

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)



12/3/20

2

***** Jingde Cheng / SUSTech *****

The Question: How to Automate Computing?

How to automate computing ?
(in the electronic digital way)

12/3/20

3

***** Jingde Cheng / SUSTech *****



The Question: What Is a Computer ?

What is a (electronic digital) computer ?

4

***** Jingde Cheng / SUSTech *****



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.”

12/3/20

5

***** Jingde Cheng / SUSTech *****



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

❖ An important fact 冯·诺依曼结构不是冯·诺依曼发明的

- ◆ The “von Neumann” architecture was NOT invented by von Neumann !

```

graph LR
    Input[Input device] --> CPU[Central processing unit<br/>Control unit<br/>Arithmetic/logic unit]
    CPU --> Memory[Memory unit]
    Memory <--> IO[Output device]
    
```

FIGURE 5.1 The von Neumann architecture.

6

***** Jingde Cheng / SUSTech *****

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.

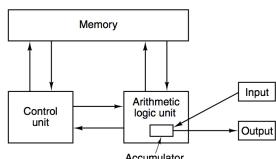


Figure 1-5. The original von Neumann machine.

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

12/3/20

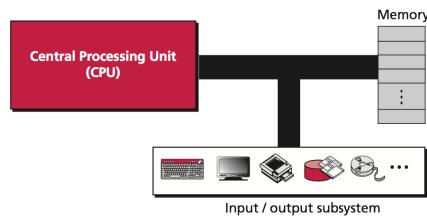
7

***** Jingde Cheng / SUSTech *****



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

Figure 5.1 Computer hardware (subsystems)



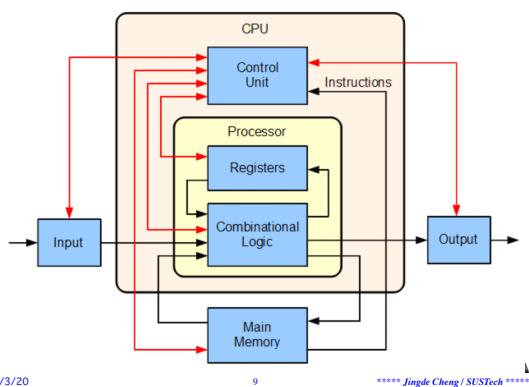
12/3/20

8

***** Jingde Cheng / SUSTech *****



Computer Architecture: What Is It? [Wikipedia]



12/3/20

9

***** Jingde Cheng / SUSTech *****

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.

12/3/20 将程序存在主存中 10 ***** Jingde Cheng / SUSTech *****



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.

操作单元与存储单元是分开的

12/3/20

11

***** Jingde Cheng / SUSTech *****

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

12/3/20

12

***** Jingde Cheng / SUSTech *****



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).

12/3/20

13

***** Jingde Cheng / SUSTech *****



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.

12/3/20

14

***** Jingde Cheng / SUSTech *****



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.”



12/3/20

15

***** Jingde Cheng / SUSTech *****

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。」



12/3/20

16

***** Jingde Cheng / SUSTech *****

Facts about “von Neumann” Architecture

✿ N. Metropolis and J. Worton’ 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”.



12/3/20

17

***** Jingde Cheng / SUSTech *****

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.



12/3/20

18

***** Jingde Cheng / SUSTech *****

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

John Vincent Atanasoff

John Vincent Atanasoff was born in Hamblinville, New York, on June 23, 1903. When he was 12 years old, his father bought a new slide rule. After Atanasoff learned how it worked, he became more interested in the mathematics involved in the slide rule's operation. His mother placed him in a local school, and he later studied at his father's old college, Alfred University, in Alfred, New York. He earned a master's degree and graduated from Alfred with a degree in electrical engineering. In 1925, he received a Ph.D. in theoretical physics, returning to Iowa State College.

Dr. Atanasoff became interested in finding a method to solve systems of linear equations, and he and his graduate students were doing. He examined computational devices in existence at that time and decided that the vacuum tube circuitry available at that time could not handle such calculations. He concluded that these machines were too slow and expensive. He also concluded that the vacuum tubes used had not yet been invented. He then suggested that if he could find a way to make the machine faster, it would be electronically operated and would compute by direct logical action rather than numerical action. This was the beginning of what would become known as digital computing, rather than decimal numbers, condensers for memory, and binary logic, rather than decimal logic and knowledge of power.

In 1939, with a \$500 grant from the school and a new assistant named Cuthbert E. Berry, Dr. Atanasoff began work on the first prototype of the Atanasoff-Berry Computer in the basement of the physics building. The first working prototype was demonstrated in 1942.

In 1941, John Mauchly, a physicist at Ursinus College where Dr. Atanasoff had met at a conference,

came to Iowa State to visit the Atanasoff group. After viewing the computer, he asked for an ABC machine. After extensive discussions, Mauchly left with papers describing the Atanasoff-Berry Computer. He then continued their work on a computation device at the Moore School of Electrical Engineering at the University of Pennsylvania. Their machine, the ENIAC, was completed in 1946.

Dr. Atanasoff went to Washington, D.C., to become director of the Underwater Acoustics Program at the Naval Ordnance Laboratory, leaving the patent application for the Atanasoff-Berry Computer with the Iowa State attorney. The patent application was filed in 1947, but it was not issued until 1973, without either Atanasoff or Berry being notified. After the war, Dr. Atanasoff was chief scientist for the Army Signal Corps, and he then became director of the Naval Ordnance Laboratory.

In 1950, he became director of the Ordnance Engineering Corporation, a research and engineering company in Falls Church, Virginia. He remained there until he retired in 1967.

In 1962, Maurice Ecken applied the patent on their ENIAC computer. Sperry Rand bought the patent and began to sue anyone who began to copy royalties. However, he declined to pay Sperry Rand brought suit. The subsequent trial took place in 1967, and the court ruled that 110 pages of testimony from the testimony of 77 witnesses, plus the patent application, showed that Atanasoff and Ecken "did not themselves first invent the electronic digital computer, but instead derived it from the work of others."

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

The Atanasoff-Berry Computer

<img alt="A detailed technical diagram of the Atanasoff-Berry Computer, showing its internal components and circuitry. Labels include: board 1/1024, board 4/1024, board 8/1024, board 16/1024, board 32/1024, board 64/1024, board 128/1024, board 256/1024, board 512/1024, board 1024/1024, board 2048/1024, board 4096/1024, board 8192/1024, board 16384/1024, board 32768/1024, board 65536/1024, board 131072/1024, board 262144/1024, board 524288/1024, board 1048576/1024, board 2097152/1024, board 4194304/1024, board 8388608/1024, board 16777216/1024, board 33554432/1024, board 67108864/1024, board 134217728/1024, board 268435456/1024, board 536870912/1024, board 1073741824/1024, board 2147483648/1024, board 4294967296/1024, board 8589934592/1024, board 17179869184/1024, board 34359738368/1024, board 68719476736/1024, board 137438953472/1024, board 274877906944/1024, board 549755813888/1024, board 1099511627776/1024, board 2199023255552/1024, board 4398046511104/1024, board 8796093022208/1024, board 17592186044416/1024, board 35184372088832/1024, board 70368744177664/1024, board 140737488355328/1024, board 281474976710656/1024, board 562949953421312/1024, board 1125899906842624/1024, board 2251799813685248/1024, board 4503599627370496/1024, board 9007199254740992/1024, board 18014398509481984/1024, board 36028797018963968/1024, board 72057594037927936/1024, board 144115188075859680/1024, board 288230376151719360/1024, board 576460752303438720/1024, board 1152921504606877440/1024, board 2305843009213754880/1024, board 4611686018427509760/1024, board 9223372036855019520/1024, board 18446744073710039040/1024, board 36893488147420078080/1024, board 73786976294840156160/1024, board 147573952589680312320/1024, board 295147905179360624640/1024, board 590295810358721249280/1024, board 1180591620717442495680/1024, board 2361183241434884991360/1024, board 4722366482869769982720/1024, board 9444732965739539965440/1024, board 18889465931479079930880/1024, board 37778931862958159861760/1024, board 75557863725916319723520/1024, board 151115727458232639467040/1024, board 302231454916465278934080/1024, board 604462909832930557868160/1024, board 1208925819665851115736320/1024, board 2417851639331702231472640/1024, board 4835703278663404462945280/1024, board 9671406557326808925890560/1024, board 19342813114653617851781120/1024, board 38685626229307235703562240/1024, board 77371252458614471407124480/1024, board 15474250491722894281428960/1024, board 30948500983445788562857920/1024, board 61897001966891577125715840/1024, board 12379400393378355425143680/1024, board 24758800786756710850287360/1024, board 49517601573513421700574720/1024, board 99035203147026843401149440/1024, board 198070406294053686802298880/1024, board 39614081258810737360459760/1024, board 79228162517621474720919520/1024, board 15845632503524294944183840/1024, board 31691265007048589888367680/1024, board 63382530014097179776735360/1024, board 12676506002819435955347120/1024, board 25353012005638871910694240/1024, board 50706024011277743821388480/1024, board 101412048022555477642776960/1024, board 202824096045110955285553920/1024, board 405648192085221910571107840/1024, board 811296384170443821142215680/1024, board 1622592768340887642284431360/1024, board 3245185536681775284568862720/1024, board 6490371073363550569137725440/1024, board 12980742146727101138275450880/1024, board 25961484293454202276550901760/1024, board 51922968586908404553101803520/1024, board 103845937173816809106203607040/1024, board 207691874347633618212407214080/1024, board 415383748695267236424814428160/1024, board 830767497390534472849628856320/1024, board 1661534994781068945699257712640/1024, board 3323069989562137891398515425280/1024, board 6646139979124275782797030850560/1024, board 13292279958248551565594061701120/1024, board 26584559916497103131188123402240/1024, board 53169119832994206262376246804480/1024, board 106338239665984412524732493608960/1024, board 212676479331968825049464987217920/1024, board 425352958663937650098929974435840/1024, board 850705917327875300197859948871680/1024, board 1701411834655750600395799897743360/1024, board 3402823669311501200791599795486720/1024, board 6805647338623002401583199590973440/1024, board 13611294677246004803166399181946880/1024, board 27222589354492009606332798363893760/1024, board 54445178708984019212665596727787520/1024, board 10889035741796803842533193345557040/1024, board 21778071483593607685066386691114080/1024, board 43556142967187215370132773382228160/1024, board 87112285934374430740265546764456320/1024, board 174224571868748861480531093528912640/1024, board 348449143737497722961062187057825280/1024, board 696898287474995445922124374115650560/1024, board 1393796574949988891844247458231301120/1024, board 2787593149899977783688494916462602240/1024, board 5575186299799955567376989832925204480/1024, board 1115037259959977113475397966585108960/1024, board 2230074519919954226950795933170217920/1024, board 4460149039839908453901591866340435840/1024, board 8920298079679816907803183732680871680/1024, board 17840596159359633815606367465361733360/1024, board 35681192318719267631212734930723466720/1024, board 71362384637438535262425469861446933440/1024, board 14272476927467707052485933972289866880/1024, board 28544953854935414104971867944579733760/1024, board 57089857709870828209943735889159467520/1024, board 11417971541974165641987467177831893040/1024, board 22835943083948331283974934355663786080/1024, board 45671886167896662567949868711327572160/1024, board 91343772335793325135899737422655144320/1024, board 18268754467158665027179867484530288640/1024, board 36537508934317330054359734969060577280/1024, board 73075017868634660028679469938121154560/1024, board 146150035737269320173589349876242301120/1024, board 292300071474538640347178699752484602240/1024, board 584600142949077280694357399505969204480/1024, board 1169200285898154561388714990011938408960/1024, board 2338400571796309122777429980023876817920/1024, board 4676801143592618245554859960047753635840/1024, board 9353602287185236491109719920095507311680/1024, board 1870720457437047282221943984019101463360/1024, board 3741440914874094564443887968038202926720/1024, board 7482881829748189128887775936076405853440/1024, board 14965763659496378257755559672152811706880/1024, board 29931527318992756515511119344305623413760/1024, board 59863054637985513030555559688611246827520/1024, board 11972610927597102606111119377222493365040/1024, board 23945221855194205212222229354444986730080/1024, board 47890443710388410424444449678889773460160/1024, board 95780887420776820848888899357779546920320/1024, board 191561774841521641697777798755590933840640/1024, board 383123549683043283395555597511181867681280/1024, board 766247099366086566785555595022363733362560/1024, board 1532494198732173133571111949944727466725120/1024, board 3064988397464346267142222949889454933450240/1024, board 6129976794928692534284444949778909866900480/1024, board 12259953589857385068568889899557819733800960/1024, board 24519907179714770137137779899115639467601920/1024, board 49039814359429540274275559898231278935203840/1024, board 98079628718859080548551119896462557870407680/1024, board 196159257437718161097077798932925115708015360/1024, board 392318514875436322194155598915850231416030720/1024, board 784637029750872644388311198903700462832061440/1024, board 1569274059501745286776622988807400925664122880/1024, board 3138548119003490573553244988614801853328245760/1024, board 6277096238006981147106488988309603706656491360/1024, board 1255419247601396229421297798861920713311282720/1024, board 2510838495202792458842595598833841426622565440/1024, board 5021676989405584917685191198867682853245130880/1024, board 1004335397881116983537082219881536570649061760/1024, board 2008670795762233967074164419883073141298123520/1024, board 4017341591524467934148328819886146282596247040/1024, board 8034683183048935868296657619881292565192494080/1024, board 1606936636609787173659315239882585131384988160/1024, board 3213873273219574347318630479885170262769976320/1024, board 6427746546439148694637260959887540525539952640/1024, board 12855493092878293389274521959885081051079853280/1024, board 25710986185756586778549043959882562102159706560/1024, board 51421972371513173557098087959885124204319413120/1024, board 102843944743026347114196175959882548408638826240/1024, board 205687889486052694228392351959882596817477652480/1024, board 411375778972105388456784703959882593634955304960/1024, board 822751557944210776913569407959882597269110609920/1024, board 1645503115888421553831388815959882594538221219840/1024, board 3291006231776843107662777631959882599076442439680/1024, board 6582012463553686215325555263959882598152888879360/1024, board 1316402493106737243065111152795988259630577758720/1024, board 2632804986213474486130222205595988259261155517440/1024, board 5265609972426948972260444411195988259522311034880/1024, board 1053121994485389794452088882229598825904462207760/1024, board 2106243988970779588904177774459598825908924415520/1024, board 4212487977941559177808355558895988259178488831040/1024, board 8424975955883118355616711117895988259356977662080/1024, board 1684991991176623671233342223578598825971395532160/1024, board 3369983982353247342466684447157859882594278864320/1024, board 6739967964706494684933378894315785988259855732640/1024, board 13479935929412989369866757788631578598825971145280/1024, board 26959871858825978739733515577263157859882594290560/1024, board 53919743717651957479467031155526315785988259851120/1024, board 107839467433203914989334062311052631578598825970240/1024, board 2156789348664078299786681246221052631578598825940480/1024, board 43135786973281565995733624924421052631578598825980960/1024, board 86271573946563131991467249848841052631578598825961920/1024, board 172543147893126263982934996897681052631578598825923840/1024, board 3450862957862525279658689937953681052631578598825947680/1024, board 69017259157250505593173799759073681052631578598825995360/1024, board 1380345183445010110663479951901473681052631578598825990720/1024, board 27606899668900202213269599238029473681052631578598825991440/1024, board 552137993378004044265391984760589473681052631578598825992880/1024, board 11042759867560080853107839695211789473681052631578598825995760/1024, board 22085519735120161606215679390423589473681052631578598825991520/1024, board 44171039470240323212431358780847189473681052631578598825993040/1024, board 883420789404806464248627175616943789473681052631578598825996080/1024, board 176684157880961291296354351123388789473681052631578598825992160/1024, board 353368315761922582592708702246775789473681052631578598825994320/1024, board 7067366315238451651854174044935515789473681052631578598825998640/1024, board 141347326304769033037083580897110315789473681052631578598825997360/1024, board 2826946526095380660741671617942206315789473681052631578598825994720/1024, board 5653893052190761321483343235884412315789473681052631578598825999440/1024, board 11307786104381522628666864471768824315789473681052631578598825999880/1024, board 22615572208763045257333728943536482315789473681052631578598825999760/1024, board 45231144417526085514667457887072964315789473681052631578598825999520/1024, board 90462288835052170829334915774145928315789473681052631578598825999040/1024, board 180924577670104341586688311548290568315789473681052631578598825999080/1024, board 36184915534020868317337662309658116315789473681052631578598825999160/1024, board 723698310680417366346753246193162315789473681052631578598825999320/1024, board 14473966213608347329343649243863264315789473681052631578598825999640/1024, board 28947932427216694658687298487726528315789473681052631578598825999280/1024, board 578958648544333893173745857754530566315789473681052631578598825999560/1024, board 11579172908866778634749171550906711315789473681052631578598825999520/1024, board 231583458177335572694983431018134226315789473681052631578598825999040/1024, board 4631669163546711453899668620362644526315789473681052631578598825999080/1024, board 9263338327093422907799337240725289126315789473681052631578598825999160/1024, board 185266766541868458155986548145057824126315789473681052631578598825999320/1024, board 370533533083736917311973096289115648126315789473681052631578598825999640/1024, board 74106706616747383462394619257823128126315789473681052631578598825999280/1024, board 1482134132335947669247892385156565616315789473681052631578598825999560/1024, board 296426826467189533849578477031313232126315789473681052631578598825999520/1024, board 5928536529343790676991569540626264643126315789473681052631578598825999040/1024, board 1185707305868795345398311981252532988126315789473681052631578598825999080/1024, board 2371414611737590690796633962505065816315789473681052631578598825999160/1024, board 47428292234751813815933378450101316315789473681052631578598825999320/1024, board 9485658446950362763186675680202633

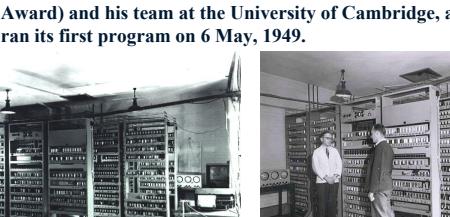
What is a Computer? (From the viewpoint of architecture)

- ◆ Three necessary requirements for computers **数字电子计算机**
- ◆ Program-control **程序控制**
- ◆ Internally-stored-program **内部存储程序**
- ◆ Changeable-program (Self-modifying code) **程序可改**
- ◆ The first computer **第一个符合以上的条件**
 - ◆ 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.



Jingde Cheng / SUSTech

Memory Unit [DL-CS-16]

- ♣ Memory unit **主存**
 - ♦ *Memory unit* is a collection of *cells*, each with a unique *physical address*.
- ♣ Addressability **可寻址的**
 - ♦ 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**

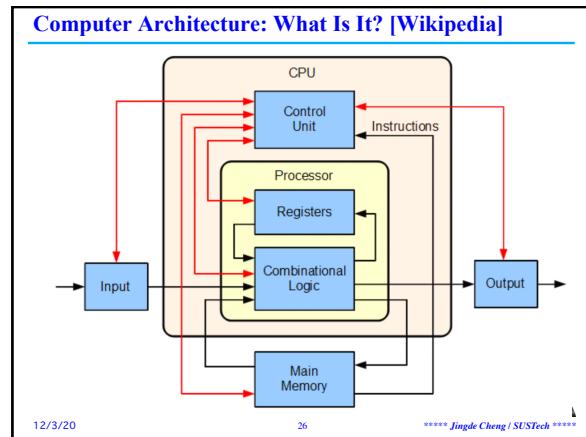
- Most modern CPUs have a small number of special storage units called **registers**.
- These registers contain **one word** and are used to store data that is needed again immediately.

General-purpose/special-purpose registers

- Some of the registers within the **register unit** are considered **general-purpose registers** (e.g., for all operations) whereas others are **special-purpose registers** (e.g., PC and IR)

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

12/3/20 25 ***** Jingde Cheng / SUSTech *****



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

12/3/20 27 ***** Jingde Cheng / SUSTech *****

IR: 当前正在执行的指令
(要执行→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**).

12/3/20 29 ***** Jingde Cheng / SUSTech *****

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

12/3/20 28 ***** Jingde Cheng / SUSTech *****

Control Unit [DL-CS-16]

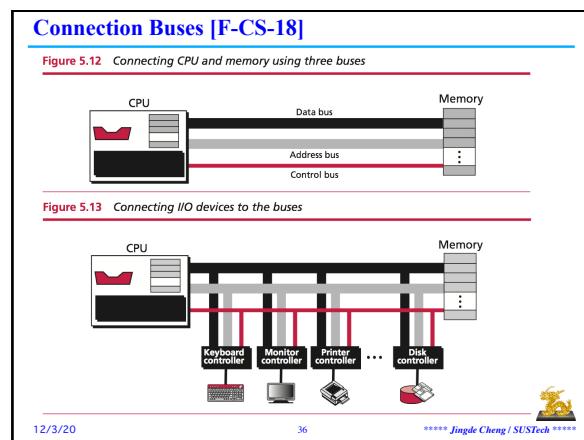
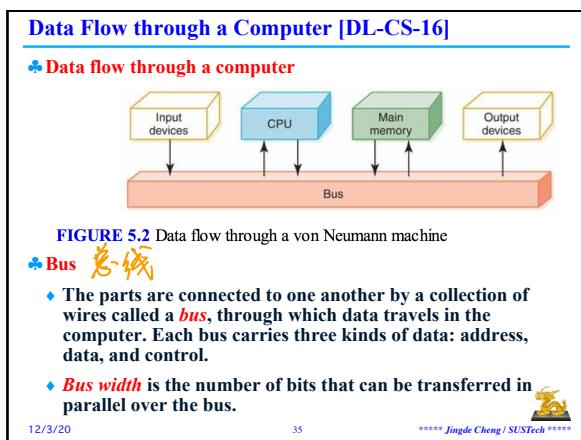
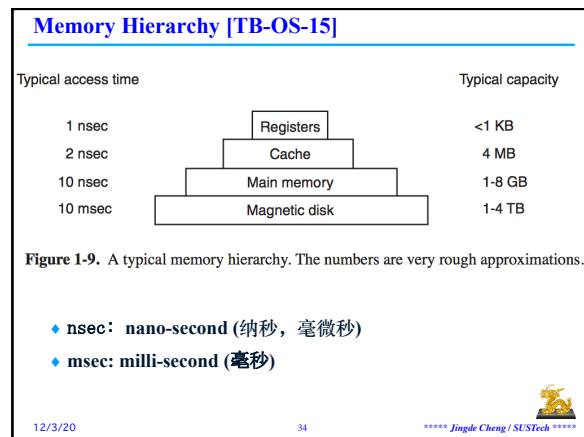
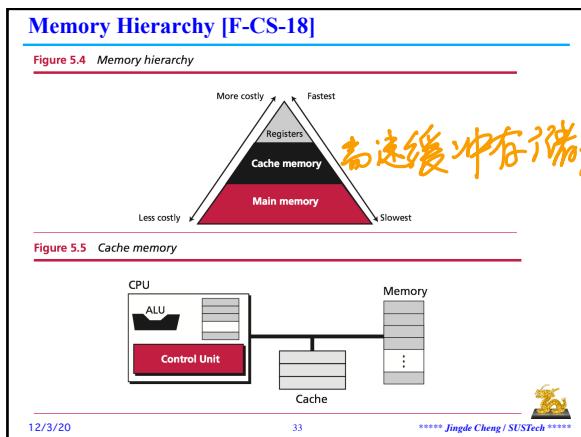
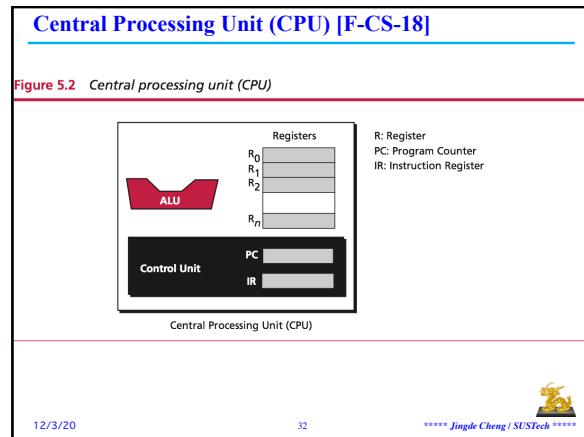
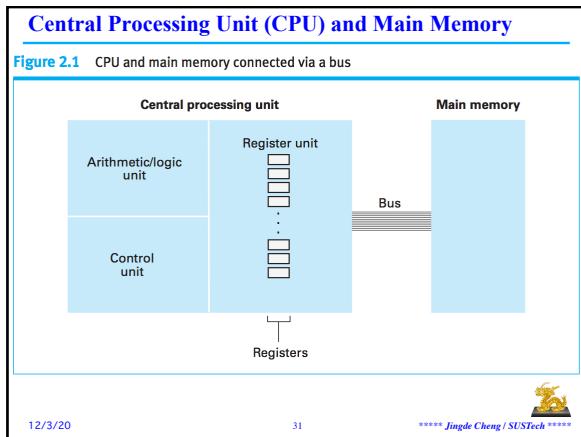
Control unit The computer component that controls the actions of the other components so as to execute instructions in sequence

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

12/3/20 30 ***** Jingde Cheng / SUSTech *****



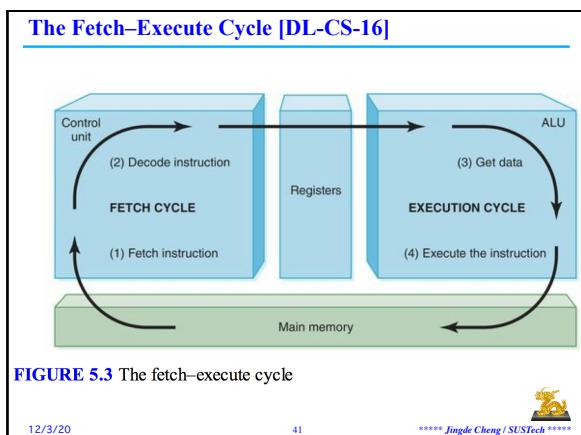
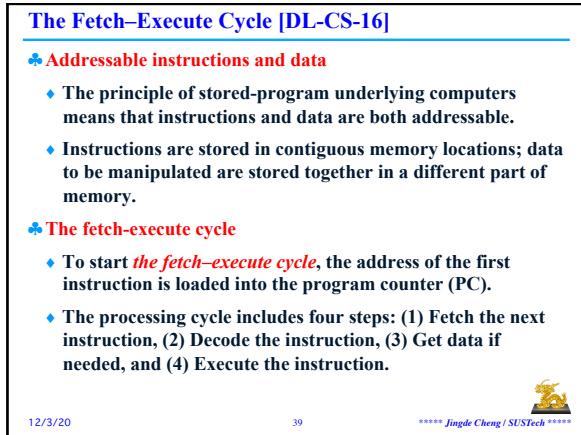
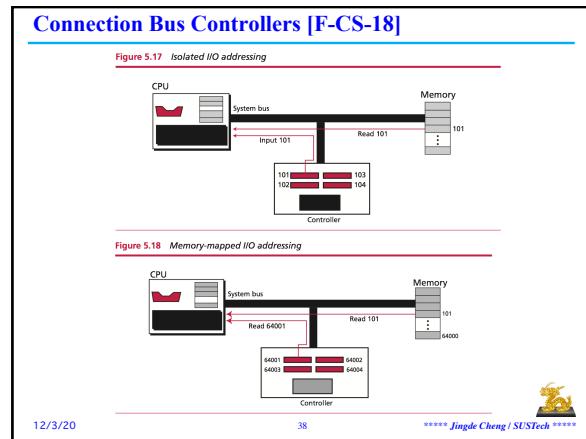
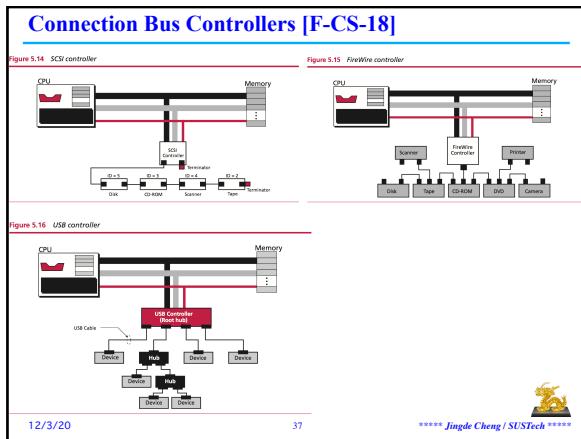
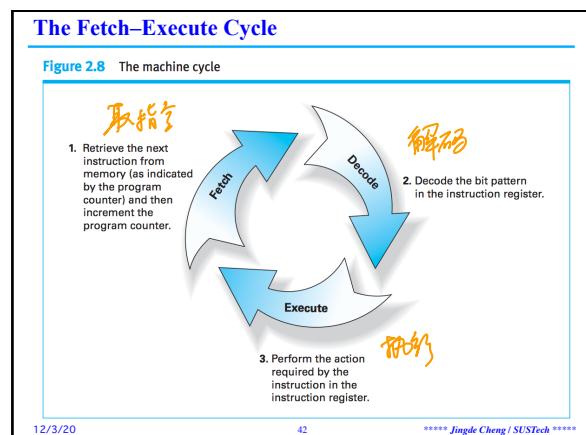
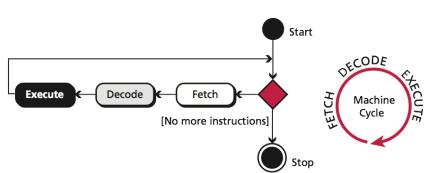


FIGURE 5.3 The fetch–execute cycle



The Fetch–Execute Cycle [F-CS-18]

Figure 5.19 The steps of a cycle



12/3/20

43

***** Jingde Cheng / SUSTech *****



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.

12/3/20

45

***** Jingde Cheng / SUSTech *****



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.

12/3/20

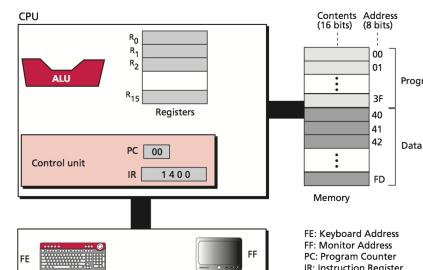
47

***** Jingde Cheng / SUSTech *****



Computer Architecture [F-CS-18]

Figure 5.30 The components of a simple computer



12/3/20

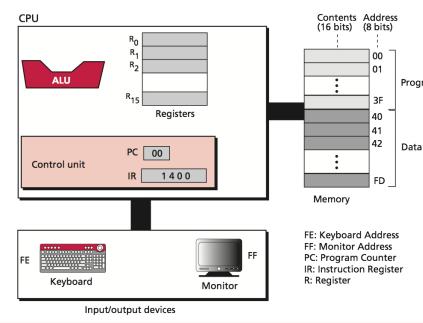
44

***** Jingde Cheng / SUSTech *****



Computer Architecture [F-CS-18]

Figure 5.30 The components of a simple computer



12/3/20

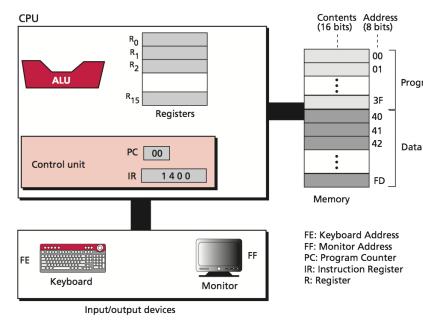
46

***** Jingde Cheng / SUSTech *****



Computer Architecture [F-CS-18]

Figure 5.30 The components of a simple computer



12/3/20

48

***** Jingde Cheng / SUSTech *****



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.

12/3/20

49

***** Jingde Cheng / SUSTech *****



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.

12/3/20

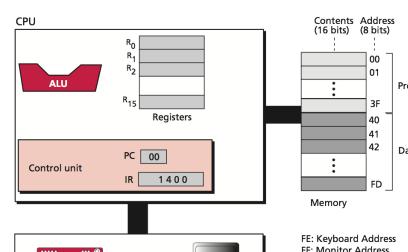
51

***** Jingde Cheng / SUSTech *****



Computer Architecture [F-CS-18]

Figure 5.30 The components of a simple computer



12/3/20

50

***** Jingde Cheng / SUSTech *****



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.

12/3/20

53

***** Jingde Cheng / SUSTech *****



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.

12/3/20

54

***** Jingde Cheng / SUSTech *****



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.

12/3/20 55 ***** Jingde Cheng / SUSTech *****

Machine Language *机器语言*

- 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

12/3/20 56 ***** Jingde Cheng / SUSTech *****

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)**.
 - The argument in favor of RISC architecture is that such a machine is efficient, fast, and less expensive to manufacture.

通过形而简单的命令逻辑
节省内存空间

12/3/20 57 ***** Jingde Cheng / SUSTech *****

CISC architecture vs. RISC architecture

- CISC architecture** *复杂指令集计算机*
 - 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

12/3/20 58 ***** Jingde Cheng / SUSTech *****

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

FIGURE 5.8 Processors in a synchronous computing environment

Processor 1 → Result 1 → Processor 2 → Result 2 → Processor 3 → Result 3 → ... → Processor N

12/3/20 59 ***** Jingde Cheng / SUSTech *****

Parallel Architectures [DL-CS-16]

FIGURE 5.10 A shared-memory parallel processor

12/3/20 60 ***** Jingde Cheng / SUSTech *****

Parallel Architectures [F-CS-18]

Figure 5.24 Pipelining

Figure 5.25 A taxonomy of computer organization

Organization	Description
SISD	Single Instruction-stream, Single Data-stream
SIMD	Single Instruction-stream, Multiple Data-stream
MISD	Multiple Instruction-stream, Single Data-stream
MIMD	Multiple Instruction-stream, Multiple Data-stream

12/3/20 61 ***** Jingde Cheng / SUSTech *****

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 floating-point calculating machine
1941	ENIAC	Moore Inst. govt	First American general-purpose computer
1944	Mark I	Aiken	
1945	ENIAC	Eckert/Mauchly	Modern computer history starts here
1945	EDSAC	Wilkes	First stored program computer
1946	Harvard Mark I	MIT	First real-time computer
1950	IAS	Von Neumann	Most current machines use this design
1951	DP-1	DEC	First minicomputer (50 words)
1951	1401	IBM	Extremely popular small business machine
1953	IBM 650	IBM	One of the most popular in the early 1960s
1956	85000	Burroughs	First machine designed for a high-level language
1961	360	IBM	First product line designed as a family
1961	6600	CDC	First scientific supercomputer
1961	POD-8	DEC	First mass-market minicomputer (80,000 sold)
1971	1802	DEC	One of the most popular in the 1970s
1971	4090	Intel	First general-purpose 8-bit computer on a chip
1971	CRAY-1	Cray	First vector supercomputer
1974	VAX	DEC	First 32-bit supercomputer
1975	PC	IBM	Start of modern personal computer era
1981	Doskone-1	Doskone	First portable computer
1981	Lisbon-1	Apple	First personal computer with a GUI
1985	386	Intel	First 32-bit ancestor of the Pentium line
1985	3270	IBM	First graphical RISC machine
1986	KC2004	Xilinx	First field-programmable gate array (FPGA)
1987	SPARC	Sun	First SPARC-based RISC workstation
1989	GridPad	Grid Systems	First commercial tablet computer
1991	PowerPC	IBM	First 32-bit parallel processor
1992	Alpha	DEC	First 64-bit parallel computer
1992	Simon	IBM	First smartphone
1993	Newton	Apple	First palm-top computer (PDA)
2001	POWER4	IBM	First dual-core chip multiprocessor

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

12/3/20 62 ***** Jingde Cheng / SUSTech *****

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)

12/3/20 63 ***** Jingde Cheng / SUSTech *****

The Question: How is Data Manipulated in a Computer?

How is data manipulated in a computer ?

(in the electronic digital way)

12/3/20 64 ***** Jingde Cheng / SUSTech *****

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**.

12/3/20 65 ***** Jingde Cheng / SUSTech *****

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**.

12/3/20 66 ***** Jingde Cheng / SUSTech *****

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.

12/3/20

67

***** Jingde Cheng / SUSTech *****



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**.

12/3/20

69

***** Jingde Cheng / SUSTech *****



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 to the left within the register.
- These operations are known as either **SHIFT** or **ROTATE** operations.

12/3/20

68

***** Jingde Cheng / SUSTech *****



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.

12/3/20

70

***** Jingde Cheng / SUSTech *****



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.

12/3/20

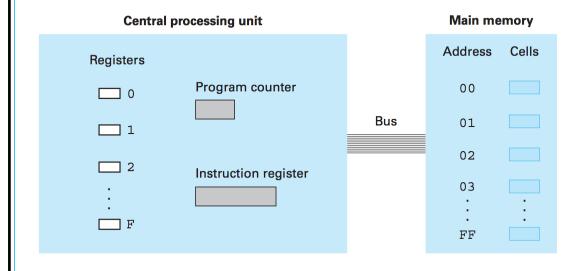
71

***** Jingde Cheng / SUSTech *****



An Illustrative Machine (Appendix C): Architecture

Figure 2.4 The architecture of the machine described in Appendix C



12/3/20

72

***** Jingde Cheng / SUSTech *****



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.



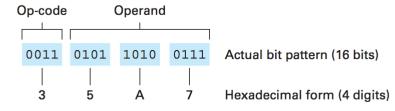
12/3/20

73

***** Jingde Cheng / SUSTech *****

An Illustrative Machine (Appendix C): Instructions

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



74

***** Jingde Cheng / SUSTech *****



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.



12/3/20

75

***** Jingde Cheng / SUSTech *****

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 A3. Example: 20A3 would cause the value A3 to be placed in register 0.

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.

76

***** Jingde Cheng / SUSTech *****



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.

12/3/20

77

***** Jingde Cheng / SUSTech *****

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.

12/3/20

78

***** Jingde Cheng / SUSTech *****



An Illustrative Machine (Appendix C): Instructions

A	ROX	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. <i>Example:</i> 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.) <i>Example:</i> 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. <i>Example:</i> C000 would cause program execution to stop.

12/3/20 79 ***** Jingde Cheng / SUSTech *****

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.”

12/3/20 80 ***** Jingde Cheng / SUSTech *****

An Illustrative Machine (Appendix C): Decoding Instructions

Figure 2.6 Decoding the instruction 35A7

12/3/20 81 ***** Jingde Cheng / SUSTech *****

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.

12/3/20 82 ***** Jingde Cheng / SUSTech *****

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.

12/3/20 83 ***** Jingde Cheng / SUSTech *****

The Fetch–Execute Cycle

Figure 2.8 The machine cycle

1. Retrieve the next instruction from memory (as indicated by the program counter) and then increment the program counter.
2. Decode the bit pattern in the instruction register.
3. Perform the action required by the instruction in the instruction register.

12/3/20 84 ***** Jingde Cheng / SUSTech *****

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.

12/3/20

85

***** Jingde Cheng / SUSTech *****



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.

12/3/20

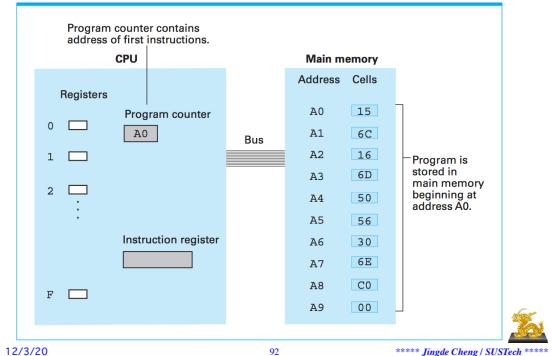
91

***** Jingde Cheng / SUSTech *****



An Example of Program Execution

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



12/3/20

92

***** Jingde Cheng / SUSTech *****



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.

12/3/20

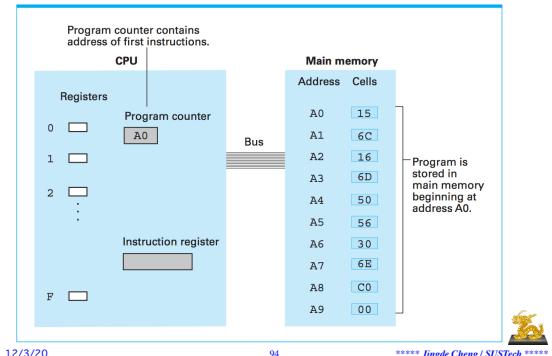
93

***** Jingde Cheng / SUSTech *****



An Example of Program Execution

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



12/3/20

94

***** Jingde Cheng / SUSTech *****



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.

12/3/20

95

***** Jingde Cheng / SUSTech *****



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
80	C0
B1	00

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

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

12/3/20

96

***** Jingde Cheng / SUSTech *****



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



12/3/20

97

***** Jingde Cheng / SUSTech *****

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	B0
AB	B0
AC	50
AD	02
AE	B0
AF	AA
BO	C0
B1	00

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

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



12/3/20

98

***** Jingde Cheng / SUSTech *****

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



12/3/20

99

***** Jingde Cheng / SUSTech *****

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
BO	C0
B1	00

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

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



12/3/20

100

***** Jingde Cheng / SUSTech *****

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



12/3/20

101

***** Jingde Cheng / SUSTech *****

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
BO	C0
B1	00

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

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



12/3/20

102

***** Jingde Cheng / SUSTech *****

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

12/3/20

103

***** Jingde Cheng / SUSTech *****



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
BO	C0
B1	00

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

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



12/3/20

104

***** Jingde Cheng / SUSTech *****

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

12/3/20

105

***** Jingde Cheng / SUSTech *****



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
BO	C0
B1	00

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

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



12/3/20

106

***** Jingde Cheng / SUSTech *****

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

12/3/20

107

***** Jingde Cheng / SUSTech *****



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
BO	C0
B1	00

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

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



12/3/20

108

***** Jingde Cheng / SUSTech *****

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



12/3/20

109

***** Jingde Cheng / SUSTech *****

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

$$\begin{array}{r} 10011010 \\ \text{AND } 11001001 \\ \hline 10001000 \end{array} \quad \begin{array}{r} 10011010 \\ \text{OR } 11001001 \\ \hline 11011011 \end{array} \quad \begin{array}{r} 10011010 \\ \text{XOR } 11001001 \\ \hline 01010011 \end{array}$$



12/3/20

110

***** Jingde Cheng / SUSTech *****

Arithmetic/Logic Instructions: AND Operations

✿ Masking: an example

$$\begin{array}{r} 00001111 \\ \text{AND } 10101010 \\ \hline 00001010 \end{array}$$

- ◆ 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.



12/3/20

111

***** Jingde Cheng / SUSTech *****

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.



12/3/20

112

***** Jingde Cheng / SUSTech *****

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.

$$\begin{array}{r} 11110000 \\ \text{OR } 10101010 \\ \hline 11111010 \end{array}$$



12/3/20

113

***** Jingde Cheng / SUSTech *****

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.

$$\begin{array}{r} 11111111 \\ \text{XOR } 10101010 \\ \hline 01010101 \end{array}$$



12/3/20

114

***** Jingde Cheng / SUSTech *****

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.

12/3/20

115

***** Jingde Cheng / SUSTech *****



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.

12/3/20

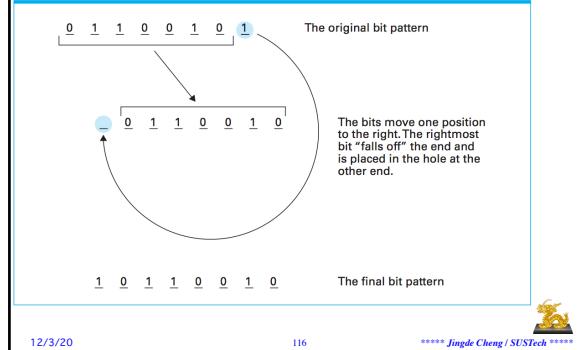
117

***** Jingde Cheng / SUSTech *****



Arithmetic/Logic Instructions: Rotation and Shift Operations

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



12/3/20

116

***** Jingde Cheng / SUSTech *****



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.

12/3/20

118

***** Jingde Cheng / SUSTech *****



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*.

12/3/20

119

***** Jingde Cheng / SUSTech *****



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).

12/3/20

120

***** Jingde Cheng / SUSTech *****



Arithmetic/Logic Instructions: Rotation and Shift Operations

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

The original bit pattern
The bits move one position to the right. The rightmost bit "falls off" the end and is placed in the hole at the other end.
The final bit pattern

12/3/20 121 ***** Jingde Cheng / SUSTech *****

Shift Operations [F-CS-18]

Figure 4.3 Simple shift operations

a. Simple right shift
b. Simple left shift

Figure 4.4 Circular shift operations

a. Circular right shift
b. Circular left shift

Figure 4.5 Arithmetic shift operations

a. Arithmetic right shift
b. Arithmetic left shift

12/3/20 122 ***** Jingde Cheng / SUSTech *****

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.

12/3/20 123 ***** Jingde Cheng / SUSTech *****

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.

12/3/20 124 ***** Jingde Cheng / SUSTech *****

Arithmetic Operations [F-CS-18]

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

R = A ± B
Start
[Subtract] [Add]
B ← (B + 1)
(X + 1) : Two's complement of X
R ← A + B
Stop

The procedure is as follows:

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

12/3/20 125 ***** Jingde Cheng / SUSTech *****

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)

12/3/20 126 ***** Jingde Cheng / SUSTech *****