ECE 331: Handout 1

Timeline of Computer History Highlights

For more details, see textbook section 1.1

|		
~ 2000 B.C. ~ 500 B.C.	Sumer (Sumerian) China	Abacus -First *calculating machine*
~1650 A.D.	Wilhelm Schickard (and others including Blaise Pascal)	First *mechanical adder/subtractor* (not programmable)
1837	Charles Babbag	First concept and design a fully programmable mechanical computer
1937	Alan Turing	Concept of the **algorithm** and computation with the **Turing machine**
1940s (WWII)		First plans for an electrical computer (not planned for personal use)
1950	John Mauchly and J. Presper Eckert (University of Pennsylvania, U.S. Army)	ENIAC (**Electronic Numerical Integrator and Computer**) –First general-purpose *electronic compute*r using relay memories and vacuum tubes
1948	William Shockley et.al. (Bell Labs)	First *semiconductor transistor*; the beginning of the microelectronics age
1958	Jack Kilby (Texas Instruments) won Nobel Prize in 2000	First *integrated circuit* (multiple transistors in one substrate); built in germanium
1959	Robert Noyce (Fairchild Semiconductor)	First *silicon integrated circuit*
1971	Intel	Intel 4004 (4-bit CPU) -First commercial single-chip *microprocessor*

Computer (Microcontroller) Architecture

Basic Computer Organization (architecture):
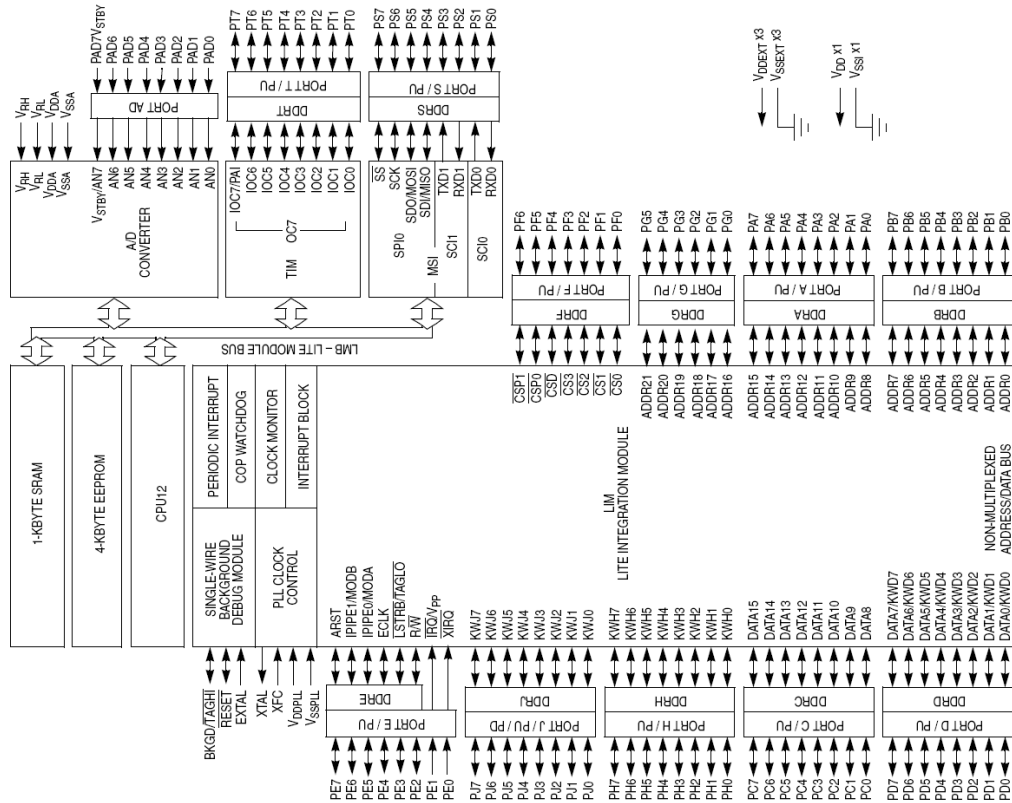Stored-program computer A.K.A. von Neumann architecture
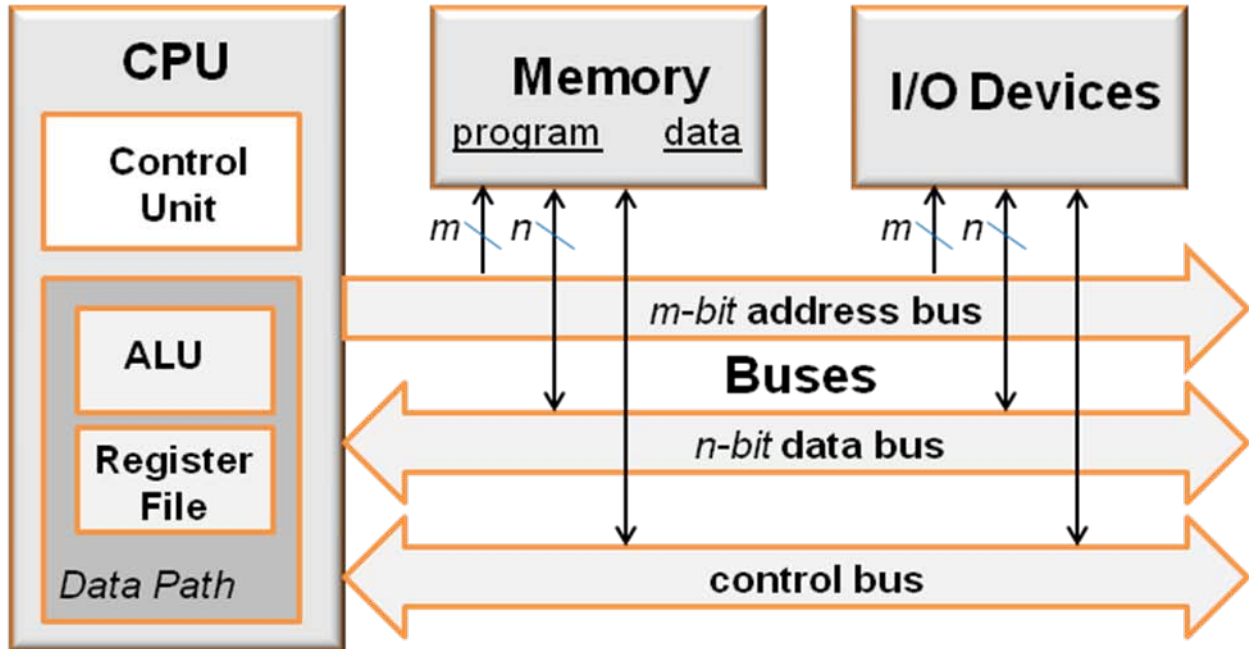
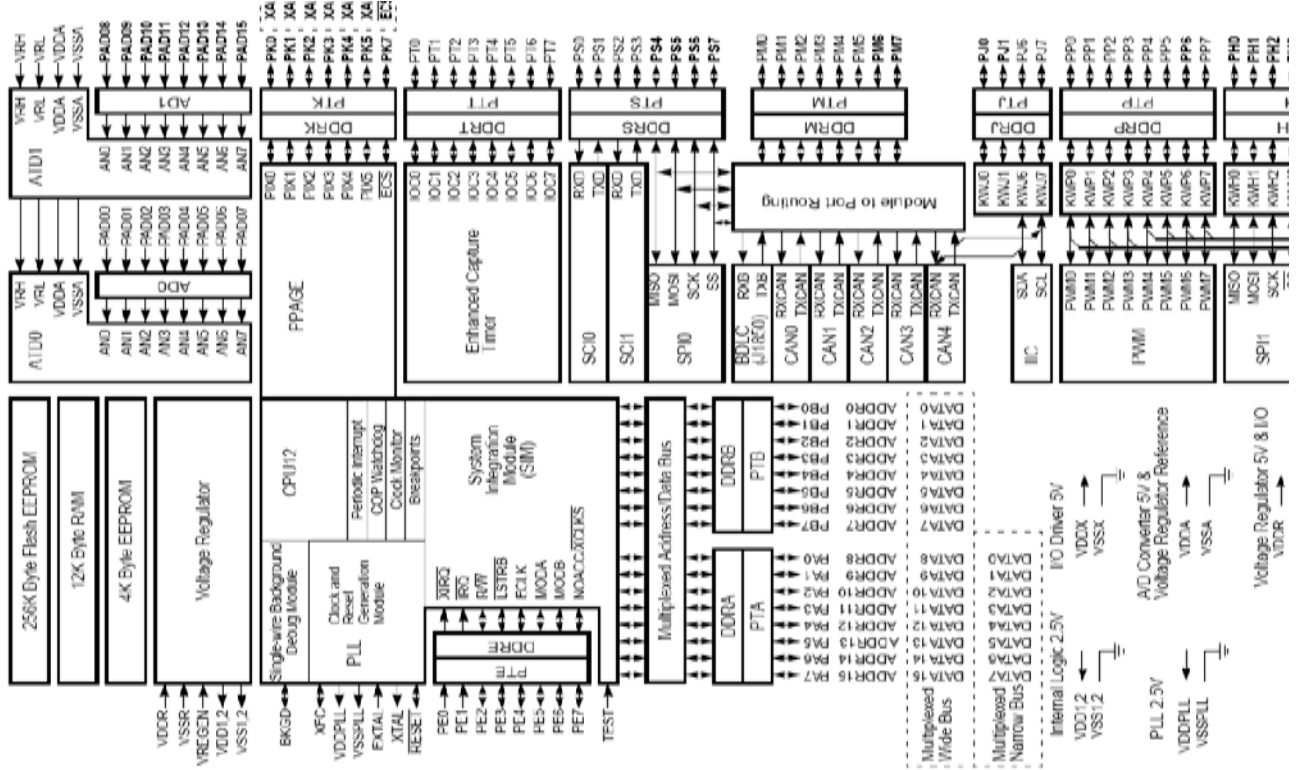Functional structure of CPU
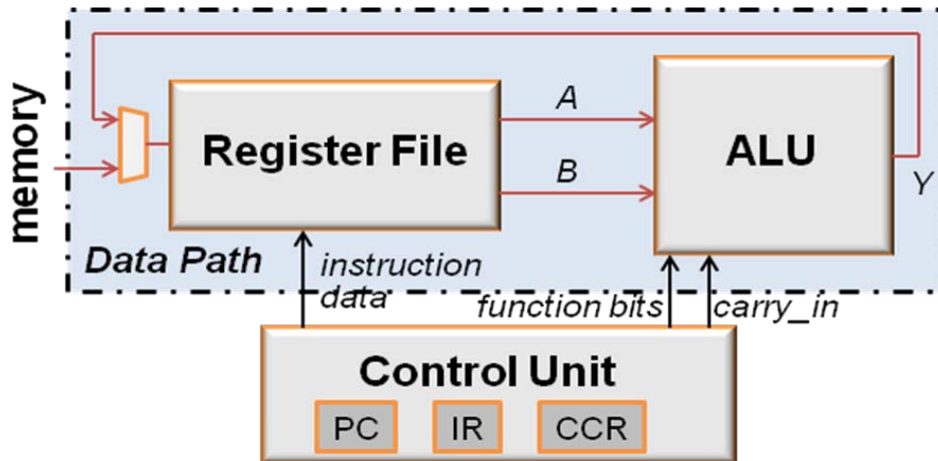
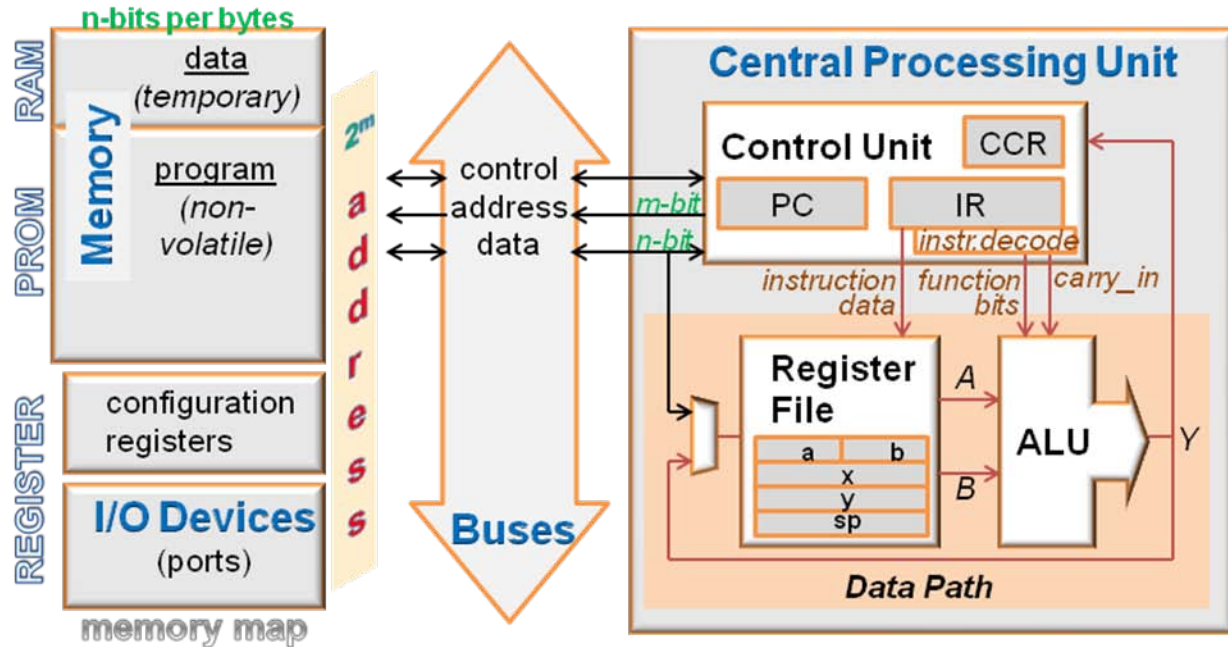# Freescale HC12/S12 Diagrams

MC68HC812A4 Block Diagram

MCS12DP256B Block Diagram

Figure 1-1 MC9S12DP256B Block Diagram

Microcontroller architecture with more details

Terminology and History of Stored-Program Computer Organization

Preface and Contents

All the text and figures below have been extracted from Wikipedia and should be credited to those authors. Please note that Wikipedia articles are user-written and not reviewed by experts; although the information is generally correct, it should not be trusted 100%.

Feb. 2010, A. Mason

Contents

von Neumann Architecture

The **von Neumann architecture** is a design model for a stored-program digital computer that uses a central processing unit (CPU) and a single separate storage structure ('memory') to hold both instructions and data. It is named after the mathematician and early computer scientist John von Neumann. Such computers implement a universal Turing machine (see below) and have a sequential architecture.
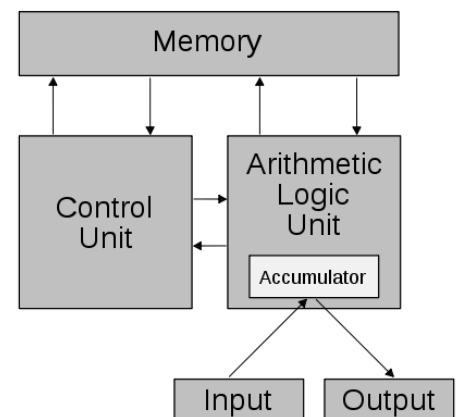
A **stored-program** digital computer is one that keeps its programmed instructions, as well as its data, in read-write, random-access memory (RAM). Stored-program computers were an advancement over the program-controlled computers of the 1940s, such as the Colossus and the ENIAC, which were programmed by setting switches and inserting patch leads to route data and to control signals between various functional units. In the vast majority of modern computers, the same memory is used for both data and program instructions. The mechanisms for transferring the data and instructions between the CPU and memory are, however, considerably more complex than the original von Neumann architecture.

The terms "von Neumann architecture" and "stored-program computer" are generally used interchangeably, and that usage is followed in this article.

The earliest computing machines had fixed programs. Some very simple computers still use this design, either for simplicity or training purposes. For example, a desk calculator (in principle) is a fixed program computer. It can do basic mathematics, but it cannot be used as a word processor or a gaming console. Changing the program of a fixed-program machine requires re-wiring, re-structuring, or re-designing the machine. The earliest computers were not so much "programmed" as they were "designed". "Reprogramming", when it was possible at all, was a laborious process, starting with flowcharts and paper notes, followed by detailed engineering designs, and then the often-arduous process of physically re-wiring and re-building the machine. It could take three weeks to set up a program on ENIAC and get it working.[1]

The idea of the stored-program computer changed all that: a computer that by design includes an instruction set and can store in memory a set of instructions (a program) that details the computation.

In computing, **SISD** (**S**ingle **I**nstruction, **S**ingle **D**ata) is a term referring to a computer architecture in which a single processor executes a single instruction stream, to operate on data stored in a single memory. This corresponds to the von Neumann architecture. SISD is one of the four main classifications as defined in Flynn's taxonomy (==see below==). In this system classifications are based upon the number of concurrent instructions and data streams present in the computer architecture. According to Michael J. Flynn, SISD can have concurrent processing characteristics. Instruction fetching and pipelined execution of instructions are common examples found in most modern SISD computers

von Neumann bottleneck

The separation between the CPU and memory leads to the *von Neumann bottleneck*, the limited throughput (data transfer rate) between the CPU and memory compared to the amount of memory. In most modern computers, throughput is much smaller than the rate at which the CPU can work. This seriously limits the effective processing speed when the CPU is required to perform minimal processing on large amounts of data. The CPU is continuously forced to wait for needed data to be transferred to or from memory. Since CPU speed and memory size have increased much faster than the throughput between them, the bottleneck has become more of a problem, a problem whose severity increases with every newer generation of CPU.

The performance problem can be alleviated (to some extent) by several mechanisms. Providing a cache between the CPU and the main memory, providing separate caches with separate access paths for data and instructions (the so-called Harvard architecture (==see below==)), and using branch predictor algorithms and logic are three of the ways performance is increased.

Universal Turing Machine

In computer science, a **universal Turing machine** is a Turing machine that can simulate an arbitrary Turing machine on arbitrary input. The universal machine essentially achieves this by reading both the description of machine to be simulated as well as the input thereof from its own tape. Alan Turing introduced this machine in 1936–1937. This model is considered by some (for example, Martin Davis (2000)) to be the origin of the stored program computer—used by John von Neumann (1946) for the "Electronic Computing Instrument" that now bears von Neumann's name: the von Neumann architecture.

Harvard Architecture

The **Harvard architecture** is a computer architecture with physically separate storage and signal pathways for instructions and data. The term originated from the Harvard Mark I relay-based computer, which stored instructions on punched tape (24 bits wide) and data in electro-mechanical counters. These early machines had limited data storage, entirely contained within the central processing unit, and provided no access to the instruction storage as data, making loading and modifying programs an entirely offline process.

Today, most processors implement such separate signal pathways for performance reasons but actually implement a Modified Harvard architecture, so they can support tasks like loading a program from disk storage as data and then executing it.

Memory details
In a Harvard architecture, there is no need to make the two memories share characteristics. In particular, the word width, timing, implementation technology, and memory address structure can differ. In some systems, instructions can be stored in read-only memory while data memory generally requires read-write memory. In some systems, there is much more instruction memory than data memory so instruction addresses are wider than data addresses.

Contrast with von Neumann architectures
Main article: Von Neumann architecture
In a computer with the contrasting von Neumann architecture (and no CPU cache), the CPU can be either reading an instruction or reading/writing data from/to the memory. Both cannot occur at the same time since the instructions and data use the same bus system. In a computer using the Harvard architecture, the CPU can both read an instruction and perform a data memory access at the same time, even without a cache. A Harvard architecture computer can thus be faster for a given circuit complexity because instruction fetches and data access do not contend for a single memory pathway. Also, a Harvard architecture machine has distinct code and data address spaces: instruction address zero is not the same as data address zero. Instruction address zero might identify a twenty-four bit value, while data address zero might indicate an eight bit byte that isn't part of that twenty-four bit value.

Contrast with Modified Harvard architecture
Main article: Modified Harvard architecture
A Modified Harvard architecture machine is very much like a Harvard architecture machine, but it relaxes the strict separation between instruction and code while still letting the CPU concurrently access two (or more) memory busses.
- The most common modification includes separate instruction and data caches backed by a common address space. While the CPU executes from cache, it acts as a pure Harvard machine. When accessing backing memory, it acts like a von Neumann machine (where code can be moved around like data, a powerful technique). This modification is widespread in modern processors such as the ARM architecture and X86 processors. It is sometimes loosely called a Harvard architecture, overlooking the fact that it is actually "modified".
- Another modification provides a pathway between the instruction memory (such as ROM or flash) and the CPU to allow words from the instruction memory to be treated as read-only data. This technique is used in some microcontrollers, including the Atmel AVR. This allows constant data, such as text strings or function tables, to be accessed without first having to be copied into data memory, preserving scarce (and power-hungry) data memory for read/write variables. Special machine language instructions are provided to read data from the instruction memory. (This is distinct from instructions which themselves embed constant data, although for individual constants the two mechanisms can substitute for each other.)

Modern uses of the Harvard architecture
The principal advantage of the pure Harvard architecture - simultaneous access to more than one memory system - has been reduced by modified Harvard processors using modern CPU cache systems. Relatively pure Harvard architecture machines are used mostly in applications where tradeoffs, such as the cost and power savings from omitting caches, outweigh the programming penalties from having distinct code and data address spaces.
- Digital signal processors (DSPs) generally execute small, highly-optimized audio or video processing algorithms. They avoid caches because their behavior must be extremely reproducible. The difficulties of coping with multiple address spaces are of secondary concern to speed of execution. As a result, some DSPs have multiple data memories in distinct address spaces to facilitate SIMD and VLIW processing. Texas Instruments TMS320 C55x processors, as one example, have multiple parallel data busses (two write, three read) and one instruction bus.

- [Microcontrollers](#) are characterized by having small amounts of program ([flash memory](#)) and data ([SRAM](#)) memory, with no cache, and take advantage of the Harvard architecture to speed processing by concurrent instruction and data access. The separate storage means the program and data memories can have different bit depths, for example using 16-bit wide instructions and 8-bit wide data. They also mean that [instruction prefetch](#) can be performed in parallel with other activities. Examples include the [8051](#), the [AVR](#) by [Atmel Corp](#), and the [PIC](#) by [Microchip Technology, Inc.](#).

Even in these cases, it is common to have special instructions to access program memory as data for read-only tables, or for reprogramming.

Flynn's Taxonomy

Flynn's taxonomy is a classification of [computer architectures](#), proposed by [Michael J. Flynn](#) in 1966. The four classifications defined by Flynn are based upon the number of concurrent instruction (or control) and data streams available in the architecture:

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

[Single Instruction, Single Data stream](#) (SISD)
> A sequential computer which exploits no parallelism in either the instruction or data streams. Examples of SISD architecture are the traditional [uniprocessor](#) machines like a [PC](#) (currently manufactured PC's have multiple processors) or old [mainframes](#).

[Single Instruction, Multiple Data streams](#) (SIMD)
> A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized. For example, an [array processor](#) or [GPU](#).

[Multiple Instruction, Single Data stream](#) (MISD)
> Multiple instructions operate on a single data stream. Uncommon architecture which is generally used for fault tolerance. Heterogeneous systems operate on the same data stream and must agree on the result. Examples include the [Space Shuttle](#) flight control computer.

[Multiple Instruction, Multiple Data streams](#) (MIMD)
> Multiple autonomous processors simultaneously executing different instructions on different data. [Distributed systems](#) are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space.

Diagram comparing classifications
Visually, these four architectures are shown below where each "PU" is a processing unit: