ECET 310-001
Chapter 1
& Course Introduction

W. Barnes, 9/06, rev. 9/07

Ref. Huang, Han-Way, *The HCS12/9S12: An Introduction to Software and Hardware Interfacing,* Thomson/Delmar.

In This Set of Slides:

1. First day of class
2. What can a microntroller do, Dragon 12 do?
3. Computer Architecture
4. µP vs. µctlr
5. HCS 12 Features
6. Memory
7. Software
8. List file of a program
9. HCS12 registers
10. More HCS12 Details
11. Memory Addressing
12. Six Basic Addressing Modes (First 5)

First Day of Class

- Basics of the class
  - Syllabus
  - Progress Reports and Teamwork
- Basics of the lab
  - Hardware and Software
  - Reports
- Number Systems (binary and hex)
  - Review Sheet In-class Assignment

What are some of the things a Microcontroller Do?

- **It can calculate** – from simple arithmetic to more complex functions.
- **It can input signals** – it will digitize a signal, and store it.
- **It can output a signal** – either a signal that is generated by a microcontroller, or from a stored input signal
- **It can modify a signal** – By performing calculations on a stored signal, you can modify all or portions of a signal.
- **It can use a stored lookup table –** Can compare a stored signal to a lookup table.
- **It can make decisions based on certain levels or attributes of a signal –** Can use logic functions to decide when to have certain outputs.
- **It can store a signal for later use –** You might want to save an input for a period of time, before outputting

A Few Examples of What the Dragon12 Board Can Do

(These are used as a startup exercise in Senior Project)

- Input a number on the computer, and output that number to the display
- Input two numbers, the computer stores the numbers, displays each number as it's input, and calculates and displays the sum.
- Save a lookup table that will convert an input, and display the sine of that number. The number you will input would be an angle, in degrees from 00 to 900, in increments of 50.
- Input a number via the keyboard. If the number is less than a threshold, nothing happens. If a number is greater than the threshold, an LED will blink.
- Input a sinusoid signal from a function generator and calculate the frequency of the signal, displaying it.

I. Computer Architecture

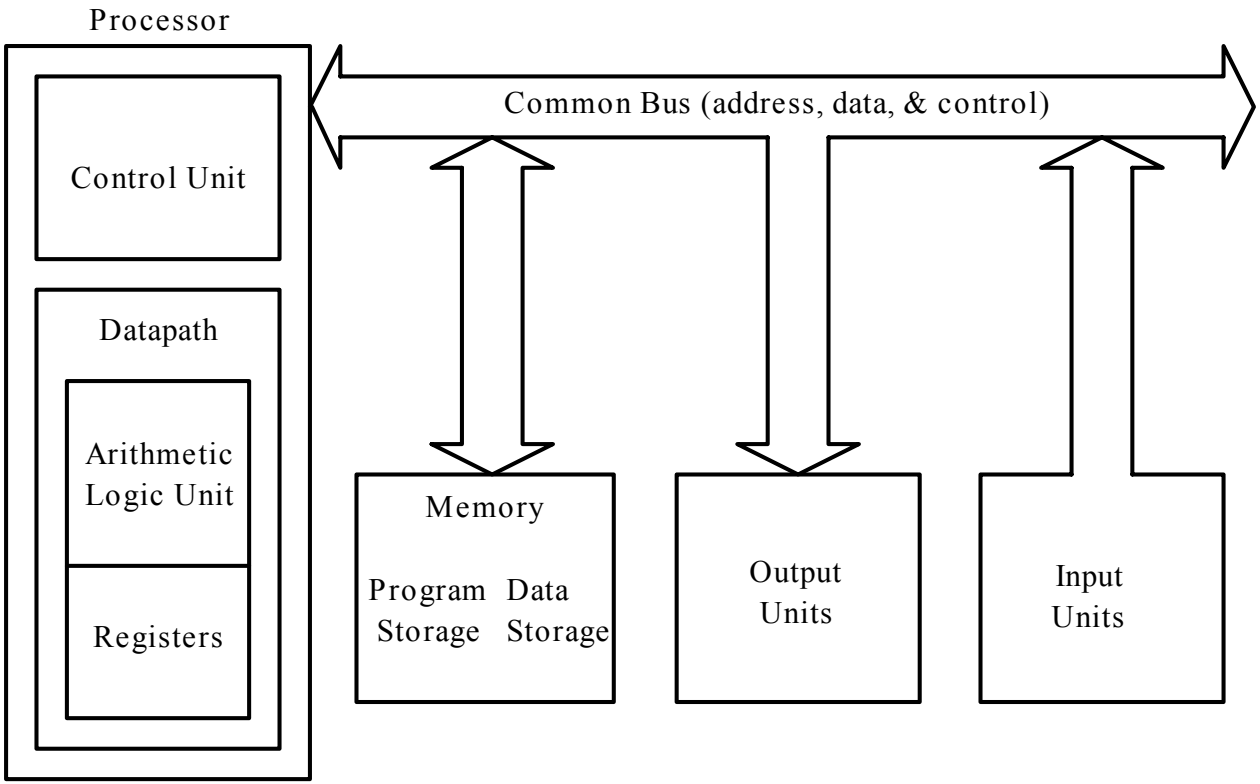

Figure 1.1 Computer Organization

II. µP vs. µctlr

- µP & its limitations (needs: ext. memory, I/O interfaces, digital glue such as decoders)

  primary use: number crunching

- µctlr features beyond µP : internal memory, timers, A/D, D/A, DMA ctlr, parallel port(s), Asynch. Serial I/O, DSP functions

  primary use: control of and response to 'outside world' – often called embedded systems

- Specific HCS 12 features- next slide

HCS12 Microcontroller Features

- 16-bit CPU
- 64 KB on-board memory space (also supports expanded memory up to 1 MB through a 16-KB window)
  - 0 KB to 4KB of EEPROM
  - 2 KB to 14 KB of on-chip SRAM
  - 32 KB to 512 KB flash memory
- Sophisticated timer functions that include: input capture, output compare, pulse accumulators, real-time interrupt, and COP timer
- Serial communication interfaces: SCI, SPI, CAN, BDLC
- Background debug mode (BDM)
- 10-bit A/D converter
- Instructions for supporting fuzzy logic function
- Controller Area Network (CAN)- useful particularly in autos

III. (Semiconductor) Memory

- RAM (SRAM and DRAM)
- ROM
  - Mask-programmed (MROM)
  - Programmable (PROM): is fuse-based
  - Erasable programmable (EPROM) by ultraviolet light in bulk only
  - Electrically Erasable programmable ROM (EEPROM) by location, row, etc.
- Flash memory
  - Electrically programmable and erasable many times in bulk or a sector at a time

IV. Software including firmware

- Machine code in binary (note some are 1 byte and some 2 bytes)
  - 0001 1000 0000 0110: A ← [A] + [B] (addition of 2 reg's)
  - 0100 0011: A ← [A] + 1 (increment a register)
  - 1000 0110 0000 0110: A ← 6 (load a register w/constant)
- Assembly language in mnemonics
  - Drawbacks: need hardware familiarity, programs hard to understand, low productivity for large projects
- High level (C) better but may execute *relatively* slowly
- (Cross) Assemblers and (Cross) Compilers convert source to object code

Program Example (from a *list file*)

line	Address	Machine code	Source Code
1			org $2000
2	2000	B6 10 00	ldaa $1000
3	2003	BB 10 01	adda $1001
4	2006	BB 10 02	adda $1002
5	2009	7A 11 00	staa $1100
6			end

- Three Important Questions:
  - Why is hexadecimal used so much?
  - What 24 bits will be stored for B6 10 00?
  - Why no machine code for end or org?
  - This program uses direct addressing (discussed later), what does it accomplish?

One Critical Question from Example

What can we assume the computer knows?

Answer: Nothing! Everything you want the computer to do you have to tell it in the minutest detail

HCS 12 Registers
(also called Programmer's Model)

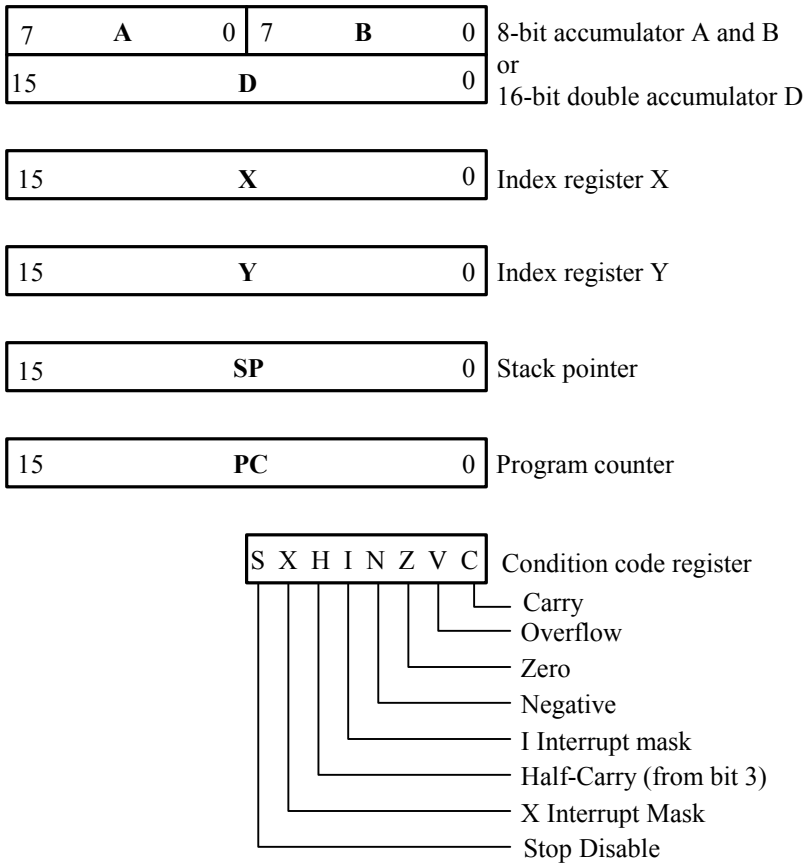

Figure 1.2 HCS12 CPU registers.

More HCS 12 Details

- LIFO stack
  - Works towards lower addresses
  - See Fig. 1.3, p. 10
- 16 bit machine (accessible single to 32 bits, signed and unsigned)
- Base Prefixes
  - None for decimal (default)
  - % for binary
  - $ for hex

Memory Addressing

- **Some Useful Definitions**
  - 1k ≡ 1024 or 2^{10} ; 1M ≡ 1,048,576 or 2^{20}
  - M[$100] ≡ contents of location $100
  - [D] ≡ contents of register D, etc.
- **The Six Basic Addressing Modes**

1. Inherent
2. Immediate
3. Direct
4. Extended
5. Relative
6. Indexed

Six Basic Addressing Modes cont'd

(1) Inherent

Characteristics: if an operand (number operated on), it is only a register

Examples: clra, nop, inx, deca

What will the processor do after accessing each of these four different instructions?

Six Basic Addressing Modes cont'd

(2) Immediate

Characteristics: operand is part of the instruction

Examples:
(ldd #$1022, adda #33, ldx #N)

What will the processor do after accessing each of these three different instructions?

What is the significance of the *label* 'N'? Note that label is replaced by an actual number during assembly.

Six Basic Addressing Modes cont'd

(3) **Direct**

Characteristics: operand accessed using 1 byte address (between $00 and $FF only)

Examples: ldaa $45, suba num1

What will the processor do after accessing each of these two instructions?

What is the significance of the *label* 'num1'?

What would be the difference between ldaa $45 and ldaa #$45?

Six Basic Addressing Modes cont'd

(4) **Extended**

- same as Direct but 16 bits
- Usually used with a label

Six Basic Addressing Modes cont'd

5. **Relative** (Used only by branch instructions)

- Conditional branch instructions use only the relative mode.
- BRCLR and BRSET instructions can also use relative mode to specify branch target.
- Short branch instructions consist of an 8-bit opcode and a signed 8-bit offset (a range of -128 ~ +127).
- Long branch instructions consist of an 8-bit opcode and a signed 16-bit offset. (a range of -32768 ~ +32767)

» Continued on next slide

Six Basic Addressing Modes cont'd

(Relative Addressing Cont'd.)

Almost always, the programmer uses a symbol (label) to specify the branch target and *the assembler will calculate the actual branch offset* (distance) from the instruction that follows the branch instruction.

Example:

```
repeat:   xxxx          ; the first instruction of this sequence
          …
          …
          bmi   repeat  ; a label rather than a number for offset
          …             ; program continues
```