An Introduction to Computer Science

Jingde Cheng

Southern University of Science and Technology

An Introduction to Computer Science

- Computer Science: What Is It and Why Study It?
- Computation: What Is It and Why Study It?
- Computability
- Computational Complexity (CS101A class only)
- Algorithms
- Data, Information, and Knowledge, and Their Representations
- Data Storage
- Computer Architecture
- Data Manipulation in Computer Systems
- Programming Languages and Compilers
- Operating Systems
- System Software and Application Software
- Software Engineering (CS101A class only)
- Knowledge Engineering and Artificial Intelligence (CS101A class only)
- Information Security Engineering (CS101A class only)

The Question: How to Automate Computing?

How to automate computing ?

(in the electronic digital way)

The Question: What Is a Computer ?

What is a (electronic digital) computer ?

Computer Architecture: What Is It?

- ♣ Fundamental questions on computers
  - What is a computer?
  - What is a computer in the sense of today's computer science?
- ♣ Computer architecture [A Dictionary of Computer Science, 7th Edition, OUP, 2016]
  - "The specification of a (digital) computer system at a somewhat general level, including description from the programming (user) viewpoint of the instruction set and user interface, memory organization and addressing, I/O operation and control, etc."
- ♣ Architecture [IEEE Standard Computer Dictionary]
  - "The organizational structure of a system or component."

Computer Architecture: What Is It? [DL-CS-16]

- ♣ An important fact
  - The "von Neumann" architecture was NOT invented by von Neumann !

冯·诺依曼结构不是冯·诺依曼发明的

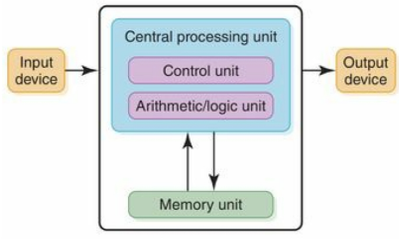


FIGURE 5.1 The von Neumann architecture.

Computer Architecture: What Is It? [TA-SCO-13]

The basic design, which he first described, is now known as a **von Neumann machine**. It was used in the EDSAC, the first stored-program computer, and even now, more than half a century later, is still the basis for nearly all digital computers. This design, and the IAS machine, built in collaboration with Herman Goldstine, has had such an enormous influence that it is worth describing briefly. Although Von Neumann's name is always attached to this design, Goldstine and others made major contributions to it as well. A sketch of the architecture is given in Fig. 1-5.

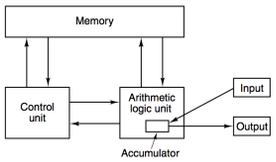
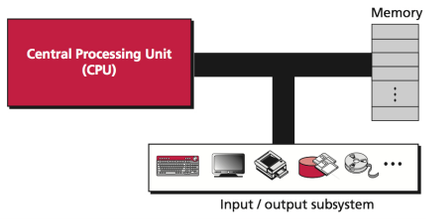


Figure 1-5. The original von Neumann machine.

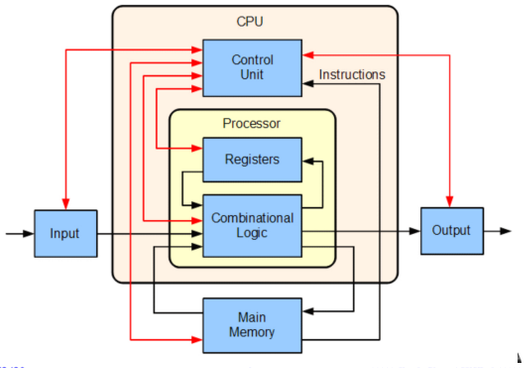
◆ -- A. S. Tanenbaum and T. Austin, "Structured Computer Organization," 6th Edition, 2013.

Computer Architecture: What Is It? [F-CS-18]

Figure 5.1 *Computer hardware (subsystems)*



Computer Architecture: What Is It? [Wikipedia]



The Stored-Program Concept

♣ Stored-program 存储程序

◆ *A program, just like data, can be encoded and stored in main memory* (credited, apparently incorrectly, to John von Neumann).

♣ Changeable-program (Self-modifying code) 可变程序(程序自修改)

◆ If the control unit is designed to extract the program from memory, decode the instructions, and execute them, *the program that the machine follows can be changed merely by changing the contents of the computer's memory instead of rewiring the CPU (run-time-changeable-program)*.

♣ The stored-program concept

◆ The idea of storing a computer's program in its main memory is called *the stored-program concept* and has become the standard approach used today.

将程序存在主存中

The Stored-Program Concept [DL-CS-16]

♣ The stored-program concept

◆ A major defining point in the history of computing was the realization in 1944-1945 that *data and instructions to manipulate the data were logically the same and could be stored in the same place*.

♣ The "von Neumann" architecture

◆ The computer design built upon *the principle of the stored-program concept*, which became known as the "von Neumann" architecture, is still the basis for computers today.

◆ Note: Another major characteristic of the "von Neumann" architecture is that the units that process data are separate from the units that store data.

数据和操作指令在逻辑上是相同的，可以存在同一位置

操作单元与存储单元是分开的

The Separation of Processing and Storage [DL-CS-16]

♣ The separation of processing and storage

◆ Another major characteristic of the "von Neumann" architecture is that the units that process data are separate from the units that store data.

♣ The "von Neumann" architecture

◆ The characteristic leads to the following five components of the "von Neumann" architecture:

- The memory unit that holds both data and instructions
- The arithmetic/logic unit that is capable of performing arithmetic and logic operations on data
- The input unit that moves data from the outside world into the computer
- The output unit that moves results from inside the computer to the outside world
- The control unit that acts as the stage manager to ensure that all the other components act in concert

Facts about "von Neumann" Architecture

- The historical fact [DL-CS-16]
  - Although the name honors John von Neumann, the idea probably originated with J. Presper Eckert and John Mauchly, two other early pioneers who worked on the ENIAC at the Moore School at the University of Pennsylvania during the same time period.
  - **The stored-program concept was considered before that von Neumann joined ENIAC project.**
- Why "von Neumann"?
  - von Neumann joined ENIAC project midway, NOT from beginning.
  - von Neumann wrote a report ("First Draft of a Report on the EDVAC"), with his single name but without confirm of other members, about design discussion of EDVAC (the successor of ENIAC).

Facts about "von Neumann" Architecture

- The von Neumann's draft
  - John von Neumann, "First Draft of a Report on the EDVAC," (Contract No.W-670-ORD-4926) Moore School of Electric Engineering, University of Pennsylvania, June 30, 1945.
- The formal report
  - A. W. Burks, H. H. Goldstine, and J. von Neumann, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," Institute for Advanced Study, Princeton, June 1946.

Facts about "von Neumann" Architecture

- M. V. Wilkes's statements in his 1976 Turing Award lecture
  - "Eckert and Mauchly appreciated that the main problem was one of storage, and they proposed for future machines the use of ultrasonic delay lines. Instructions and numbers would be mixed in the same memory. … von Neumann was, at that time, associated with the Moore School group in a consultative capacity. ... The computing field owes a very great debt to von Neumann. He appreciated at once ... the potentialities implicit in the stored program principle. That von Neumann should bring his great prestige and influence to bear was important, since the new ideas were too revolutionary for some, and powerful voices were being raised to say that the ultrasonic memory would not be reliable enough, and that to mix instructions and numbers in the same memory was going against nature. ... Subsequent developments have provided a decisive vindication of the principles taught by Eckert and Mauchly."

Facts about "von Neumann" Architecture

- M. V. Wilkes's statement
  - As far as I know, Von Neumann never said once that the only inventor of the stored-program method is himself. For that reason, I don't like to use the term 'von Neumann computer' as the general name of stored-program computers.
  - 「私の知る限りでは、プログラム内蔵方式の唯一の発明者は自分であるとフォン・ノイマンがいったことは一度もない。そうした理由で、フォン・ノイマン型コンピュータという用語をプログラム内蔵方式コンピュータの一般名称として使うのは、私は好きではない。」

Facts about "von Neumann" Architecture

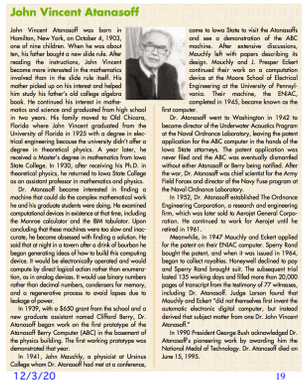
- N. Metropolis and J. Worlton' statements
  - "The stored-program concept predates von Neumann's participation in the EDVAC design. That von Neumann is often given credit for this fundamental concept is likely due to the fact that he wrote a preliminary report which summarized the earlier work on the EDVAC design, including the stored-program concept. Von Neumann contributed significantly to the development of this concept, but to credit him with its invention is an historical error."
- J. Copeland's statement
  - It is "historically inappropriate, to refer to electronic stored-program digital computers as 'von Neumann machines'".
- Homework
  - Read M. V. Wilkes's ACM Turing Lecture (1967), "Computers Then and Now" and J. Copeland's "A Brief History of Computing".

Who Invented What?

Who Invented What?

Awarding a single individual credit for an invention is always a dubious undertaking. Thomas Edison is credited with inventing the incandescent lamp, but other researchers were developing similar lamps, and in a sense Edison was lucky to be the one to obtain the patent. The Wright brothers are credited with inventing the airplane, but they were competing with and benefited from the work of many contemporaries, all of whom were preempted to some degree by Leonardo da Vinci, who toyed with the idea of flying machines in the fifteenth century. Even Leonardo's designs were apparently based on earlier ideas. Of course, in these cases the designated inventor still has legitimate claims to the credit bestowed. In other cases, history seems to have awarded credit inappropriately—an example is the stored-program concept. Without a doubt, John von Neumann was a brilliant scientist who deserves credit for numerous contributions. But one of the contributions for which popular history has chosen to credit him, the stored-program concept, was apparently developed by researchers led by J. P. Eckert at the Moore School of Electrical Engineering at the University of Pennsylvania. John von Neumann was merely the first to publish work reporting the idea and thus computing lore has selected him as the inventor.

J. V. Atanasoff and The Atanasoff–Berry Computer [DL-CS-02]


John Vincent Atanasoff

John Vincent Atanasoff was born in Hamilton, New York, on October 4, 1903, one of nine children. When he was about ten, his father bought a new slide rule. After reading the instructions, John Vincent became more interested in the mathematics involved than in the slide rule itself. His mother picked up on his interest and helped him study his father's old college algebra book. He continued his interest in mathematics and science and graduated from high school in two years. His family moved to Old Chicora, Florida where John Vincent graduated from the University of Florida in 1925 with a degree in electrical engineering because the university didn't offer a degree in theoretical physics. A year later, he received a Master's degree in mathematics from Iowa State College. In 1930, after receiving his Ph.D. in theoretical physics, he returned to Iowa State College as an assistant professor in mathematics and physics.

Dr. Atanasoff became interested in finding a machine that could do the complex mathematical work he and his graduate students were doing. He examined computational devices in existence at that time, including the Monroe calculator and the IBM tabulator. Upon concluding that these machines were too slow and inaccurate, he became obsessed with finding a solution. He said that at night in a tavern after a drink of bourbon he began generating ideas of how to build his computing device. It would be electronically operated and would compute by direct logical action rather than enumeration, as in analog devices. It would use binary numbers rather than decimal numbers, condensers for memory, and a regenerative process to avoid lapses due to leakage of power.

In 1939, with a $650 grant from the school and a new graduate assistant named Clifford Berry, Dr. Atanasoff began work on the first prototype of the Atanasoff Berry Computer (ABC) in the basement of the physics building. The first working prototype was demonstrated that year.

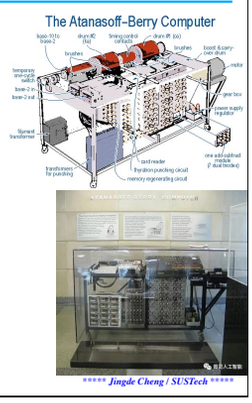
In 1941, John Mauchly, a physicist at Ursinus College whom Dr. Atanasoff had met at a conference, came to Iowa State to visit the Atanasoffs and see a demonstration of the ABC machine. After extensive discussions, Mauchly left with papers describing its design. Mauchly and J. Presper Eckert continued their work on a computation device at the Moore School of Electrical Engineering at the University of Pennsylvania. Their machine, the ENIAC, completed in 1945, became known as the first computer.

Dr. Atanasoff went to Washington in 1942 to become director of the Underwater Acoustics Program at the Naval Ordnance Laboratory, leaving the patent application for the ABC computer in the hands of the Iowa State attorneys. The patent application was never filed and the ABC was eventually dismantled without either Atanasoff or Berry being notified. After the war, Dr. Atanasoff was chief scientist for the Army Field Forces and director of the Navy Fuze program at the Naval Ordnance Laboratory.

In 1952, Dr. Atanasoff established The Ordnance Engineering Corporation, a research and engineering firm, which was later sold to Aerojet General Corporation. He continued to work for Aerojet until he retired in 1961.

Meanwhile, in 1947 Mauchly and Eckert applied for the patent on their ENIAC computer. Sperry Rand bought the patent, and when it was issued in 1964, began to collect royalties. Honeywell declined to pay and Sperry Rand brought suit. The subsequent trial lasted 135 working days and filled more than 20,000 pages of transcript from the testimony of 77 witnesses, including Dr. Atanasoff. Judge Larson found that Mauchly and Eckert "did not themselves first invent the automatic electronic digital computer, but instead derived that subject matter from one Dr. John Vincent Atanasoff."

In 1990 President George Bush acknowledged Dr. Atanasoff's pioneering work by awarding him the National Medal of Technology. Dr. Atanasoff died on June 15, 1995.

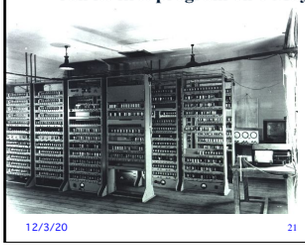


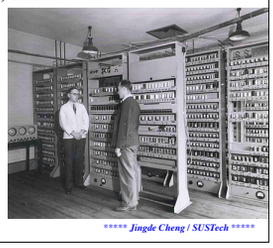
What is a Computer? (From the viewpoint of architecture)

- Three necessary requirements for computers
  - **Program-control**
  - **Internally-stored-program**
  - **Changeable-program (Self-modifying code)**
- The first computer
  - EDSAC (Electronic Delay Storage Automatic Calculator), University of Cambridge, 6 May, 1949.

EDSAC (Electronic Delay Storage Automatic Calculator)

- EDSAC
  - The machine was constructed (inspired by von Neumann's First Draft of a Report on the EDVAC) by Maurice V. Wilkes (The second recipient of the ACM A. M. Turing Award) and his team at the University of Cambridge, and ran its first program on 6 May, 1949.





ACM Turing Award

- 1967, Maurice V. Wilkes
  - Citation: Professor Wilkes is best known as the builder and designer of the EDSAC, the first computer with an internally stored program. Built in 1949, the EDSAC used a mercury delay line memory. He is also known as the author, with Wheeler and Gill, of a volume on "Preparation of Programs for Electronic Digital Computer" in 1951, in which program libraries were effectively introduced.
  - Maurice Vincent Wilkes (born June 26, 1913 in Dudley, Staffordshire, England) is a British computer scientist, credited with several important developments in computing.

Memory Unit [DL-CS-16]

- Memory unit
  - *Memory unit* is a collection of *cells*, each with a unique *physical address*.
- Addressability
  - We use the generic word cell here rather than byte or word, because the number of bits in each addressable location, called the memory's *addressability*, varies from one machine to another.
  - Today, most computers are *byte addressable*.

Addressability The number of bits stored in each addressable location in memory

ALU: Arithmetic/Logic Unit [DL-CS-16]

- ALU: Arithmetic/logic unit
  - *The arithmetic/logic unit* is capable of performing basic *arithmetic operations* such as adding, subtracting, multiplying, and dividing two numbers.
  - This unit is also capable of performing *logical operations* such as AND, OR, and NOT.
- The word length
  - *The word length* of a computer is the number of bits processed at once by the ALU.

Arithmetic/logic unit (ALU) The computer component that performs arithmetic operations (addition, subtraction, multiplication, and division) and logical operations (comparison of two values)

Register Unit [DL-CS-16]

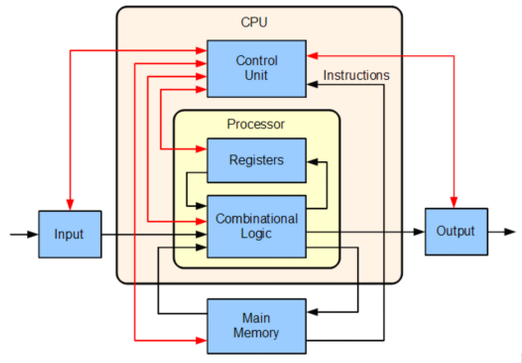
- **Registers** 寄存器
  - Most modern CPUs have a small number of special storage units called *registers*.
  - These registers contain one word and are used to store data that is needed again immediately.
- **General-purpose/special-purpose registers**
  - Some of the registers within the *register unit* are considered *general-purpose registers* (e.g., for a/l operations) whereas others are *special-purpose registers* (e.g., PC and IR)

当前正在执行

下一条要执行

Register A small storage area in the CPU used to store intermediate values or special data

Computer Architecture: What Is It? [Wikipedia]



Input/Output Unit [DL-CS-16]

- **Input unit**
  - *An input unit* is a device through which data and programs from the outside world are entered into the computer.
  - The first input units interpreted holes punched on paper tape or cards.
  - Modern-day input devices include the keyboard, the mouse, and the scanning devices used at supermarkets.

Input unit A device that accepts data to be stored in memory

Input/Output Unit [DL-CS-16]

- **Output unit**
  - *An output unit* is a device through which results stored in the computer memory are made available to the outside world.
  - The most common output devices are printers and displays.

Output unit A device that prints or otherwise displays data stored in memory or makes a permanent copy of information stored in memory or another device

IR: 当前正在执行的指令
(要执行之快执行完)

PC: 下一条指令地址

Control Unit and CPU [DL-CS-16]

- **Control unit** 控制单元
  - *The control unit* is the organizing force in the computer, for it is in charge of *the fetch–execute cycle*.
  - There are two special registers in the control unit.
  - *The instruction register* (*IR*) contains the instruction that is being executed, and *the program counter* (*PC*) contains the address of the next instruction to be executed.
- **CPU: the central processing unit**
  - Because the ALU and the control unit work so closely together, they are often thought of as one unit called *the central processing unit*, or *CPU* (often referred to as merely *the processor*).

Control Unit [DL-CS-16]

Control unit The computer component that controls the actions of the other components so as to execute instructions in sequence

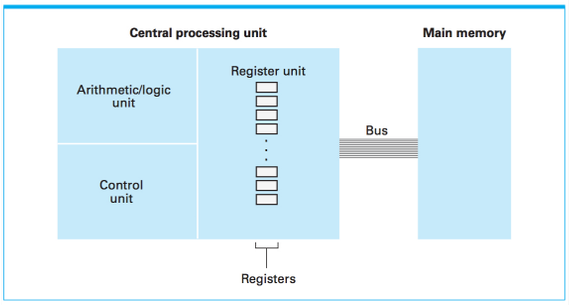
Instruction register (IR) The register that contains the instruction currently being executed

Program counter (PC) The register that contains the address of the next instruction to be executed

CPU The central processing unit, a combination of the arithmetic/logic unit and the control unit; the "brain" of a computer that interprets and executes instructions

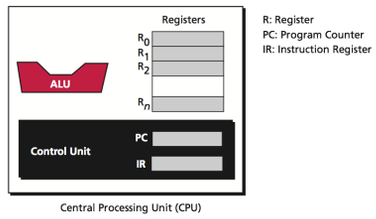
Central Processing Unit (CPU) and Main Memory

Figure 2.1 CPU and main memory connected via a bus



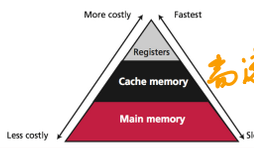
Central Processing Unit (CPU) [F-CS-18]

Figure 5.2 *Central processing unit (CPU)*



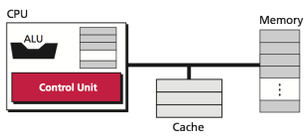
Memory Hierarchy [F-CS-18]

Figure 5.4 *Memory hierarchy*



高速缓冲存储器

Figure 5.5 *Cache memory*



Memory Hierarchy [TB-OS-15]

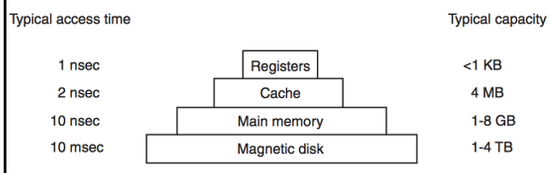


Figure 1-9. A typical memory hierarchy. The numbers are very rough approximations.

- nsec：nano-second (纳秒，毫微秒)
- msec: milli-second (毫秒)

Data Flow through a Computer [DL-CS-16]

♣ Data flow through a computer

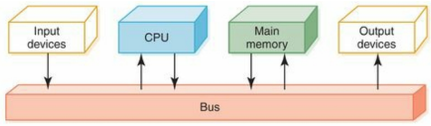


FIGURE 5.2 Data flow through a von Neumann machine

♣ Bus 总线

- The parts are connected to one another by a collection of wires called a *bus*, through which data travels in the computer. Each bus carries three kinds of data: address, data, and control.
- *Bus width* is the number of bits that can be transferred in parallel over the bus.

Connection Buses [F-CS-18]

Figure 5.12 *Connecting CPU and memory using three buses*

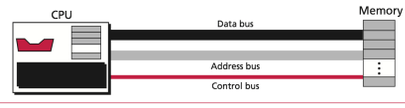
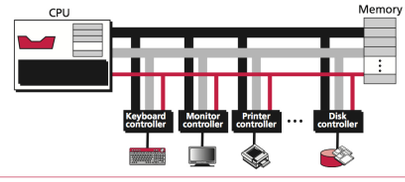


Figure 5.13 *Connecting I/O devices to the buses*



Connection Bus Controllers [F-CS-18]

Figure 5.14 SCSI controller

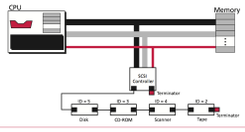


Figure 5.15 FireWire controller

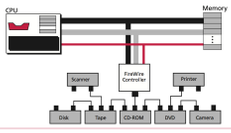
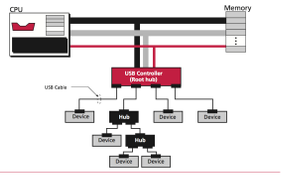


Figure 5.16 USB controller



Connection Bus Controllers [F-CS-18]

Figure 5.17 Isolated I/O addressing

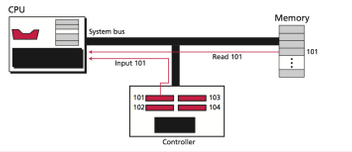
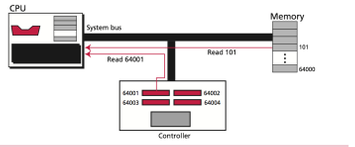


Figure 5.18 Memory-mapped I/O addressing



The Fetch–Execute Cycle [DL-CS-16]

- **Addressable instructions and data**
  - The principle of stored-program underlying computers means that instructions and data are both addressable.
  - Instructions are stored in contiguous memory locations; data to be manipulated are stored together in a different part of memory.
- **The fetch-execute cycle**
  - To start *the fetch–execute cycle*, the address of the first instruction is loaded into the program counter (PC).
  - The processing cycle includes four steps: (1) Fetch the next instruction, (2) Decode the instruction, (3) Get data if needed, and (4) Execute the instruction.

The Fetch–Execute Cycle [DL-CS-16]

- **Alan Jay Perlis**
  (April 1, 1922 – February 7, 1990)
  - The first recipient of the ACM A. M. Turing Award.
- **The only universal in the computing field**

> **"Sometimes I think the only universal in the computing field is the fetch–execute cycle." – A. J. Perlis, 1981.**

The Fetch–Execute Cycle [DL-CS-16]

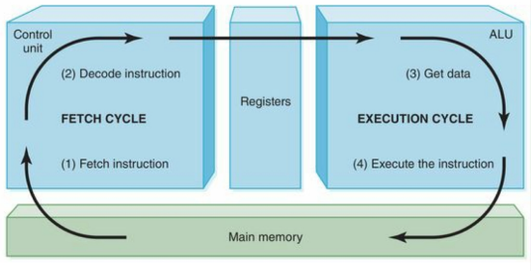
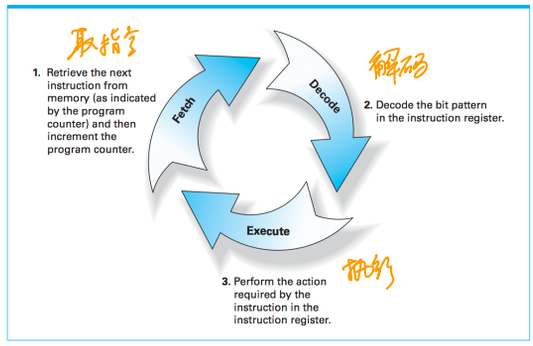


FIGURE 5.3 The fetch–execute cycle

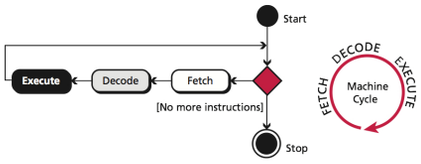
The Fetch–Execute Cycle

Figure 2.8 The machine cycle



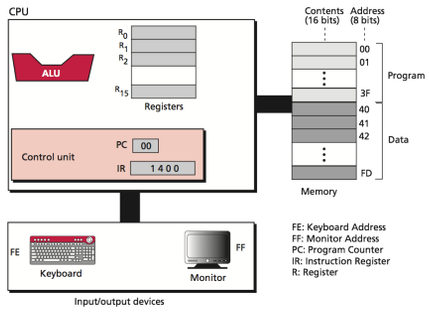
The Fetch–Execute Cycle [F-CS-18]

Figure 5.19 *The steps of a cycle*



Computer Architecture [F-CS-18]

Figure 5.30 *The components of a simple computer*



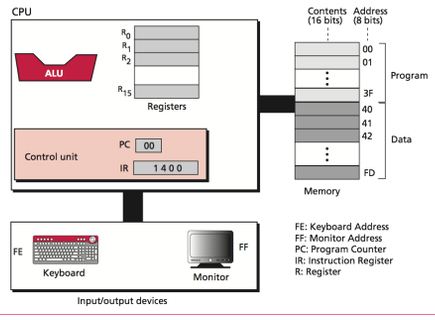
The Fetch–Execute Cycle: Fetch [DL-CS-16]

♣ **Fetch the next instruction**

- The program counter (PC) contains the address of the next instruction to be executed, so the control unit goes to the address in memory specified in the PC, makes a copy of the contents, and places the copy in the instruction register (IR).
- At this point the IR contains the instruction to be executed.
- Before going on to the next step in the cycle, the PC must be updated to hold the address of the next instruction to be executed when the current instruction has been completed.
- Because the instructions are stored contiguously in memory, adding the number of bytes in the current instruction to the PC should put the address of the next instruction into the PC.
- Thus the control unit increments the PC.
- It is possible that the PC may be changed later by the instruction being executed.

Computer Architecture [F-CS-18]

Figure 5.30 *The components of a simple computer*



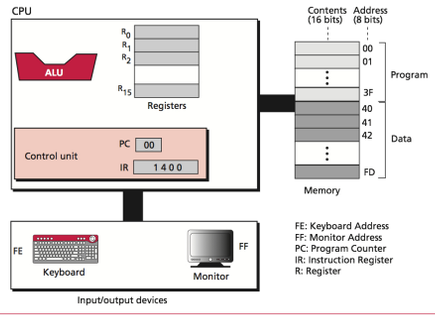
The Fetch–Execute Cycle: Decode [DL-CS-16]

♣ **Decode the instruction**

- To execute the instruction in the instruction register (IR), the control unit has to determine what instruction it is.
- It might be an instruction to access data from an input device, to send data to an output device, or to perform some operation on a data value.
- At this phase, the instruction is decoded into control signals.
- That is, the logic of the circuitry in the CPU determines which operation is to be executed.
- This step shows why a computer can execute only instructions that are expressed in its own machine language.
- The instructions themselves are literally built into the circuits.

Computer Architecture [F-CS-18]

Figure 5.30 *The components of a simple computer*



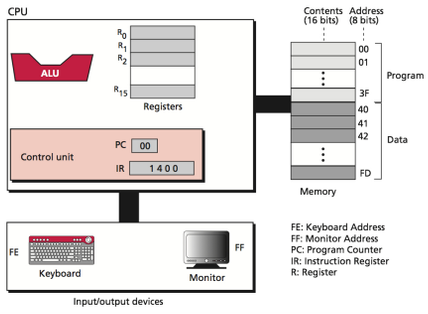
The Fetch–Execute Cycle: Get Data [DL-CS-16]

♣ **Get data if needed**

- The instruction to be executed may potentially require additional memory accesses to complete its task.
- For example, if the instruction says to add the contents of a memory location to a register, the control unit must get the contents of the memory location.
- In the case of an instruction that must get additional data from memory, the ALU sends an address to the memory bus, and the memory responds by returning the value at that location.

Computer Architecture [F-CS-18]

Figure 5.30 *The components of a simple computer*



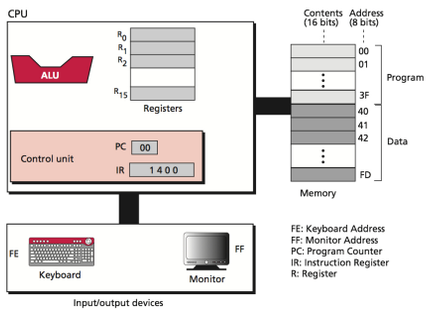
The Fetch–Execute Cycle: Execute [DL-CS-16]

♣ **Execute the instruction**

- Once an instruction has been decoded and any operands (data) fetched, the control unit is ready to execute the instruction.
- Execution involves sending signals to the arithmetic/logic unit (ALU) to carry out the processing.
- In the case of adding a number to a register, the operand is sent to the ALU and added to the contents of the register.

Computer Architecture [F-CS-18]

Figure 5.30 *The components of a simple computer*



The Fetch–Execute Cycle: The Next Cycle [DL-CS-16]

♣ **The next cycle**

- When the execution is complete, the cycle begins again.
- If the last instruction was to add a value to the contents of a register, the next instruction probably says to store the results into a place in memory.
- However, the next instruction might be a control instruction -- that is, an instruction that asks a question about the result of the last instruction and perhaps changes the contents of the program counter PC.

An Example: Adding Values Stored in Memory

Figure 2.2 Adding values stored in memory

Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

Machine Instruction 机器指令

♣ **Machine instruction**

- Each computer must have a defined set of *machine instructions*.
- To apply the stored-program concept, CPUs are designed to recognize machine instructions encoded as bit patterns.

♣ **Notes**

- The set of machine instructions that a typical CPU must be able to decode and execute is quite short.
- In fact, once a machine can perform certain elementary but well-chosen tasks, adding more features does not increase the machine's theoretical capabilities.

Machine Language

♣ **Machine language** 机器语言

- The set of instructions along with the encoding system is called the *machine language*.

Machine language The language made up of binary-coded instructions that is used directly by the computer

CISC architecture vs. RISC architecture

♣ **Two philosophies of CPU architecture**

- One is that a CPU should be designed to execute a minimal set of machine instructions, another is that a CPU should be designed to execute a large number of complex instructions, even though many of them are technically redundant.

♣ **RISC architecture**

- The first approach leads to what is called a *reduced instruction set computer* (*RISC*). 简单指令计算机
- The argument in favor of RISC architecture is that such a machine is efficient, fast, and less expensive to manufacture.

通过一般简单的命令建构
所有的命令

CISC architecture vs. RISC architecture

♣ **CISC architecture** 复杂指令计算机

- The second approach leads to what is called a *complex instruction set computer* (*CISC*).
- The argument in favor of CISC architecture is that the more complex CPU can better cope with the ever increasing complexities of today's software.
- With CISC, programs can exploit a powerful rich set of instructions, many of which would require a multi-instruction sequence in a RISC design.

♣ **Examples**

- CISC: Intel processors, AMD processors
- RISC: PowerPC processors, ARM processors

Parallel Architectures [DL-CS-16] 并行处理

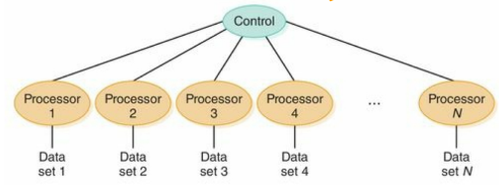


FIGURE 5.8 Processors in a synchronous computing environment

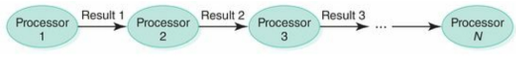


FIGURE 5.9 Processors in a pipeline

Parallel Architectures [DL-CS-16]

共享存储器

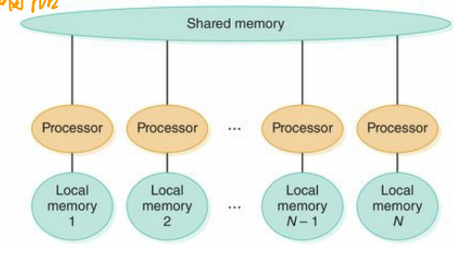


FIGURE 5.10 A shared-memory parallel processor

Parallel Architectures [F-CS-18]

Figure 5.24 *Pipelining*

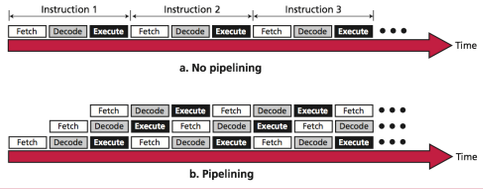
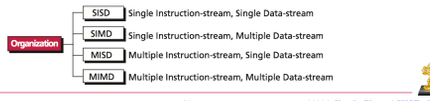


Figure 5.25 *A taxonomy of computer organization*



Milestones in Development of Computers [TA-SCO-13]

Year	Name	Made by	Comments
1834	Analytical Engine	Babbage	First attempt to build a digital computer
1936	Z1	Zuse	First working relay calculating machine
1943	COLOSSUS	British gov't	First electronic computer
1944	Mark I	Aiken	First American general-purpose computer
1946	ENIAC	Eckert/Mauchley	Modern computer history starts here
1949	EDSAC	Wilkes	First stored-program computer
1951	Whirlwind I	M.I.T.	First real-time computer
1952	IAS	Von Neumann	Most current machines use this design
1960	PDP-1	DEC	First minicomputer (50 sold)
1961	1401	IBM	Enormously popular small business machine
1962	7094	IBM	Dominated scientific computing in the early 1960s
1963	B5000	Burroughs	First machine designed for a high-level language
1964	360	IBM	First product line designed as a family
1964	6600	CDC	First scientific supercomputer
1965	PDP-8	DEC	First mass-market minicomputer (50,000 sold)
1970	PDP-11	DEC	Dominated minicomputers in the 1970s
1974	8080	Intel	First general-purpose 8-bit computer on a chip
1974	CRAY-1	Cray	First vector supercomputer
1978	VAX	DEC	First 32-bit supermini computer
1981	IBM PC	IBM	Started the modern personal computer era
1981	Osborne-1	Osborne	First portable computer
1983	Lisa	Apple	First personal computer with a GUI
1985	386	Intel	First 32-bit ancestor of the Pentium line
1985	MIPS	MIPS	First commercial RISC machine
1985	XC2064	Xilinx	First field-programmable gate array (FPGA)
1987	SPARC	Sun	First SPARC-based RISC workstation
1989	GridPad	Grid Systems	First commercial tablet computer
1990	RS6000	IBM	First superscalar machine
1992	Alpha	DEC	First 64-bit personal computer
1992	Simon	IBM	First smartphone
1993	Newton	Apple	First palmtop computer (PDA)
2001	POWER4	IBM	First dual-core chip multiprocessor

Figure 1-4. Some milestones in the development of the modern digital computer.

An Introduction to Computer Science

- Computer Science: What Is It and Why Study It?
- Computation: What Is It and Why Study It?
- Computability
- Computational Complexity (CS101A class only)
- Algorithms
- Data, Information, and Knowledge, and Their Representations
- Data Storage
- Computer Architecture
- Data Manipulation in Computer Systems
- Programming Languages and Compilers
- Operating Systems
- System Software and Application Software
- Software Engineering (CS101A class only)
- Knowledge Engineering and Artificial Intelligence (CS101A class only)
- Information Security Engineering (CS101A class only)

The Question: How is Data Manipulated in a Computer?

How is data manipulated in a computer ?

(in the electronic digital way)

Various Machine Instructions

♣ Three groups of machine instructions

- Regardless of the choice between RISC and CISC, a machine's instructions can be categorized into three groups:
  (1) the *data transfer group*,
  (2) the *arithmetic/logic group*, and
  (3) the *control group*.

Various Machine Instructions: Data Transfer Instructions

♣ Data transfer instructions

- The data transfer (actually a misnomer, "copy" is more suitable term) group consists of instructions that request the movement of data from one location to another.
- The process involved in a transfer instruction is more like copying the data rather than moving it. Thus terms such as copy or clone better describe the actions of this group of instructions.

♣ Load and Store instructions

- A request to fill a general-purpose register with the contents of a memory cell is commonly referred to as a *LOAD instruction*; conversely, a request to transfer the contents of a register to a memory cell is called a *STORE instruction*.

Various Machine Instructions: Data Transfer Instructions

- **Input and Output instructions**
  - An important group of instructions within the data transfer category consists of the commands for communicating with devices outside the CPU-main memory context (printers, keyboards, display screens, disk drives, etc.).
  - Since these instructions handle the input/output (I/O) activities of the machine, they are called *I/O instructions* and are sometimes considered as a category in their own right.
  - We shall consider the I/O instructions to be a part of the data transfer group.

Various Machine Instructions: Arithmetic/Logic Instructions

- **Arithmetic/logic instructions**
  - The arithmetic/logic group consists of the instructions that tell the control unit to request an activity within the arithmetic/logic unit.
  - The arithmetic/logic unit is capable of performing *the basic arithmetic operations* (e.g., *ADD*) and *the logical (Boolean) operations* (e.g., *AND*, *OR*, and *XOR*).
  - Another collection of operations available within most arithmetic/logic units allows the contents of registers to be moved to the right or the left within the register.
  - These operations are known as either *SHIFT* or *ROTATE* operations.

Various Machine Instructions: Control Instructions

- **Control instructions**
  - The control group consists of those instructions that direct the execution of the program rather than the manipulation of data.
  - This group contains many of the interesting instructions in a machine's repertoire, such as the family of *JUMP* (or *BRANCH*) instructions used to direct the CPU to execute an instruction other than the next one in the list.
  - These JUMP instructions appear in two varieties: *unconditional jumps* and *conditional jumps*.

An Example: Dividing Values Stored in Memory

Figure 2.3 Dividing values stored in memory

Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

Step 3. If this second value is zero, JUMP to Step 6.

Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

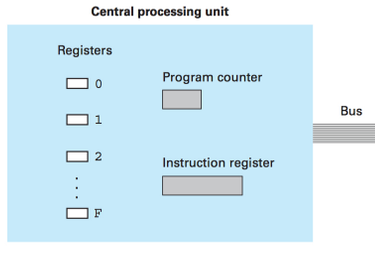
Step 6. STOP.

An Illustrative Machine (Appendix C): Architecture

- **The architecture**
  - It has 16 general-purpose registers and 256 main memory cells, each with a capacity of 8 bits.
- **Labels and addresses**
  - We label the registers with the values 0 through 15 and address the memory cells with the values 0 through 255.
  - For convenience we think of these labels and addresses as values represented in base two and compress the resulting bit patterns using hexadecimal notation.
  - Thus, the registers are labeled 0 through F, and the memory cells are addressed 00 through FF.

An Illustrative Machine (Appendix C): Architecture

Figure 2.4 The architecture of the machine described in Appendix C

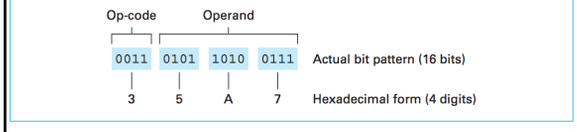


An Illustrative Machine (Appendix C): Instructions

- The encoded version of a machine instruction
  - The encoded version of a machine instruction consists of two parts: the op-code (short for operation code) field and the operand field.
- The bit pattern of a machine instruction
  - The bit pattern appearing in the op-code field indicates which of the elementary operations, such as STORE, SHIFT, XOR, and JUMP, is requested by the instruction.
  - The bit patterns found in the operand field provide more detailed information about the operation specified by the op-code.
  - Each of these instructions is encoded using a total of 16 bits, represented by four hexadecimal digits.

An Illustrative Machine (Appendix C): Instructions

Figure 2.5 The composition of an instruction for the machine in Appendix C



An Illustrative Machine (Appendix C): Instructions

- Machine instructions
  - Each machine instruction is two bytes long.
  - The first 4 bits provide the op-code; the last 12 bits make up the operand field.
  - The table that follows lists the instructions in hexadecimal notation together with a short description of each.
  - The letters R, S, and T are used in place of hexadecimal digits in those fields representing a register identifier that varies depending on the particular application of the instruction.
  - The letters X and Y are used in lieu of hexadecimal digits in variable fields not representing a register.

An Illustrative Machine (Appendix C): Instructions

Op-code	Operand	Description
1	RXY	LOAD the register R with the bit pattern found in the memory cell whose address is XY. *Example:* 14A3 would cause the contents of the memory cell located at address A3 to be placed in register 4.
2	RXY	LOAD the register R with the bit pattern XY. *Example:* 20A3 would cause the value A3 to be placed in register 0.

An Illustrative Machine (Appendix C): Instructions

3	RXY	STORE the bit pattern found in register R in the memory cell whose address is XY. *Example:* 35B1 would cause the contents of register 5 to be placed in the memory cell whose address is B1.
4	0RS	MOVE the bit pattern found in register R to register S. *Example:* 40A4 would cause the contents of register A to be copied into register 4.
5	RST	ADD the bit patterns in registers S and T as though they were two's complement representations and leave the result in register R. *Example:* 5726 would cause the binary values in registers 2 and 6 to be added and the sum placed in register 7.
6	RST	ADD the bit patterns in registers S and T as though they represented values in floating-point notation and leave the floating-point result in register R. *Example:* 634E would cause the values in registers 4 and E to be added as floating-point values and the result to be placed in register 3.

An Illustrative Machine (Appendix C): Instructions

7	RST	OR the bit patterns in registers S and T and place the result in register R. *Example:* 7CB4 would cause the result of ORing the contents of registers B and 4 to be placed in register C.
8	RST	AND the bit patterns in registers S and T and place the result in register R. *Example:* 8045 would cause the result of ANDing the contents of registers 4 and 5 to be placed in register 0.
9	RST	EXCLUSIVE OR the bit patterns in registers S and T and place the result in register R. *Example:* 95F3 would cause the result of EXCLUSIVE ORing the contents of registers F and 3 to be placed in register 5.

An Illustrative Machine (Appendix C): Instructions

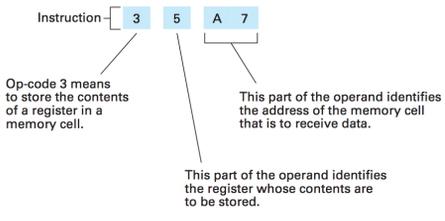
A	R0X	ROTATE the bit pattern in register R one bit to the right X times. Each time place the bit that started at the low-order end at the high-order end. *Example:* A403 would cause the contents of register 4 to be rotated 3 bits to the right in a circular fashion.
B	RXY	JUMP to the instruction located in the memory cell at address XY if the bit pattern in register R is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution. (The jump is implemented by copying XY into the program counter during the execute phase.) *Example:* B43C would first compare the contents of register 4 with the contents of register 0. If the two were equal, the pattern 3C would be placed in the program counter so that the next instruction executed would be the one located at that memory address. Otherwise, nothing would be done and program execution would continue in its normal sequence.
C	000	HALT execution. *Example:* C000 would cause program execution to stop.

An Illustrative Machine (Appendix C): Decoding Instructions

- **Decoding the operand field of an instruction**
  - **The operand field of each instruction consists of three hexadecimal digits (12 bits), and in each case clarifies the general instruction given by the op-code.**
- **Example (Figure 2.6)**
  - **If the first hexadecimal digit of an instruction were 3 (the op-code for storing the contents of a register), the next hexadecimal digit of the instruction would indicate which register is to be stored, and the last two hexadecimal digits would indicate which memory cell is to receive the data.**
  - **Thus the instruction 0011010110100111 (bit pattern) / 35A7 (hexadecimal) translates to the statement "STORE the bit pattern found in register 5 in the memory cell whose address is A7."**

An Illustrative Machine (Appendix C): Decoding Instructions

Figure 2.6 Decoding the instruction 35A7



An Example: Adding Values Stored in Memory

Figure 2.2 Adding values stored in memory

Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

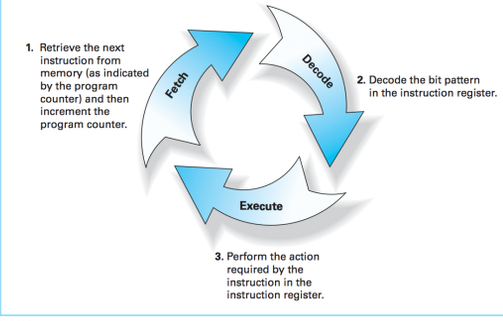
An Encoded Version of the Instructions in Figure 2.2

Figure 2.7 An encoded version of the instructions in Figure 2.2

Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.

The Fetch–Execute Cycle

Figure 2.8 The machine cycle

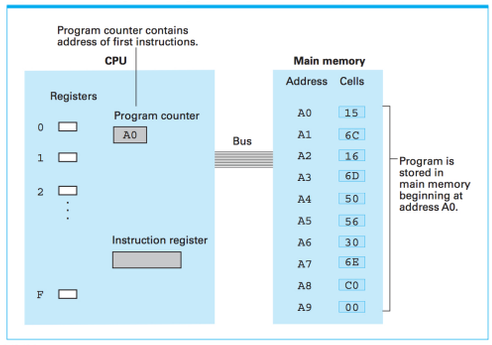


An Example of Program Execution

- **Example program (Figure 2.7)**
  - The program gets two values from main memory, computes their sum, and stores that total in a main memory cell.
- **Putting program in memory**
  - We first need to put the program somewhere in memory. We suppose the program is stored in consecutive addresses, starting at address A0 (hexadecimal).
  - With the program stored in this manner, we can cause the machine to execute it by placing the address (A0) of the first instruction in the PC and starting the machine.
- **Putting data in memory**
  - We also need to put data in memory cells 6C and 6D.

An Example of Program Execution

Figure 2.10 The program from Figure 2.7 stored in main memory ready for execution

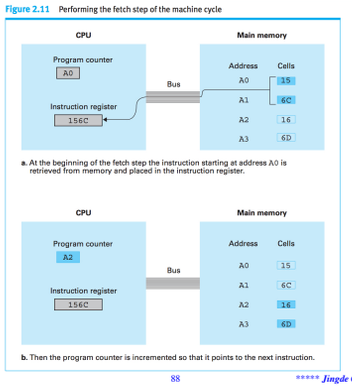


An Example of Program Execution: The 1st Cycle

- **Performing the fetch step of the first machine cycle**
  - The CPU begins the fetch step of the machine cycle by extracting the instruction stored in main memory at location A0 and placing this instruction (156C) in its IR.
  - Thus the entire instruction to be fetched occupies the memory cells at both address A0 and A1.
  - The CPU is designed to take this into account so it retrieves the contents of both cells and places the bit patterns received in the instruction register, which is 16 bits long.
  - The CPU then adds two to the PC so that this register contains the address of the next instruction.

An Example of Program Execution: The 1st Cycle

Figure 2.11 Performing the fetch step of the machine cycle

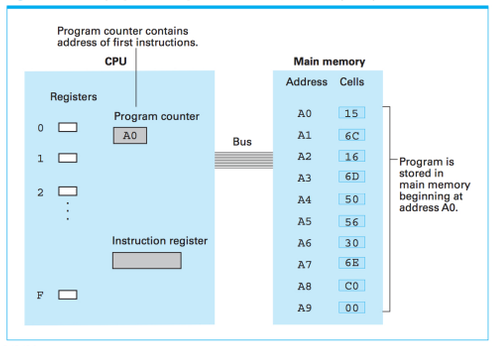


An Example of Program Execution: The 1st Cycle

- **Performing the decoding step of the first machine cycle**
  - The CPU analyzes the instruction in its instruction register and concludes that it is to load register 5 with the contents of the memory cell at address 6C.
- **Performing the executing step of the first machine cycle**
  - The load activity is performed during the execution step of the machine cycle, and the CPU then begins the next cycle.

An Example of Program Execution

Figure 2.10 The program from Figure 2.7 stored in main memory ready for execution



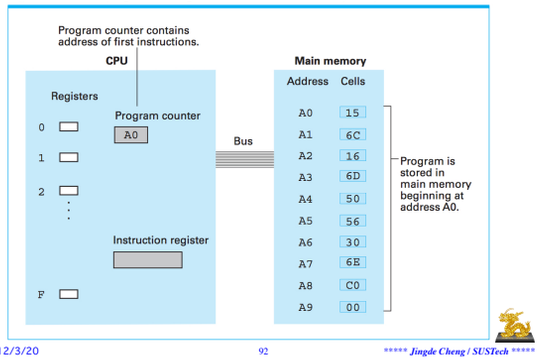
An Example of Program Execution: The 2nd Cycle

♣ **Performing the second machine cycle**

- The second cycle begins by fetching the instruction 166D from the two memory cells starting at address A2.
- The CPU places this instruction in the instruction register and increments the program counter to A4.
- After performing the fetch step of the second cycle, the values in the PC and IR therefore become the following: PC: A4; IR: 166D
- Now the CPU decodes the instruction 166D and determines that it is to load register 6 with the contents of memory address 6D.
- The CPU then executes the instruction. It is at this time that register 6 is actually loaded.

An Example of Program Execution

Figure 2.10 The program from Figure 2.7 stored in main memory ready for execution



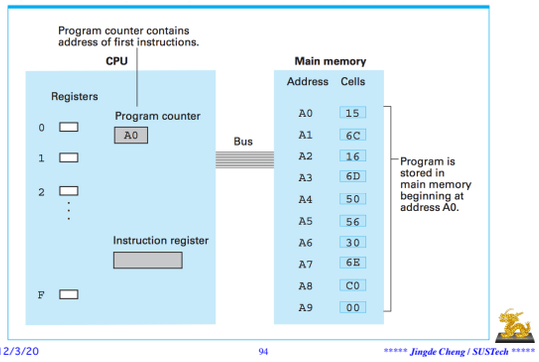
An Example of Program Execution: The 3rd Cycle

♣ **Performing the third machine cycle**

- Since the PC now contains A4, the CPU extracts the next instruction starting at this address.
- The result is that 5056 is placed in the IR, and the PC is incremented to A6.
- The CPU now decodes the contents of its IR and executes it by activating the two's complement addition circuitry with inputs being registers 5 and 6.
- During this execution step, the arithmetic/logic unit performs the requested addition, leaves the result in register 0 (as requested by the control unit), and reports to the control unit that it has finished.
- The CPU then begins the next machine cycle.

An Example of Program Execution

Figure 2.10 The program from Figure 2.7 stored in main memory ready for execution



An Example of Program Execution: The 4th and 5th Cycles

♣ **Performing the fourth machine cycle**

- The CPU fetches the next instruction (306E) from the two memory cells starting at memory location A6 and increments the PC to A8.
- This instruction is then decoded and executed.
- At this point, the sum is placed in memory location 6E.

♣ **Performing the fifth machine cycle**

- The next instruction is fetched starting from memory location A8, and the PC is incremented to AA.
- The contents of the IR (C000) are now decoded as the halt instruction.
- Consequently, the machine stops during the execute step of the machine cycle, and the program is completed.

An Example of Program Execution

3. Suppose the memory cells at addresses A4 to B1 in the machine described in Appendix C contain the (hexadecimal) bit patterns given in the following table:

Address	Contents
A4	20
A5	00
A6	21
A7	03
A8	22
A9	01
AA	B1
AB	B0
AC	50
AD	02
AE	B0
AF	AA
B0	C0
B1	00

When answering the following questions, assume that the machine is started with its program counter containing A4.

a. What is in register 0 the first time the instruction at address AA is executed?
b. What is in register 0 the second time the instruction at address AA is executed?
c. How many times is the instruction at address AA executed before the machine halts?

An Example of Program Execution: The 1st – 3rd Cycles

- Performing the 1st machine cycle
  - PC: A4, IR: 2000, PC: A6
  - LOAD the register 0 with the bit pattern 00
  - R0: 00
- Performing the 2nd machine cycle
  - PC: A6, IR: 2103, PC: A8
  - LOAD the register 1 with the bit pattern 03
  - R1: 03
- Performing the 3rd machine cycle
  - PC: A8, IR: 2201, PC: AA
  - LOAD the register 2 with the bit pattern 01
  - R2: 01

An Example of Program Execution

3. Suppose the memory cells at addresses A4 to B1 in the machine described in Appendix C contain the (hexadecimal) bit patterns given in the following table:

Address	Contents
A4	20
A5	00
A6	21
A7	03
A8	22
A9	01
AA	B1
AB	B0
AC	50
AD	02
AE	B0
AF	AA
B0	C0
B1	00

When answering the following questions, assume that the machine is started with its program counter containing A4.

a. What is in register 0 the first time the instruction at address AA is executed?
b. What is in register 0 the second time the instruction at address AA is executed?
c. How many times is the instruction at address AA executed before the machine halts?

An Example of Program Execution: The 4th – 5th Cycles

- Performing the 4th machine cycle
  - PC: AA, IR: B1B0, PC: AC
  - JUMP to the instruction located in the memory cell at address B0, if the bit pattern in register 1 is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution.
  - R0: 00, R1: 03, NOT JUMP
- Performing the 5th machine cycle
  - PC: AC, IR: 5002, PC: AE
  - ADD the bit patterns in registers 0 and 2 as though they were two's complement representations and leave the result in register 0
  - R0: 00, R2: 01, R0: 01

An Example of Program Execution

3. Suppose the memory cells at addresses A4 to B1 in the machine described in Appendix C contain the (hexadecimal) bit patterns given in the following table:

Address	Contents
A4	20
A5	00
A6	21
A7	03
A8	22
A9	01
AA	B1
AB	B0
AC	50
AD	02
AE	B0
AF	AA
B0	C0
B1	00

When answering the following questions, assume that the machine is started with its program counter containing A4.

a. What is in register 0 the first time the instruction at address AA is executed?
b. What is in register 0 the second time the instruction at address AA is executed?
c. How many times is the instruction at address AA executed before the machine halts?

An Example of Program Execution: The 6th – 7th Cycles

- Performing the 6th machine cycle
  - PC: AE, IR: B0AA, PC: B0
  - JUMP to the instruction located in the memory cell at address AA, if the bit pattern in register 0 is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution.
  - JUMP, PC: AA
- Performing the 7th machine cycle
  - PC: AA, IR: B1B0, PC: AC
  - JUMP to the instruction located in the memory cell at address B0, if the bit pattern in register 1 is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution.
  - R0: 01, R1: 03, NOT JUMP

An Example of Program Execution

3. Suppose the memory cells at addresses A4 to B1 in the machine described in Appendix C contain the (hexadecimal) bit patterns given in the following table:

Address	Contents
A4	20
A5	00
A6	21
A7	03
A8	22
A9	01
AA	B1
AB	B0
AC	50
AD	02
AE	B0
AF	AA
B0	C0
B1	00

When answering the following questions, assume that the machine is started with its program counter containing A4.

a. What is in register 0 the first time the instruction at address AA is executed?
b. What is in register 0 the second time the instruction at address AA is executed?
c. How many times is the instruction at address AA executed before the machine halts?

An Example of Program Execution: The 8th – 9th Cycles

- Performing the 8th machine cycle
  - PC: AC, IR: 5002, PC: AE
  - ADD the bit patterns in registers 0 and 2 as though they were two's complement representations and leave the result in register 0
  - R0: 01, R2: 01, R0: 02
- Performing the 9th machine cycle
  - PC: AE, IR: B0AA, PC: B0
  - JUMP to the instruction located in the memory cell at address AA, if the bit pattern in register 0 is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution.
  - JUMP, PC: AA

An Example of Program Execution

3. Suppose the memory cells at addresses A4 to B1 in the machine described in Appendix C contain the (hexadecimal) bit patterns given in the following table:

Address	Contents
A4	20
A5	00
A6	21
A7	03
A8	22
A9	01
AA	B1
AB	B0
AC	50
AD	02
AE	B0
AF	AA
B0	C0
B1	00

When answering the following questions, assume that the machine is started with its program counter containing A4.

a. What is in register 0 the first time the instruction at address AA is executed?
b. What is in register 0 the second time the instruction at address AA is executed?
c. How many times is the instruction at address AA executed before the machine halts?

An Example of Program Execution: The 10th – 11th Cycles

- Performing the 10th machine cycle
  - PC: AA, IR: B1B0, PC: AC
  - JUMP to the instruction located in the memory cell at address B0, if the bit pattern in register 1 is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution.
  - R0: 02, R1: 03, NOT JUMP
- Performing the 11th machine cycle
  - PC: AC, IR: 5002, PC: AE
  - ADD the bit patterns in registers 0 and 2 as though they were two's complement representations and leave the result in register 0
  - R0: 02, R2: 01, R0: 03

An Example of Program Execution

3. Suppose the memory cells at addresses A4 to B1 in the machine described in Appendix C contain the (hexadecimal) bit patterns given in the following table:

Address	Contents
A4	20
A5	00
A6	21
A7	03
A8	22
A9	01
AA	B1
AB	B0
AC	50
AD	02
AE	B0
AF	AA
B0	C0
B1	00

When answering the following questions, assume that the machine is started with its program counter containing A4.

a. What is in register 0 the first time the instruction at address AA is executed?
b. What is in register 0 the second time the instruction at address AA is executed?
c. How many times is the instruction at address AA executed before the machine halts?

An Example of Program Execution: The 12th – 13th Cycles

- Performing the 12th machine cycle
  - PC: AE, IR: B0AA, PC: B0
  - JUMP to the instruction located in the memory cell at address AA, if the bit pattern in register 0 is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution.
  - JUMP, PC: AA
- Performing the 13th machine cycle
  - PC: AA, IR: B1B0, PC: AC
  - JUMP to the instruction located in the memory cell at address B0, if the bit pattern in register 1 is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution.
  - R0: 03, R1: 03, JUMP, PC: B0

An Example of Program Execution

3. Suppose the memory cells at addresses A4 to B1 in the machine described in Appendix C contain the (hexadecimal) bit patterns given in the following table:

Address	Contents
A4	20
A5	00
A6	21
A7	03
A8	22
A9	01
AA	B1
AB	B0
AC	50
AD	02
AE	B0
AF	AA
B0	C0
B1	00

When answering the following questions, assume that the machine is started with its program counter containing A4.

a. What is in register 0 the first time the instruction at address AA is executed?
b. What is in register 0 the second time the instruction at address AA is executed?
c. How many times is the instruction at address AA executed before the machine halts?

An Example of Program Execution: The 14th Cycle

- Performing the 14th machine cycle
  - PC: B0 , IR: C000, PC: B2
  - STOP
- Answers to the questions
  - a. What is in register 0 the first time the instruction at address AA is executed?
    Answer: R0: 00
  - b. What is in register 0 the second time the instruction at address AA is executed?
    Answer: R0: 01
  - c. How many times is the instruction at address AA executed before the machine halts?
    Answer: 4

Arithmetic/Logic Instructions: Logic Operations

- Logic operations on bit strings
  - Those logic operations AND, OR, and XOR (exclusive or) that combine two input bits to produce a single output bit can be extended to operations that combine two strings of bits to produce a single output string by applying the basic operation to individual columns.
- Examples of logic operations on bit strings

```
  10011010        10011010        10011010
AND 11001001    OR 11001001    XOR 11001001
  10001000        11011011        01010011
```

Arithmetic/Logic Instructions: AND Operations

- Masking: an example

```
  00001111
AND 10101010
  00001010
```

  - Without knowing the contents of the second operand, we still can conclude that the four most significant bits of the result will be 0s. Moreover, the four least significant bits of the result will be a copy of that part of the second operand.
- Masking
  - One of the major uses of the AND operation is for placing 0s in one part of a bit pattern while not disturbing the other part.
  - This use of the AND operation is an example of the process called ***masking***: one operand, called a ***mask***, determines which part of the other operand will affect the result.

Arithmetic/Logic Instructions: AND Operations

- Using AND operation in manipulating a bit map
  - A ***bit map*** is a string of bits in which each bit represents the presence or absence of a particular object.
  - Suppose that the 8 bits in a memory cell are being used as a bit map, and we want to find out whether the object associated with the third bit from the high-order end is present.
  - We merely need to AND the entire byte with the mask 00100000, which produces a byte of all 0s if and only if the third bit from the high-order end of the bit map is itself 0.
  - Moreover, if the third bit from the high-order end of the bit map is a 1, and we want to change it to a 0 without disturbing the other bits, we can AND the bit map with the mask 11011111 and then store the result in place of the original bit map.

Arithmetic/Logic Instructions: OR Operations

- Using OR operation to duplicate a part of a bit string
  - Where the AND operation can be used to duplicate a part of a bit string while placing 0s in the non-duplicated part, the OR operation can be used to duplicate a part of a bit string while putting 1s in the non-duplicated part.
  - For this we again use a mask, but this time we indicate the bit positions to be duplicated with 0s and use 1s to indicate the non-duplicated positions.
  - For example, ORing any byte with 11110000 produces a result with 1s in its most significant 4 bits while its remaining bits are a copy of the least significant 4 bits of the other operand.

```
  11110000
OR 10101010
  11111010
```

Arithmetic/Logic Instructions: XOR Operations

- Using XOR operation to duplicate a part of a bit string
  - A major use of the XOR operation is in forming the complement of a bit string.
  - XORing any byte with a mask of all 1s produces the complement of the byte.
  - For example, note the relationship between the second operand and the result in the following example.

```
  11111111
XOR 10101010
  01010101
```

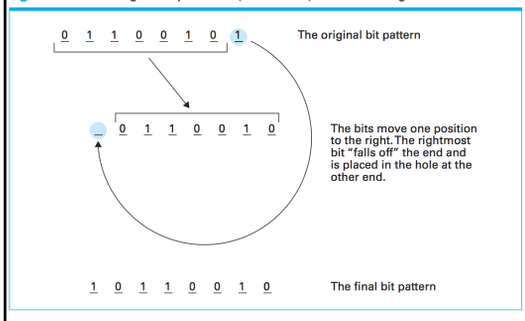
Arithmetic/Logic Instructions: Rotation and Shift Operations

♣ **Rotation and shift operations**

- The operations in the class of rotation and shift operations provide a means for moving bits within a register and are often used in solving alignment problems.
- These operations are classified by the direction of motion (right or left) and whether the process is circular.
- Consider a register containing a byte of bits. If we shift its contents 1 bit to the right, we imagine the rightmost bit falling off the edge and a hole appearing at the leftmost end.
- What happens with this extra bit and the hole is the distinguishing feature among the various shift operations.

Arithmetic/Logic Instructions: Rotation and Shift Operations

Figure 2.12 Rotating the bit pattern 65 (hexadecimal) one bit to the right



Arithmetic/Logic Instructions: Rotation and Shift Operations

♣ **Rotation (circular shift)**

- One technique is to place the bit that fell off the right end in the hole at the left end. The result is a *circular shift*, also called a *rotation*.
- Thus, if we perform a right circular shift on a byte-size bit pattern eight times, we obtain the same bit pattern we started with.

Arithmetic/Logic Instructions: Rotation and Shift Operations

♣ **Logical shift**

- Another technique is to discard the bit that falls off the edge and always fill the hole with a 0.
- The term *logical shift* is often used to refer to these operations.
- Such shifts to the left can be used for multiplying two's complement representations by two.
- After all, shifting binary digits to the left corresponds to multiplication by two, just as a similar shift of decimal digits corresponds to multiplication by ten.

Arithmetic/Logic Instructions: Rotation and Shift Operations

♣ **Arithmetic shift**

- Moreover, division by two can be accomplished by shifting the binary string to the right.
- In either shift, care must be taken to preserve the sign bit when using certain notational systems.
- Thus, we often find right shifts that always fill the hole (which occurs at the sign bit position) with its original value.
- Shifts that leave the sign bit unchanged are sometimes called *arithmetic shifts*.

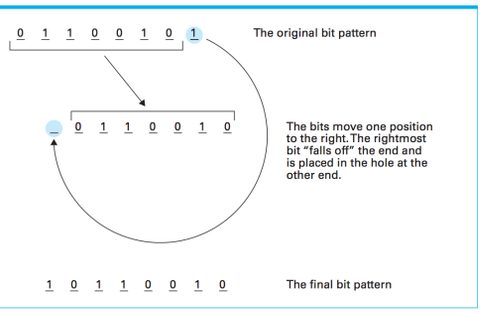
Arithmetic/Logic Instructions: Rotation and Shift Operations

♣ **Right circular shift**

- The machine language described in Appendix C contains only a right circular shift, designated by op-code A.
- In this case the first hexadecimal digit in the operand specifies the register to be rotated, and the rest of the operand specifies the number of bits to be rotated.
- Thus the instruction A501 means "Rotate the contents of register 5 to the right by 1 bit."
- In particular, if register 5 originally contained the bit pattern 65 (hexadecimal), then it would contain B2 after this instruction is executed (Figure 2.12).

Arithmetic/Logic Instructions: Rotation and Shift Operations

Figure 2.12 Rotating the bit pattern 65 (hexadecimal) one bit to the right

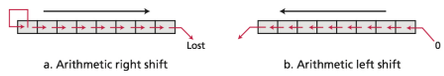


Shift Operations [F-CS-18]

Figure 4.3 *Simple shift operations*

Figure 4.4 *Circular shift operations*

Figure 4.5 *Arithmetic shift operations*



Arithmetic/Logic Instructions: Arithmetic Operations

♣ **Subtraction, multiplication, and division**

- Subtraction can be simulated by means of addition and negation.
- Multiplication is merely repeated addition.
- Division is repeated subtraction.
- Some small CPUs are designed with only the add or perhaps only the add and subtract instructions.

♣ **Notes**

- Numerous variations exist for each arithmetic operation.

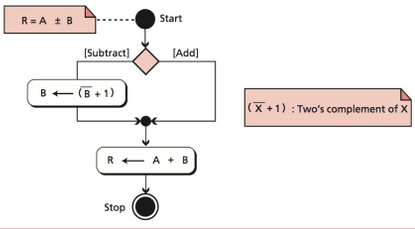
Arithmetic/Logic Instructions: Arithmetic Operations

♣ **Different addition**

- If the values to be added are stored in two's complement notation, the addition process must be performed as a straightforward column by column addition.
- If the operands are stored as floating-point values, the addition process must extract the mantissa of each, shift them right or left according to the exponent fields, check the sign bits, perform the addition, and translate the result into floating-point notation.
- Thus, although both operations are considered addition, the action of the machine is not the same.

Arithmetic Operations [F-CS-18]

Figure 4.6 *Addition and subtraction of integers in two's complement format*



The procedure is as follows:

1. If the operation is subtraction, we take the two's complement of the second integer. Otherwise, we move to the next step.
2. We add the two integers.

An Introduction to Computer Science

- Computer Science: What Is It and Why Study It?
- Computation: What Is It and Why Study It?
- Computability
- Computational Complexity (CS101A class only)
- Algorithms
- Data, Information, and Knowledge, and Their Representations
- Data Storage
- Computer Architecture
- Data Manipulation in Computer Systems
- Programming Languages and Compilers
- Operating Systems
- System Software and Application Software
- Software Engineering (CS101A class only)
- Knowledge Engineering and Artificial Intelligence (CS101A class only)
- Information Security Engineering (CS101A class only)