Computer Systems Organization

Druhan Rajiv Shah

Spring - 2023

Contents

Course Structure

The first half (upto, but not including Assembly code) will be taught by Suryajith Chillara, while the second half will be taught by Girish Verma.

Textbooks

- Computer Systems: A Programmer's Perspective - Bryant & O'Hallon
- Computer Systems Organization - Patterson & Henessey

Topics The ranges in the parentheses are sections in the first textbook.

- Computer Arithmetic (2.1 - 2.3)
- Assembly language (3.4 - 3.7)
- Processor architecture and design (4.1 - 4.5)
- Memory hierarchy (6.1 - 6.4)
- System calls: Intro to process control (8.2 - 8.4)

Grading

Assignments (Programming-based) - 20%

Deep Quizzes - 20%

Midsem Lab Exam - 15%

Endsem - 25%

Endsem Lab Exam - 20%

1 Introduction

How is this different? In previous courses we studied how computers worked in a sort of abstract way, without actually worrying about what goes on under the table. This 'under the table' stuff is exactly what we will be discussing in this course.

2 Computer System structure

Overall structure In a typical system, there are several functional sections, like the CPU, which contains the Program counter, register file and ALU; I/O Bridge, controllers, adapters and Main memory; each of which is interconnected through buses. These buses transfer fixed-side chunks of bytes (called words).

Now, all of the information is contained in bits (which, recall, are just states of low vs high voltage). The bandwidth of these buses is what determines the word length of the system. This would be more familiar as whether a system has a 32-bit or 64-bit architecture.

Note, the difference between a controller and an adapter is whether it is directly connected to the motherboard or not. So, mouse and keyboard ports are termed controllers, while a monitor or a display would need an adapter. Earlier, RAMs used to be connected using adapters as opposed to controllers.

CPU A CPU contains a PC, a register file and an ALU. Program counters are necessary for parallel processing of instructions. An ALU is self-explanatory and does most of the calculation here. Note that this is what CPUs looked like a good while ago. Nowadays it's a lot more complicated.

What CPUs do essentially, is they Load data from the RAM into the register; Store a copy form the register to the main memory; Operate by copying contents of a register to the ALU and load the output back into the register; and Jump by extracting a word from instrutions and overwrite into the PC. (Load, Store, Operate, Jump is gonna be a recurring theme in formalizations)

Dynamic RAM The main memory is organized into a linear array of bytes, each with its unique address (recall CPro). The Program Counter back in the CPU points to the curent instruction in the main memory.

Register file It's a small storage device that contains a collection of word-sized registrs, each with unique names.

Running an executable First, the system accepts the input of the command from the USB controllers, dumps it into the main memory and then processes it, loads the instructions from the disk into main memory, and finally runs each of the instructions in sequence.

Processor-Memory gap Moore's law says that the processing power doubles every constant amount of time (paraphrase :P).

RAM types The damage this causes can be mitigated by involving Cache memories and Registers, both of which are implemented using Static RAMs, which use flip-flops as opposed to capacitors and transistors in DRAM. DRAMs have a constant recharging of the capacitors, the rate of which is called the refresh rate of the DRAM. However, SRAMs use flip-flops which are significantly more stable. This makes SRAMs more complicated to design while making them significantly faster. This is also why SRAMs are used inside he CPU, while the main memory uses DRAM.

Note that both SRAMs and DRAMs are both volatile, meaning that the moment the system is shut down, the information stored in the RAM is destroyed.

More RAM types! There is a hierarchy of 7 types of RAMs, which are labelled from L0 to L6.

The faster, more efficient types of RAM also happen to be way more expensive than their cheaper counterparts and use more energy, which is why we need to optimize where we use each type of RAM and how much it would cost.

OS Cameo Operating Systems behave as a sort of middleman between the Application programs and the hardware of the system. They have two main purposes:

- To protect the hardware from misuse by runaway programs
- To provide a simple and uniform mechanism for programs to manipulate complicated and varied hardware devices (this is done through abstractions that are handled by device drivers)

The abstraction reaches comical levels when considering the implementation of a mouse and a keyboard as simply reading from a file associated with each (on Linux at least)! This is not too slow because these files are specially implemented to be a part of the main memory instead of disk, so reading and writing these files is not too slow.

The abstraction for running a program is a Process. This is convenient to handle concurrency of processes.

3 Computer Arithmetic

Numerical representation We usually use bit representations of numbers, which is convenient because storing information in bits is a lot more efficient because of the 'ON/HIGH' versus 'OFF/LOW' states. This was especially useful back in the days of punch card programming.

However, the representation is rather verbose. (In fact, base b is $\mathcal{O}\left(\frac{\log b}{\log a}\right)$ times as verbose as base a representation). On top of that, we are used to using base 10 due to the languages we speak and the writing systems we use. Conversion from base 10 to base 2 and vice versa is rather cumbersome.

To overcome this, we use the hexadecimal system instead, which has the added consequence of making conversion from binary to hexadecimal much more convenient. In fact, for a compiler, you can save some computation power and use a lookup table instead of calculating the conversion each time at the cost of a tiny amount of space.

Words Word size is a measure of the size of integer and pointer data.

Note that this means 32-bit computers have a hard limit on the size of virtual memory because there are only $2^{32} \approx 4 \times 10^9$ possible address indices.

Enter Jonathan Swift? The Lilliputian argument of Big-Endians vs Little-Endians is still strong in bit representations as well, because whether the first bit should be for the most or least significant bit respectively. *e.g.* Sun, IBM machines prefer the Big-Endian approach, while Intel machines prefer the Little-Endian approach.

The Big-Endian vs Little-Endian approach also applies to memory location indexing. Here the Little-Endian approach would require the least significant digits in the hexadecimal form of an address or integer to be stored and processed earlier in the memory.

Now this is dangerous because multiple machines which use different approaches in the same system could possibly lead to disaster if not spotted.

ASCII what you did there Strings can now be represented as strings of integers, each of which corresponds to each character. The generally accepted system of encoding the characters as integers is called the American Standard Code for Information Interchange (ASCII). However, there is also Unicode which is gaining more popularity over time.

3.1 Boolean operations

Basic operators The standard NOT, AND, OR and XOR are also used nowadays for bit operations in computers, even though the notation used is different from that used in Propositional Logic (because of the character set allowed by ASCII). These operations can be applied on bit vectors bit-by-bit to make 'bitwise' operators.

Additionally, $\oplus, \wedge, \neg$ can be thought of as $+, \times$, negation in the Boolean Ring.

As a bonus, bit vectors can be used to represent subsets (obviously), so the operators behave analogously to set operations in the Power Set Ring.

More representation Pure binary representation can only store nonnegative integers. If we were to store negative numbers as well, we would need alternative methods. One possibility is the signed n-bit notation, which assigns a sign bit at the cost of absolute range.

Another possibility, which is more often used, is the 2's complement encoding, which makes operations a lot simpler. Recall DSM.

$$\text{Signed } 2'\text{s complement} : \mathrm{B2T}_w\left(\vec{x}\right) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

$$\text{Unsigned } 2'\text{s complement} : \mathrm{B2U}_w\left(\vec{x}\right) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

Clearly, both of these functions are bijective over w-bit numbers.