First:_____ Last:_____

This is a closed book exam. You must put your answers in these boxes only. You have 3 hours, so allocate your time accordingly. ***Please read the entire exam before starting.***

(3) Question 1.	**(3) Question 15.**
(3) Question 2.	**(3) Question 16.**
(3) Question 3.	**(3) Question 17.**
(3) Question 4.	**(3) Question 18.**
(3) Question 5.	**(3) Question 19.**
(3) Question 6.	**(3) Question 20.**
(3) Question 7.	**(3) Question 21.**
(3) Question 8.	**(3) Question 22.**
(3) Question 9.	**(3) Question 23.**
(3) Question 10.	**(3) Question 24.**
(3) Question 11.	**(3) Question 25.**
(3) Question 12.	**(3) Question 26.**
(3) Question 13.	**(3) Question 27.**
(3) Question 14.	**(3) Question 28.**

(8) Question 29. Show the assembly main program

(8) Question 30. Show the assembly main program

(3) Question 1. Consider a matrix with 4 rows and 6 columns, stored in column-major zero-index format. Each element is 16 bits. Which equation correctly calculates the address of the element at row I and column J?

A) base+I+J
B) base+4*I+J
C) base+I+4*J
D) base+6*I+J
E) base+I+6*J
F) base+2*I+2*J
G) base+8*I+2*J
H) base+2*I+8*J
I) base+12*I+2*J
J) base+2*I+12*J

(3) Question 2. Consider the following C program.

```
short function(const short in){
   return in+5;
}
```

Where is the parameter **in** allocated?

A) global RAM
B) local RAM
C) global ROM.
D) local ROM.
E) None of these answers is correct.

(3) Question 3. What is drop out?

A) Drop out is the error that occurs when the result of a calculation exceeds the range of the number system.
B) Drop out is the error that occurs after a right shift or a divide, and the consequence is that an intermediate result looses its ability to represent all of the values.
C) Drop out is when both the Carry and the Overflow bits are set.
D) Drop out is data is lost when the software does not respond fast to an I/O event.
E) None of these answers is correct.

(3) Question 4. Which direction does data flow on the data bus during a write cycle?

A) From 6812 to memory, or from 6812 to output device
B) From memory to 6812, or from input device to 6812
C) From input device to memory
D) From memory to output device
E) None of these answers is correct

(3) Question 5. Which of the following statements best describes the action that will set the **RDRF** bit in the **SCISR1** register on the 6812?

A) The software writes a 1 to the **RDRF** bit in the **SCISR1** register.
B) The software reads **SCISR1** when **RDRF** is one, followed by reading **SCIDRL**.
C) The software wants new input data.
D) The software writes to the serial data register, **SCIDRL**.
E) The receive hardware is idle, ready to receive another input.
F) The receive shift register is busy, currently receiving a new input.
G) None of these choices is correct.

(3) Question 6. Which of the following statements best describes the action that will clear the **RDRF** bit in the **SCISR1** register on the 6812?

A) The software writes a 0 to the **RDRF** bit in the **SCISR1** register.
B) The software reads **SCISR1** when **RDRF** is one, followed by reading **SCIDRL**.
C) The software wants to transmit new output data.
D) The software reads from the serial data register, **SCIDRL**.
E) The receive hardware is idle, ready to receive another input.
F) The receive shift register is busy, currently receiving a new input.
G) None of these choices is correct.

(3) Question 7. What event triggers the start of an ADC conversion on the 6812?

A) The software writes to the **ATDCTL3** register.
B) The software writes to the **ATDCTL4** register.
C) The software writes to the **ATDCTL5** register.
D) The ADC is automatically started by hardware.
E) Software sets the **ADPU** bit in the **ATDCTL2** register.
F) Software read **ATDSTAT0** with **SCF** set, followed by reading the result register.
G) None of these choices is correct.

(3) Question 8. What is the bug in the following initialized global variable on the 9S12C32?

```
        org  $3A00
Count   fdb  100
```

A) $3A00 is not RAM
B) RAM is volatile
C) RAM is nonvolatile
D) 100 is 8-bits, and fdb specifies 16-bits
E) Global variables are poor style and should never be used.
F) No error, this definition is acceptable.

(3) Question 9. The measurement system range is 0 to 399.9 and a resolution of 0.1. What is the precision in decimal digits?

Consider the result of executing the following two 6812 assembly instructions.

```
ldab #101
subb #110
```

(3) Question 10. What will be the value of the carry (C) bit?
(3) Question 11. What will be the value of the overflow (V) bit?

(3) Question 12. What will be the value in Register D after executing the following 6812 assembly instructions?

```
ldd  #100
ldy  #670
emul
```

(3) Question 13. You may assume all RAM locations are initially 0, and assume Reg Y equals $1234, Reg D is $5678. What is in Reg Y after these instructions are executed?

```
pshy
pshb
inc 1,s
puly
```

(3) Question 14. An **swi** pushes the following registers on the stack in this order PC, Y, X, A, B, CCR, with CCR on top. Initially, assume RegX=$4321, RegY=$ABCD, RegD=$8765. What is the resulting hexadecimal value in RegX after these instructions execute?

```
stx  5,s
std  3,s
rti
```

(3) Question 15. Show the machine code generated by the instruction

```
orab -5,y
```

Questions 16 and 17 involve the following assembly code. The subroutine returns the result by value on the stack.

```
main lds #$4000
     leas -2,s   ; make space for out parameter on stack
     jsr GetT
     puly        ; Fetch the return value from the stack
     stop
data set  xxx    ; binding of 16-bit local variable
out  set  yyy    ; binding of 16-bit output parameter
GetT leas -2,s   ; allocate 16-bit local variable called data
;****body of the subroutine
     ldd  TCNT   ; get time
     std  data,s ; place time into local variable data
     ldd  TCNT   ; get time again
     subd data,s ; time difference in RegD
     std  out,s  ; return by value on the stack
;****end of body
     leas 2,s    ; deallocate data
     rts         ; return
```

(3) Question 16. What value should you use in the **xxx** position to implement the binding of the local variable, **data**?

(3) Question 17. What value should you use in the **yyy** position to implement the binding of the parameter, **out**?

(3) Question 18. A SCI is configured at a baud rate of 1200 bits/sec, with 9 bit data, two stop bits, and no parity? What is the bandwidth in bits/sec?

(3) Question 19. Assume the 9S12C32 sequence length is configured to perform one ADC sample (**S8C S4C S2C S1C** in **ATDCTL3** is set to 0001). After the ADC is triggered to sample channel 5, into which register is the digital conversion stored? In other words, out of which register does the software read the ADC result?

(3) Question 20. A signed 8-bit binary fixed-point number has a resolution of $1/16 = 2^{-4}$. If the integer value stored in memory is $F0, what value does it represent?

(3) Question 21. Which term best describes an interfacing method that the software checks the status of an I/O device, and proceeds once the device is ready?

(3) Question 22. Which data structure has the following features? It can hold a variable number of fixed-size elements. It has two main operations, one to store data into itself, and a second operation to remove data. The data is removed in a "first come first served" order.

(3) Question 23. Assuming the variable, **N**, has an 8-bit signed value, does the following operation potentially cause overflow? Answer Yes or No.

```
     ldab  N
     sex   b,x  ;promote to 16-bits
     leax  10,x ;16-bit add RegX=RegX+10
     tfr   x,b  ;demote
     stab  N
```

Consider the following assembly subroutine that creates two local variables, called **p** and **q**. The variable **p** is 8-bits and initialized to 50. The variable **q** is 16-bits and initialized to 500. The local variable binding is created using the **set** pseudo-ops.

```
p    set   xxx          ; binding
q    set   yyy          ; binding
sub1 pshy               ; save register Y
     tsy                ; stack frame
     movb #50,1,-sp     ; allocate and initialize p
     movb #500,2,-sp    ; allocate and initialize q
;... stuff
     ldaa p,y  ; read from p
     ldx  q,y  ; read from q
;... more stuff
     leas 3,s    ; deallocate p,q
     puly        ; restore register Y
     rts         ; return
```

(3) Question 24. What value should you use in the **xxx** position to implement the binding of **p**?

(3) Question 25. What value should you use in the **yyy** position to implement the binding of **q**?

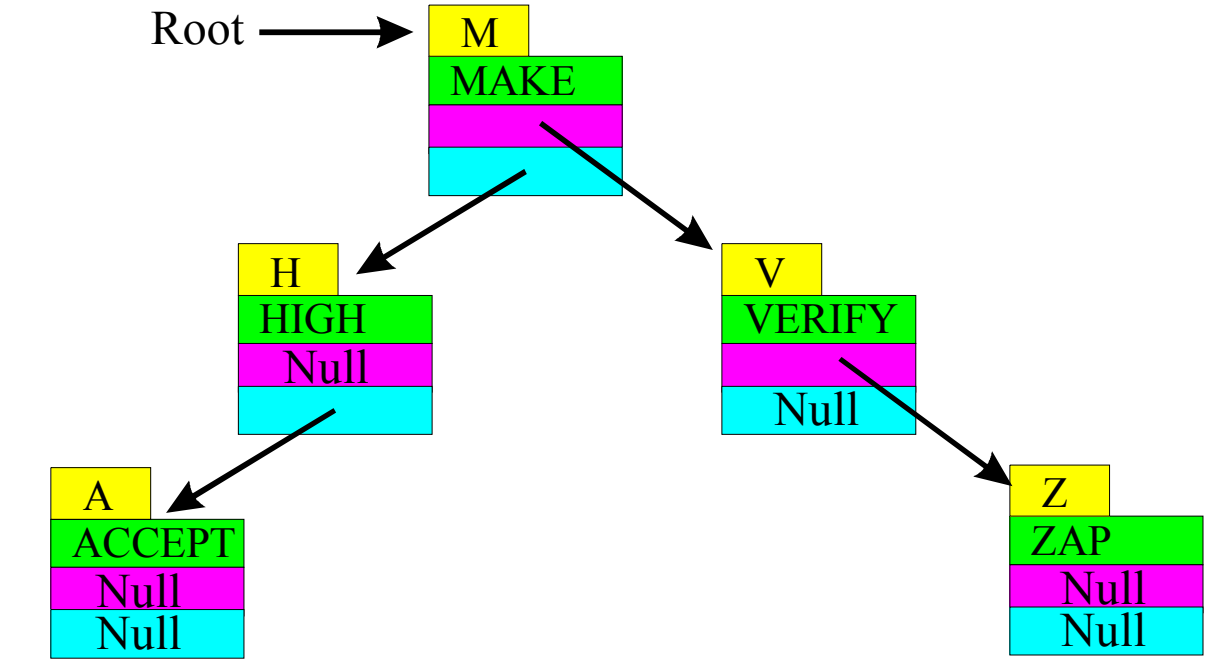
(3) Question 26. Sketch the output waveform occurring on the PS1=TxD output as one character (ASCII 'C' = $43) is transmitted. Assume 8 bit data, no parity and 1 stop bit.

(3) Question 27. In order to observe the where and when our software is executing, we can connect unused output pins to an oscilloscope and set/clear these pins at various important locations within our software. What is this debugging process is called? Choose from *stabilization*, *profiling*, *desk checking*, or *dump*.

(3) Question 28. Assume **`PTT`** is an 8-bit input and **`PTM`** is an output. The goal of this code is to clear **`PTM`** if **`PTT`** bit 0 is set. Which op code should be used in the **`???`** position?

```
     ldaa PTT
     anda #1
     ???  skip
     clr  PTM
loop
```

(8) Question 29. Write an assembly main program that implements an interpreter using a Tree data structure. You may assume the SCI device driver is available. In other words, you can call **`SCI_Init`** to initialize the SCI and call **`SCI_InChar`** to receive a character (returned in Reg A). There are five nodes in the binary tree as shown in the figure. The tree data structure, shown below, is given and cannot be changed. For example, if the operator types V, your interpreter will call the function **`Verify`**. Ignore letters which do not match any of the nodes.




```
Root     fdb  CMDM
CMDM     fcc  'M'    Command letter
         fdb  MAKE      Ptr to subroutine
         fdb  CMDH      Ptr to left entry
         fdb  CMDV      Ptr to right entry
CMDH     fcc  'H'    Command letter
         fdb  HIGH      Ptr to subroutine
```

```
         fdb  CMDA      Ptr to left entry
         fdb  null      None to the right
CMDA     fcc  'A'    Command letter
         fdb  ACCEPT    Ptr to subroutine
         fdb  null      None to the left
         fdb  null      None to the right
CMDV     fcc  'V'    Command letter
         fdb  VERIFY    Ptr to subroutine
         fdb  null      None to the left
         fdb  CMDZ      Ptr to right entry
CMDZ     fcc  'Z'    Command letter
         fdb  ZAP       Ptr to subroutine
         fdb  null      None to the left
         fdb  null      None to the right

MAKE     ldx  #MAKEString
         jsr  SCI_OutString
         rts
MAKEString fcb CR,"MAKE command",EOT

HIGH     ldx  #HIGHString
         jsr  SCI_OutString
         rts
HIGHString fcb CR,"HIGH command",EOT

ACCEPT   ldx  #ACCEPTString
         jsr  SCI_OutString
         rts
ACCEPTString fcb CR,"ACCEPT command",EOT

VERIFY   ldx  #VERIFYString
         jsr  SCI_OutString
         rts
VERIFYString fcb CR,"VERIFY command",EOT

ZAP      ldx  #ZAPString
         jsr  SCI_OutString
         rts
ZAPString fcb CR,"ZAP command",EOT
```

(8) Question 30. 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 8-bit indices (e.g., 1 means S1) rather than 16-bit pointers. Each state has 16 outputs and 16 next-state links. The input is on Port T bits 3,2,1,0 and the output is on Port M bits 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. You need not be friendly, but do initialize the direction registers. The repeating execution sequence is input, output, next.

```
      org  $4000  Put in EEPROM so it can be changed
* Finite State Machine
S0  fcb  0,0,5,6,3,9,3,0,1,2,3,4,5,6,7,8  Outputs for inputs 0 to 15
    fcb  0,0,0,1,1,1,2,2,2,0,0,0,1,2,0,1  Next states for inputs 0 to 15
S1  fcb  1,2,3,9,6,5,3,3,3,3,4,5,9,1,0,0  Outputs for inputs 0 to 15
    fcb  2,2,2,0,0,0,2,2,2,1,1,1,2,1,1,0  Next states for inputs 0 to 15
S2  fcb  1,2,3,9,6,5,3,3,3,3,4,5,9,1,0,0  Outputs for inputs 0 to 15
```

```
fcb  0,0,0,0,0,0,2,2,2,2,2,2,0,0,2,1  Next states for inputs 0 to 15
```

ORAB Inclusive OR B ORAB

Operation: (B) + (M) ⇒ B

Description: Performs bitwise logical inclusive OR between the content of accumulator B and the content of memory location M. The result is placed in B. Each bit of B after the operation is the logical inclusive OR of the corresponding bits of M and of B before the operation.

Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

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

Source Form	Address Mode	Object Code	Cycles	Access Detail
ORAB #*opr8i*	IMM	CA ii	1	P
ORAB *opr8a*	DIR	DA dd	3	rfP
ORAB *opr16a*	EXT	FA hh ll	3	rOP
ORAB *oprx0_xysp*	IDX	EA xb	3	rfP
ORAB *oprx9,xysp*	IDX1	EA xb ff	3	rPO
ORAB *oprx16,xysp*	IDX2	EA xb ee ff	4	frPP
ORAB [D,*xysp*]	[D,IDX]	EA xb	6	fIfrfP
ORAB [*oprx16,xysp*]	[IDX2]	EA xb ee ff	6	fIPrfP

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

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

```
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
asrb   8-bit arith right shift to RegB
bcc    branch if carry clear
bclr   clear bits in memory
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,
brn    branch never
brset  branch if bits are set
bset   set bits in memory
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
dbne   decrement and branch if result?0
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
fdiv   16-bit unsigned fractional divide
ibeq   increment and branch if result=0
ibne   increment and branch if result?0
idiv   16-bit unsigned divide, X=D/X
idivs  16-bit signed divide, X=D/X
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
movw   16-bit move memory to memory
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=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
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 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)

Motorola 6812 addressing modes

Pseudo op	meaning
org 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

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