Overview of Microcontrollers, Embedded Control and the Motorola HC12 family

Computer: Built by combining a microprocessor with other components such as memory and I/O.

Microprocessor: Central processing unit (CPU) (ALU, decoder, registers) on a single circuit.

Microcontroller: (MCU) Combines a microprocessor, memory and I/O all into a single chip.

“MCU designed specifically to interface with its environment”

Examples of:

<table>
<thead>
<tr>
<th>High level code</th>
<th>Assembly</th>
<th>Machine code</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta=30;</td>
<td>LDAA</td>
<td>#$1C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$B6 $1C</td>
</tr>
</tbody>
</table>
Microcontroller use on the rise

Use of 8-bit microcontrollers rose by 34% between 2005 and 2006, according to the third edition of a report called "Marketing to Your Embedded Engineering Customer - 2006." The report by The William Baldwin Group, Palo Alto, Calif., bases its results on the responses of 640 engineers polled about the devices and tools they use. The survey also shows a substantial increase in the use of 32-bit MCUs (16.3%), while FPGA (field-programmable gate array) use dropped 8.8% and ASIC (application-specific integrated circuit) use dropped 17.4%.

"Traditionally, FPGAs have been used as glue logic and to accelerate repetitive calculations," says study author Nancy B. Green. "Newer microcontrollers are so highly integrated that they may be reducing the need for glue logic. What's more, high performance MCUs that offer 150-plus MIPS and DSP instruction extensions may obviate the need for hardware acceleration. Shorter product life cycles may also be contributing to the shift. It's a lot easier to write code in C/C++ and run it on a processor, than it is to write code in C/C++ and then translate it to a hardware description language."

The study also found FPGA use is closely tied to ASIC use; 74% of FPGAs are used with an ASIC. And assembly language is not dead. Nearly half of engineers do some programming in assembly; 15% use it exclusively. Finally, engineers say controller features, tools, and technical support are equally important in selecting MCU vendors.

From Machine Design, Nov. 22, 2006
## MCU Review

<table>
<thead>
<tr>
<th>MCU</th>
<th>Company</th>
<th>Size</th>
<th>Flash/Ram</th>
<th>ATD</th>
<th>Timer</th>
<th>Interrupts</th>
<th>Speed</th>
<th>Features</th>
<th>Cost</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>HC11</td>
<td>Motorola</td>
<td>8-bit</td>
<td>0k/1k</td>
<td>8 – 8bit</td>
<td>Int.,Ext.</td>
<td>8MHz</td>
<td>SCI, SPI</td>
<td>$12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HC12B32</td>
<td>Motorola</td>
<td>16-bit</td>
<td>32k/1k</td>
<td>8-10bit</td>
<td>2 16-bit Timers</td>
<td>Int.,Ext.</td>
<td>8MHz</td>
<td>Fuzzy logic commands, SCI, SPI</td>
<td>$20</td>
<td>EVB Programmer $100+</td>
</tr>
<tr>
<td>Atmega128</td>
<td>Atmel</td>
<td>16-bit</td>
<td>128k/4k</td>
<td>8-10bit</td>
<td>2 8-bit and 2 16-bit Timers</td>
<td>Int.,Ext.</td>
<td>16MHz</td>
<td>8 PWM ports Internal Osc.</td>
<td>$18</td>
<td>Programmer $29</td>
</tr>
<tr>
<td>PIC16877</td>
<td>Microchip</td>
<td>14-bit</td>
<td>14.2k/368</td>
<td>8-10bit</td>
<td>2 8-bit and 1 16-bit Timers</td>
<td>20MHz</td>
<td>$9</td>
<td>MPLAB IDE simulator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z8 Encore</td>
<td>Zilog</td>
<td>8-bit</td>
<td>64k/4k</td>
<td>12-10-bit</td>
<td>3 timers</td>
<td>Int.,Ext.</td>
<td>20MHz</td>
<td>2 UARTS (RS232)</td>
<td>$15</td>
<td>Programmer $49</td>
</tr>
<tr>
<td>TMS370Cx6x</td>
<td>TI</td>
<td>8 bit</td>
<td></td>
<td>8</td>
<td>2</td>
<td></td>
<td>1 SCI, 1 SPI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMD AM186</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel 8051</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Consider a specific MCU: Motorola M68HC12:

The Motorola M68HC12 is a HCMOS which contains a CPU, memory, a timer and a variety of I/O features (these are shown on the HC12 block diagram, see page 39 of the HC12 Advanced Information book)

Figure 1-1 MC9S12DP256B Block Diagram
The MCU Consists of:
Processor (CPU):
contains a code decoder, an math logic unit, and registers

Bus: Highway connecting the CPU to memory and other I/O. There are address and data buses.

*HC12 is a 16-bit architecture with 16-bit data and address buses

Memory/Output units/Input units

Memory: Memory is a sequence of addressable locations, better referred to as information units.

Registers: CPU registers and I/O registers

I/O ports

RAM

ROM

EEPROM

FLASH

(Please refer to memory map on page 124 of the HC12 Advanced Information book)
Figure 1-2 MC9S12DP256B Memory Map

- **REGISTERS** (Mappable to any 2k block within the first 32k)
  - $0000
  - $03FF

- **4K Bytes EEPROM** (Mappable to any 4k block)
  - $0000
  - $0FFF
  - $1000
  - $3FFF

- **12K Bytes RAM** (Mappable to any 16K and alignable to top or bottom)
  - $4000
  - $7FFF

- **16K Fixed Flash Page $3E = 62** (This is dependent on the state of the ROMHM bit)
  - $6000
  - $BFFF

- **16K Page Window 10 x 16K Flash EEPROM pages**
  - $C000
  - $FFF

- **16K Fixed Flash Page $3F = 63**
  - $FF00
  - $FFFF

* Assuming that a '0' was driven onto port K bit 7 during MCU is reset into normal expanded wide or narrow mode.
### 1.6 Detailed Register Map

The following tables show the detailed register map of the MC9S12DP256B.

#### $0000 - $000F

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Read</th>
<th>Write</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>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>PORTA</td>
<td></td>
<td></td>
<td>Bit 7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$0001</td>
<td>PORTB</td>
<td></td>
<td></td>
<td>Bit 7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>Bit 0</td>
</tr>
<tr>
<td>$0002</td>
<td>DDRA</td>
<td></td>
<td></td>
<td>Bit 7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>Bit 0</td>
</tr>
<tr>
<td>$0003</td>
<td>DDRB</td>
<td></td>
<td></td>
<td>Bit 7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>Bit 0</td>
</tr>
<tr>
<td>$0004</td>
<td>Reserved</td>
<td></td>
<td>Write</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$0005</td>
<td>Reserved</td>
<td></td>
<td>Write</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$0006</td>
<td>Reserved</td>
<td></td>
<td>Write</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$0007</td>
<td>Reserved</td>
<td></td>
<td>Write</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$0008</td>
<td>PORTE</td>
<td>Read</td>
<td>Write</td>
<td>Bit 7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td></td>
<td>Bit 0</td>
</tr>
<tr>
<td>$0009</td>
<td>DDRRE</td>
<td>Read</td>
<td>Write</td>
<td>Bit 7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td></td>
<td>Bit 2</td>
<td>0</td>
</tr>
<tr>
<td>$000A</td>
<td>PEAR</td>
<td>Read</td>
<td>Write</td>
<td>NOACCE</td>
<td>0</td>
<td>PIPOE</td>
<td>NECLK</td>
<td>LSTRE</td>
<td>RDWE</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>$000B</td>
<td>MODE</td>
<td>Read</td>
<td>Write</td>
<td>MODC</td>
<td>MODB</td>
<td>MODA</td>
<td>IVIS</td>
<td>0</td>
<td>EMK</td>
<td>EME</td>
<td></td>
</tr>
<tr>
<td>$000C</td>
<td>UCR</td>
<td>Read</td>
<td>Write</td>
<td>PUPKE</td>
<td>0</td>
<td>0</td>
<td>PUPEE</td>
<td>0</td>
<td>PUPBE</td>
<td>PUPAE</td>
<td></td>
</tr>
<tr>
<td>$000D</td>
<td>DDRIV</td>
<td>Read</td>
<td>Write</td>
<td>RDPK</td>
<td>0</td>
<td>0</td>
<td>ROPE</td>
<td>0</td>
<td>RDPB</td>
<td>RDPB</td>
<td></td>
</tr>
<tr>
<td>$000E</td>
<td>EBICTL</td>
<td>Read</td>
<td>Write</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ESTR</td>
<td></td>
</tr>
<tr>
<td>$000F</td>
<td>Reserved</td>
<td></td>
<td>Write</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Registers come in two types: CPU registers and I/O/Memory registers

CPU Registers: (no location in memory)
Accumulators A, B, D

Index Registers X & Y

Stack pointer

Program counter

CCR

<table>
<thead>
<tr>
<th>Accumulator A (7 – 0)</th>
<th>Accumulator B (7 – 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Double Accumulator D (15 – 0)</td>
</tr>
<tr>
<td></td>
<td>Index Register X (15 – 0)</td>
</tr>
<tr>
<td></td>
<td>Index register Y (15 – 0)</td>
</tr>
<tr>
<td></td>
<td>Stack Pointer (15 – 0)</td>
</tr>
<tr>
<td></td>
<td>Program Counter (15 – 0)</td>
</tr>
<tr>
<td></td>
<td>Condition Code Register</td>
</tr>
<tr>
<td></td>
<td>S X H I N Z V C (7 – 0)</td>
</tr>
</tbody>
</table>

Control Registers (I/O registers, 512 memory locations, initially located at $0000$-$01FF$)

Examples:
DDRA (Data direction register A)

DDRB

etc.
Addressing Modes:

Immediate (#)
ex: LDAA #!64

Direct
ex: LDAA $64

Extended
ex: LDAA $1064

Indexed
ex: LDAA 0,x

Inherent

Relevant

Instruction Execution Cycle
Programming Style for Microcontrollers

Microcontrollers (MCU's) are generally programmed in one of two ways:

1) Use a standard programming language (usually C) to develop your code, and then compiling this code (using a compiler such as Code Warrior) to yield assembly suitable for your microcontroller. This approach is probably the most common in microcontroller applications. However, to be proficient in this approach one needs a thorough understanding of how a microcontroller works at an assembly level.

2) Write assembly language directly for your MCU using its available instruction set, and is then assembled to result in machine code (for example using the ASM12 assembler). This approach generally results in a more efficient use of MCU resources.

The remainder of this discussion will focus on assembly language programming.

Assembly language programming style:

An assembler converts source files to machine code. Code written in assembly language has a nearly direct conversion to machine code (every command in assembly has a direct counterpart in machine code). Assembly is written in mnemonics; short 3-5 letter identifiers of a given command. For example, LDDA stands for the instruction load accumulator A, BRA stands for Branch, etc. Machine code is written in hex, and ultimately binary. Some people do this so much they can actually read binary code and enjoy it.

The following is an example of a standard format that can serve as an outline for all your programs:

Program Header: Programmer, date, briefly describe program
Assembler directives: Define constants/variables to be used
Program location: Tells assembler were program resides
Program initialization: Initialize stack pointer and other variables
Main: Body of the main program
End: End of the main program
Subroutines
Data constants: Locate constants in data in Flash
Data variables: Locate variables in RAM

Examples of these elements follow:

Program Header:

*  Program Example.asm
*  Stephen Canfield, 1/15/2000
*  This program will demonstrate a standard format for creating
  * assembly programs with the HC12.

Assembler Directives:
Assembler directives (generally equates) are not MCU code, but rather information the assembler uses when assembling your program. The predominant use is in defining data names using the statement EQU. Here are some examples:

<table>
<thead>
<tr>
<th>Label</th>
<th>EQU</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORTA</td>
<td>EQU</td>
<td>$0000</td>
<td>Assign the word PORTA with the value $0000</td>
</tr>
<tr>
<td>DDRA</td>
<td>EQU</td>
<td>$0002</td>
<td>Assign DDRA with the value $0002</td>
</tr>
<tr>
<td>STACK</td>
<td>EQU</td>
<td>$8000</td>
<td>Location of the STACK</td>
</tr>
<tr>
<td>PROG</td>
<td>EQU</td>
<td>$0800</td>
<td>Starting location for the program</td>
</tr>
<tr>
<td>hitime</td>
<td>EQU</td>
<td>20</td>
<td>Data for the program</td>
</tr>
<tr>
<td>lowtime</td>
<td>EQU</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>ORG</td>
<td>PROG</td>
<td></td>
<td>Start loading the program at PROG</td>
</tr>
</tbody>
</table>

Program Initialization:
The stack pointer must be initialized before it is used, and variables (data in memory) must be initialized in the program.

```assembly
lds #STACK ;Load the stack pointer with the number STACK
```

Main:
The body of your program exists here. If your program has more than one task, it is best to make the main program consist of calls to specific modules in the program.

```assembly
* Main
ldd #STACK ;Load the stack pointer with the number STACK
main   jsr     get_IR
       jsr     get_sona
       jsr     driv_mot
       bra     main
```

An assembly program is written in any text editor and is organized into the following fields;

<table>
<thead>
<tr>
<th>Field</th>
<th>Field</th>
<th>Field</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Opcode</td>
<td>Operand</td>
<td>Comment</td>
</tr>
<tr>
<td>ex.</td>
<td>start</td>
<td>LDAA</td>
<td>#$FF</td>
</tr>
<tr>
<td>repeat</td>
<td>repeat</td>
<td>BRA</td>
<td>start</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>;load ACCA with FF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>;branch to start</td>
</tr>
</tbody>
</table>

All fields must be separated by a whitespace. The label and comment fields are optional. The comment field must begin with a semicolon.
Program subroutines:
Subroutines provide a valuable tool to create and debug code, and should be used as part of a good programming technique. As shown above, the main body of the program can make calls to subroutines in the program, making program operation clear and compartmentalized. This allows the program to be written and tested piece by piece. Once a good subroutine is written, it can be used time after time. You can pass information too and from a subroutine using accumulators and the stack, making it a functional block. Good programming also dictates the use of subroutines rather than branches or jump commands, these should be reserved for implementing if-then or for loops.

* Subroutine get_IR
* This subroutine will take readings from each IR sensor, and store this data in the variables, IRdata1 – Irdatan.
* Accumulators A and B are modified in this subroutine

get_IR

; subroutine code here

rts ; return from subroutine

* End subroutine get_IR

Constant Data
Constants can be defined in ROM.

Variable Data
Variable data is stored in RAM.

Other issues:
Other issues in assembly programming style include the following:
- Numbers used in the code should be named with and equate in general
- Indentation is generally fixed by a required syntax for the assembler
- Assembly text is not case sensitive, use case to make your code more readable.
- You can use the include command to include code common to multiple programs, for example:
  
  include f:\mechatronics\include.h

Creating If-Then-Else loops:
The following code demonstrates and if-then-else loop.

* Somewhere in a program
* This code takes a reading from an analog sensor and sets PA0 low if the value is below a threshold, and sets PA0 high if the value is above or equal to the threshold.

ldaa AT_Port0 ; get the sensor reading
cmpa #Min_valu ; compare with minimum value
bhs Else ; goto Else if higher than or the same
ldab PA0high ; set PA0 high
stab PORTA
bra End_if ; end if loop
Creating For loops (or Do While loops)
Below is an example of a for loop (or use of a counter).

* Somewhere in a program
* This code completes an operation n times, with n defined in the equates

    ldx  #n
* Start loop
Do
    ; Enter tasks that need to be performed
    dex ; decrement x
    bne Do ; branch not equal to zero (while x not equal to zero)

OR:

    ldx  #n
* Start loop
Do
    ; Enter tasks that need to be performed
    dbne x,Do ; decrement x, branch not equal to zero

Summary:
Writing solid assembly programs comes with practice. Look for other programs to gain experience and get new ideas in programming as well.

Example:
Below is an example program, demonstrating some of the programming style elements discussed in this chapter.

;***EQUATE STATMENTS***
PORTA EQU $0000 ;ADDRESS OF PORT A
DDRA EQU $0002 ;ADDRESS OF DATA DIRECTION REGISTER FOR PORTA
INIT EQU %00000010 ;INIT CONNECTED TO PA1
ECHO EQU %00000001 ;ECHO CONNECTED TO PA0
OUT4HEX EQU $F698 ;JSR [OUT4HEX,PCR] FROM ACC. B
TCNT EQU $84 ;ADDRESS OF 16-BIT TIMER
TSCR EQU $86 ;ADDRESS OF TIMER CONTROL REGISTER
TMSK2 EQU $8D ;ADDRESS OF TIMER PRESCALER

ORG $0800
    MOV $%11111110,DDRA ;PORTA = ALL OUTPUTS EXCEPT PA0
    BSET TSCR,$%10000000 ;START CLOCK
    BSET TMSK2,$%00000011 ;PRESCALE CLOCK
;********************
;*****MAIN PROGRAM*****
;***********************

MAIN
    JSR SONAR ; JUMP TO SONAR SUBROUTINE
    JSR DISPLAY ; JUMP TO DISPLAY SUBROUTINE
    SPIN BRA SPIN

;***************************************************************
;**********SONAR SUBROUTINE GETS DISTANCE.**********
;***************************************************************

SONAR
    BSET PORTA,INIT ; SET INIT PIN
    LDD TCNT ; LOAD CURRENT TIME INTO ACC D
    STD START ; STORE THAT TIME AS THE START TIME
    WAIT BRCLR PORTA,ECHO,WAIT ; WAIT FOR THE ECHO TO COME BACK
    LDD TCNT ; LOAD CURRENT TIME INTO ACC D
    SUBD START ; FIND THE DIFFERENCE BETWEEN CURRENT TIME AND
                 ; THE START TIME
    STD DISTANCE ; STORE THIS TIME DIFFERENCE AS THE DISTANCE
    BCLR PORTA,INIT ; RESET THE INIT PIN, READY FOR NEXT READING
    RTS ; RETURN FROM SUBROUTINE

;***************************************************************
;**********DISPLAY SUBROUTINE DISPLAYS DISTANCE TO TERMINAL**
;***************************************************************

DISPLAY
    LDD DISTANCE ; DISTANCE VALUE INTO ACC D
    JSR [OUT4HEX,PCR] ; OUTPUT CURRENT DISTANCE TO SCREEN
    JSR DELAY ; ALLOW FOR CHARACTERS TO BE TRANSMITTED
    RTS

;***************************************************************
;********** DELAY SUBROUTINES **********
;***************************************************************

DELAY   PSHX ; push the contents of X onto the stack
PSHY ; push the contents of y onto the stack
LDY #$001F ; load Y with the number $001F
DELAYY  LDX #$0018
DELAYX  DEX
BNE DELAYX
DEY
BNE DELAYY
PULY ; Pull from the stack and load in Y
PULX ; pull from the stack and load in x (note reverse order)
RTS ; return from subroutine

DISTANCE DS 2 ; DEFINE SPACE FOR DISTANCE VALUE