INTEGRATED TECHNICAL EDUCATION CLUSTER
AT ALAMEERIA

E-626-A
Real-Time Embedded Systems (RTES)

Lecture #2
Basics of Computer Architecture & Assembly Language

Instructor:
Dr. Ahmad El-Banna

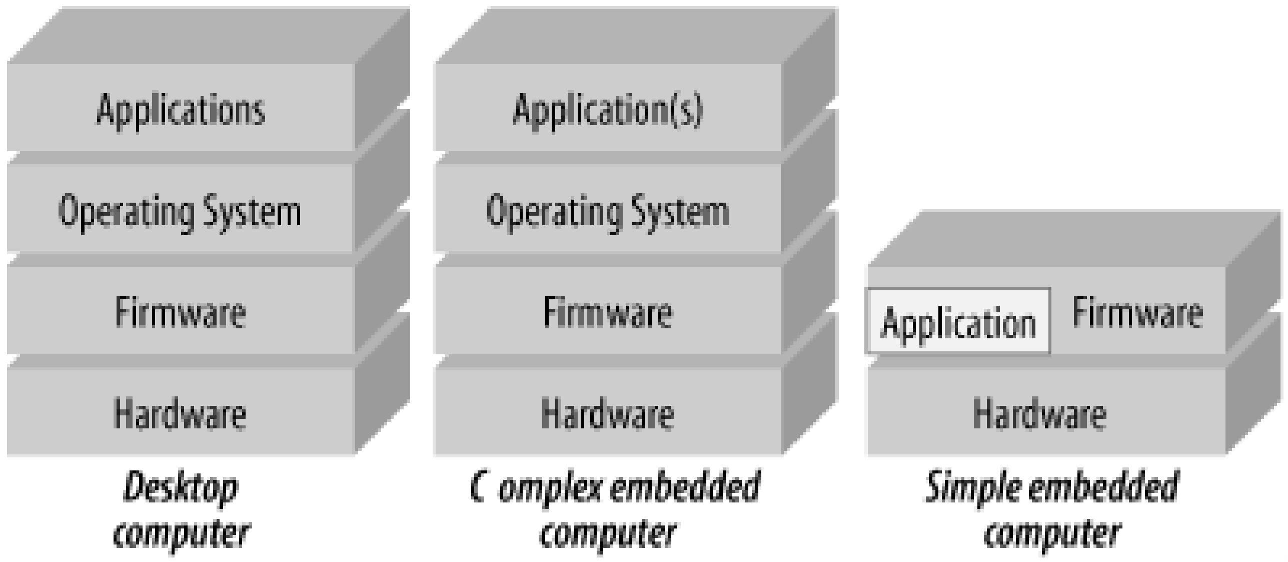
Agenda

- Concepts of Computer Architecture
- Memory
- Embedded Computer Architectures
- Basics of Assembly Language
- Registers & Machine Codes

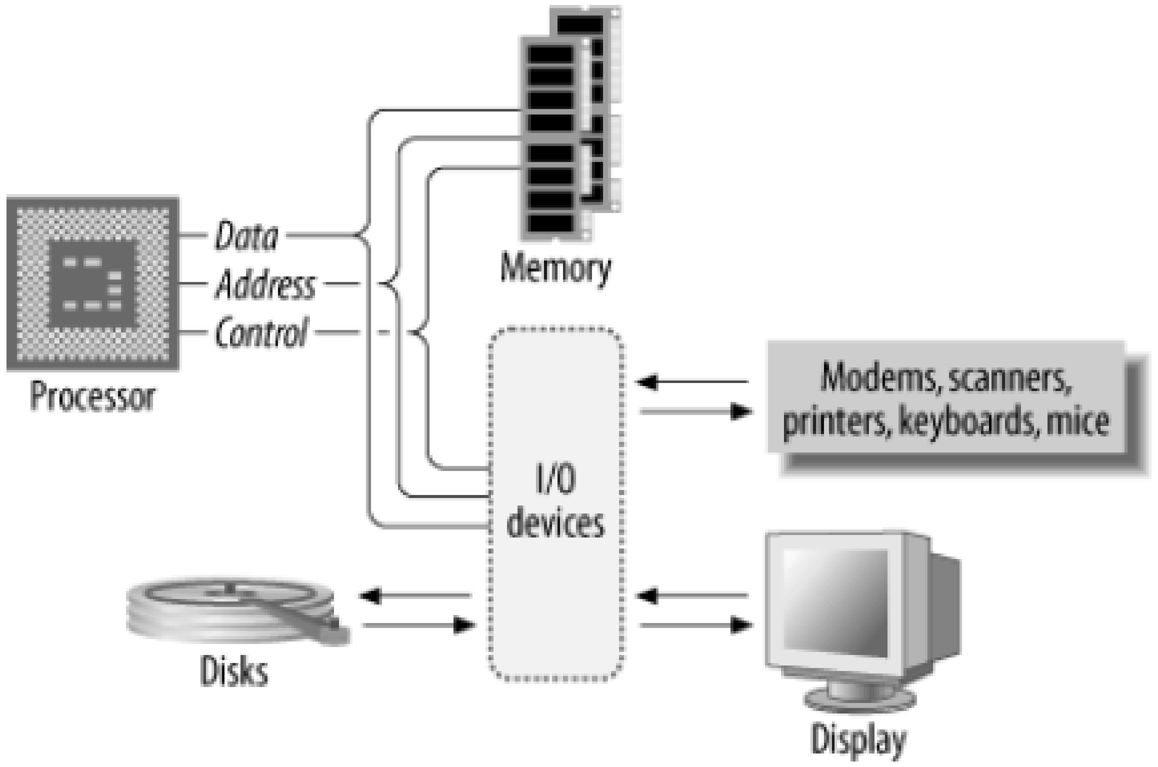
BASICS OF COMPUTER ARCHITECTURE

Software Layers

- The software controls the operation and functionality of the computer.
- There are many "layers" of software in the computer.

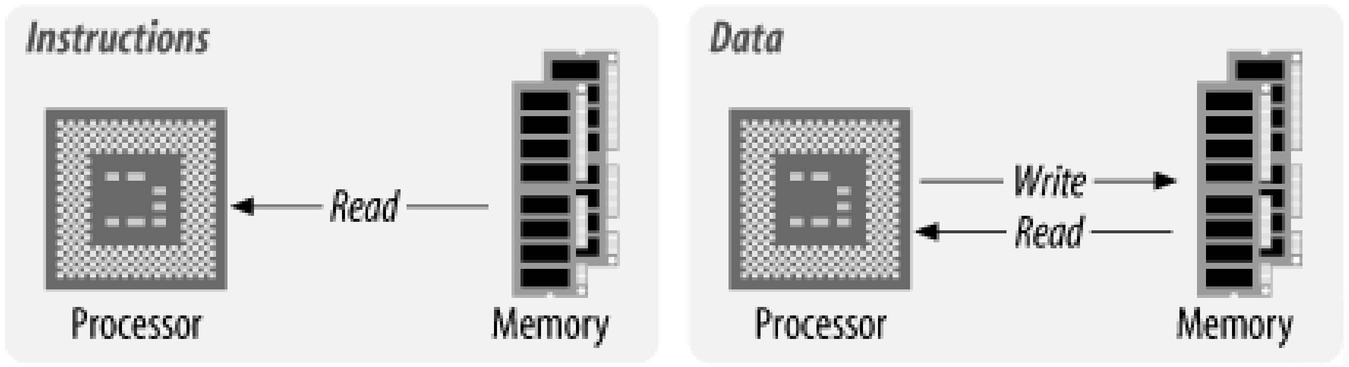


Basic Computer Systems



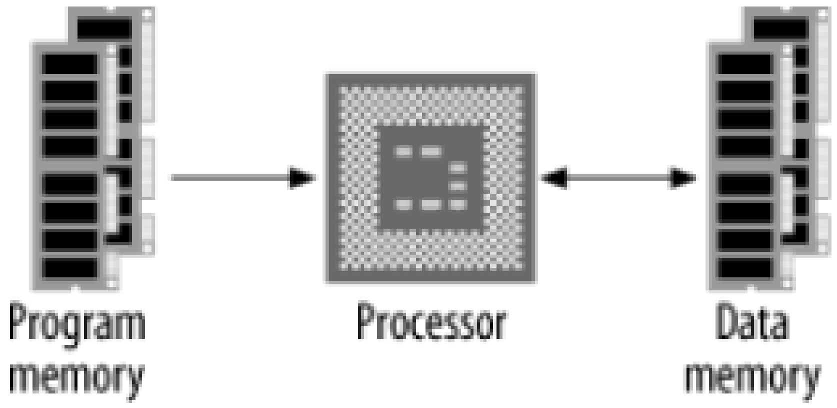
Data Flow (Von Neumann)

- Nearly all modern computers follow this form.
- Von Neumann computers are what can be termed control-flow computers.
- Characteristics:
  - There is no real difference between data and instructions.
  - Data has no inherent meaning.
  - Data and instructions share the same memory.
  - Memory is a linear (one-dimensional) array of storage locations.



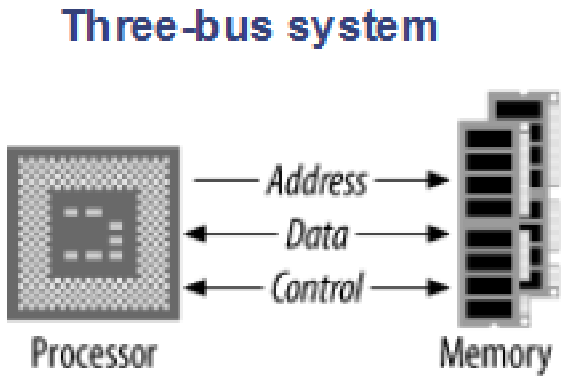
Harvard Architecture

- The instructions and data have different memory spaces with separate address, data, and control buses for each memory space.
- Advantages:
  - instruction and data fetches can occur concurrently, and
  - the size of an instruction is not set by the size of the standard data unit (word).



Buses

- A bus is a physical group of signal lines that have a related function.
- Buses allow for the transfer of electrical signals between different parts of the computer system and thereby transfer information from one device to another.



Interrupts

- Interrupts (also known as traps or exceptions in some processors) are a technique of diverting the processor from the execution of the current program so that it may deal with some event that has occurred.
- Interrupts free the processor from having to continuously check the I/O devices to determine whether they require service. Instead, the processor may continue with other tasks.
- When an interrupt occurs, the usual procedure is for the processor to save its state by pushing its registers and program counter onto the stack.
- processor then loads an interrupt vector into the program counter.
- The interrupt vector is the address at which an interrupt service routine (ISR) lies.

Hardware interrupts

- There are two ways of telling when an I/O device is ready for the next sequence of data to be transferred.
- The first is ***busy waiting*** or ***polling***, where the processor continuously checks the device's status register until the device is ready.
- This wastes the processor's time but is the simplest to implement. For some time-critical applications, polling can reduce the time it takes for the processor to respond to a change of state in a peripheral.
- The other technique of servicing an interrupt is by using ***vectored interrupts***, by which the interrupting device provides the interrupt vector that the processor is to take.
- Vectored interrupts reduce considerably the time it takes the processor to determine the source of the interrupt.
- If an interrupt request can be generated from more than one source, it is therefore necessary to assign priorities (levels) to the different interrupts.

Software interrupts

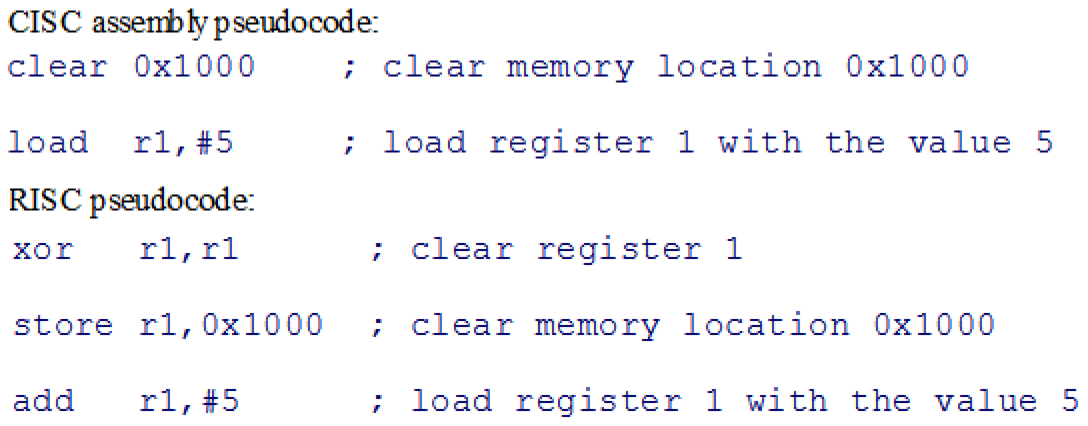
- A software interrupt is generated by an instruction.
- It is the lowest-priority interrupt and is generally used by programs to request a service to be performed by the system software (operating system or firmware).

CISC/RISC

- There are two major approaches to processor architecture:
  - Complex Instruction Set Computer (CISC, pronounced "Sisk") processors and
  - Reduced Instruction Set Computer (RISC) processors.
- Classic CISC processors are the Intel x86, Motorola 68xxx, and National Semiconductor 32xxx processors, and, to a lesser degree, the Intel Pentium.
- Common RISC architectures are the Freescale/IBM PowerPC, the MIPS architecture, Sun's SPARC, the ARM, the Atmel AVR, and the Microchip PIC.

CISC/RISC..

- CISC processors have a single processing unit, external memory, and a relatively small register set and many hundreds of different instructions.
- RISC processors have a number of distinguishing characteristics.
- They have large register sets (in some architectures numbering over 1,000), thereby reducing the number of times the processor must access main memory.
- Often-used variables can be left inside the processor, reducing the number of accesses to (slow) external memory.


CISC assembly pseudocode:

```
clear  0x1000     ; clear memory location 0x1000

load   r1,#5      ; load register 1 with the value 5
```

RISC pseudocode:

```
xor    r1,r1       ; clear register 1

store  r1,0x1000   ; clear memory location 0x1000

add    r1,#5       ; load register 1 with the value 5
```

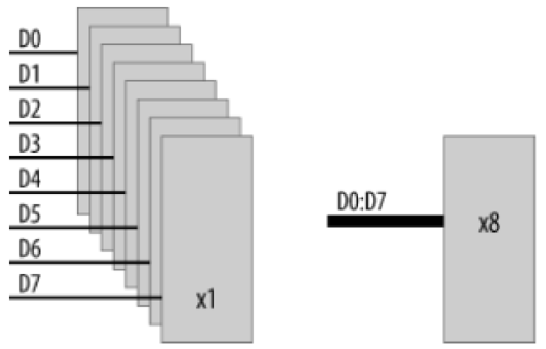
Memory

- Memory is used to hold data and software for the processor.
- There is a variety of memory types, and often a mix is used within a single system.
- Some memory will retain its contents while there is no power, yet will be slow to access.
- Other memory devices will be high-capacity, yet will require additional support circuitry and will be slower to access.
- Still other memory devices will trade capacity for speed, yielding relatively small devices, yet will be capable of keeping up with the fastest of processors.

Memory..

- Memory chips can be organized in two ways, either in word-organized or bit-organized schemes.
- In the word-organized scheme, complete nibbles, bytes, or words are stored within a single component.
- Whereas with bit-organized memory, each bit of a byte or word is allocated to a separate component.
- Memory chips come in different sizes, with the width specified as part of the size description.
- For instance, a DRAM (dynamic RAM) chip might be described as being 4Mx1 (bit-organized), whereas a SRAM (static RAM) may be 512Kx8 (word-organized).

Eight bit-organized 8x1 devices and one word-organized 8x8 device



RAM

- RAM stands for Random Access Memory.
- This is a bit of a misnomer, since most (all) computer memory may be considered "random access."
- RAM is the "working memory" in the computer system.
- RAM is generally volatile, losing its contents when the system loses power.
- RAMs generally fall into two categories: static RAM (also known as SRAM) and dynamic RAM (also known as DRAM).
- SRAMs use pairs of logic gates to hold each bit of data. SRAMs are the fastest form of RAM available, require little external support circuitry, and have relatively low power consumption.
- DRAM uses arrays of what are essentially capacitors to hold individual bits of data. The capacitor arrays will hold their charge only for a short period before it begins to diminish. Therefore, DRAMs need continuous refreshing, every few milliseconds or so.

ROM

- ROM stands for Read-Only Memory. This is also a bit of a misnomer, since many (modern) ROMs can also be written to.
- ROMs are nonvolatile memory, requiring no power to retain their contents.
- They are generally slower than RAM, and considerably slower than fast static RAM.
- The primary purpose of ROM within a system is to hold the code (and sometimes data) that needs to be present at power-up.
- Many microcontrollers contain on-chip ROM, thereby reducing component count and simplifying system design.
- *One-Time Programmable (OTP)* ROMs, as the name implies, can be burned once only.

EPROM

- OTP ROMs are great for shipping in final products, but they are wasteful for debugging, since with each iteration of code change, a new chip must be burned and the old one thrown away.
- As such, OTPs make for a very expensive development option. No sane person uses OTPs for development work.
- A (slightly) better choice for system development and debugging is the Erasable Programmable Read-Only Memory, or EPROM.
- Shining ultraviolet light through a small window on the top of the chip can erase the EPROM, allowing it to be reprogrammed and reused.

EEPROM

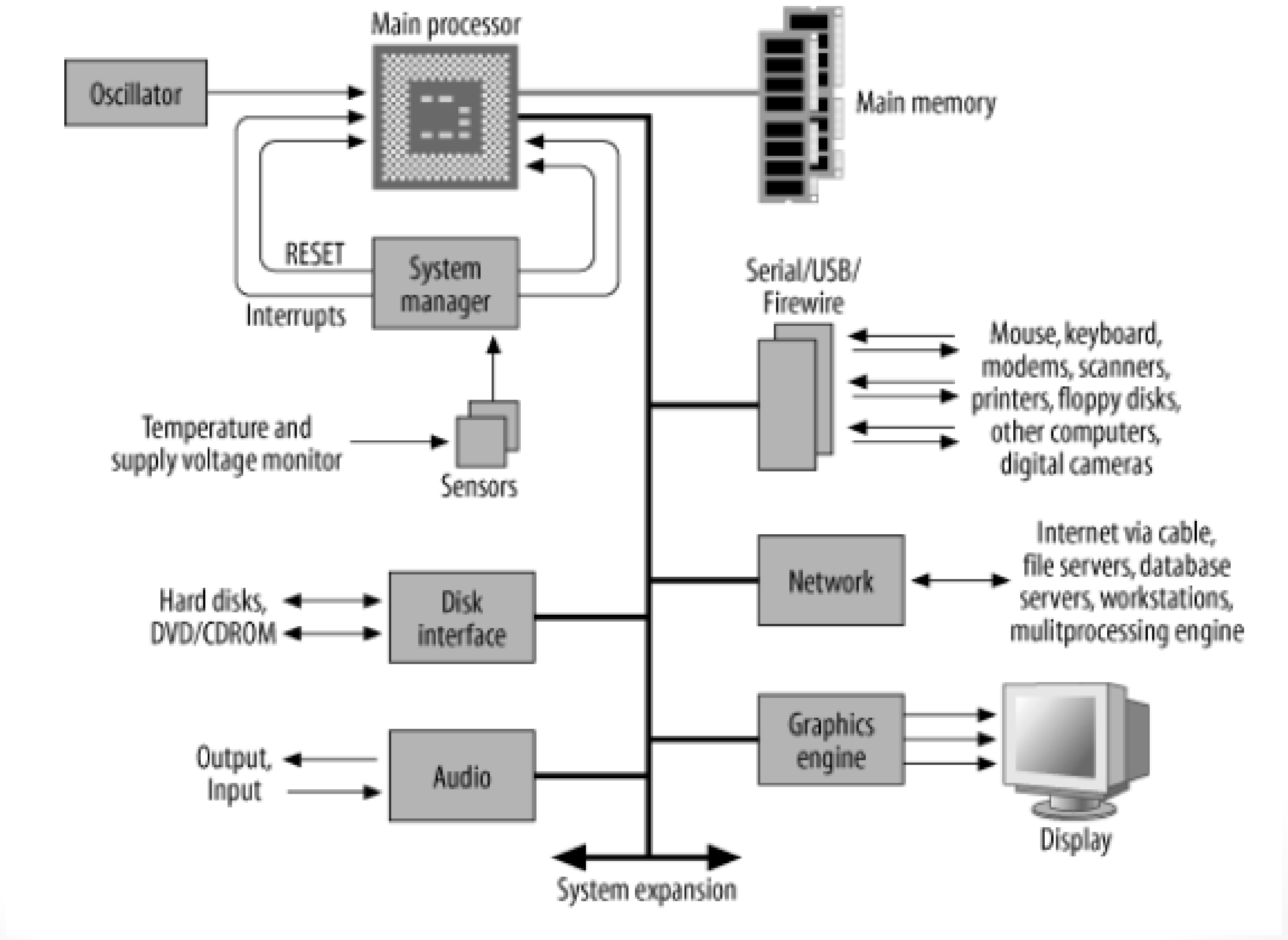
- EEROM is Electrically Erasable Read-Only Memory, also known as EEPROM (Electrically Erasable Programmable Read-Only Memory).
- Very rarely, it is also called Electrically Alterable Read-Only Memory (EAROM). EEROM can be pronounced as either "e-e ROM" or "e-squared ROM," or sometimes just "e-squared" for short.
- EEROMs can be erased and reprogrammed in-circuit. Their capacity is significantly smaller than standard ROM (typically only a few kilobytes), and so they are not suited to holding firmware.
- Instead, they are typically used for holding system parameters and mode information to be retained during power-off.

Flash

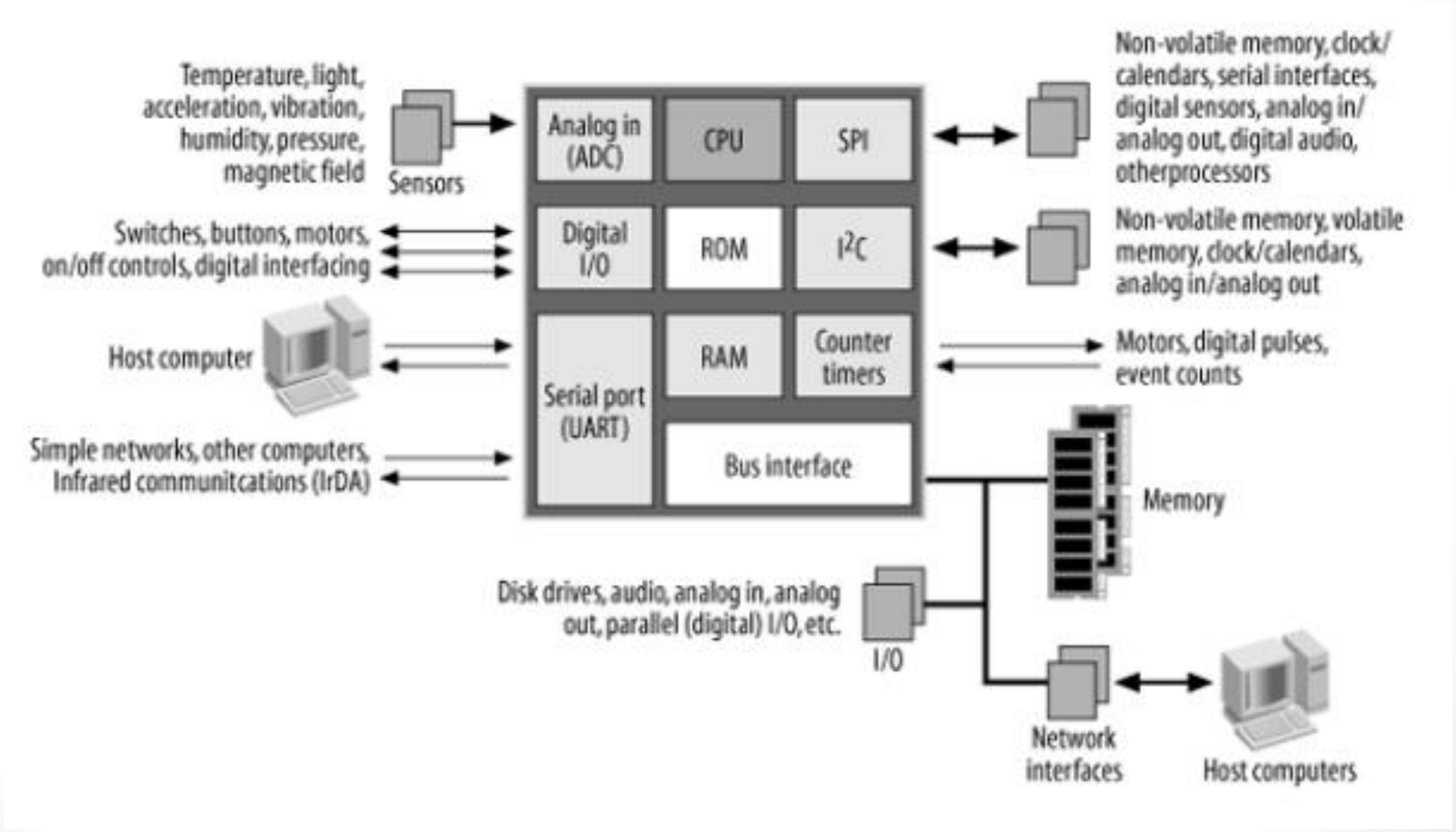
- Flash is the newest ROM technology and is now dominant. Flash memory has the re-programmability of EEROM and the large capacity of standard ROMs.
- Flash chips are sometimes referred to as "flash ROMs" or "flash RAMs." Since they are not like standard ROMs or standard RAMs.
- Flash is normally organized as sectors and has the advantage that individual sectors may be erased and rewritten without affecting the contents of the rest of the device.
- Typically, before a sector can be written, it must be erased. It can't just be written over as with a RAM.

EMBEDDED COMPUTER ARCHITECTURES

Block diagram of a generic computer


Main processor
Oscillator
Main memory
RESET
System manager
Interrupts
Serial/USB/ Firewire
Mouse, keyboard, modems, scanners, printers, floppy disks, other computers, digital cameras
Temperature and supply voltage monitor
Sensors
Internet via cable, file servers, database servers, workstations, mulitprocessing engine
Network
Hard disks, DVD/CDROM
Disk interface
Graphics engine
Output, Input
Audio
Display
System expansion

Block Diagram of an Embedded Computer


Temperature, light, acceleration, vibration, humidity, pressure, magnetic field
Sensors
Switches, buttons, motors, on/off controls, digital interfacing
Host computer
Simple networks, other computers, Infrared communitcations (IrDA)
Analog in (ADC)
CPU
SPI
Digital I/O
ROM
I²C
Serial port (UART)
RAM
Counter timers
Bus interface
Non-volatile memory, clock/calendars, serial interfaces, digital sensors, analog in/analog out, digital audio, otherprocessors
Non-volatile memory, volatile memory, clock/calendars, analog in/analog out
Motors, digital pulses, event counts
Memory
Disk drives, audio, analog in, analog out, parallel (digital) I/O, etc.
I/O
Network interfaces
Host computers

BASICS OF ASSEMBLY LANGUAGE

Intro.

- Assembly-language programming works down at the machine level, and as such, you really get a feel for what the processor is doing and how your computer actually works.
- With assembly language, you work with data in single bytes or words at a time.
- Assembly language has been described as the "nuts and bolts language," because you are writing code directly for the processor.
- For a lot of the software you will write, a high-level language like C will be the language of choice.
- Assembly and machine code, are "hand-written," can be finely tuned to get optimum performance out of the processor and computer hardware.

Registers

bit 0	bit 1	….	bit N

- Registers are the internal (working) storage for the processor. The number of registers varies significantly among processor architectures.
- Typically, the processor will have one or more accumulators. These are registers that may have arithmetic operations performed on them.
- In some architectures, all the registers function as accumulators, whereas in others, some registers are dedicated for storage only and have limited functionality.
- Some processors have index registers that can function as pointers into the memory space.
- All processors have a program counter (also known as an instruction pointer) that tracks the location in memory of the next instruction to be fetched and executed.
- Some processors also have one or more control registers consisting of configuration bits that affect processor operation and the operating modes of various internal subsystems.

Machine Code

- Everything that a processor deals with is expressed as numbers in memory. That applies to data, and to software as well.
- This is known as machine code, and each numeric instruction is called an opcode.

```
Machine code    Assembly     Comment
86 41           LDAA #$41    ; anything after a semi-colon
B7 01 00        STAA $0100   ; is a comment
BD 02 00        JSR $0200    ;
```

Comparison of some different assembly languages

Processor	Instruction
Motorola 6800/68HC11	`LDAA #$41`
Intel x86	`moval, 41h`
Motorola 680x0	`move.b #$41,D0`
PIC16xx	`movlw 0x41`
Motorola 56000	`move #$0041,A`
Intel 80960	`lda 0x41,r4`

Addressing modes

- The different ways in which an instruction can reference a register or memory location are known as the addressing modes of the processor.
- The types of addressing modes available within different architectures vary.

Types:

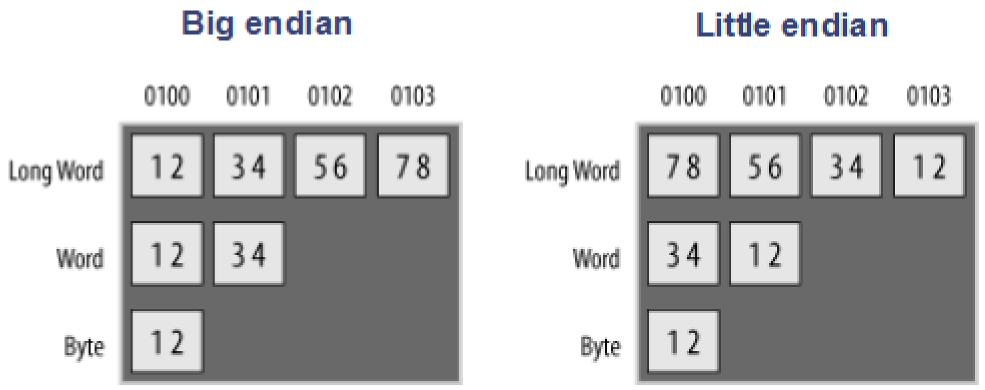
- Inherent
  - The instruction deals purely with registers. CLRB clears the B accumulator, for example.
- Immediate/Literal
  - The instruction has a literal number as an operand.
- Direct
  - The instruction accesses a memory location, specified by a short address.

Addressing modes..

- Extended
  - The instruction accesses a memory location, specified by the full address.
- Indexed
  - The instruction uses the contents of a register as a pointer into memory.
- Relative
  - An offset is specified as part of the addressing.

Big-Endian and Little-Endian Addressing

- Microprocessors are either big endian or little endian in their architecture.
- This refers to the way in which the processor stores data (16 bits or greater) to memory.
- A big-endian processor stores the most significant byte at the least significant address.
- A little-endian processor stores the most significant byte at the most significant address.



Coding in Assembly

- let's say we want a 68HC11 processor to add three numbers together (0x10, 0x1F, and 0x0C) and store the result at address 0x0027.
- This problem is easily broken down into four steps.
  - We take the first number (step one),
  - add the second number (step two),
  - add the third number (step three), and
  - store the result (step four).
- We want to do some arithmetic, so we choose an accumulator to hold our numbers, since accumulators are registers designed for this type of operation.

Coding in Assembly..

label, s used by the assembler during compilation to identify a given address location. It has no direct meaning in the machine code

```
add_numbers

   LDAA   #$10   ; load first number in the A accumulator

   ADDA   #$1F   ; add the second number to A

   ADDA   #$0C   ; add the third number to A

   STAA   $0027  ; store the contents of A to address 0x0027

   RTS           ; return to main program
```

To call our subroutine from our main program, we simply use a jump-to-subroutine instruction

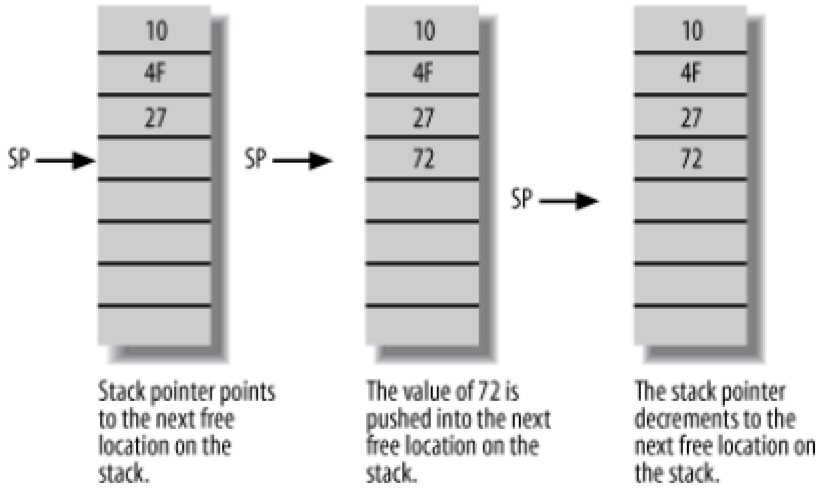
```
  JSR    add_numbers
```

Loops

```
        LDAA #$0F   ; load the count into the A accumulator
again               ;
        DECA        ; count down (decrement the A accumulator)
        BNE again   ; repeat until we have counted down to zero
```

Stacks

- Many processors implement one or more stacks, which serve as temporary storage in external memory.
- The processor can push a value from a register on the stack to preserve it for later use.
- The processor retrieves this value by popping from the stack back into a register. In some processor architectures, popping is also known as pulling.



SP
10
4F
27
SP
10
4F
27
72
SP
10
4F
27
72
Stack pointer points to the next free location on the stack.
The value of 72 is pushed into the next free location on the stack.
The stack pointer decrements to the next free location on the stack.

```
fred

      PSHB  ; save B accumulator onto the stack

            ;

            ; (subroutine code here)

            ;

      PULB  ; restore B accumulator from the stack

      RTS   ;
```

- For more details, refer to:
  - Chapter 1,2 John Catsoulis, **Designing Embedded Hardware**, 2005.
- The lecture is available online at:
  - http://bu.edu.eg/staff/ahmad.elbanna-courses/12134
- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg