

ECE 265 Comprehensive Notes

Christopher J. Atherton

April 15, 2009

1 Brief Overview of Microcontrollers (Lectures 1–3)

Microcontrollers are

- small microprocessors,
- embedded in products,
- not computationally intensive (not for, say, spreadsheets on your PC), and
- numerous (billions are around), and cheap (our MC68HC11 costing about \$8.59 in hundreds quantities).

1.1 Basic Structure

The basic structure of the microcontroller has not undergone a dramatic change since the von Neumann architecture came about in the EDVAC computer in 1945. There are four general components.

- CPU (containing registers and ALU)
- Memory
- Control unit
- I/O unit

Often the I/O for the device is *memory mapped*, meaning some locations in memory are reserved for I/O. The term *microprocessor* refers to just the CPU and the control unit, while *microcomputer* refers to all four elements in combination.

The Thermostat Example Consider the thermostat (a common application of microcontrollers). It possesses

- an analog-to-digital convertor, which interfaces the microcontroller to the thermo couple,
- memory to store programmed data, and
- a CPU, which has been programmed to perform all of the thermostat's functions.

About the M68HC11 Microcontroller The M68HC11 is a family of 8-bit microcontrollers. These chips, like many microcontrollers carry two modes of operation, a *single-chip mode* and *expanded mode*. Single-chip mode is intended for use in the final product, while expanded mode is used as a debugging aid. The exact name of each chip in the family is meaningful and contains pertinent information. For instance, the MC68HC11A8 has a high-density CMOS (HC), 8k ROM (A8), as well as 512k EEPROM and 256 bytes RAM. The MC68HC11E1 version has 512 EEPROM, 512 RAM, 8k ROM (disabled), and runs at clock speed 2 Mhz.

1.2 Memory

Memory comes in 51 flavors.

- Electrically Erasable Programmable Read-Only Memory (EEPROM) - Just read-only memory that can be electrically erased and reprogrammed.
- Erasable Programmable Read-Only Memory (EPROM) - This is read-only memory that needs to be exposed to ultraviolet light to be erased. Reprogramming also needs special equipment.

Memory also sometimes isn't real memory. *Memory-mapped I/O* is a common technique that interfaces I/O using memory addresses. In embedded applications, there is 64k of memory addressed from \$0000-\$FFFF. In the MC68HC11E1, in particular, RAM occupies addresses \$0000-\$00FF. \$E000-\$FFFF is mapped to EPROM. This EPROM is coded with what is called a *monitor*, debugging software provided from the manufacturer, and is essentially a BIOS. Finally, some of the address space is used for memory-mapped I/O. Output 13 of the MC68HC11E1 is located at \$1004.

1.3 Ports

Here is an overview of the MC68HC11E1 ports (places for data to go in and out).

Port	Input Pins	Output Pins	Bidirectional Pins	Shared Functions
A	3	3	2	Timer/Counter
B	-	8	-	High-order address
C	-	-	8	Low-order address or data bus
D	-	-	6	SCI, SPI
E	8	-	-	A/D converter

And more detailed description.

- Port A
 - Provides a 16-bit timer system (used for keeping time)
 - Provides a counter (used for counting)
- Port B
 - Normally used for I/O
 - In expanded mode, provides a memory address bus or data bus
- Port D
 - Two serial ports
 - Serial communication interface (SCI) [2-bit]
 - Serial peripheral interface (SPI) [4-bit]
- Port E
 - A/D converter
 - 8-channel [8-bit]

1.4 Other Components

(Review) info dump.

- Register - set of D flip-flops that store data
- Accumulator - register to “accumulate” intermediant results
- Bus - collection of wires for data transportation

2 High Level Microcontroller Operation (Lecture 3)

This lecture was an info dump on how data flows around the microcontroller as it performs operations. The diagrams cannot be found here yet (maybe in future), so for now these diagrams will only be available in the notes notes and a handout.

3 Register Transfer Notation (Lecture 3)

Register transfer notation is a tool for documenting computer architecture. For example, $A + \text{Mem}(\text{MA}) \implies A$ says to take the data in the A accumulator, add it to the data in memory at MA and store the result into the A accumulator. You can specify specific bit in a register using angle brackets like so $A \equiv A \langle 7 : 0 \rangle$ (all 8 bits of A) or $A \equiv A \langle 3 : 0 \rangle$ (least significant nibble of A). Arrows, \implies , show the transfer of data. $R \langle a : b \rangle \implies S \langle c : d \rangle$ is a *copy operation* for R to S. Note that $a - b$ must equal $c - d$ for stuff to make sense. Another example, $A \langle 7 : 1 \rangle \implies A \langle 6 : 0 \rangle$ is a shift right one (bit 7 doesn't change). A *load operation* might have any of the following syntaxes: $n \implies R \langle i : j \rangle$, $147 \implies B \langle 7 : 0 \rangle$, or $\$E \implies A \langle 3 : 0 \rangle$. Another thing to note is that when we specify a span of bits, the other bits in the register are unaffected by whatever is going on. Earlier, we used the Mem(MA) syntax. The argument to Mem is a pointer to something in memory.

4 Basic Instruction Organization and Timing (Lecture 3)

Instructions are stored in memory as *opcodes*. An *assembler* converts the human-intelligible *mnemonics* into machine-intelligible opcodes.

5 Overview of Registers (Lecture 4)

Some registers.

5.1 Program Counter (PC)

The *program counter* register (PC) maintains the location of the next instruction in memory.

5.2 Instruction Register (IR)

The *instruction register* maintains the opcode of the current instruction.

5.3 Stack Pointer (SP)

The *stack pointer* register maintains a pointer to the top of the stack and is incremented and decremented by pushes and pulls to the stack.

5.4 Temporary (TMP)

The *temporary registers* were not explained, and I believe prof said that these are not used directly, instead they are used to execute certain instructions.

5.5 X and Y Index

The *X and Y index registers* are used to form pointers and were not explained in detail yet.

5.6 Condition Code Register

The *condition code register* maintains various infos related to results of arithmetic or logic operations and is also used for controlling interrupts. And here are some of those bits (Lecture 6).

- 1 - Carry (C) - Carry bit (C_7)
- 2 - Overflow (V) - Overflow for signed numbers (C_7 xor C_8)
- 3 - Zero (Z) - If result is \$00, set on
- 4 - Significand (N) - Most significant bit of result (can be used to indicate sign)
- 6 - Half carry (H) - Another carry bit (C_4)

6 Instruction Interpretation and Execution (Lectures 4–5)

Here we layout the process for executing an instruction. In this example, an ADDA operation (adds a number in memory to the number in accumulator A and stores the result in accumulator A).

1. Fetch

(a)

$$\text{Mem(PC)} \implies \text{IR}$$

$$\text{PC} + 1 \implies \text{PC}$$

(b)

$$\text{Mem(PC)} \implies \text{MA} \langle 15 : 8 \rangle$$

$$\text{PC} + 1 \implies \text{PC}$$

(c)

$$\text{Mem(PC)} \implies \text{MA} \langle 7 : 0 \rangle$$

$$\text{PC} + 1 \implies \text{PC}$$

2. Execute

(a)

$$A + \text{Mem(MA)} \implies A$$

8 Operations

8.1 ABA (Lecture 5)

Adds A to B and stores the result in A.

- Mnemonic - ABA
- Operation - Add B to A
- Register transfer - $A + B \implies A$
- Opcode - \$1B
- Bytes - 1
- Changes - H, N, Z, V, C (in CCR)
- Cycles - 2

Here's the transfers for ABA.

Fetch (1)	$\text{Mem(PC)} \implies IR$
	$PC + 1 \implies PC$
Execute (2)	$A + B \implies A$

8.2 PSH and PUL (Lecture 5)

PSH and PUL push and pull numbers onto and off of the stack.

PULA	PSHA
$SP + 1 \implies SP$	$A \implies \text{Mem(SP)}$
$\text{Mem(SP)} \implies A$	$SP - 1 \implies SP$

9 Operations with Signed Numbers (Lecture 5)

Know the three ways to convert a number to its two's complement.

10 Addressing Modes (Lecture 6)

Data is stored in a bunch of different places.

- Memory
- Registers
- Memory (for programs)

So we say there is a variety of ways to refer to said data, which we call addressing modes.

- Immediate mode - data stored with instruction codes
For example, LDAA #\$20 loads \$20 into A. The \$20 is stored in memory along with the program (right after the corresponding opcode as an argument).
- Inherent mode - data location is integrated as part of the instruction opcode
For example, ABA adds the A and B registers together and stores the result in A. The operation implicitly tells where to find the pertinent data.
- Extended mode - data stored in main memory
For example, LDAA \$032C loads the data stored at memory location \$032C into A. The data is pointed to by the argument of the operation (data tells how to find other data).