 5b const	−11,X 5b const	6,+X pre-inc	6,X+ post-inc	5,Y 5b const	−11,Y 5b const	6,+Y pre-inc	6,Y+ post-inc	5,SP 5b const	−11,SP 5b const	6,+SP pre-inc	6,SP+ post-inc	5,PC 5b const	−11,PC 5b const	B,X B offset	B,SP B offset
06	16	26	36	46	56	66	76	86	96	A6	B6	C6	D6	E6	F6
6,X 5b const	−10,X 5b const	7,+X pre-inc	7,X+ post-inc	6,Y 5b const	−10,Y 5b const	7,+Y pre-inc	7,Y+ post-inc	6,SP 5b const	−10,SP 5b const	7,+SP pre-inc	7,SP+ post-inc	6,PC 5b const	−10,PC 5b const	D,X D offset	D,SP D offset
07	17	27	37	47	57	67	77	87	97	A7	B7	C7	D7	E7	F7
7,X 5b const	−9,X 5b const	8,+X pre-inc	8,X+ post-inc	7,Y 5b const	−9,Y 5b const	8,+Y pre-inc	8,Y+ post-inc	7,SP 5b const	−9,SP 5b const	8,+SP pre-inc	8,SP+ post-inc	7,PC 5b const	−9,PC 5b const	[D,X] D indirect	[D,SP] D indirect
08	18	28	38	48	58	68	78	88	98	A8	B8	C8	D8	E8	F8
8,X 5b const	−8,X 5b const	8,−X pre-dec	8,X− post-dec	8,Y 5b const	−8,Y 5b const	8,−Y pre-dec	8,Y− post-dec	8,SP 5b const	−8,SP 5b const	8,−SP pre-dec	8,SP− post-dec	8,PC 5b const	−8,PC 5b const	n,Y 9b const	n,PC 9b const
09	19	29	39	49	59	69	79	89	99	A9	B9	C9	D9	E9	F9
9,X 5b const	−7,X 5b const	7,−X pre-dec	7,X− post-dec	9,Y 5b const	−7,Y 5b const	7,−Y pre-dec	7,Y− post-dec	9,SP 5b const	−7,SP 5b const	7,−SP pre-dec	7,SP− post-dec	9,PC 5b const	−7,PC 5b const	−n,Y 9b const	−n,PC 9b const
0A	1A	2A	3A	4A	5A	6A	7A	8A	9A	AA	BA	CA	DA	EA	FA
10,X 5b const	−6,X 5b const	6,−X pre-dec	6,X− post-dec	10,Y 5b const	−6,Y 5b const	6,−Y pre-dec	6,Y− post-dec	10,SP 5b const	−6,SP 5b const	6,−SP pre-dec	6,SP− post-dec	10,PC 5b const	−6,PC 5b const	n,Y 16b const	n,PC 16b const
0B	1B	2B	3B	4B	5B	6B	7B	8B	9B	AB	BB	CB	DB	EB	FB
11,X 5b const	−5,X 5b const	5,−X pre-dec	5,X− post-dec	11,Y 5b const	−5,Y 5b const	5,−Y pre-dec	5,Y− post-dec	11,SP 5b const	−5,SP 5b const	5,−SP pre-dec	5,SP− post-dec	11,PC 5b const	−5,PC 5b const	[n,Y] 16b indr	[n,PC] 16b indr
0C	1C	2C	3C	4C	5C	6C	7C	8C	9C	AC	BC	CC	DC	EC	FC
12,X 5b const	−4,X 5b const	4,−X pre-dec	4,X− post-dec	12,Y 5b const	−4,Y 5b const	4,−Y pre-dec	4,Y− post-dec	12,SP 5b const	−4,SP 5b const	4,−SP pre-dec	4,SP− post-dec	12,PC 5b const	−4,PC 5b const	A,Y A offset	A,PC A offset
0D	1D	2D	3D	4D	5D	6D	7D	8D	9D	AD	BD	CD	DD	ED	FD
13,X 5b const	−3,X 5b const	3,−X pre-dec	3,X− post-dec	13,Y 5b const	−3,Y 5b const	3,−Y pre-dec	3,Y− post-dec	13,SP 5b const	−3,SP 5b const	3,−SP pre-dec	3,SP− post-dec	13,PC 5b const	−3,PC 5b const	B,Y B offset	B,PC B offset
0E	1E	2E	3E	4E	5E	6E	7E	8E	9E	AE	BE	CE	DE	EE	FE
14,X 5b const	−2,X 5b const	2,−X pre-dec	2,X− post-dec	14,Y 5b const	−2,Y 5b const	2,−Y pre-dec	2,Y− post-dec	14,SP 5b const	−2,SP 5b const	2,−SP pre-dec	2,SP− post-dec	14,PC 5b const	−2,PC 5b const	D,Y D offset	D,PC D offset
0F	1F	2F	3F	4F	5F	6F	7F	8F	9F	AF	BF	CF	DF	EF	FF
15,X 5b const	−1,X 5b const	1,−X pre-dec	1,X− post-dec	15,Y 5b const	−1,Y 5b const	1,−Y pre-dec	1,Y− post-dec	15,SP 5b const	−1,SP 5b const	1,−SP pre-dec	1,SP− post-dec	15,PC 5b const	−1,PC 5b const	[D,Y] D indirect	[D,PC] D indirect

Addr	Bit 7	6	5	4	3	2	1	Bit 0	Name
$00C8	BTST	BSPL	BRLD	SBR12	SBR11	SBR10	SBR9	SBR8	SCIBD
$00C9	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0	
$00CB	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCICR2
$00CC	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	SCISR1
$00CF	R7T7	R6T6	R5T5	R4T4	R3T3	R2T2	R1T1	R0T0	SCIDRL

SCIBD is 16-bit SCI baud rate register, let **n** be the 16-bit number Baud rate is 250 kHz/**n**

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 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: $(Y_H : Y_L) \Rightarrow M : M + 1$

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 one).

Source Form	Address Mode	Object Code
STY *opr8a*	DIR	5D dd
STY *opr16a*	EXT	7D hh ll
STY *oprx0_xysp*	IDX	6D xb
STY *oprx9,xysp*	IDX1	6D xb ff
STY *oprx16,xysp*	IDX2	6D xb ee ff
STY [D,*xysp*]	[D,IDX]	6D xb
STY [*oprx16,xysp*]	[IDX2]	6D xb ee ff

BSR

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

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.

Source Form	Address Mode	Object Code
BSR *rel8*	REL	07 rr