

Data Manipulation

- Now that we have seen how a computer *represents* information, we will look at how it *manipulates* it
- By “manipulate,” we mean everything that can a computer can do with bits
 - ◆ Storage, moving, copying
 - ◆ Calculations, computations, modifications
 - ◆ Transfer across devices, or translation of bits to/from the “real world” (it’s OK to draw analogies with *The Matrix* here)

Computer Architecture

- Let’s get to know the machine itself — the subspecialty of computer science that deals with the components of a computer and how they interact is called *computer architecture*
- Despite the wide, wide variety of computing devices available today, you may be surprised to learn that deep down, they are ultimately the same machine — something called the *von Neumann architecture*
 - ◆ Though named after Austro-Hungarian mathematician John von Neumann (1903–1957), credit for this architecture should actually be shared among many other pioneers of the era, such as J. Presper Eckert and John William Mauchly

The CPU

- The core of a “von Neumann machine” is the *central processing unit* or CPU, which, these days, is typically delivered as inch-size (or less!) squares of silicon, gallium arsenide, or other semiconductor material
- The CPU itself has three main subcomponents:
 - ◆ Control unit
 - ◆ Arithmetic/logic unit (ALU)
 - ◆ Registers (general- or special-purpose)
- Figure 2.1 in the textbook illustrates this structure, as well as the CPU’s connection to main memory through a collection of wires called a *bus*

- A CPU’s “life” consists entirely of performing the following sequence *ad infinitum*, or else until someone cuts the power:
 1. Read an instruction from main memory (*fetch*)
 2. Determine what the instruction is specifying (*decode*)
 3. Perform the instruction (*execute*)
- With few exceptions, the *instruction set* that a CPU “understands” falls into a handful of primary categories
 - ◆ Transfer data from main memory into a register (*load*)
 - ◆ Transfer data from a register to main memory (*store*)
 - ◆ Perform a calculation on one or more registers
 - ◆ Direct the CPU to the next instruction

Notes on This *Stored-Program Concept*

- You have just been given what amounts to the “prime directive” of modern computers — just as all data boils down to bit patterns, then all computer activity boils down to this “fetch-and-execute” cycle
- Observe that there is *only one* main memory: both the instructions *and* the data manipulated by those instructions *come from the same place*
- This is the *stored-program concept*, and is the fundamental innovation behind the von Neumann (or Eckert, or Mauchly) architecture or machine

Tie-Ins to Jargon that You Might Have Heard

- Most of you probably recognize terms such as “CPU,” “bus,” and “main memory” from trips to Fry’s and Best Buy...and now you know what they truly mean
- Here’s how other terms that you might have heard before tie in to this whole thing:
 - ◆ “CPU speed,” measured by frequency (MHz, GHz), refers to how many *clock cycles* a CPU performs per second, roughly delimiting the smallest possible chunk of work that a CPU can perform — a “fetch-decode-execute” sequence takes one or more clock cycles
 - ◆ Along with CPU speed comes *bus speed*, which expresses how quickly information can pass between the CPU and main memory or external devices
 - ◆ Of course, there’s much more to all of these terms than can be said here, but at least you now have an overview of what they mean

Variations on the Theme

- The picture of a CPU fetching and performing instructions from main memory is a very simplified, though accurate, explanation of what goes on
- The reality (which you would learn in detail if you specialize in computer architecture) is that each one of these concepts has a wide range of variations:
 - ◆ There may be more than one CPU (or core) in a machine
 - ◆ Sometimes, the fetching and execution of instructions may overlap, forming a *pipeline* that tries to enhance the overall performance of the machine
 - ◆ The number and variety of registers varies by CPU
 - ◆ Main memory isn't necessarily a single physical chunk of circuitry; intervening "memory layers" called *caches* try to speed things up where applicable

Machine Language

- Back to that limited set of instructions that a CPU fetches and performs forever — for every CPU, these instructions are well-defined and precisely specified, forming that processor's *machine language*
- The *machine instructions* are, of course, expressed represented as specific bit patterns — but very few humans can ever truly memorize all of these
- To help us remember these machine instructions, an equivalent *assembly language* is also defined, which translates the machine code to *mnemonic* abbreviations

Available Instructions

- Two major philosophies exist regarding what instructions should be provided in machine language:
 - ◇ The *reduced instruction set computer* (RISC) philosophy prefers to limit the instruction set to the smallest and simplest possible collection
 - ◇ The *complex instruction set computer* (CISC) school of thought believes that more is better — have lots of instructions that perform more work per instruction
- Traditionally, Intel and AMD CPUs have been associated with CISC, while CPUs from Sun, IBM, ARM, and others are associated with RISC — but the reality is that the RISC/CISC distinction is very blurred today, as both sides have learned from each other
- As mentioned, machine instructions belong to a small number of categories, and everything that we do on a computer — surfing the net, playing music, writing papers — just strings these instructions together
 - ◇ *Data transfer*: These instructions have to do with moving data to/from the CPU and main memory
 - ◇ *Arithmetic/logic*: These instructions perform actual changes or calculations, using values in either registers or main memory, or both; the result of the calculation then appears in some predetermined location, again either a register or main memory
 - ◇ *Control*: These instructions affect where the next instruction should come from, and come in two major flavors — *conditional* and *unconditional*

Fetch-Decode-Execute in Greater Detail

- To fill out the whole data manipulation picture, we'll take the fetch-decode-execute cycle to one additional layer of detail
- First, let's look at two *special-purpose registers* that virtually all CPUs have
 - ◇ The *program counter* is a register that holds the main memory address of the next instruction to fetch
 - ◇ The *instruction register* is a register that holds the instruction that is currently being executed
- With these registers, we can now define the *fetch* phase a little more precisely: it loads the instruction located in the address found in the program counter into the instruction register
 - ◇ In addition to the instruction loading, the fetch phase also *increments* the program counter so that its address now points to the next instruction in main memory
- In the *decode* phase, the CPU's control unit figures out the specifics of the instruction: the action to perform (load, store, add, jump, etc.), and any *operands* of that action (registers affected, new values, etc.)
- Finally, in the *execute* phase, the instruction is performed, with calculations and comparisons taken care of by the ALU, and results appearing either in registers or main memory; conditional instructions affect the value of the program counter