CPE 323
MODULE 01
COMPUTER SYSTEMS: A BRIEF REVIEW

Aleksandar Milenković

Email: milenka@uah.edu

Web: http://www.ece.uah.edu/~milenka

Overview

This module reviews main components of a computer system (processor, memory, input/output peripherals, and interconnect) and principles of stored-program execution.

Objectives

Upon completion of this module learners will be able to:

- *Describe 4 main components of a computer system*
- *Describe main steps of an instruction execution*

Contents

1 Introduction

Computers or computing devices have become an indispensable parts of our lives. They come in many forms: starting from wearable electronics (e.g., fitness trackers), smart watches, smartphones, personal computers, to warehouse-scale computers running the majority of cloud-based services such as email, file sharing, photo sharing, social networks, shopping web sites, and others. Computers are also present in many other devices we interact with every day – in cars, smart speakers, TVs, and others. Broadly speaking, computers can be classified into the following major categories: embedded computers, personal devices, personal computers, and servers/warehouse scale computers. The embedded computers refer to computing devices that are a part of other devices (e.g., car, microwave oven, refrigerator, network router, to name just a few), and are not necessarily perceived as computing devices. Embedded computers are highly specialized and designed to perform a single task (or a subset of tasks) throughout their lifetime.

2 Four Components of a Computer System

Regardless of functionality, size, or cost, any computer system includes four major components: the processor (a.k.a., central processing unit or CPU), memory, input/output peripherals, and an interconnect. Figure 1 shows a block diagram of a computer system that uses a bus (collection of wires) to interconnect major components. The CPU is the brain of a computer system responsible for executing machine instructions. The main memory stores programs and data. Input/Output (I/O) peripheral devices enable interfacing to the outside world (e.g., switches, LEDs – light emitting diodes, LCDs – liquid crystal displays, and others). The interconnect fabric connects all these components together.

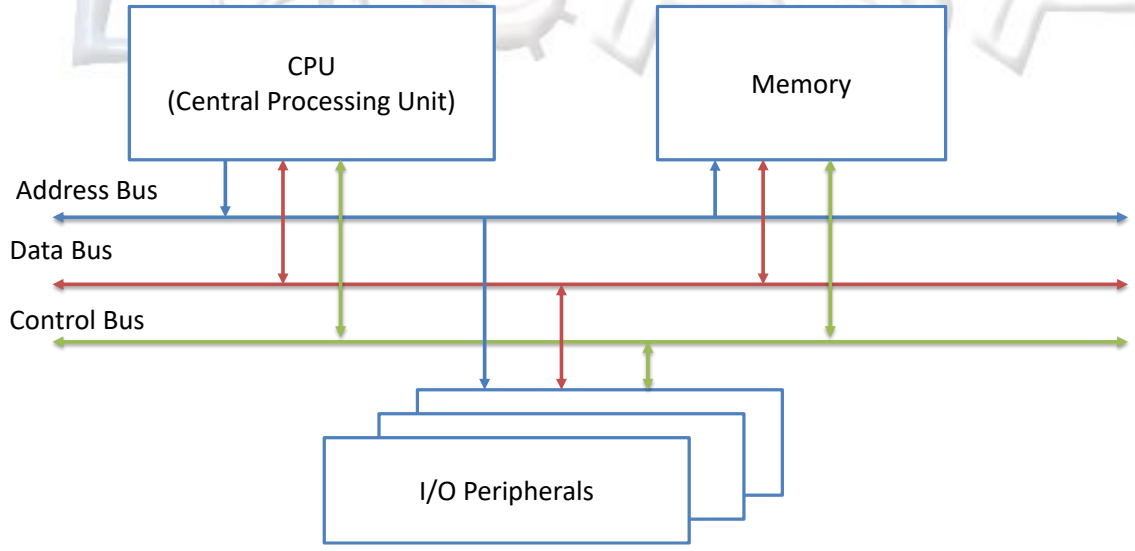

Figure 1. Block diagram of a computer system with 4 major components.

Things to remember 2-1. Four components of a computer system.

The four main components of any computer system are: CPU, memory, I/O peripherals, and interconnect fabric.

3 CPU

The Central Processing Unit (CPU) is the brain of a computer system, responsible for executing machine instructions. It can be divided into two major sub-modules: the datapath and the control unit. The datapath includes registers, functional units (e.g., integer ALU – arithmetic logic unit, floating-point adders, floating-point multiplier, etc.), and internal buses connecting these logic blocks. Thus, the datapath is where instructions are executed and temporary data are stored and processed. The control unit includes a sequencer and combinational logic responsible for generating control signals needed to carry out machine instructions in the datapath. You can think about resources of datapath as individual members of an orchestra and about the control unit as a conductor of the orchestra.

To execute a machine instruction the CPU goes through a sequence of steps as follows:

1. Fetch Instruction
2. Decode Instruction
3. Fetch Operands
4. Execute Instruction
5. Store Result(s)
6. Process Exceptions or Interrupts.

Fetch Instruction. Machine instructions are stored in the main memory. Thus, the first step of the instruction execution is fetching the next instruction from memory. For this purpose, the CPU typically maintains a special register called Program Counter (PC). This register always points to the next instruction to be executed, i.e., its content is the address of memory location that contains the next instruction. Fetching an instruction involves placing the content of the PC on the address bus, setting a control signal to trigger a read from memory (RD line on the control bus is asserted), and then waiting for memory to respond. The memory responds by placing the instruction on the data bus. The CPU takes the content from the data bus (the fetched instruction) and stores it into a dedicated register, let us call it, the Instruction Register (IR). The PC is then incremented to point to the next instruction in the program.

Using so-called *Register-Transfer-Level (RTL)* notation that captures movement of data and instructions inside the CPU, we can describe the instruction fetch phase as follows:

1. IR ← M[PC]
2. PC ← PC + Size_Of_Instruction

We can describe the first step as follows: "IR gets the next instruction fetched from memory from the address currently contained in register PC." The second step is updating PC to point to the next instruction in sequence. Here we provided a simplified view of the instruction fetching that involves one read operation from memory. Generally, an instruction fetch can involve multiple reads from memory in case that the instruction size exceeds the size of data that can be read from memory in one read operation.

Decode Instruction. Once fetched, the instruction is decoded. The instruction is encoded in binary. The binary contains multiple fields that tell us the following: (a) what the instruction is (e.g., ADD, SUB, MOV, …), (b) where the operands are (registers or memory), and (c) if the operands are in memory, what addresses they are placed at. The CPU relies on decoders that analyze individual fields of the instruction to figure out answers to these questions.

Fetch Operand(s). If the operands are in the main memory, the processor issues reads from the memory to fetch operands. The CPU determines the address of a source operand (Effective Address of Source – EAS) and then reads the operand from memory: the EAS is placed on the address bus and a memory read control signal is asserted (set to active value). Once the memory responds by placing the operand on the data bus, it is grabbed by the CPU and stored into a temporary register, let us call it, S1.

Using the RTL notation, we can describe the operand fetch as follows:

1. S1 ← M[EAS]

We can describe this step as follows: "S1 gets the value of the operand from memory from the address of the source operand, EAS." If the instruction specifies multiple operands in memory, multiple "trips" to memory are made to fetch all source operands.

Execute Instruction. This stage involves actual instruction execution. For arithmetic-logic instructions, the specified operation is performed using corresponding functional units. For transfer instructions that read from or write into the main memory, the effective address of the operand(s) is calculated. For control-flow instructions, a branch condition is evaluated, and if the condition is true, a new target address is computed and stored into the PC (branch is taken). Thus, this phase is instruction-specific. In the case of arithmetic-logic instructions, the result of the instruction is ready at the end of this stage.

Store Result. If the result of the instruction is to be stored in a general-purpose register, it is done so at the end of instruction execution stage. If the result is to be stored in the main memory, the CPU carries out a memory write operation. The address of the destination operand is placed on the address bus, the result is placed on the data bus, and a control signal to initiate write into memory (WR) is asserted. Once the memory updates its content, this step is over. The result of an instruction may also include some side-effects. E.g., in CPUs that support flags that indicate correct or incorrect results, these flags are set accordingly. The flags are typically held in a dedicated register called Status Register – SR.

Processing Exceptions (Interrupts). The last stage of an instruction execution is to check for pending exceptions (interrupts). Typically there are no exceptional situations (e.g., irregular conditions, incorrect results, etc.) and the CPU can go back and repeat the sequence of steps

for the next instruction. However, if there are pending interrupts, the processor needs to handle them. Handling interrupts involves a sequence of steps carried out in hardware. First, the CPU saves the context of the current program, so it can be resumed. Next, the CPU determines what event needs to be serviced first in case multiple events are pending. After that, the starting address of the corresponding handler or Interrupt Service Routine (ISR) is fetched from a so-called Interrupt Vector table. And finally, the starting address of the corresponding ISR is moved into PC. Thus, at the end of this stage the flow of the program execution is changed and the CPU continues execution of instructions from the corresponding ISR. Once the ISR is executed, the program resumes its execution from the following instruction. Think about ISRs as special subroutines designed to handle exception events. More about this will be discussed later in the course.

`Things to remember 3-1. Datapath and control unit.`

Any CPU can be divided into two parts: (a) datapath that includes registers and functional units where computation takes place, and (b) control unit that generates control signals needed to carry out instructions.

`Things to remember 3-2. Phases of instruction execution.`

The main phases on an instruction execution are as follows: (a) fetch instruction; (b) decode instruction; (c) fetch operands; (d) execute instruction; (e) store results; and (f) process exceptions/interrupts.

4 Memory

The main memory stores programs (machine instructions) and data. There are many types of memories. You have probably heard of two terms: ROM – Read Only Memory (it is a non-volatile memory that you can only read from) and RAM – Random Access Memory (it is a volatile memory that you can read from and write into). In both cases, we think about a memory as a collection of storage units, where each storage unit has its own address. The following is a list of frequently used storage units:

- Nibble – 4 bits
- Byte – 8 bits
- Word – 16 bits
- Long Word – 32 bits

Please keep in mind that Byte is always 8-bit long, whereas Words or Long Words are processor specific terms. The definitions for Words and Long Words in this text correspond to the MSP430 processor that we are using in this course. The most common addressable unit – the smallest unit that has an address – is a byte. There are many historical and practical reasons for that. Thus, we see memory as a collection or an array of byte-sized locations.

Memory sizes are usually expressed in Kilobytes, Megabytes, etc. Below are common definitions that use a power of 2:

KiB – 2^{10} bytes (1 kibibyte in International Electrotechnical Commission, 1 KB JEDEC)

MiB – 2^{20} bytes (1 mebibyte IEC, 1 MB JEDEC)

GiB – 2^{30} bytes (1 gibibyte IEC, 1 GB JEDEC)

TiB – 2^{40} bytes (1 tebibyte IEC)

PiB – 2^{50} bytes (1 pebibyte IEC)

Please note that sometimes people will use KB to refer to 1,000 bytes, not 1,024 or 2^{10} bytes. We will almost exclusively refer to memory sizes in power of 2.

Check yourself 4-1

How many bytes do you have in a 4 KiB memory?

Figure 2 gives a logical view of a byte-addressable 64 KiB memory. The memory contains 2^{16} or 65,536 bytes. To distinguish between 2^{16} locations, we need 16 bits to represent the address of a memory location (also, the address bus will need 16 address lines A_{15} … A_0). The address range is from 0 to 65,535 decimally. We usually express addresses in hexadecimal notation for brevity. The first location has the address 0000_0000_0000_0000 b (binary) or 0x0000 (or 0000 h) in hexadecimal notation and the last location has the address 1111_1111_1111_1111 b or 0xFFFF. The bit at position 0 in a byte is referred to as the Least Significant Bit or LSB and bit 7 as the Most Significant Bit or MSB.

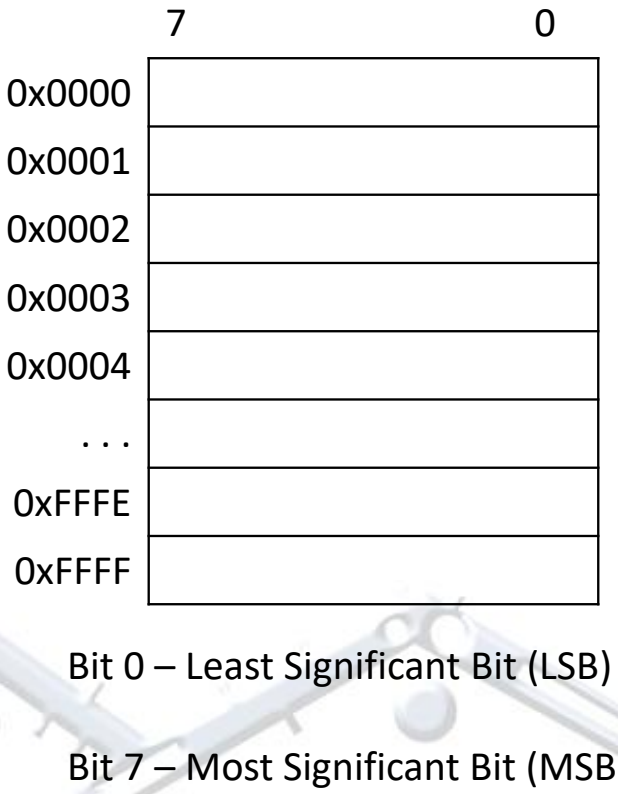

Figure 2. Logical view of a memory with 65,536 (216) bytes.

Figure 3 shows an alternative logical view of a byte-addressable 64 KiB memory. Here, two bytes are grouped to form a single 16-bit word. This view is helpful when we have a CPU that can read and write words in a single memory operation. Word addresses are even 0x0000, 0x0002, etc., pointing to the address of the first byte in a word. This view of memory is word-aligned, little-endian – the byte at the address 0x0000 corresponds to lower 8 bits of a word and byte at the address 0x0001 corresponds to the upper byte in a word.

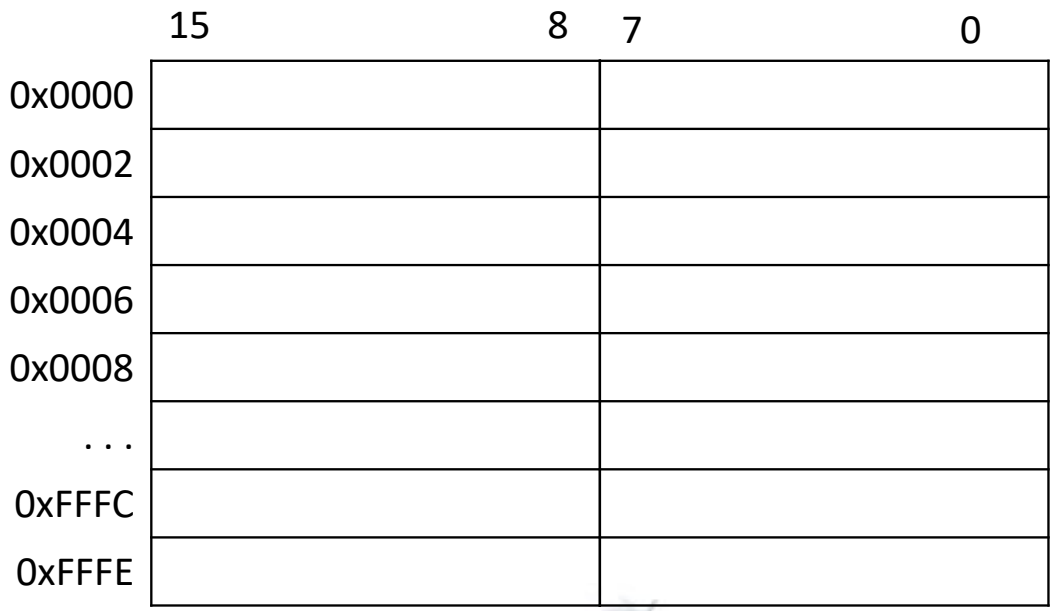

Figure 3. Logical view of a memory with 65,536 (2^{16}) bytes organized in words.

Check yourself 4-2

You are given memory modules with 16 KiB (16,384 x 8 bits). Using these modules you are asked to design a memory organized in 32 kilo 16-bit words. How many modules do you need? What other logic components do you need to properly connect these modules?

`Things to remember 4-1. Main memory.`

Main memory is organized as a collection of addressable locations. A byte or 8 bits is typically smallest addressable unit in modern processors.

5 I/O Peripherals

Input/Output peripherals are components of a computer system that allow us to interface with the outside world. Students often list keyboards and monitors as I/O peripherals, but to be precise, those are external devices that we interface through a standard set I/O peripherals, such as:

- Parallel Ports - used to interface switches, LEDs, 7-segment displays;
- Timers - used to measure time;
- Serial Communication Interfaces – used to carry out communication protocols, such as UART, SPI, I^2C, CAN bus, Ethernet, and others;

- Analog-to-Digital Converters (ADC) - used to convert analog signals into their digital counterparts; and
- Digital-to-Analog Converters (DAC) – used to convert digital values into analog signals.

The design of each I/O peripheral is function-specific, but we typically see all peripherals through their control, status, and data registers that are mapped in the processor's address space. To configure a peripheral to perform a certain task, we usually initialize its control register by setting certain bits to appropriate values. To monitor the status of a peripheral we can read the current value of its status register that reflects the current state of the peripheral (e.g., whether the task has been completed or it is still in progress). And finally, to exchange data between the CPU and peripheral, we read from and write into its data register(s). From the software perspective, there are three principal ways how we can interface I/O peripherals:

- Using Polling;
- Using the Interrupt Mechanism;
- Using Direct Memory Controllers (DMAs).

`Things to remember 5-1. I/O peripherals.`

IO peripherals are components of a computer system that allow us to interface with the outside world, such as parallel ports, timers, serial communication interfaces, analog-to-digital converters, digital-to-analog converters, and others. Our programs interact with peripherals through their control, status, and data registers that are mapped in the processor's address space.

6 Bus

The simplest type of an interconnect is a bus. The bus consists of a number of wires that carry signals between components. At any point of time we can have only one component to drive signals on the bus. When no component is driving the wire, we assume it has a high-impedance state ('Z') – neither 0 nor 1. The bus consists of address lines, data lines, and control lines. The address lines carry address bits. The data lines carry data bits. The control signals allow the CPU to issue a read from memory or an I/O peripheral (RD) or write to memory or an I/O peripheral (WR).

Figure 4 illustrates a system bus with 16 address lines (A_{15} to A_0), 16 data lines (D_{15} to D_0), and control bus (typically will include RD and WR control signals, but can have more).

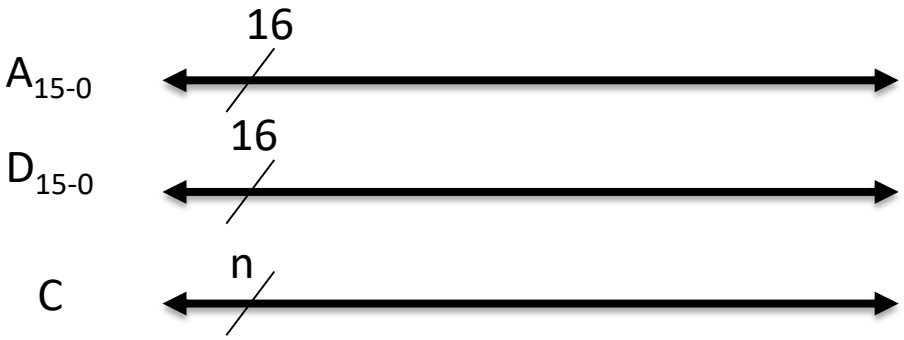

Figure 4. System Bus with 16 address lines, 16 data lines, and control lines.

`Things to remember 6-1. Bus.`

A bus is the simplest form of an interconnect. It consists of a number of wires that carry signals between components, specifically address lines, data lines, and control lines.

7 Exercises

Q #1.

List the 4 main components of any computer system and their functions.

Q #2.

What are the main phases on an instruction execution?

Q #3.

Assume you have a processor that includes a machine instruction that sums up two operands in memory at locations A and B as follows:

ADD A, B; M[B] <= M[A] + M[B]

Using RTL notation, describe steps taken in the Instruction Fetch stage and the Operand Fetch stage (assume operands A and B are fetched into temporary registers S1 and S2 inside the processor), the Execute Stage (assume the result is temporarily stored in register S3), and the Store Result stage? Assume the instruction is 6 bytes long.

Q #4.

Assume a processor with a 16 MiB address space. A byte is the smallest addressable unit and we can read 16-bits of data from memory in one read operation (providing they are aligned to even addresses).

How many address lines does this processor have on its bus?

How many data lines does this processor have on its bus?

What is the address range of the address space (provide the address of the first and the last byte in the address space)?