|  |  |  |
| --- | --- | --- |
| Explanation of the execution of a program, using a simple three-instruction example |  | This presentation looks at the data flow between the CPU and memory as a program is executed.  Both the program and the data it is working on are stored in the same memory, and instructions are fetched to the CPU where they are executed.  This basic architecture was described by von Neumann, Eckert and Mauchly in the mid-1940s and is still with us, although it is now much improved.  Let’s begin by looking back at our diagram of the components making up a computer. |
| A technology topic |  | It fits within the technology portion of our class. |
| Review of the functional components of a computer system |  | This diagram shows the functional components of a computer.  Nearly all computers from the largest supercomputer to the computer in your cell phone have these components.  In this presentation, we will focus on the CPU and memory, since the CPU executes programs that are in memory. |
| Review of CPU and memory hardware |  | The CPU and memory are electronic chips.  They’re installed on a circuit board inside a computer.  The photo at the bottom shows a portion of a circuit board.  As you see, the wires that connect the devices are printed on the board.  Since they are packaged together this way, information can be transferred at very high speed between the CPU and memory.  Now let’s take a look at the execution of a sample program. |
| The operating system, the application program and the application data are all in memory while the program executes. |  | For the CPU to execute a program, it must be in memory along with the data it is working on.  The operating system is also in memory.  Let’s drill down to see what happens when a program executes. |
| A simple, three-instruction machine-language program |  | Consider this program segment, which has been loaded into the computer’s memory.  It consists of three machine-language instructions that direct the computer to add the numbers in memory locations X and Y and leave the result in memory location Z.  The instructions will be executed in order.  Study it before going on – see if you can anticipate what will happen when it executes. What will be the final value of Z after execution of the third instruction?  When you are ready, let’s take a closer look. |
| A closer look at the CPU and the program and its data in memory |  | Here we see more detail.  Both the program and the data it operates on are in memory.  We also have a more detailed view of the CPU.  It has a small amount of internal memory – a place to hold an instruction while it is being executed and a place to hold the number it is working on, called a “register.”  Before execution begins, the program and the numbers to be added are in memory, but the CPU has no current instruction and it’s register is empty.  Let’s look at the execution of the first instruction. |
| Fetch and execute the first instruction |  | As shown here, the first step in executing the program is to copy the first instruction from memory to the CPU.  (A geek would say the instruction is “fetched”).  Once the instruction is in the CPU, the CPU executes it.  Can you anticipate how our diagram will change when the first instruction is executed?  Don’t go on until you think about it. |
| After the first instruction has executed |  | We see here that the first instruction has executed, loading the number 25 into the CPU register.  Note that the data was copied, not moved, so memory location X still contains 25.  Next, the second instruction will be copied to the CPU and executed.  Before going on, try to anticipate the way our diagram will change. |
| After the second instruction has executed |  | We see that the second instruction has been loaded into the CPU and executed.  The number 10 was copied from location Y and added to the 25 that was already in the register, leaving 35 in the register.  Next, we will execute the third instruction.  What will our diagram look like after that completes? |
| After the third instruction has executed |  | We see that the third instruction has been fetched into the CPU and executed.  The sum, 35, has been copied from the register to location Z.  It can now be operated on by another part of the program, sent to an output device like a printer or display or stored.  That completes our short example. |
| The fetch-execute cycle |  | As we have seen, an instruction is fetched from memory into the CPU, then executed.  Then the next instruction is fetched and executed.  The process continues until the program stops. |
| This basic von Neumann architecture was designed in the mid-1940s. |  | Computers that implement the fetch-execute model and store both the program and the data it is working on in the same memory are called are called “von Neumann machines,” after the famous Hungarian mathematician, physicist and engineer John von Neumann.  Von Neumann, shown on the left, and two colleagues published an influential paper outlining this computer architecture in 1946.  The photo on the right shows John Mauchly and John Presper Eckert (standing), who designed and built what most consider to be the first programmable electronic computer, the ENIAC (Electronic Numerical Integrator And Computer), at the University of Pennsylvania.  Around the time von Neumann’s report was published, they established the Eckert-Mauchly Computer Corporation, where they designed the first commercial computer.  They sold their company to Remington Rand, where they continued work on their first computer, called the UNIVAC (Universal Automatic Computer).  (It was “universal” since it could be used for both scientific and business work).  In 1950, they sold a UNIVAC 1 to the US Census Bureau.  While the ENIAC was not a von Neumann machine, UNIVAC was. |
| Today’s CPUs are more complex, but still adhere to the basic von Neumann architecture. |  | Our example has been simplified to make the basic idea clear.  The earliest computers worked on one instruction at a time, as in our example.  By the late 1950s, the largest, most expensive computers could afford some of the parallel execution we have today.  We used only one register in our example, but a modern CPU would have many.  We showed only one instruction being executed at a time, but a modern CPU would work on several instructions at once.  We showed only one instruction being in the CPU, but modern CPUs fetch instructions ahead so they don’t have to wait for transfers from memory.  Furthermore, we are packaging multiple CPUs on a single chip.  Today’s CPUs are also billions of times faster, smaller and more power efficient than the UNIVAC, but the basic architecture is unchanged. |
| Today’s CPUs are billions of times faster and smaller than those of the 1940s. |  | You may be thinking that the sorts of instructions we used in our example – load a number into a CPU register, add two numbers, copy a number from the CPU to memory – are simple.  It is true that one instruction does very little.  But, today’s CPUs execute billions or even trillions of instructions per second.  That enables us to write programs that do complicated things using simple machine-language instructions.  But, writing a program with millions of machine-language instructions would be tedious and error prone, so programmers write programs using development tools and high-level languages which are translated automatically into machine language. |
| Summary |  | We used a three-instruction program segment to illustrate the information flow between the CPU and memory when a program executes.  We saw that instructions are copied (fetched) from memory to the CPU, where they are executed.  While an individual instruction does very little work, they are executed so fast, that we can program computers to do complex things.  Programmers use higher-level programming languages and tools to generate machine language programs.  Today’s computers still implement the fetch-execute model and store both the program and the data it is working on in memory.  While the early computers executed only one instruction at a time, today’s computers are much faster and work on several instructions in parallel, but the basic operation is the same. |

**Self-study questions**

What would happen if the programmer of our three instruction example accidently used a subtract instruction instead of an add instruction in step 2?

What would happen if the programmer of our three instruction example accidently switched the order of steps 1 and 2?

What would happen if the programmer of our three instruction example accidently switched the order of steps 2 and 3?

**Resources**

Arthur W. Burks, Herman H. Goldstine  and John von Neumann, *Preliminary discussion of the logical design of an electronic computing instrument,* The Institute for Advanced Study, Princeton University, 28 June 1946.

<http://www.fdi.ucm.es/profesor/mozos/EC/burks.pdf>

The paper outlines the architecture of a “von Neumann machine,“ going into considerable detail on the design of the CPU.

Also see the Wikipedia artcles on von Neumann and Eckert and Mauchley.

<http://spectrum.ieee.org/podcast/computing/software/from-ballistics-to-programming>

A terrific podcast on the role of the women who were computing artillery shell trajectories during World War II and became the first ENIAC programmers.