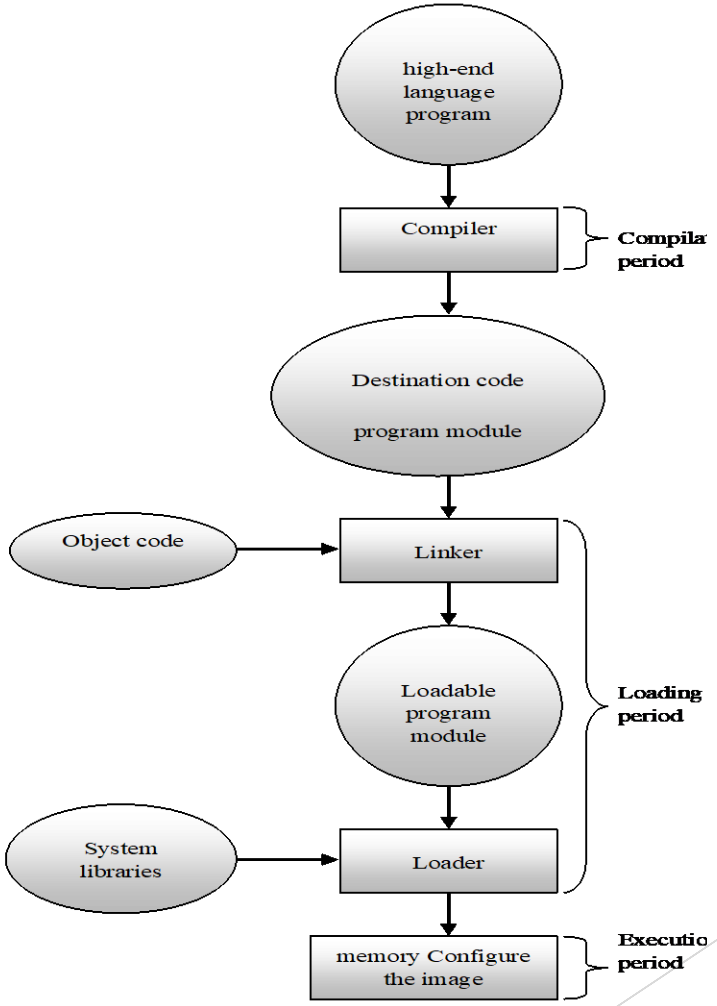
Chapter 3 Instruction Set Architecture

- 3.1 Program execution process and hierarchical structure
- 3.2 Execution of Commands
- 3.3 Format of Instructions
- 3.4 Data Addressing Mode
- 3.5 Design and consideration of instruction sets
- 3.6 Encoding and decoding of commands

Command set architecture

- Computers, like humans, need to provide a complete set of language for people to fully communicate with it to complete correct calculations.
- The most basic unit of computer language is "instruction", which is a collection of all the instructions of a computer, which is called the computer's instruction set.

3.1 Program execution flow architecture



The flowchart of the computer program execution and its hierarchy

3.1 Program execution flow and hierarchy

- Compilation period
- Loading period
- Execution period

3.1.1 Compilation period

- Compilation time, also known as compile time, is the first period experienced by a computer program from the "human" world to the "computer" world.
- The compiler uses the source code of a high-level language as its *input*, and the output is a assembly language program, or a further purpose program

（object program）。

3.1.1 Compilation period

- The destination program contains the following three records：
  - Header record: Describes the name, start address, and size of the program, usually assigned by the operating system.
  - Text record: The main body of the program, represented by an object code.
  - End record: Records the starting position when the program runs.

3.1.1 Compilation period

- Example of a destination program：

```
H^SWAP  ^002000^003FFF
T^002000^1E^10xxxx^...
T^00201E^1E^42xxxx^...
          ⋮
E^003FFF
```

Examples of destination program properties include header records, body records, and ending records.

3.1 Program execution flow and hierarchy

- Compilation period
- Loading period
- Execution period

3.1.2 Loading period

- Load time, also known as load time, includes:
  - linking
  - loading
- After the program is converted to object code, it cannot be regarded as an executable program. Because assembly language allows programmers to write specific complex actions into independent routines (subroutines), the corresponding routines can be called wherever these actions are needed in the program to complete the necessary operations.

3.1.2 Loading period

- During program execution, some lower-level actions need to be completed by the lower-level functions provided by the system, and the collection of these system functions is called the system library. Whether it is a routine written by a programmer or a system function, it is itself a small program, so after completing the assembly process, it will be translated into independent program modules. These compiled modules must go through linked actions.
- Give the relative address of the routine to the command calling the routine, so that the command calling the routine knows where to find the code of the routine.

3.1.2 Loading period

- The program responsible for this linking action is called the link editor（linkage editor）or link programs（linker）。

- The linked program is at best a load module that does not work immediately. It must be arranged in memory in a specific way before it can be handed over to the CPU for execution, this process is called loading, and the program responsible for loading is called a loader or loader（loader）。

3.1.2 Loading period

- Before loading memory, any unresolved compilation issues at compile time must be dealt with together:
  - Undefined tags(label).
  - Other error information.

- In order to reduce the need to re-perform assembly, address calculations, or add external references after each modification, the following records are required to provide information about the program re-addressing.
- modification record
    - Contains the command address that needs to be relocated and the length of the address that needs to be modified.

3.1.2 Loading period

- define record
  - Record information about external references for later linking actions
- refer record
  - Symbols used when recording external references.

```
H^SWAP   ^002000^003FFF
D^OSYM1  ^000030^OSYM2  ^0050
R^REF1   ^REF2   ^REF3   ^...
...
T^002000^1E^10xxxx^...
T^00201E^1E^42xxxx^...
...
M^000016^05^+REF1
M^000019^05^+REF2
...
E^003FFF
```

Contains the purpose process for correcting records, defining records, and referencing records

3.1 Program execution flow and hierarchy

- Compilation period
- Loading period
- Execution period

3.1.3 Execution period

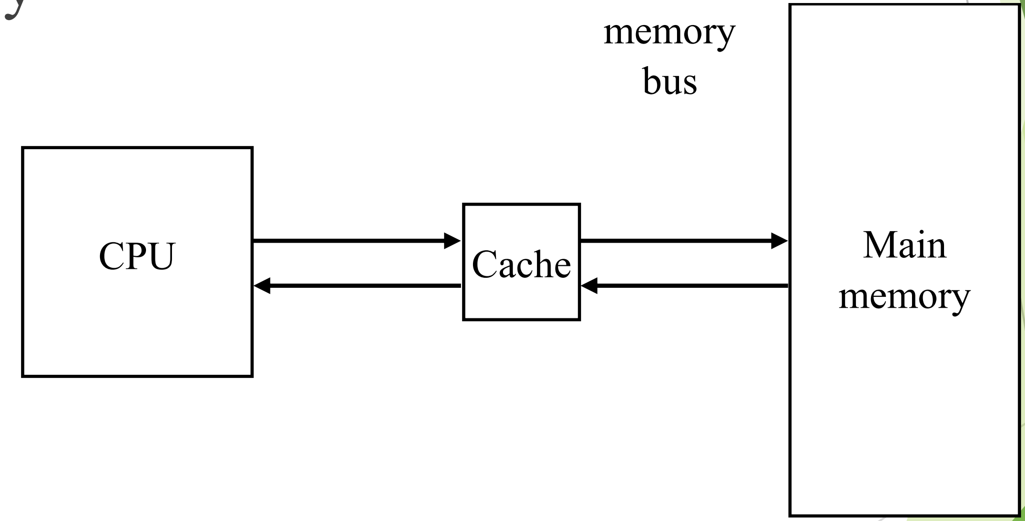
- After the program is fully loaded into memory, the system will "remember" the address of the program's starting point, and the CPU will start to follow the logical flow of the program to extract the machine commands in the correct address one by one until the program ends. In this process, each instruction will have its own execution process, or instruction cycle.

3.2 Execution of commands

- The hardware architecture of the command
- The execution period of the command

3.2.1 The hardware architecture of the command

- Critical hardware to execute commands：
  - main memory
  - CPU
  - Cache
  - memory bus



The important hardware that executes the instructions and their relationships

3.2.1 The hardware architecture of the command

- main memory

After the computer program is loaded, it is initially stored in the main memory, so all the instructions to be executed come from memory. Main memory is generally composed of a semiconductor device called dynamic random access memory (DRAM), which can read and write the values through current when it is powered on, and keep the data content unchanged. However, once the power is lost, the contents of the DRAM disappear. DRAM is considered a fairly high-speed storage, but compared to the computing speed of the CPU, the speed of reading an instruction from the main memory is still many times slower than executing an instruction by the CPU.

3.2.1 The hardware architecture of the command

- CPU
  - The unit that is really responsible for executing instructions is actually composed of a bunch of logic circuits with different functions, the main components are the data path and the control unit (CU). The former includes ALUs and other computing units used to perform specified operations, as well as registers with access speeds comparable to those of each computing unit. the latter is responsible for interpreting (decoding) the meaning of instructions and controlling the process of execution.

3.2.1 The hardware architecture of the command

- Cache
  - The CPU has a high computing speed, which cannot match the *read and write* speed of ordinary DRAM, so if you want to increase the speed of instruction execution, you must start with the bottleneck of improving memory access speed. In addition to increasing the speed of the main memory, the improvement method can actually add some smaller but faster memory to temporarily store the instructions or data to be accessed. It will first copy some instructions or data blocks that may be about to be accessed from the autonomous memory, and when the CPU wants to retrieve the next instruction for execution, or when it wants to access a piece of data, it can prioritize searching for it in the cache memory to improve access performance.

3.2.1 The hardware architecture of the command

- memory bus

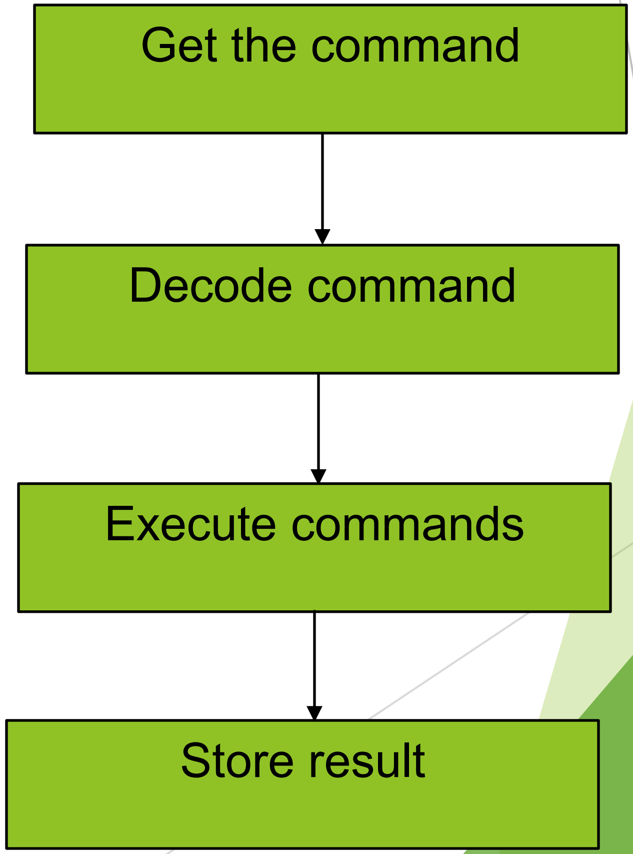
The speed of the CPU is different from the main memory, and to ensure that the data in memory is correctly obtained, the CPU usually accesses the memory through a channel called a bus (bus). It is actually a set of dedicated transmission pipelines that span between the CPU and main memory, and typically separates the lines that send data from the memory addresses. There is usually an interface between the bus and the connected device, on which there is usually a register to temporarily store the transmitted data, as well as some transmission control mechanism.

3.2 Execution of commands

- The hardware architecture of the command
- The execution period of the command

3.2.2 The execution period of the command

- The execution cycle of a command consists of *four* phases：
  - Get the command
  - Decoding commands
  - Execute commands
  - Store result



3.2.2 The execution period of the command

- Get the command：
  - When the program is linked and loaded, each command will have its own unique command address. In the CPU, there is a special program counter (PC) register that is used to store the address of the next instruction to be executed.
  - According to the content of the program counter, the CPU fetches the next instruction to be executed to the correct address in memory, and returns the instruction through the data bus and stores it in a special instruction register (IR) inside the CPU for execution.

3.2.2 The execution period of the command

- If the program is executed sequentially according to the memory address, the contents of the PC register will be automatically added by 1 after each execution to map (or point to) the address where the next command is located; However, if there is a situation where the execution is not in the order of the address, such as when encountering a branch command, the correct address of the next command to be executed will be directly saved to the PC register.

3.2.2 The execution period of the command

- Decoding commands
  - When the machine language instruction in IR is returned to the CPU, the most important action is to interpret it, a step called decode.
  - Each machine instruction indicates the action it wants to perform, as well as all the data involved in the action, the former is called the operator in the command, and the latter is called the operand.
  - Mapped to a script, the part representing the action is called the operation code (op_code), and the remaining code is used to indicate where the data is.

3.2.2 The execution period of the command

- The CPU first decides how to obtain the operand or its address based on the operand content. Since each machine code represents a unique kind of operation, the process of program compilation (assembly) must first go through the coding process to convert assembly language instructions into unique binary code according to established encoding rules.
- When the command is to be executed, the binary machine code must be interpreted and disassembled into operators, and then based on the operator's definition and the remaining script content, the value of the operator to participate in the operation must be found.

3.2.2 The execution period of the command

- Execute command
  - After the instructions are decoded, the CPU goes further to get the required operators and put them into the correct registers. After all the operators are taken out and in place, the data is sent to the data path and the operation is completed by starting the operation unit specified by the command through the control circuit.

3.2.2 The execution period of the command

- Save the result
  - The results obtained after calculation must be saved to the correct location for the command to be completed. This location is also determined based on the result of the aforementioned decoding, which may be a register or a memory address derived from the operator content. On many computers that store execution results with registers, "saving results" may not be a separate runtime, it may be stored in registers or memory when the runtime is complete. However, the action of storing the results of the operation into main memory can sometimes be another independent instruction.

3.3 format of the instruction

- An operator is the code of a command that specifies which operation to perform.
- The meaning represented by the operator is not fixed：
  - The register's codename, memory address, and actual data content.
- The command format varies depending on the hardware design or the type of instruction.

3.3 format of the instruction

- Four-Address Instruction
- Three-Address Instruction
- Two-Address Instruction
- One-Address Instruction
- Zero-Address Instruction

3.3.1 Four-Address Instruction

OP code	**address 1**	**address 2**	**address 3**	**next instruction address**

op_code：opcode
address 1：Addressing of the output operand
address 2：The address of the first input operator
address 3：The second input is the addressing of the operator
next instruction address：The address of the next command to be executed

3.3.1 Four-Address Instruction

- If the instruction set is encoded in a four-bit manner, there is no need to use a program counter （program counter，PC）：
  - The address of the next command exists in the current command.
  - makes its length longer.
- If you write a program with the same function, the four-address format requires the least number of commands.

3.3 format of the instruction

- Four-Address Instruction
- Three-Address Instruction
- Two-Address Instruction
- One-Address Instruction
- Zero-Address Instruction

3.3.2 Three-Address Instruction

OP code	**address 1**	**address 2**	**address 3**

op_code ： opcode
address 1 ： Addressing of the output operand
address 2 ： The address of the first input operator
address 3 ： The second input is the addressing of the operator

3.3.2 Three-Address Instruction

- With three-bit encoding, a PC is required to determine the order of execution.
- Advantages：
  - The instruction length is shorter than the four-bit format, easy to understand, and high program readability.

3.3.2 Three-Address Instruction

- Example: Suppose we want to calculate the following multiplication formula

$$A = B \times C$$

where B is the Multiplied, C is the Multiplier, and A is the product. As long as you use three addresses, you can get the result of multiplying two numbers:

```
MUL   A, B, C       ; A ← B * C
```

3.3.2 Three-Address Instruction

If the encoding length of each register code is 3, if the codes of the three sets of registers are A:001, B:010, C:011, and the multiplication instruction is 0011, the above command can be translated into the following 16-bit three-bit format：

4 bits	3 bits	3 bits	3 bits
0011	001	010	011

3.3 format of the instruction

- Four-Address Instruction
- Three-Address Instruction
- Two-Address Instruction
- One-Address Instruction
- Zero-Address Instruction

3.3.3 Two-Address Instruction

OP code	**address 1**	**address 2**

op_code： opcode
address 1：addressing of the first input-output operand
address 2：The second input is the addressing of the operator

3.3.3 Two-Address Instruction

- The command length of the dual address is shorter than that of the four and three addresses, and the address of the next instruction also needs to be determined by the PC.
- Example: Suppose we want to calculate the following multiplication formula

$$A = B \times C$$

If you directly use a command in a dual address format to perform the calculation

```
MUL    A, B      ; A ← A * B
```

The product after completing the multiplication is stored in A. In this way, the value in the original A will be overwritten.

3.3.3 Two-Address Instruction

To avoid this situation, A must first be moved into the register as the output:

```
MOV    C, A         ; C ← A
MUL    C, B         ; C ← C * B
```

In this way, C will be the product of the multiplication of A and B, and the contents of the A register will not be changed.

3.3 format of the instruction

- Four-Address Instruction
- Three-Address Instruction
- Two-Address Instruction
- One-Address Instruction
- Zero-Address Instruction

3.3.4 One-Address Instruction

OP code	**address**

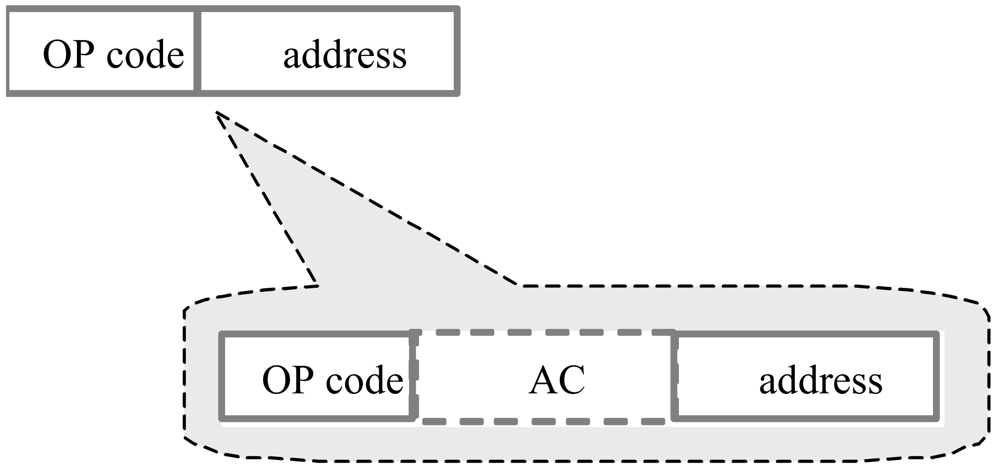
op_code ：opcode
address ：Enter the addressing of the operator

3.3.4 One-Address Instruction

- When the unit address instruction responds to the operation of two inputs, it needs to use a specific accumulator (AC) as a workspace to replace the first operator in the aforementioned two-address instruction, which not only provides another input source, but also serves as the destination for storing the operation results.
- AC itself is also a register, its length will match the needs of the operation, and the programmer cannot specify other registers to replace it 。

3.3.4 One-Address Instruction

- You can think of the format of the unit address as a two-bit format, but the first field of the operator is "hidden" and AC replaces the role of the address：



3.3.4 One-Address Instruction

- Example: Here are two arithmetic instructions represented by dual addresses

```
ADD    B, C        ; B← B + C
SUB    A, B        ; A← A – B
```

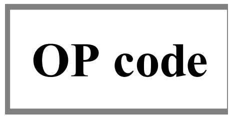
If the above two commands are expressed in the unit address format, then:

```
LOAD   B   ; AC← B
ADD    C   ; AC ←AC + C
STOR   B   ; B ←AC
LOAD   A   ; AC ←A
SUB    B   ; AC←AC – B
STOR   A   ; A ←AC
```

3.3 format of the instruction

- Four-Address Instruction
- Three-Address Instruction
- Two-Address Instruction
- One-Address Instruction
- Zero-Address Instruction

3.3.5 Zero-Address Instruction



op_code：opcode

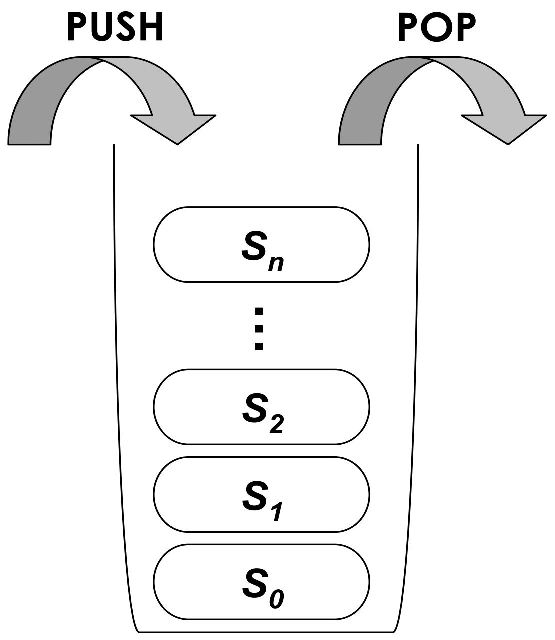
3.3.5 Zero-Address Instruction

- The zero address instruction itself is op_code, omitting all operators; However, not all commands can be represented in this format.
- The zero-address instruction uses a data structure called a stack to assist in arithmetic or logical operations.
- Stacked structure：
    - Last-in-first-out (LIFO) storage.
    - Data can only be put in and taken out from the top.

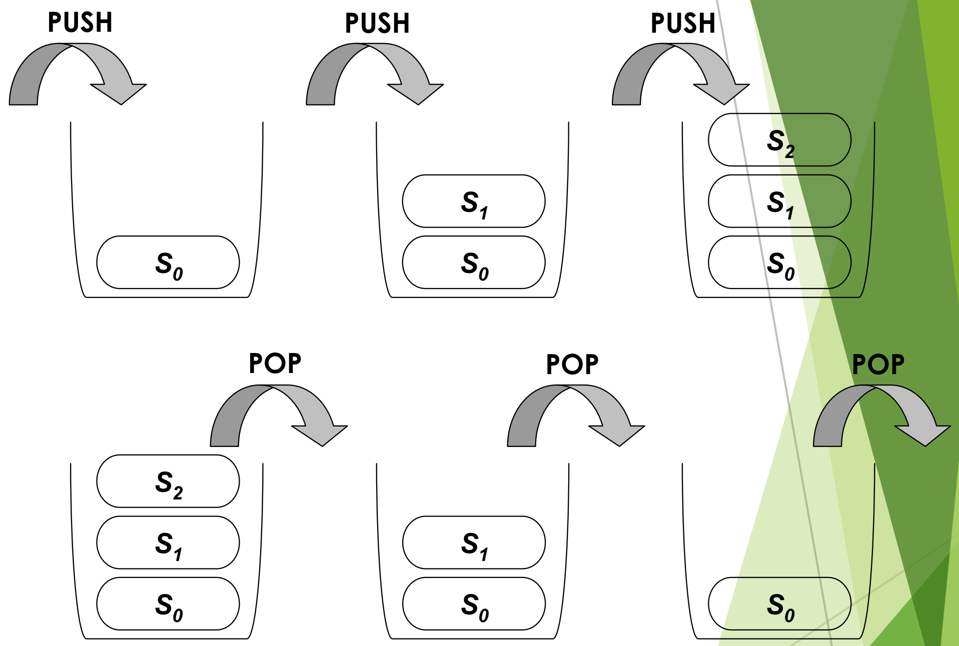
3.3.5 Zero-Address Instruction

- The data structure of the operator is stacked, so in addition to the command being represented only in the format of zero addresses, two unit address instructions are required:
- Save stack command（PUSH）
    - PUSH X ; Save X to the top of the stack
    - Pop the stack command（POP）
    - POP X ; Take the element at the top of the stack and put it in X

3.3.5 Zero-Address Instruction



structure



Stack operation process example

3.3.5 Zero-Address Instruction

- When computing an expression using a stack architecture, its expression must be represented by a postfix notation：
  - The operator must be placed after the corresponding operator.
- example：

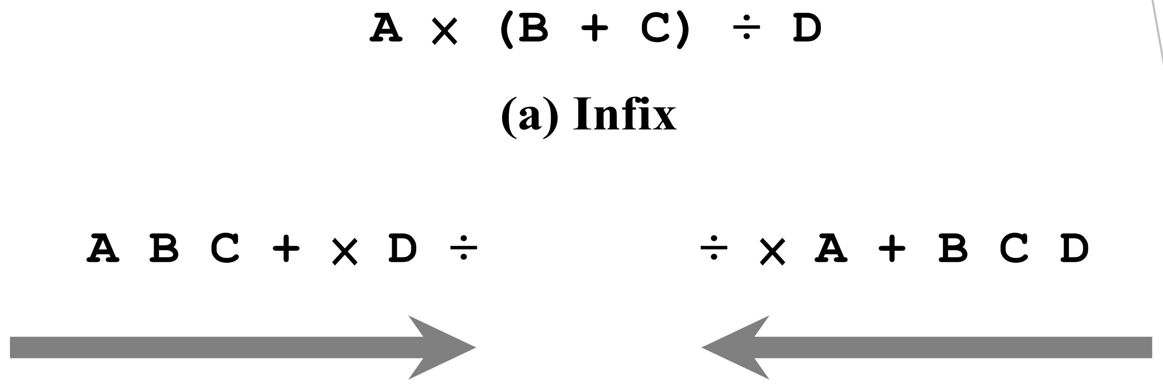
  The expression presented by the central notation：

  $$A + B \times C$$

  Replaced with post-post notation：

  $$A\ B\ C \times +$$

3.3.5 Zero-Address Instruction



(a) Infix

(b) Postfix

(c) Prefix

(a)General central notation

(b)Stack input order of post-prefix notation, from left to right

(c) Stack input order of pre-representation from right to left

3.3 format of the instruction

【example】 There is an equation as follows

$$Y = (A – B) ÷ (C + D * E)$$

Try to express the command formats of the four addresses, three addresses, double addresses, unit addresses, and zero addresses mentioned above. The addition, subtraction, multiplication, and division operators are ADD, SUB, MUL, and DIV, respectively; There are also two commands: STORE and LOAD.

3.3 format of the instruction

【answer】

Four addresses：

```
I1: SUB R1, A, B, I2        ；R1 ← A – B
I2: MUL R2, D, E, I3        ；R2 ← D * E
I3: ADD R3, C, R2, I4       ；R3 ← C + R2
I4: DIV Y, R1, R3, I5       ；Y ← R1 ÷ R3
```

3.3 format of the instruction

【answer】

Three addresses：

```
SUB R1, A, B        ；R1 ← A – B
MUL R2, D, E        ；R2 ← D * E
ADD R3, C, R2       ；R3 ← C + R2
DIV Y, R1, R3       ；Y ← R1 ÷ R3
```

3.3 format of the instruction

【answer】

Dual address：

SUB A, B	；$A \leftarrow A - B$
MUL D, E	；$D \leftarrow D * E$
ADD C, D	；$C \leftarrow C + D$
DIV A, C	；$A \leftarrow A \div C$
MOV Y, A	；$Y \leftarrow A$

3.3 format of the instruction

【answer】

unit address：

```
LOAD   A    ；AC ← A
SUB    B    ；AC ← AC – B
STORE  R1   ；R1 ← AC
LOAD   D    ；AC ← D
MUL    E    ；AC← AC * E
ADD    C    ；AC←AC + C
STORE  R2   ；R2← AC
LAOD   R1   ；AC ← R1
DIV    R2   ；AC← AC ÷ R2
STORE  Y    ；Y ←AC
```

3.3 format of the instruction

【answer】

Zero address：

First (A–B)÷(C+D*E) **Changed to post-position AB–CDE*+÷**

	Stack content	
PUSH A	A	
PUSH B	A B	
SUB	R1	；R1 ← A – B
PUSH C	R1C	
PUSH D	R1CD	
PUSH E	R1CDE	
MUL	R1CR2	；R2 ← D * E
ADD	R1R3	；R3 ← C * R2
DIV	R4	；R4← R1 ÷ R3
POP Y		；Y ← R4

3.4 Data addressing

- addressing
  - It is a method used to determine how the computer obtains the instruction operand and the efficient address (EA) of the operand.

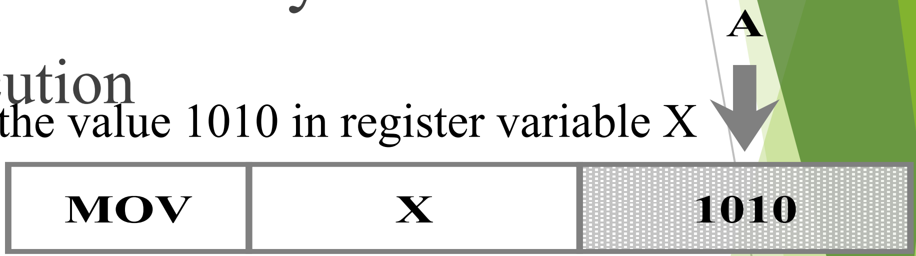
3.4 Data addressing

- Immediate Addressing
- Direct Addressing
- Indirect Addressing
- Register Addressing
- Register Indirect Addressing
- Displacement Addressing
- Implicit Addressing

3.4.1 Immediate Addressing

- The value of the data is in the instruction, and after extracting the operator, the data can be obtained "immediately" without accessing the memory.
  - virtue：Faster execution

```
MOV  X, 1010    ; Store the value 1010 in register variable X
```



  - demerit：The numeric range of data is limited by the length of the operator field.

The immediate addressing method, assuming that the operand A=1010, can obtain the data immediately after extracting the oper

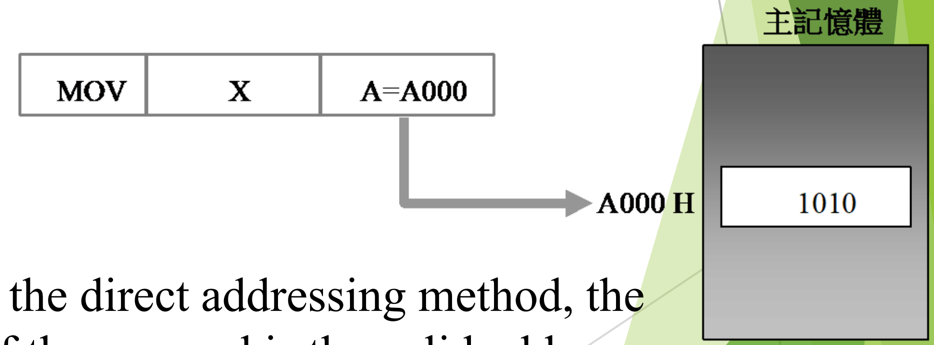
3.4 Data addressing

- Immediate Addressing
- Direct Addressing
- Indirect Addressing
- Register Addressing
- Register Indirect Addressing
- Displacement Addressing
- Implicit Addressing

3.4.2 Direct Addressing

- It is a simple addressing mode, and the address referred to by the operator is the data.
  - virtue：Data can be obtained with just one memory access.
  - demerit：Limited by the length of the command address field, the range of addressability is very limited.

MOV X, [1010] ; Deposit the value in memory address 1010 into register variable X



Using the direct addressing method, the data of the operand is the valid address EA=A, in this case, the value is A000h, and the real data value 1010 can be obtained after accessing the memory address

3.4 Data addressing

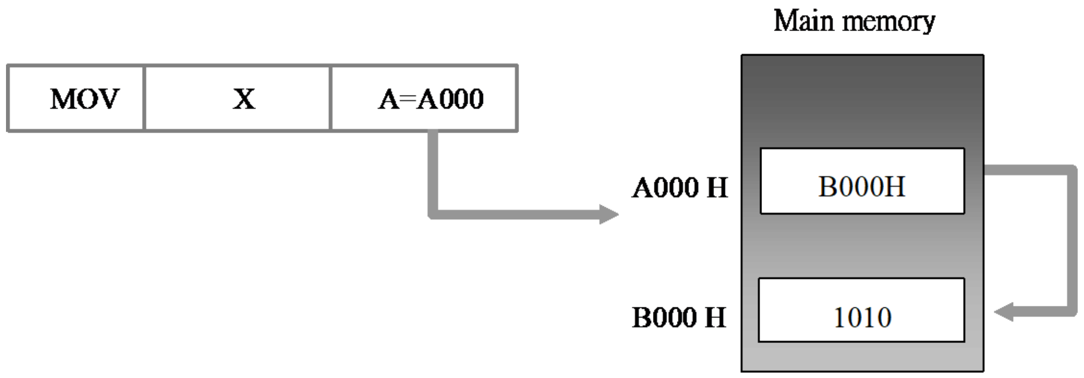
- Immediate Addressing
- Direct Addressing
- Indirect Addressing
- Addressing
- Register Indirect Addressing
- Displacement Addressing
- Implicit Addressing

3.4.3 Indirect Addressing

- The indirect address of the command is stored in memory with the true valid address of the operator.
- After obtaining the memory address of the valid address from the operator field of the command, the memory must be accessed again to obtain the valid address. After that, the valid address is obtained to obtain its property value, which is the real computational metadata, which requires two memory accesses.
  - Advantages：The valid address of the operator is stored in memory, and the addressing range can be increased.
  - Cons：Increases the execution time of the command.

3.4.3 Indirect Addressing

MOV X, A; First, take the value stored in the operator field A as the memory address, take out the valid address from it, then take the value from the valid address, and store it in the register variable X



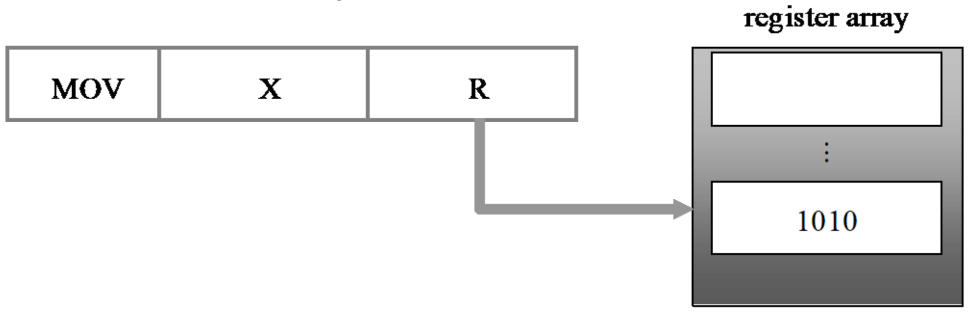
Using the indirect addressing method, if the value obtained from the operator field is A=A000h, the valid address EA=B000h must be obtained from the corresponding position in memory with A as the address, and then the real data used for calculation is obtained 1010 based on this EA; Therefore, we can see that EA=(A)

3.4 Data addressing

- Immediate Addressing
- Direct Addressing
- Indirect Addressing
- Register Addressing
- Register Indirect Addressing
- Displacement Addressing
- Implicit Addressing

3.4.4 Register Addressing

- Similar to the direct addressing method, the register addressing method also uses the operator field to specify the location of the data, but the difference is that the location it specifies is the register.
  - virtue：Access data from registers at high speed.
  - demerit：The number of registers is limited, and the program formed by the command is limited by the number of registers.



MOV X, R ; The value is first taken from the register indicated by the operator field R, and then stored in the register variable X

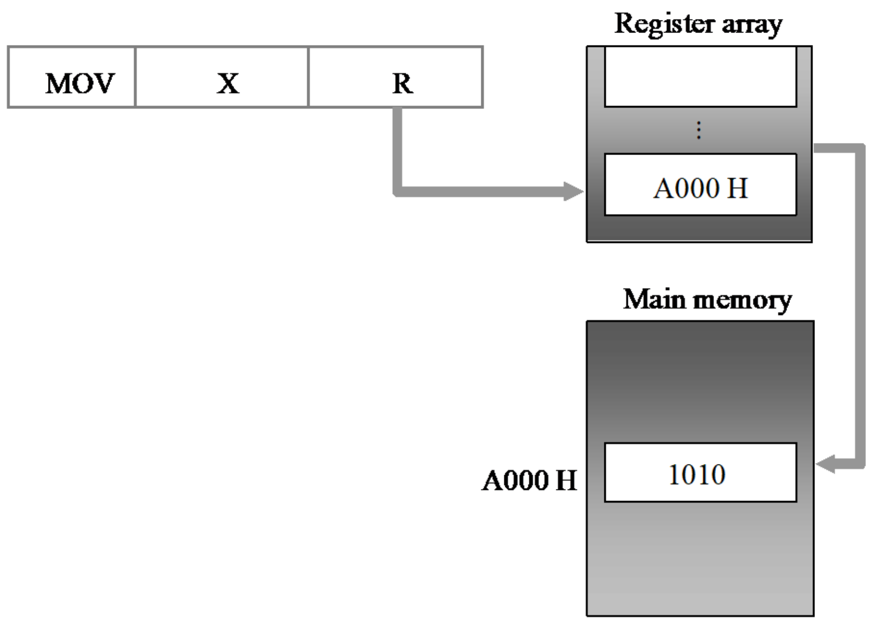
Using the register addressing method, the register content referred to by the operand field is the data, where EA=R, and the true operand value is 1010

3.4.5 Register Indirect Addressing

- The valid address is stored in the register specified by the operand field, so the valid address of the data must be retrieved from the corresponding register according to the operand field, and then the valid address must be used to retrieve the data from memory.
- Two access actions to obtain the true computational metadata, once for registers and once for memory.
  - Advantages ：The addressability of the word width to speed up the acquisition of valid addresses.
  - Cons ：Additional memory access is required.

3.4.5 Register Indirect Addressing

MOV X, [R]; First, take out the valid address from the register pointed to by the operator field R, then take out the value from the memory location of the valid address and store it in the register variable X



Using the register addressing method, the register content referred to by the operand field is the data, where EA=R, and the true operand value is 1010

3.4 Data addressing

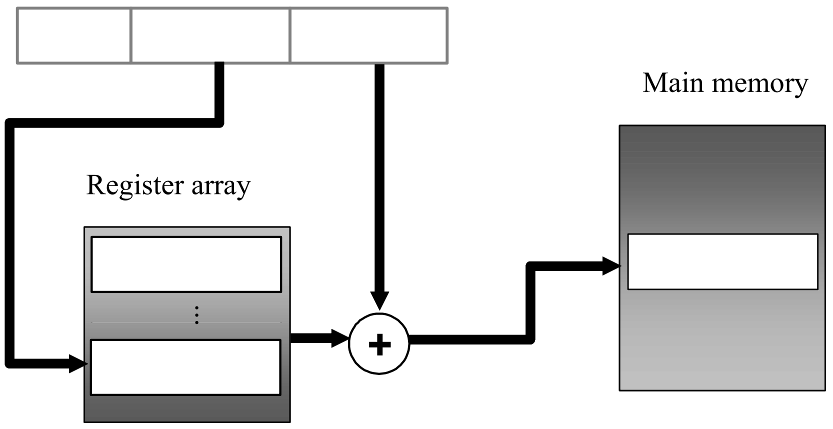
- Immediate Addressing
- Direct Addressing
- Indirect Addressing
- Register Addressing
- Register Indirect Addressing
- Displacement Addressing
- Implicit Addressing

3.4.6 Displacement Addressing

- Add a base address to a displacement to get a "true valid address".
- The base address can be:
  - The value of a register.
  - The contents of an operator.
- The amount of displacement is the difference between the true valid address and the base address.

3.4.6 Displacement Addressing

- The displacement addressing method can be subdivided into three types：
  - relative addressing
  - base-register addressing
  - indexing

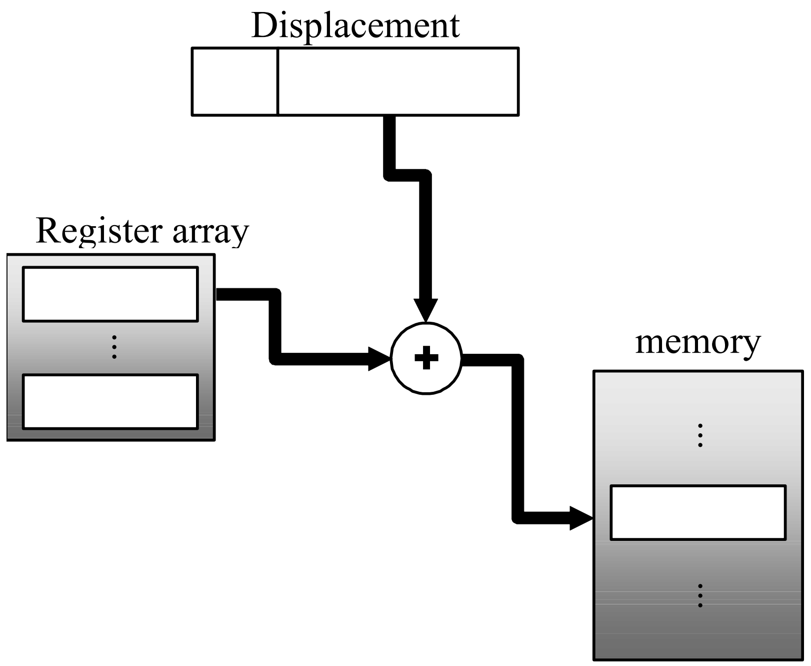


Conceptual diagram of displacement addressing method

3.4.6 Displacement Addressing

- Relative Addressing
  - It is mainly used for branch or jump commands.
  - When a program generates a branch due to a judgment or condition, the PC value must be re-given, and the method is to provide the command's "actual effective address relative to the PC" of the command to the computer through the operator field to change the content of the PC.
    - EA=(PC)＋Operator content
  - Instructions using relative addressing only need one Oporto to find the EA and access the memory, but the addressing ability is limited by the length of the operand field.

3.4.6 Displacement Addressing

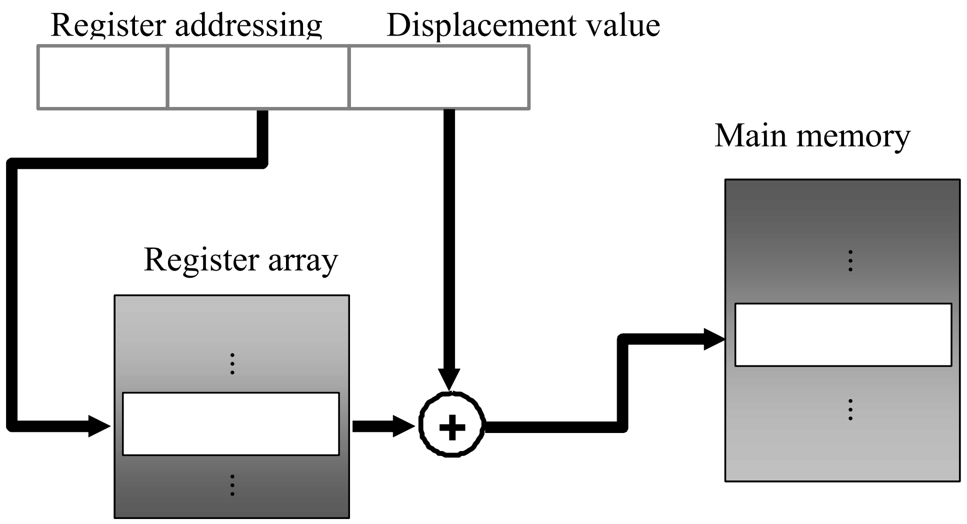


Relative addressing method

3.4.6 Displacement Addressing

- Base-Register Addressing
  - When the program needs to address more neighboring data (such as array data structures) at the same time, it is very convenient to use the base addressing method to access the data.
  - The start address of the program or data fragment is used as the base address and stored in the specified register; The memory addresses in the program or data fragment are all offsets relative to the start address.
  - The way to get EA is：
    - The amount of displacement in the base + operand
    - The base addressing method can be used to achieve relocating programs.

3.4.6 Displacement Addressing

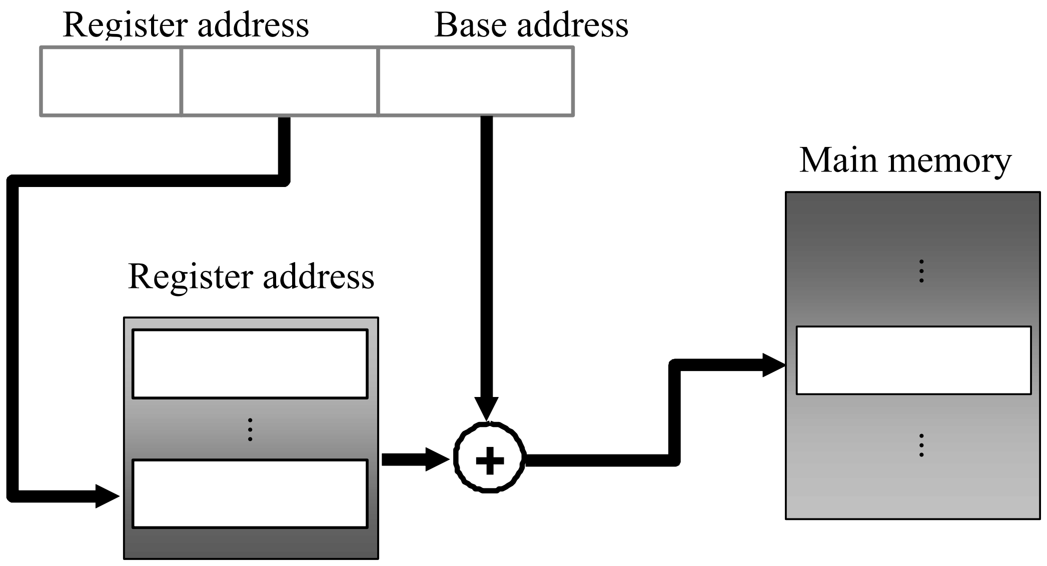


Base addressing method

3.4.6 Displacement Addressing

- Indexing
  - Similar to base addressing, the only difference is that the "start address" and "offset" are stored in opposite positions.
    - The From Address is the operation column for the command.
    - Displacement has registers.
  - The valid address is calculated as follows：
    - Operator content + register content
    - Since the displacement exists in the register, the amount of displacement can be automatically changed through incremental commands (such as INC), which is suitable for array access.

3.4.6 Displacement Addressing



indexing
method

3.4.6 Displacement Addressing

【example】

Please explain the similarities and differences between the "base addressing method" and the "indexing method".

3.4.6 Displacement Addressing

【answer】

same :

(1) How to calculate EA； EA = (R) + (operand)；

(2) It is suitable for working with data structures in arrays or other contiguous spaces.

Different:

(1) The register of the base addressing method stores the start address, and the operator stores the displacement; The index law is the opposite, its displacement is in the register, and the start address is in the opposition;

(2) The registered value of the base addressing method cannot be changed by command, and the index method can change the registered value (such as INC, DEC commands).

3.4 Data addressing

- Immediate Addressing
- Direct Addressing
- Indirect Addressing
- Register Addressing
- Register Indirect Addressing
- Displacement Addressing
- Implicit Addressing

3.4.7 Implicit Addressing

- Implicit addressing is commonly found in commands in a single operand format.
- Many operations require two input operators and one output operator, and usually by default an AX register (accumulator) is used as the multiplier and also as the output object, so it does not need to be specified in the command.
- MUL BX;
  - Multiply the value of AX with the BX register and save the result to the AX register.

3.4 Data addressing

Addressing mode	Valid address	memory Number of visits	Pros and cons
Immediate addressing method	Operand=A	0 times	Advantages：No memory access required, fast speed Cons：The operator size is limited by the number of registers
Direct addressing method	EA=A	1 time	Advantages：It's simpler Cons：Address space is limited
Indirect addressing method	EA=(A)	2 times	Advantages：It has more address space Cons：Multiple memory accesses, slow
Register addressing method	EA=R	0 times	Advantages：No memory access required, fast speed Cons：The address space is limited by the number of registers

3.4 Data addressing

Addressing mode	Valid address	memory Number of visits	Pros and cons
Register indirect addressing method	EA=(R)	1 time	Advantages：It has more address space Cons：Additional memory access is required
Displacement addressing method	EA=A+(R)	1 time	Advantages：The addressing ability is more flexible Cons：The acquisition of EA needs to be calculated
Implicit addressing method	EA=Default register	0 times	Advantages：Fast speed, saving instruction length Cons：Requires specific registers

3.5 Design and consideration of instruction sets

- Completeness of the instruction set
- Instruction set optimization

3.5.1 Completeness of the instruction set

- There are six types of instructions that a complete general-purpose computer usually needs：
  - Data transfer command
  - arithmetic, logic, and comparison instructions
  - branch instructions
  - Export/Input Commands
  - Floating-point operation instructions
  - Special system call commands

3.5.1 Completeness of the instruction set

- Data transfer command
  - Computers mainly process data, which must be moved or copied between registers and memory during program execution, so any instruction set must have such instructions.
  - Common ones such as：
    - MOV、LDW（load word）、STW（store word）、LDB（load byte）、STB（store byte）
  - With these migration instructions, the computer can smoothly retrieve data from memory for calculations, and can also save the results back to the specified memory space.

3.5.1 Completeness of the instruction set

- arithmetic, logic, and comparison instructions
  - The most powerful function of a computer is numerical and logical operations, so this type of instruction must be indispensable.
  - The numerical operation referred to here is integer operation, because it is the basis of all arithmetic operations, and the hardware line is relatively easy to implement.
  - Arithmetic：
    - adder（ADD）、subtract（SUB）、multiply（MUL）、divide（DIV），There is also taking the opposite number (i.e., the complement of taking two)
  - Logical operations：
    - AND、OR、NOT 、XOR、XAND

3.5.1 Completeness of the instruction set

- Compare operations：
  - COMP：Set the value of a specific register or flag based on the size of the two input values.
- With these arithmetic, logical, and comparative instructions, the computer can perform various numerical and logical operations.

3.5.1 Completeness of the instruction set

- branch instructions
  - Usually is:
    - conditional branch：Common BRANCH commands.

      BRZ（branch on zero） command, based on the previous comparison Whether the result of the operation or arithmetic operation is 0 to determine the next decision

      The command address of the line.
    - Unconditional branch: Also known as jump, when the program executes this command, it will determine the address of the next command to be executed based on the parameters of the command.

3.5.1 Completeness of the instruction set

- Output/Input Commands
  - The computer must be able to read data from the outside world and "display" the processed data in order to be useful.
  - Output/input instructions are a series of instructions that allow the CPU to read data from peripherals into memory or CPU, or to send data to various output/input devices. The format of such commands is affected by the perimeter addressing method.

3.5.1 Completeness of the instruction set

- Floating-point operation instructions
  - Floating-point arithmetic is a numerical operation with a higher complexity than integers, and in practical applications, the range of values is already quite large, and the use of floating-point numbers can make it easier for computers to meet such needs.
  - In fact, as long as you have complete integer computing power, you can use programs to complete various floating-point operations, but the performance is not ideal, especially when facing applications that require a lot of floating-point operations, it will form serious bottlenecks. If a computer provides floating-point operation instructions on hardware, it will greatly improve the performance of numerical operations and make the computer more powerful.
  - Common floating-point commands：
    - ADD、SUB、MUL、DIV、SQRT、LOG

3.5.1 Completeness of the instruction set

- Special system call commands
  - These commands are usually related to the design of the microprocessor itself, and some microprocessors provide special functions, such as shutting down the computer, interrupting services, etc., which can be used by such system call commands, and the instructions provided by them vary depending on the design goals and concepts of the processor.

3.5 Design and consideration of instruction sets

- Completeness of the instruction set
- Instruction set optimization

3.5.2 Instruction set optimization

- The most important and difficult topic to choose when designing a computer processor ：
  - Find the most appropriate or optimal set of instructions.
- Common issues：
  - number of instructions
  - Format of the instruction - the number of operators
  - Addressing mode

3.5.2 Instruction set optimization

- number of instructions
  - The number of instructions affects the performance of program execution and also affects the hardware design of the computer.
  - In addition to a complete basic instruction set, computers usually provide some additional instructions for some commonly used operations, so that many actions that originally required a combination of multiple instructions can now be processed with only one or two instructions, and the performance is naturally improved.
  - However, if too many dedicated instructions are added, the number of components on the processor will swell rapidly, which not only increases the difficulty of design, but also affects the execution performance of a single instruction.

3.5.2 Instruction set optimization

- Format of the instruction - the number of operators
  - The format of the command has a considerable impact on the hardware design.
  - The more operones an instruction can provide, the more flexibility it can be used, because more things can be done in one instruction.
  - With fewer operators, the length of the instruction can be shortened, and the actions of obtaining computational data or saving results can be more standardized, and the execution speed of a single instruction can be improved.

3.5.2 Instruction set optimization

- Addressing mode
  - The data involved in the calculation comes from registers, and the instruction operation is fast.
    - operators in RISC instructions.
    - The data transfer between registers and memory is handled by load and store commands, known as load/store mode.
  - If the data source for a single instruction is memory, there may be one or more memory access actions in a single instruction cycle, and too diverse execution processes will greatly increase the complexity of the hardware design.

3.6 Encoding and decoding of instructions

- Fixed Encoding
- Hybrid Encoding
- Variable Length Encoding

3.6.1 Fixed Encoding

- Commands of different formats have the same length of opcode, that is, the number of bits of the opprint is fixed, and the length is only related to the total number of commands.

Instruction format	number	Script code
Zero address	00	000000
	~	~
	04	000100
A address	05	000101
	~	~
	16	010000
Two addresses	17	010001
	~	~
	40	101000
Not used	41	101001
	~	~
	63	111111

Fixed encoded instruction sets

3.6.1 Fixed Encoding

- Advantages :
  - The software and hardware design is relatively simple.
    - In the compilation stage, as long as the script comparison table is first built, and then a simple table lookup method is used to convert the operation code into a binary op_code.
    - When decoding, only a fixed-length op_code is taken out and fed into the decoding line.
- Cons：
  - There will be unused coding space, resulting in waste.

3.6 Encoding and decoding of instructions

- Fixed Encoding
- Hybrid Encoding
- Variable Length Encoding

3.6.2 Hybrid Encoding

- Mixing the encoding of opcodes of different lengths, it adjusts the number of bits in the Opcode Field depending on the Instruction Format.
- example：
  - If the instruction set contains commands in the format of double address, unit address, and zero address, the possible encoding methods are as follows：

Instruction format	field		
Zero address	**OP code**		
unit address	**OP code**	**address 1**	
Dual address	**OP code**	**Address 1**	**address 2**

3.6.2 Hybrid Encoding

Instruction format	number	Script code								
two address	00	0	0	0	0	0	addr1		addr2	
	01	0	0	0	0	1	addr1		addr2	
				…			…			
	22	1	0	1	1	0	addr1		addr2	
	23	1	0	1	1	1	addr1		addr2	
Single address	96	1	1	0	0	0	0	0	addr2	
	97	1	1	0	0	0	0	1	addr2	
					…				…	
	106	1	1	0	1	0	1	0	addr2	
	107	1	1	0	1	0	1	1	Addr2	
Zero address	432	1	1	0	1	1	0	0	0	0
	433	1	1	0	1	1	0	0	0	1
						…				
	436	1	1	0	1	1	0	1	0	0

Hybrid encoded instruction sets

3.6.2 Hybrid Encoding

- Advantages ：
  - Reducing the number of bits that the Opf Field occupies can provide more bits for the two-bit format instruction operator.
  - For commands with unit addresses and zero addresses, since relatively few "operand fields" are required, unused operand fields can also be included in the scope of encoding to obtain additional encoding space.
- Cons：
  - The decoding process is much more complicated.
  - If the decoding line is designed for the maximum possible length of the op_code, the circuit complexity is high.

3.6 Encoding and decoding of instructions

- Fixed Encoding
- Hybrid Encoding
- Variable Length Encoding

3.6.3 Variable Length Encoding

- Variable instruction encoding for both instruction length and operator length provides maximum flexibility and allows you to extend or extend your instruction set with instructions of different lengths at any time.
  - IA-32 instruction set for Intel 80x86.

a. JE EIP + Number of bits

- Several different typical IA-32 directives：

4	4	8
JE	Jump conditions	Displacement

3.6.3 Variable Length Encoding

b. CALL

8	32
CALL	Offset

c. MOV EBX, [EDI+45]

6	1	1	8	8
MOV	d	w	r/m Postfix bytes (Postbyte)	Displacement