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.*

<table>
<thead>
<tr>
<th>Question</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3) Question 1.</td>
<td>(3) Question 15.</td>
</tr>
<tr>
<td>(3) Question 2.</td>
<td>(3) Question 16.</td>
</tr>
<tr>
<td>(3) Question 3.</td>
<td>(3) Question 17.</td>
</tr>
<tr>
<td>(3) Question 4.</td>
<td>(3) Question 18.</td>
</tr>
<tr>
<td>(3) Question 5.</td>
<td>(3) Question 19.</td>
</tr>
<tr>
<td>(3) Question 6.</td>
<td>(3) Question 20.</td>
</tr>
<tr>
<td>(3) Question 7.</td>
<td>(3) Question 21.</td>
</tr>
<tr>
<td>(3) Question 8.</td>
<td>(3) Question 22.</td>
</tr>
<tr>
<td>(3) Question 9.</td>
<td>(3) Question 23.</td>
</tr>
<tr>
<td>(3) Question 10.</td>
<td>(3) Question 24.</td>
</tr>
<tr>
<td>(3) Question 11.</td>
<td>(3) Question 25.</td>
</tr>
<tr>
<td>(3) Question 12.</td>
<td>(3) Question 26.</td>
</tr>
<tr>
<td>(3) Question 13.</td>
<td>(3) Question 27.</td>
</tr>
<tr>
<td>(3) Question 14.</td>
<td>(3) Question 28.</td>
</tr>
</tbody>
</table>
(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.

```c
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 loses 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?

\[
\begin{align*}
&\text{pshy} \\
&\text{pshb} \\
&\text{inc 1,s} \\
&\text{puly}
\end{align*}
\]

(3) Question 14. An \text{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?

\[
\begin{align*}
&\text{stx 5,s} \\
&\text{std 3,s} \\
&\text{rti}
\end{align*}
\]

(3) Question 15. Show the machine code generated by the instruction \text{orab -5,y}

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

\begin{verbatim}
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
\end{verbatim}

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

(3) Question 17. What value should you use in the \text{yyy} position to implement the binding of the parameter, \text{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=Tx-D 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
```

(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.
(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.

```assembly
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 1,2,3,9,6,5,3,3,3,4,3,4,5,9,1,0,0  Outputs for inputs 0 to 15
  fcb 2,2,2,0,0,2,2,2,1,1,1,2,1,1,0  Next states for inputs 0 to 15
S2 fcb 1,2,3,9,6,5,3,3,3,4,3,4,5,9,1,0,0  Outputs for inputs 0 to 15
```

```
Jonathan W. Valvano     December 19, 2005      9a-12n
```
**ORAB**

**Inclusive OR B**

**Operation:** $(B) + (M) \Rightarrow B$

**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:**

$$
\begin{array}{cccccccc}
S & X & H & I & N & Z & V & C \\
\hline
0 & - & - & - & \Delta & \Delta & 0 & -
\end{array}
$$

$N$: Set if MSB of result is set; cleared otherwise.
$Z$: Set if result is $0000$; cleared otherwise.
$V$: 0; Cleared.

**Source Form**

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
<th>Cycles</th>
<th>Access Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORAB #cpr8i</td>
<td>IMM</td>
<td>CA ii</td>
<td>1</td>
<td>$F$</td>
</tr>
<tr>
<td>ORAB cpr8a</td>
<td>DI</td>
<td>DA dd</td>
<td>3</td>
<td>rfP</td>
</tr>
<tr>
<td>ORAB cpr16a</td>
<td>EXT</td>
<td>FA hh ll</td>
<td>3</td>
<td>rOP</td>
</tr>
<tr>
<td>ORAB cpx0,yxp</td>
<td>IDX</td>
<td>EA xb</td>
<td>3</td>
<td>rfP</td>
</tr>
<tr>
<td>ORAB cpx9,yxp</td>
<td>IDX1</td>
<td>EA xb ff</td>
<td>3</td>
<td>fP0</td>
</tr>
<tr>
<td>ORAB cpx16,yxp</td>
<td>IDX2</td>
<td>EA xb ef</td>
<td>4</td>
<td>fIPfP</td>
</tr>
<tr>
<td>ORAB [D,yxp]</td>
<td>[D,IDX]</td>
<td>EA xb</td>
<td>6</td>
<td>fIPfP</td>
</tr>
<tr>
<td>ORAB [opx16,yxp]</td>
<td>[IDX2]</td>
<td>EA xb ef</td>
<td>6</td>
<td>fIPfP</td>
</tr>
</tbody>
</table>

**Address**

<table>
<thead>
<tr>
<th>Address</th>
<th>Bit 7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>Bit 0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0082$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0083$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0084$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0085$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0086$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0087$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0088$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0089$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0090$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0091$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0092$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0093$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0094$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0095$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0096$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0097$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Addr**

<table>
<thead>
<tr>
<th>Addr</th>
<th>Bit 7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>Bit 0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00C8$</td>
<td>BTST</td>
<td>BSPL</td>
<td>BRLD</td>
<td>SBR12</td>
<td>SBR11</td>
<td>SBR10</td>
<td>SBR9</td>
<td>SBR8</td>
<td>SCIBD</td>
</tr>
<tr>
<td>$00C9$</td>
<td>SBR7</td>
<td>SBR6</td>
<td>SBR5</td>
<td>SBR4</td>
<td>SBR3</td>
<td>SBR2</td>
<td>SBR1</td>
<td>SBR0</td>
<td></td>
</tr>
<tr>
<td>$00CB$</td>
<td>TIE</td>
<td>TCIE</td>
<td>RIE</td>
<td>ILIE</td>
<td>TE</td>
<td>RE</td>
<td>RWU</td>
<td>SBK</td>
<td>SCICR2</td>
</tr>
</tbody>
</table>

Jonathan W. Valvano     December 19, 2005      9a-12n
<table>
<thead>
<tr>
<th>$00CC</th>
<th>TDRE</th>
<th>TC</th>
<th>RDRF</th>
<th>IDLE</th>
<th>OR</th>
<th>NF</th>
<th>FE</th>
<th>PF</th>
<th>SCISR1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$00CF</td>
<td>R7T7</td>
<td>R6T6</td>
<td>R5T5</td>
<td>R4T4</td>
<td>R3T3</td>
<td>R2T2</td>
<td>R1T1</td>
<td>R0T0</td>
<td>SCIDRL</td>
</tr>
</tbody>
</table>
aba 8-bit add RegY=RegX+RegB
																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
tdmc 8-bit logical and to RegCC

dasl/lsla 8-bit left shift RegA

dasl/lsls 8-bit arith left shift Memory

dasr 8-bit arith right shift RegD

dasrsl 8-bit arith left shift RegD

adx 8-bit add with carry to RegA

add 8-bit add to RegA

addh 8-bit add to RegB

addw 16-bit add to RegD

addx 8-bit add with carry to RegB

addz 8-bit add to RegA

dadd 8-bit add to RegA

daddw 16-bit add to RegD

daddx 8-bit add with carry to RegB

daddz 8-bit add to RegA

ddhh 8-bit add with high carry to RegA

ddhw 16-bit add with high carry to RegD

ddhx 8-bit add with high carry to RegB

ddhz 8-bit add with high carry to RegA

ddhhw 16-bit add with high carry to RegD

ddhxl 16-bit add with high carry to RegB

ddhl 16-bit add with high carry to RegA

ddhhzz 8-bit add with high carry to RegA

adu 16-bit unsigned add

adul 16-bit unsigned add

aduu 16-bit unsigned add

adux 16-bit unsigned add with carry

aduux 16-bit unsigned add with carry

addu 16-bit unsigned add

addux 16-bit unsigned add with carry

adb 16-bit add with borrow to RegA

addb 16-bit add with borrow to RegB

addbc 16-bit add with borrow and carry to RegA

addb 16-bit add with borrow to RegA

adb 16-bit add with borrow to RegB

addb 16-bit add with borrow to RegB

addbc 16-bit add with borrow and carry to RegB

addw 16-bit add with borrow to RegA

addw 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

addbc 16-bit add with borrow and carry to RegB

addz 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

addbc 16-bit add with borrow and carry to RegB

addw 16-bit add with borrow to RegA

addw 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

addbc 16-bit add with borrow and carry to RegB

addw 16-bit add with borrow to RegA

addw 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

addbc 16-bit add with borrow and carry to RegB

addw 16-bit add with borrow to RegA

addw 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

addbc 16-bit add with borrow and carry to RegB

addw 16-bit add with borrow to RegA

addw 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

addbc 16-bit add with borrow and carry to RegB

addw 16-bit add with borrow to RegA

addw 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

addbc 16-bit add with borrow and carry to RegB

addw 16-bit add with borrow to RegA

addw 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegA

addz 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

adb 16-bit add with borrow to RegB

addbc 16-bit add with Java's worst code completion
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
rorw 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
sb a 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
staa 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
tral 8-bit compare RegA=RegA-RegB
tralp trap illegal op code, or software trap
staa 8-bit store memory from RegA
staa 8-bit store memory from RegB
staa 8-bit store memory from RegD
wait wait for interrupt
std 16-bit store memory from RegD
wav weighted Fuzzy logic average
xgdx exchange RegD with RegX
xgdy exchange RegD with RegY

example addressing mode Effective Address
ldaa #u immediate EA is 8-bit address (0 to 255)
ldaa u direct EA is 8-bit address
ldaa U extended EA is a 16-bit address
ldaa m,r 5-bit index EA=r+m (-16 to 15)
ldaa v,+r pre-increment EA=r+v, EA=r (1 to 8)
ldaa v,-r pre-decrement EA=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

<table>
<thead>
<tr>
<th>Pseudo op</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>org</td>
<td>Specific absolute address to put subsequent object code</td>
</tr>
<tr>
<td>=</td>
<td>equ Define a constant symbol</td>
</tr>
<tr>
<td>set</td>
<td>Define or redefine a constant symbol</td>
</tr>
<tr>
<td>dc.b</td>
<td>db</td>
</tr>
<tr>
<td>fcc</td>
<td>Create an ASCII string (no termination character)</td>
</tr>
<tr>
<td>dc.w</td>
<td>dw</td>
</tr>
<tr>
<td>dl</td>
<td>.long Allocate 32-bit long word(s) of storage with initialized values</td>
</tr>
<tr>
<td>ds</td>
<td>ds.b</td>
</tr>
<tr>
<td>ds.w</td>
<td>.blkw Allocate bytes of storage without initialization</td>
</tr>
<tr>
<td>ds.l</td>
<td>.blkl Allocate 32-bit words of storage without initialization</td>
</tr>
<tr>
<td>Column 1</td>
<td>Column 2</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>Value 1</td>
<td>Value 2</td>
</tr>
</tbody>
</table>