### W1-1 计算机组成

1. 计算机的组成
2. 水瓶结构
3. 类型
4. 硬件发展
5. 硬件
6. VHDL
7. 操作系统 OS
8. Moore's Law
9. 发展趋势
10. 硬件
11. 软件
12. CPU 指令集

### W1-2 冯诺依曼体系

1. Von Neumann
2. 三个基本原则
3. 内容
4. 瓶颈 bottleneck
5. 潜在问题
6. 解决方案
7. 哈佛体系

### W2-1 程序执行

1. HLL
2. 三种翻译
3. Compiler
4. Interpreter
5. 特殊的 java
6. 代码 share&reuse
7. Source-level subroutines and macro library
8. Pre-translated relocatable binary library
9. Dynamic library and dynamic linking

### W2-2 数据

1. data，information，knowledge
2. claude Shannon
3. bit
4. GB-MB-KB-B
5. 二进制转十进制
6. N 进制转十进制
7. 十进制转二进制
8. 十六进制
9. 十六进制转十进制

### W3-1 编码

1. Alphanumeric character
2. 编码
3. 字符分类
4. ASCII
5. Unicode
6. 数字表示

### W3-2 操作系统

1. 操作系统发展
2. 操作系统
3. 操作系统分层结构
4. Kernel
5. CLI 命令接口层
6. 多层次操作系统
7. 两种交互方式
8. 计算机网络
9. 客户机 client
10. TCP/IP 协议

### W4-1 机器周期

1. 主板 motherboard
2. 总线 buses
3. 优缺点
4. CPU
5. 寄存器 register
6. 协处理器 coprocessor
7. 执行指令-机器循环
8. 机器周期 machine cycle
9. 不同长度指令集
10. 输出硬件（软硬拷贝）
11. 通信硬件 communication hardware
12. 端口 Port
13. 其他硬件

### W4-2 汇编语言

1. 汇编器
2. Label 标签
3. 字
4. EAX
5. EBX
6. ECX
7. EIP
8. ESI
9. EDI

### W5-1 栈

1. 状态标志位
2. SF
3. CF
4. ZF
5. OF
6. 内联汇编 inline assembler
7. 堆栈 stack
8. Upside-down stack 颠倒堆栈

9. 堆栈的作用

### W5-2 寻址方式
1. 指令
2. 寻址方式作用
3. 寻址方式种类

### W6-1 传递参数和程序跳转
1. Printf
2. Scanf
3. Scanf+printf
4. 占位符
5. Unconditional jump
6. Conditional jump

### W6-2 控制程序流
1. 双循环指令
2. If-else
3. For
4. While
5. Do-while

### W7-1 子程序
1. 子程序
2. 优势
3. 格式
4. Call
5. 嵌套

### W7-2 参数的传递和返回
1. 参数的两种形式
2. 传递参数的两种形式
3. 用地址交换两个变量
4. Stack frame 堆栈帧
5. 程序调用 stack frame

### W8-1
1. 递归
2. 阶乘

### W8-2 数字的表示
1. 有符号整数表示
2. 无符号整数表示

### W9-1 补码
1. 十进制补码
2. 溢出 overflow
3. 二进制补码
4. 减法
5. Java 数据类型表示

### W9-2 浮点数
1. 指数表示法
2. 浮点数格式
3. 余 n 表示法
4. 浮点数标准化
5. 二进制表示法
6. IEEE754 浮点数格式
7. 小数转十进制

### W10-1 存储
1. 主存的意义
2. RAM 两类
3. 高速缓存 cache memory
4. 显存 video memory
5. 大量存储 Mass storage
6. HDD 硬盘驱动器

### W10-2 存储层次
1. Maximal memory length
2. Memory module 内存条
3. 存储层次
4. 分层的意义
5. 越往下
6. Localization 本地化缓存

### W11-1 硬盘和虚拟存储
1. 硬盘结构
2. 存储容量
3. 寻址 addressing 两种
4. Disk cache 磁盘缓存
5. 写入读取
6. 虚拟内存 virtual memory
7. 交换区域
8. 虚拟内存管理
9. 32 位逻辑地址
10. Memory management unit

### W11-2 数字电路
1. 布尔值
2. 布尔门
3. 布尔电路
4. 滤波器 filter
5. 选择电路 selector circuit

### W12-1 电路设计
1. 选择电路逻辑简化
2. 多选择器 data selector
3. 双线译码器 multiplexer
4. 范式提取

### W12-2 加法器和触发器

1. 半加器 half adder
2. 其他半加器
3. 全加器 full adder
4. 组合逻辑电路 combinational
5. 时序逻辑电路 sequential
6. SR 触发器
7. 其他触发器
8. D 触发器

## W1-1 计算机组成

1. 计算机的组成: processor, controller, primary memory, secondary memory, peripheral(外围设备)
2. 水平结构: assembly language—instruction set—micro architecture—digital logic(二进制)
3. 类型: 60s mainframe—70s super—80s workstation—80s micro—80s pc—80s microcontroller—80s server—chip; 性能提高，体积减小
4. 硬件发展: vacuum tube(电子管)—transistor(晶体管)—IC(微型电子设备)—VLSI(超大规模 integrated circuit 集成电路)
5. 硬件: input, processing, output, storage, communication; 向下兼容：新设备要支持旧设备的软件
6. VHDL: 把硬件的运行抽象成代码表示
7. 操作系统 OS: 用户给出指令，具体工作由管家来操作
8. Moore's Law: Intel 创始人，可放在芯片上的电路晶体管数量每 24 月翻倍，实际是 18 月
9. 发展趋势：科学计算—商业计算—个人计算—普适计算—移动计算；得益于摩尔定律
10. 硬件：CPU(Central Processing Unit) 主存(primary storage) 辅存(secondary story) 输入设备(键盘鼠标扫描仪) 输出设备(显示器扬声器打印机)
11. 软件：命令的集合体，执行有意义的一系列操作
12. CPU 执行机器指令，每个 CPU 有自己的指令集

## W1-2 冯诺依曼体系

1. Von Neumann: Input—Processor+Memory—Output;
2. 三个基本原则：二进制原则(binary logic) 程序存储执行( program storage and execution) 计算机由五个部分组成;
3. 内容: Program is a list of instructions used to direct a task; Both program and data are held in memory and represented by binary code; Memory is re-writeable; Processor is part of the machine that execute the program instruction;
4. 瓶颈 bottleneck: CPU 被迫等待数据和指令传输到内存上，大脑运算太快，手脚跟不上
5. 潜在问题: 数据和指令都是二进制，计算机怎么区分；16 位的指令代码可能表示 1/2 个字符
6. 方案: 计算机有自己的读取节奏，指令—数据—指令—数据，会同时获得下一个读取地址；数据和指令采取特殊编码形式让 CPU 区分
7. 哈佛体系：program memory<—CPU—>data memory; 两个独立的 memory 和读取线路；但冯诺依曼依然主流

## W2-1 程序执行

1. HLL: high level language，和机器指令有语义鸿沟(semantic gap)，需要翻译
2. 翻译三种: 编译器 Compiler, .cpp--.exe, C++，一次性，速度快；汇编器 Assembler, .asm--.exe; 解释器 Interpreter，动态翻译边读边运行，python

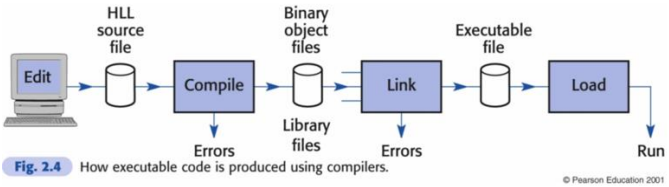


Fig. 2.4 How executable code is produced using compilers.


3. Compiler: 第一个 error 检查语法错误，第二个 error 缺少模块之间的链接，library file 为 link 提供必要的函数，执行速度快，编译慢，编译后占空间
4. Interpreter: Java, BASIC, Python, R 也叫脚本语言，指令转换成 token 中间形式，然后传递给 decoder 解码器，执行速度慢，无需编译，体积小传输快，方便调整
5. 特殊的 java：.java—compiler--.class—interpreter--.exe; 因此 java 兼容性强，跟 CPU 执行方式类似，可以看做 virtual machine
6. 代码 share&reuse: 有三种解决方案
7. Source-level subroutines and macro library: 源代码级别的子程序，macro 类似 method，替换宏库之后整个一起翻译
8. Pre-translated relocatable binary library: 预编译的二进制库，库代码预先转成二进制，连接到新代码中，但不能更改；有利于软件开发，但每个程序都有个子程序副本浪费空间和时间
9. Dynamic library and dynamic linking: 动态编译，早已进入内存的被标记成 public，与数据进行一一映射，避免翻译多个重复的程序

## W2-2 数据

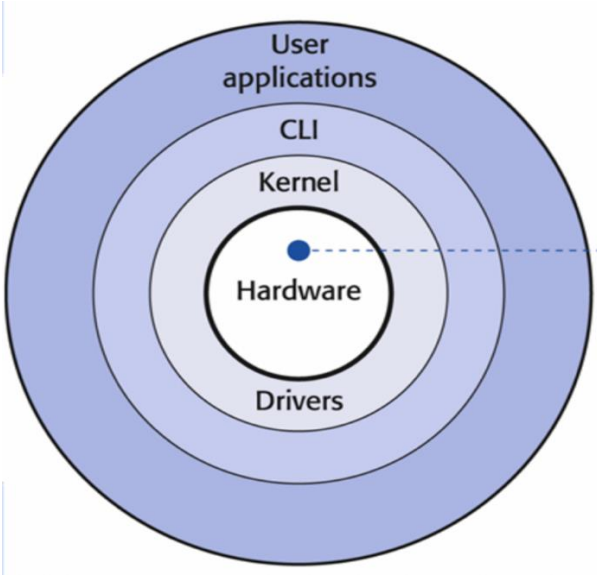
1. Data, information, knowledge: 数据就是单纯的数据，信息是在某个 condition 下的有效数据，知识是有效数据背后的逻辑规律
2. Claude Shannon：信息论和编码论，引入信息熵（entropy）定量描述信息量
3. bit：1/0 最小的信息单元
4. GB—MB—KB—B: 1024 Byte—bit: 8
5. 二进制转十进制：1011=2^0+2^1+2^3
6. n 进制转十进制：（a1,a2,a3,⋯,ak） n=ak*n^0+a(k-1)*n^1+⋯+a2*n^(k-2)+a1*n^(k-1)
7. 十进制转二进制：除 2，取余数
8. 十六进制：每 1 位十六进制可以写作 4 位二进制
9. 十六进制转十进制：可以除 16 取余数，也可以先化二进制，然后二进制转十进制

## W3-1 编码

1. Alphanumeric Character: 字母，数字，其他字符
2. 编码：同样一个字段 1011，在不同环境要对应不同的含义，所以要在前面加上不同字段适应不同环境；三种常见的：EBCDIC code（8 位），ASCII code，Unicode
3. 字符分类: 打印字符；控制字符：控制输出位置（TAB，回车），某些动作发生（BEL 铃铛），计算机与 IO 设备传送（ctrl+c/v）
4. ASCII：美国专用，国际标准是 7bit，计算机按照 8bit 储存，8bit 后面变成扩展版 ASCII。打印字符 0*20-0*7E；控制字符 0*00-0*1E。前四位是列（其实是 3bit），后四位是行（4bit）。从 1 到 2，0011 0001 到 0011 0010，行+1。从 A 到 a，0100 0001 到 0110 0001，列+10.
5. Unicode：16bit，包括所有语言和符号，万国码
6. 数字表示：4byte，32 位 bit；负数用补码表示，二者相加为 0；浮点数采用 IEEE754 标准定义。Char 是 8bit，int 是 16bit

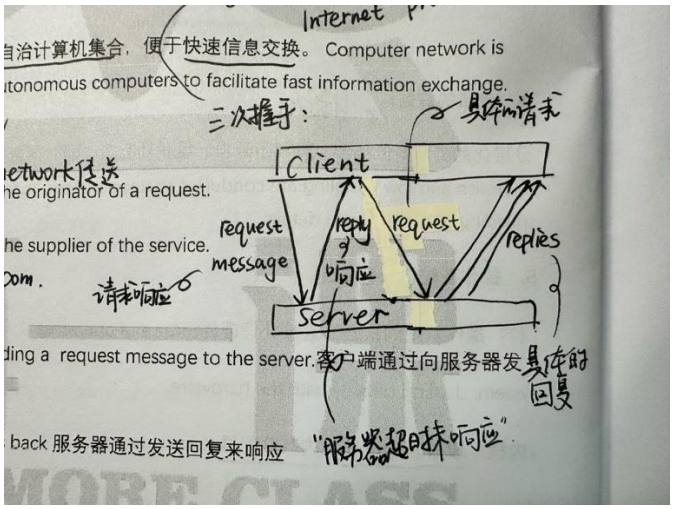
## W3-2 操作系统

1. 操作系统发展：60s OS/360(兼容)—70s Unix(分时)—80s MS-DOS & Mac-OS(图形化)—90s Windows 95/98/NT(兼容，是后续演变的基础)—2001 Mac OS X 基于 Unix 开发(内存管理)—Linux(兼容高性能)
2. 操作系统: 听从 User 和 Application 的命令，目的在于更好的管理底层硬件，并提供各种功能：interaction with the user/allow user protected/efficient/fair access to the facilities of the machine.
3. 操作系统分层结构：




cohesion and low coupling 高内聚低耦合，有利于开发维护升级，屏蔽底层细节

4. Kernel: device driver, memory allocator 驱动程序，内存分配器
5. CLI（command line interpreter）命令接口层: 提供用户对系统的访问权限 accessibility
6. 多程序操作系统: memory manager 内存管理器为每个程序分配内存，scheduler 调度器为每个程序分配 cpu 的时间，security kernel 安全内核维护每个程序的完整性
7. 两种交互方式：CLI，比如 DOS 在框中键入命令，Unix/Linux 中脚本，Windows/MacOS 鼠标单击；API 用户程序内部的函数调用
8. 计算机网络：是一个相互连接的自制计算机集合 interconnected collection of autonomous computer to facilitate fast information exchange
9. 客户机 client：发出 request；服务器 Server：接受请求提供服务。
10. TCP/IP 协议：三次握手



## W4-1 机器周期

1. 主板 MotherBoard：主要三个子系统，CPU，main memory, IO units
2. 总线 Buses：主板上所有组件通过总线互联，是一束 bundle 导线 conductor，电线 wire，轨道 track。有三类总线，地址 address，数据 data，控制 control。每个总线包含多条信号线。
3. 优缺点：是一种建立复杂系统的简单方法，插入或更换设备时几乎没有中断。点对点的连接需要大量线路但性能高，总线避免了点对点但性能会相应降低
4. CPU：包括 ALU（arithmetic logic unit 逻辑控制单元，进行逻辑运算）CU（control unit，控制器控制机器循环，需要 register 寄存器）
5. 寄存器 Register：一小块高速存储单元，存在 CPU 内部。有三类：IP（instruction pointer 存储下一条指令的地址）AX（accumulator 通用寄存器，16 位，8AH，8HL，EAX 扩展版，32 位）IR（instruction register，存储正在执行的指令）MAR（memory address register 在总线传输期间临时保存内存位置）MBR/MDR（存储数据）
6. 协处理器 Coprocessor：有特殊功能（数字/图像），是 CPU 不擅长的功能
7. 执行指令-机器循环：fetch-execute cycle/machine cycle，先 I-cycle（read from memory，decode）然后 E-cycle（execute）
8. 机器周期 machine cycle：IP—把地址复制到总线，然后地址复制到 MAR—MAR；IP 递增；M—地址上的内容复制到 MDR—MDR；MDR—从数据总线上复制到 IR—IR；IR—开始对指令解码—CU
9. 不同长度指令集：CISC（Intel，*86），RISC（ARM）（complex/reduced instruction set computer）.RISC 更快：vacated area of the chip can be used to accelerate the performance of more commonly used instructions, rather than compensating for thoses rarely used instruction; easier to optimize the design; simplify translation from HLL into smaller instruction set that hardware understands, resulting in more efficient program
10. 输出硬件：软拷贝输出（softcopy output）临时的展示包括视频音频，硬拷贝输出（hardcopy output）包括信物图片有实体支持可以打印
11. 通信硬件 Communication hardware：调制解调器 modem，集线器 hubs，其他设备
12. 端口 Port：连接设备和计算机，并行端口（parallel port）包括打印机部分扫描仪，串行端口（serial port）调制解调器扫描仪鼠标。这些都被标准化成 USB（universal serial bus）通用串口总线，因为并线由于线之间互相干扰影响速度
13. 其他硬件：接头 Connector，电源 power supply 由电涌保护器或不间断电源装置 UPS 组成

## W4-2 汇编语言

1. 汇编器：.asm—assembler—binary code。作用：翻译成二进制代码，把 label 和地址相连，生成二进制机器目标代码程序 machine object code program
2. Label 标签：是一个单元块，表示循环的开始，和内存地址关联。程序必须加载到主存里才能执行，且必须是 relocatable 可重新定位
3. 字：是由 bit 组成的一行储存单元，每个 word 都有 address，而内存采用随机访问 RAM（random access memory）。字节 byte，位 bit。
4. EAX: MOV EAX, 123H 把值赋给 EAX; INC EAX 递增; MOV maxval，EAX 把 EAX 的值赋给变量；DIV CX，除以 16 位寄存器 CX 的值，返回结果 10%3=3…1，EAX%CX=AX…DX。al，前八位
5. EBX: base 专门存地址的寄存器。LEA EBX, marks 把变量的地址赋给 EBX；MOV AL，[EBX] 把这个位置上的值作为 1byte 赋给 AL。bl，前八位
6. ECX：count 计数寄存器。MOV ECX，100……label1:……LOOP label1 循环 100 次，LOOP 会自动递减 ECX，直到=0。
7. EIP：保存下一条指令的地址，JMP 强制在 EIP 中转地址
8. ESI：MOV AX, [EBX+ESI] 将下一个 16 位值赋给 AX
9. EDI：MOV EAX, [ESI] 将 32 位值从 ESI 位置转移到 EDI 位置

## W5-1 栈

1. 状态标志位 CPU status flag，EFLAG 标志寄存器
2. Sign flag(SF)：值是负，SF=1
3. Carry flag(CF)：值进位，CF=1
4. Zero flag(ZF)：值为 0，ZF=1
5. Overflow flag(OF)：值溢出，OF=1
6. Inline assembler 内联汇编，在 C 语言中插入汇编。_asm{…} (多行) _asm …(单行) ; //(comment)
7. Stack 堆栈：采用 LIFO（后进先出 last in first out）.ESP 寄存器存储 stack 顶部的地址，EBP 指向底部地址。Push 堆，pop 取
8. Upside-down stack 颠倒堆栈，push 实际上 ESP--，pop 实际上 ESP++。也就是说，每次 push 最后要 add esp，位数/8.
9. 堆栈的作用：当作暂存器临时存储，传递参数

## W5-2 寻址方式

1. 指令：三个部分，action 操作，operand 操作对象，result

结果

2. 寻址方式作用：form operand address; compute an effective address; offer various addressing modes support better the needs of HLLs when they need to manipulate large data structure
3. 寻址方式种类：immediate mode 立即寻址：mov eax, 104；data register direct 寄存器直接寻址：mov eax, ebx；indexed register indirect with displacement 索引寄存器：mov eax, [table+esi] 或 mov eax, table[esi]

## W6-1 传递参数和程序跳转

1. Printf:

```
_asm
{
   push myint          // push the value of the variable onto the stack
   lea eax,format      // address of the format string is saved in eax
   push eax            // push the address of the string to the stack
   call printf // call printf, it will take two parameters from the stack
   add esp,8           //clean up top two positions in the stack
}
```

2. Scanf

```
_asm
{
   lea eax,input
   push eax
   lea eax, format  //address of the format string is saved in eax
   push eax         // push the address of the string to the stack
   call scanf       // call scanf, it will take two parameters from
                    // the stack scanf(%d,&input);
                    // user's input will be put in the 'input' variable
   add esp,8        // clean top two positions in the stack
}
```

3. Scanf+printf

```
_asm{
 lea eax,input
 push eax
 lea eax, format                    // address of the format string is saved in ea
 push eax                           // push the address of the string to the stack
 call scanf     // call scanf, it will take two parameters from the stack; scanf(%d,&input);
                                    // user's input will be put in the 'input' variable
 add esp,8                          // clean top two positions in the stack
 push input                         // push the value of the input onto the stack
 lea eax,message                    // address of the message string is saved in eax
 push eax                           // the value of eax is pushed onto the stack
 call printf   //call printf, it will take two parameters from the stack; printf("Your numbe
 add esp,8                  // clean top two positions in the stack
}
```

4. 占 位 符

- printf 中可以使用更多限定符（类型）：
  - %c print a character
  - %d, or %i print a signed decimal number
  - %s print a string of characters
- scanf 中可以使用更多限定符（类型）：
  - %c read a single character
  - %d read a signed decimal integer
  - %s read a string of characters until a white space or terminator (blank, new line, tab) is found

5. Unconditional jump: JMP label
6. Conditional jump:

| Instruction | Jump if | |
|---|---|---|
| JC/JB | Carry flag is set | (=1) |
| JNC/JNB | Carry flag is clear | (=0) |
| JE/JZ | Zero flag is set | (=1) |
| JNE/JNZ | Zero flag is clear | (=0) |
| JS | Sign flag is set | (=1) |
| JNS | Sign flag is clear | (=0) |
| JO | Overflow flag is set | (=1) |
| JNO | Overflow flag is clear | (=0) |

| | |
|---|---|
| JG | First operand is **g**reater. |
| JLE | First operand is **l**ess or **e**qual. |
| JL | First operand is **l**ess. |
| JGE | First operand is **g**reater, or **e**qual. |

## W6-2 控制程序流

1. 双循环指令：loopne = loop not equal

```
      mov ecx,200     ; Set counter
next: ...             ; Set label
      ...             ; 执行循环任务
      cmp eax,ebx     ; (eax==ebx? ) or (200次了?)
      loopne next     ; No? Go to next.
      ...             ; Yes? 向下执行，跳出循环.
```

(Z==1? or ECX ==0 ? )

2. If-else:

- In Java:

```
if (c > 0)
  pos = pos + c;
else
  neg = neg + c;
```

- Equivalent in the assembly code:

```
      mov eax,c
      cmp eax,0
      jg positive
negative:add neg,eax
      jmp endif
positive:add pos,eax
endif:...
```

3. For

- In Java:

```
for(int x=3;x<20;x=x+2)
{
  y = y + x;
}
```

- Equivalent in the assembly code:

```
    mov eax,3
for_loop: cmp eax,20
    jge end
    add y,eax
    inc eax
    inc eax
    jmp for_loop
end: ……
    ……
```

4. While

- In Java:

```
while (fib2 < 1000)
{
   fib0 = fib1;
   fib1 = fib2;
   fib2 = fib1 + fib0;
}
```

- Equivalent in the assembly code:

```
while:mov eax,fib2
    cmp eax,1000
    jge end_while
    mov eax,fib1
    mov fib0,eax
    mov eax,fib2
    mov fib1,eax
    add eax, fib0
    mov fib2,eax
    jmp while
end_while: …
```

while(fib < 10

do{…}

5. Do-while

- In Java:

```
do{
   fib0 = fib1;
   fib1 = fib2;
   fib2 = fib1 + fib0;
}while (fib2 < 1000)
```

- Equivalent in the assembly code:

```
while: //do{…}
    ……
    ……
    //while{
    mov eax, fib2
    cmp eax,1000
    jl while
```

## W7-1 子程序

1. 子程序：类似 method，是 part of the code，可以被 repeatedly used
2. 优势：save effort in programming; reduce the size of program; share the code; facilitate code reuse; encapsulate, package, hide complication from user; provide easy access to try and test code

```
label PROC
    …
    …
    …
    RET          ; return
label ENDP
```

3. 格式：
4. Call：将 EIP 的当前值记录为返回地址以便 return，然后把子程序地址放进 EIP 里
5. 嵌套 nested call：需要用 stack 来保存多个返回地址

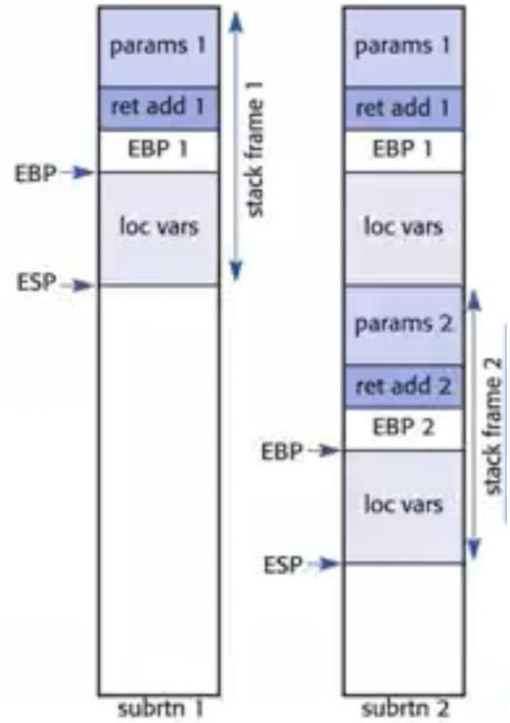
```
    …
    CALL SUB1      * call first subroutine
    …
SUB1: …
    …
    CALL SUB2      *call second subroutine
    …
    RET
SUB2: …
    …
    RET
```

## W7-2 参数的传递与返回

1. 参数的两种形式：value parameter，reference parameter（地址）
2. 传递参数的两种形式：register，stack

```
        …
        lea eax, first
        lea ebx, second
        call swap
finish: …
  ;
swap proc
        mov temp, [eax]
        mov [eax],[ebx]
        mov [ebx],temp
        ret
swap endp
```

3. 用地址交换两个变量
4. Stack frame 堆栈帧：包括一个子程序的所有参数，including 子程序的参数，返回地址，局部变量（Local variable，子程序自己定义的变量用于临时储存）
5. 程序调用 stack frame



## W8-1 递归

1. 递归：子程序调用自身

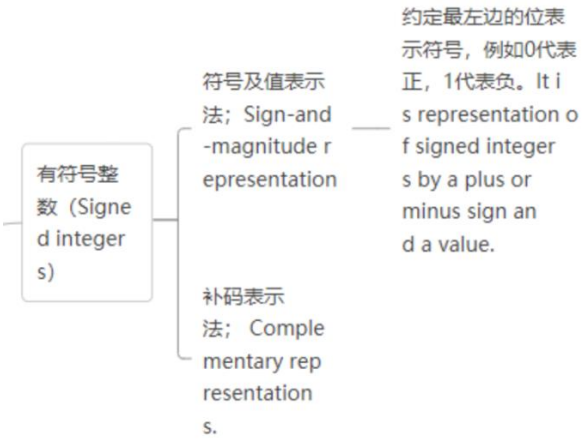
```
multiply PROC
        pop eax
        mov aux, eax
        pop eax
        mul eax, aux
        ret
multiply ENDP
```

2. 阶　　乘　　：

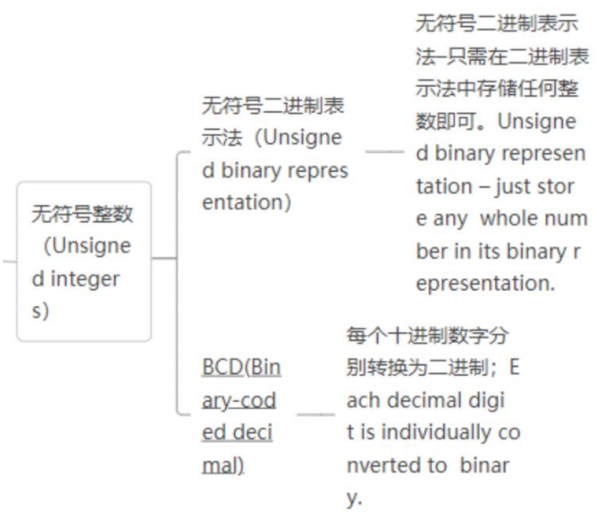
```
factorial PROC   //input n in eax
    push eax     // push current value onto the stack
    dec eax      // decrease the value of n
    jz finish    // if it is zero go to finish
    call factorial  // otherwise call factorial
    push eax        // push the result of last
                    // factorial's call to the stack
    call multiply   // call multiply subroutine
    ret             // return
finish: pop eax     // pop the parameter from the
                    // stack into eax
    ret             // return with the result in eax
factorial ENDP      // side effect?
```

## W8-2 数字的表示

1. 有　　符　　号　　整　　数　　表　　示

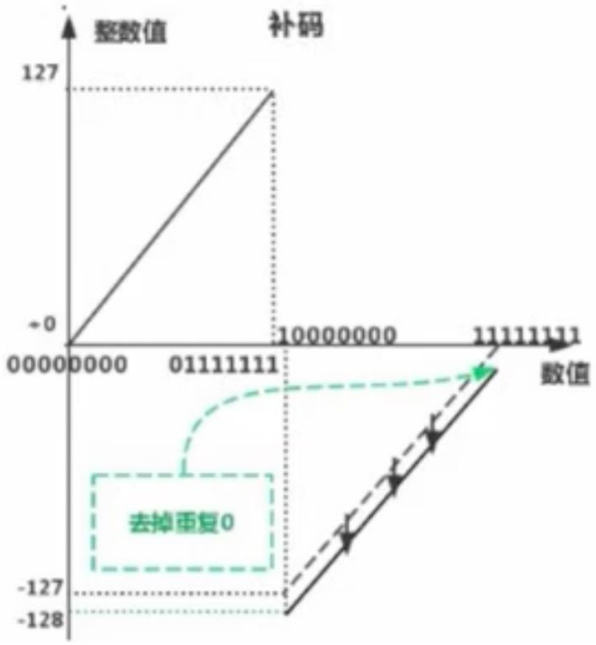



2. 无　　符　　号　　整　　数　　表　　示




## W9-1 补码

1. 十进制补码：800+300，700 实际是-200，最终结果 100；也可以 800+300=1100，然后 1100-1000
2. 溢　出　overflow　：　正　+　正　=　负　/　负　+　负　=　正




3. 二进制补码：以 8 位为例，0 开头是正数，1 开头是负数。正数的补码和二进制一样，负数则须根据正数运算得到：反转然后+1
4. 减　法　：　A-B=A+(-B).10010010-01010101 = 10010010+(-01010101)=10010010+10101011=00111101
5. Java　数　据　类　型　表　示　：
   - **byte** 8-bit: integers from $-2^{8-1}$ to $2^{8-1}-1$.
   - **short** 16-bit: integers from $-2^{16-1}$ to $2^{16-1}-1$.
   - **int** 32-bit: integers from $-2^{32-1}$ to $2^{32-1}-1$.
   - **long** 64-bit: integers from $-2^{64-1}$ to $2^{64-1}-1$.

## W9-2 浮点数

1. 指　　数　　表　　示　　法　　：

$$-1.2 * 10^{-2}$$

明如何存储
个组件部分

- 数字的符号；The **sign** of the number.
- 数字的大小---尾数；The **magnitude** of the number, known as the **mantissa**.
- 小数点的位置 The location of the **decimal point**.
- 指数的符号；The sign of the **exponent**.
- 指数的大小；The magnitude of the exponent.
- 指数的基数（如10或2）The base of the exponent (e.g., 10 or 2).

2. 浮　　点　　数　　格　　式


3. 余 n 表示法 excess-n (统一减去偏移量，以 50 为例)：

| | |
|---|---|
| *Representation* | 0 . . . 49 \| 50 . . . 99 |
| *Exponent being represented* | −50 . . . − 1 \| 0 . . . 49 |

4. 浮点数标准化：1.0000*10^n
5. 二进制表示法：32bit，1bit 表示 mantissa 正负，8bit 表示指数（包含正负，根据第一位），23 位表示 mantissa（默认 1.00000 所以实际 24 位）。偏移量是 10000000。指数代表小数点往左右移几位

   - Consider the code:

     0 10000001 11001100000000000000000

   - Sign of mantissa is '+' (leftmost bit is 0)
   - Mantissa is 1.11001100000000000000000

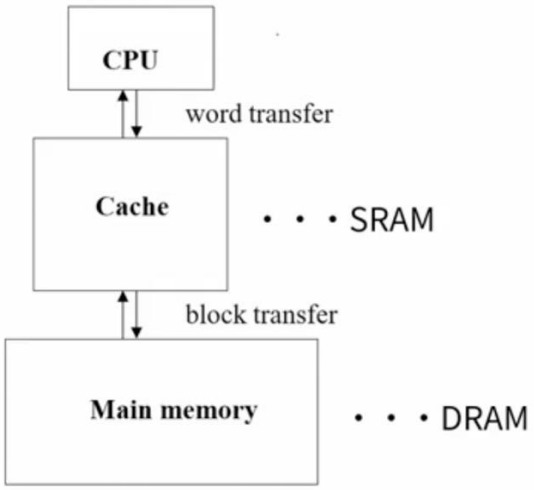
     Assumed
   - Exponent is 00000001 (=10000001-10000000)

     128
   - The number represented is +11.10011000…000

6. IEEE754 浮点数格式：指数是 8 位，偏移量是 2^7-1=127. 如果指数是 0，小数是 0，那么结果是+-0；如果指数全 1，小数是 0，那么结果是+-∞；如果指数全 1，小数不是 0，那么结果是 NaN（非数字）。
7. 小数（1.01）转十进制：2^1+2^(-2)=1.25

## W10-1 存储

1. 主存的意义：决定一次可以执行多少程序，一个程序可以分配多少空间。使用 RAM，随机访问，对所有存储项目的访问时间相同。
2. RAM 两类：动态 DRAM，便宜但速度慢，通过电容器 capacitor 实现，需要不停刷新 refresh；静态 SRAM，贵但速度快，通过触发器 flip-flop。两种都是易失的 volatile，电源关闭会丢失内容
3. 高速缓存 cache memory，是 CPU 和主存之间的桥梁，提高速度



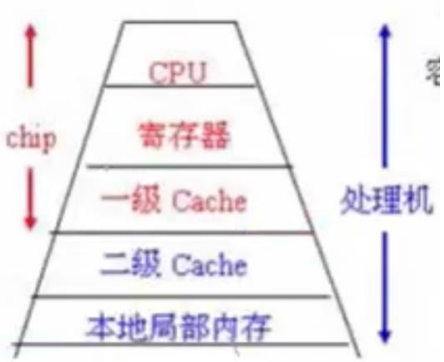

4. 显存 video memory：显卡
5. Mass storage 大量储存：hard disk 硬盘，optical disk 光盘（CD，DVD），USB（取代内存小的 floppy disk 软盘）
6. HDD 硬盘驱动器，永久储存器。与其他 mass storage 的区别：size 大，speed 快，permanence 永久性

## W10-2 存储层次

1. Maximal memory length 取决于 address width

| Address width | | Maximal memory length |
|---|---|---|
| 16 | | 64 Kbytes |
| 20 | | 1 Mbytes |
| 24 | | 16 Mbytes |
| 32 | Pentium | 4 Gbytes |
| 64 | 64-bit architectures | > 17 billion Gbytes |

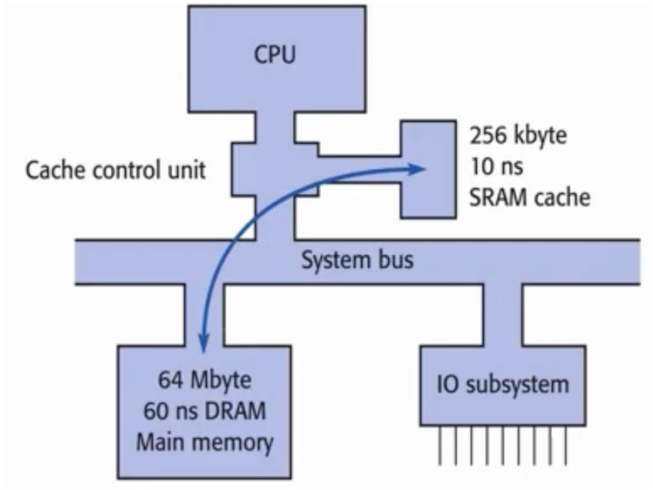
2. 一个 DRAM 有 8-16 个内存条 memory module，两种方式：单列直插 SIMM(single inline memory module)，双列直插 DIMM(dual inline memory module). 由于有多个内存芯片，需要前几位用于定位芯片，后面用于定位具体地址。
3. 存储层次：如果没有在缓存中找到，则会把主存中的相关内容挪到缓存里




4. 分层的意义：满足更快 CPU 的需求，限制系统成本，应对不断扩展的软件系统
5. 越往下：容量增加，访问时间增加，处理器访问内存的

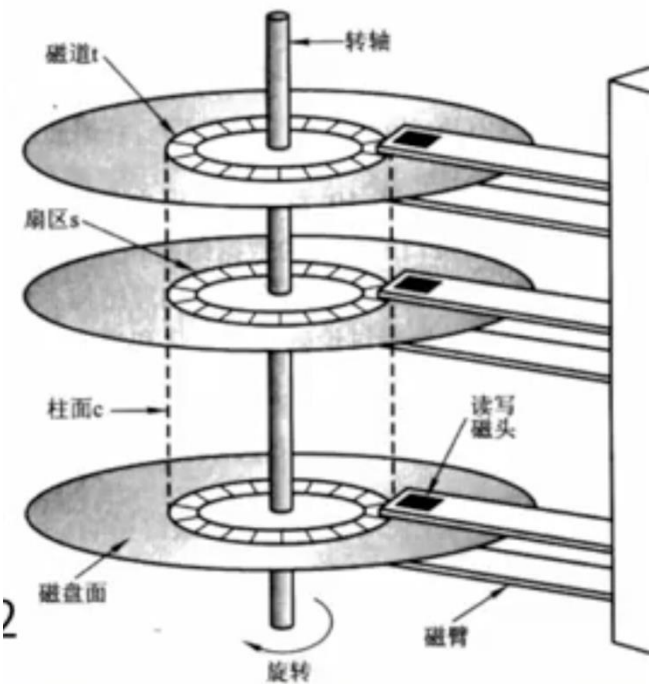
频率降低，成本降低

6. Localization 本地化：缓存的根本思想，计算机往往会在一段时间访问内存的同一位置，放在离 CPU 较近的位置。缓存里的内容管理则是由 cache control unit 来操作。

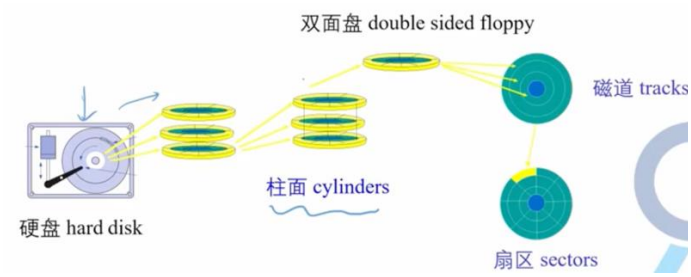



## W11-1 硬盘和虚拟存储

1. 硬盘结构：每个 disk 多个 flatter 盘，绕在 cylinder 柱面上，每个盘两面（头 head）都有信息，同心圆是磁道 track，扇形是扇区 sector。




2. 存储容量=磁面数*磁道数*每道扇区数*每扇区字节。




3. 寻址 addressing 两种：CHS(cylinder, head, sector), LBA(large block addressing，给 sector 编号)。先找到所需信息的位置，然后找到磁盘上的位置，然后向驱动器发送请求，请求读取扇区。
4. Disk cache 磁盘缓存：主存中的一部分作为缓冲区（buffer）保留部分磁盘信息。磁盘写入是 clustered，累计一定量再一次性写入。
5. 写入磁盘 write，读取磁盘 read。

**Files** : e.g. PERSONNEL FILE

**Records** : Adam's personal data

Adam Smith 35 Manager Purchasing

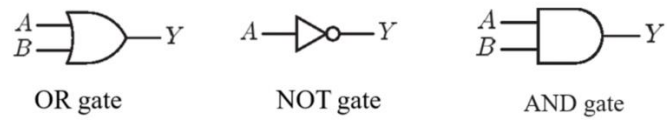
**Fields** : e.g. name, age, position, job function

**Key** : e.g. Adam Smith

6. 虚拟内存 virtual memory：硬盘中的部分空间，作为虚拟内存，有时内容太多光靠主存不够。将数据和程序溢出到磁盘上，提高灵活性。
7. 交换区域：磁盘上用作虚拟内存的区域称为 swap area。
8. 虚拟内存管理：主存分为帧 frame，每个 4KB。程序也被分为页面 page，当程序被调用时，只有需要的 page 会被加载到主存中，其他都复制到 swap area 上。
9. 32 位逻辑地址：lower 12 位—address within a page; upper 20 位—page number。
10. memory management unit：把逻辑地址转为页码和页内地址。

## W11-2 数字电路

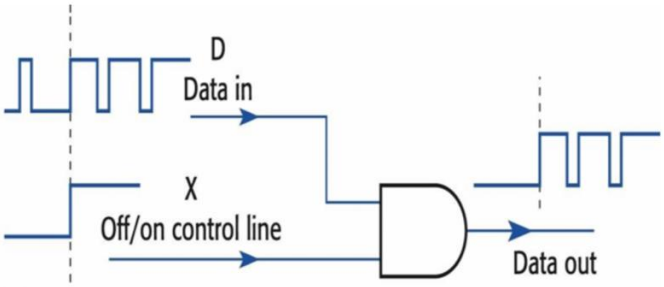
1. 布尔值：元件通电(flow): 1; 元件不通电(not flow): 0.
2. 布尔门 boolean gate：逻辑运算



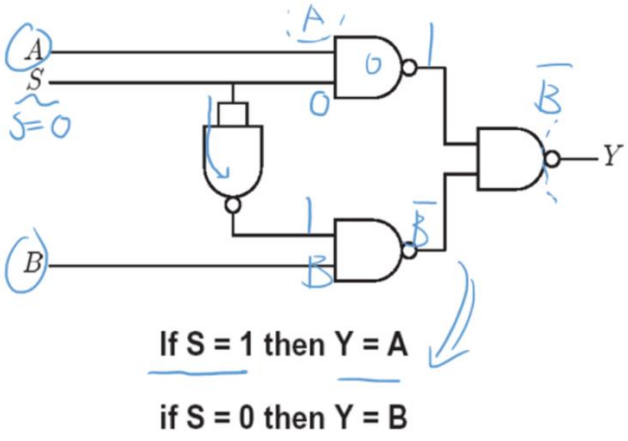


圆圈代表 not，可以和其他门搭配使用

3. 布尔电路 boolean circuit 由布尔门组成，实现布尔函数 boolean function
4. 滤波器 filter：本质是 1^A=A

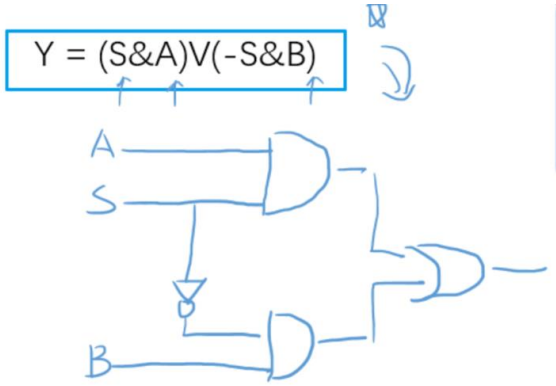


5. 选择电路 selector circuit(利用滤波器思想):

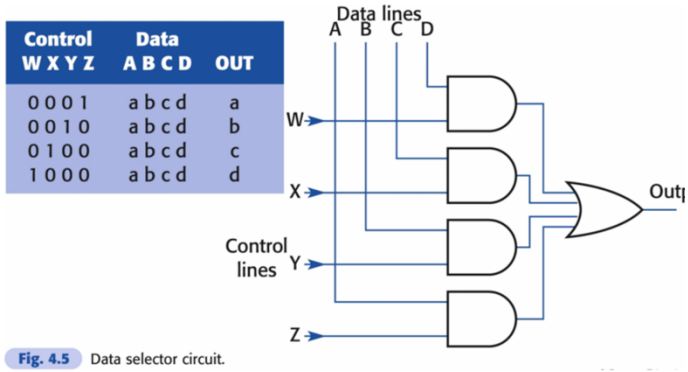



## W12-1 电路设计

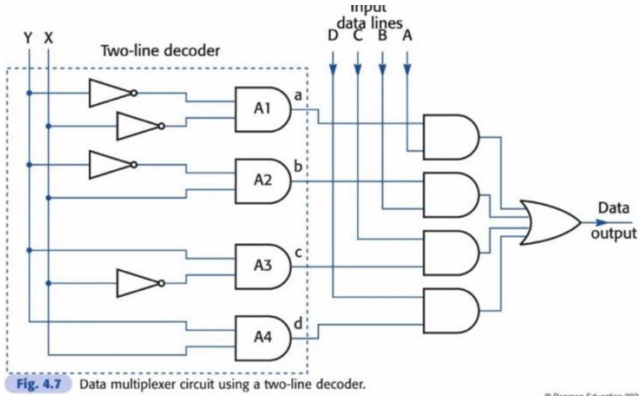
1. Selector circuit 的逻辑简化:



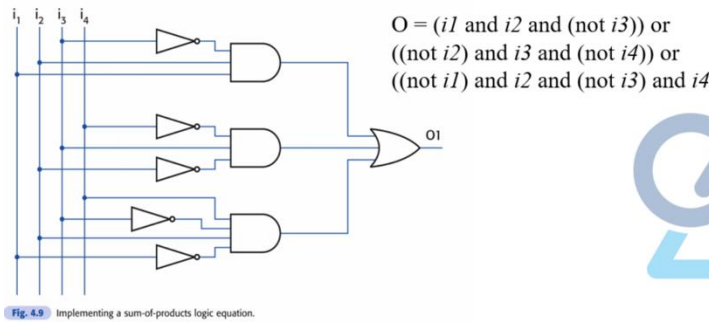

2. 多选择器 data selector(但同时按多个就不起效了)




3. 双线译码器 multiplexer: 只给两条线, 一共有四种组合的结果, 但相应的成本 cost 更高 (根据 gate 的数量)

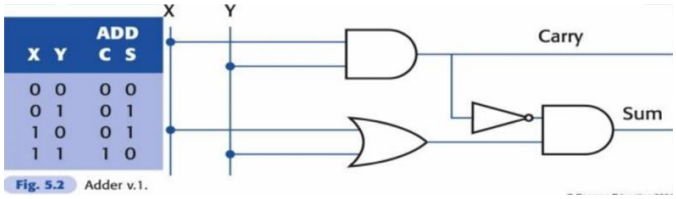



4. 范式提取: 根据 truth table, 挑出所有结果是 1 的, 然后合并同类项 (有些元件对结果无影响, 不用连接)

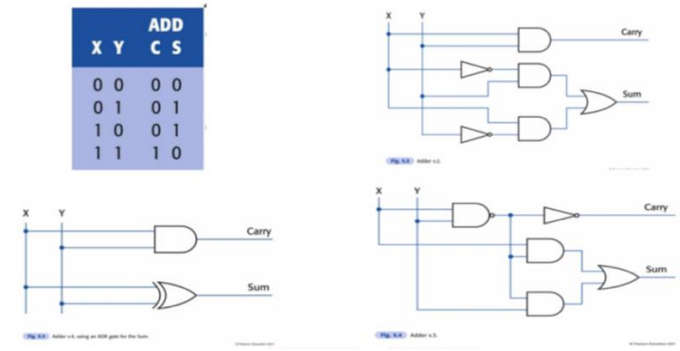



## W12-2 加法器和触发器

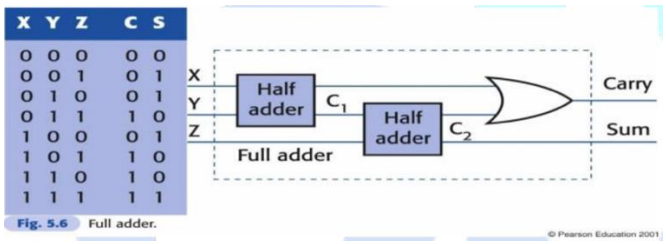
1. 半加器 half adder: carry 是进位, sum 是加位




2. 其他半加器:

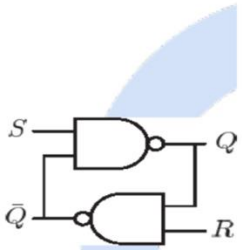



3. 全加器 full adder: 利用半加器实现多个二进制数的加法



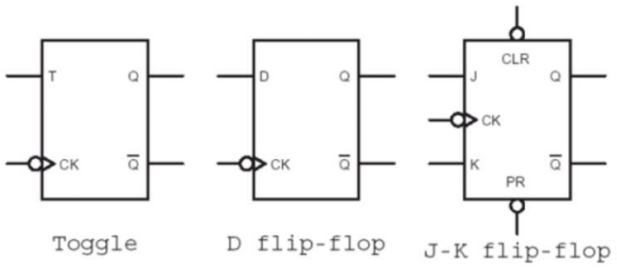

4. 组合逻辑电路 combinational(combinatorial) logical circuit: 瞬间输出, 没有记忆
5. 时序逻辑电路 sequential logical circuit: 有记忆, 包括触发器 flip-flop, 锁存器 latch, 广义的触发器包括锁存器
6. SR 触发器: 假设 R=1, S=1, 此时如果稳定在 Q=1, -Q=0。此时如果 R 突然变 0, 那么 Q=0, -Q=1。然后 R 突然变回去, 此时 Q 和-Q 也在稳定状态, 不会改变。如果 S 突然变 0, 再突然变回去, 那么 Q=1, -Q=0。因此我们可以认为, 如果呈现 Q=1, 说明 S 刚刚变动。如果呈现 Q=0, 说明 R 刚刚变动。SR 可以记忆, 最后

的状态可以显示哪个元件被最后设置为 0.




| R | S | Q |
|---|---|---|
| 0 | 0 | ? (Q=~Q) |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $Q_{prev}$ |

7. 其他触发器：哪个小尖尖是 clock，能让结果同时输出。CLR 是清除，PR 是预设。

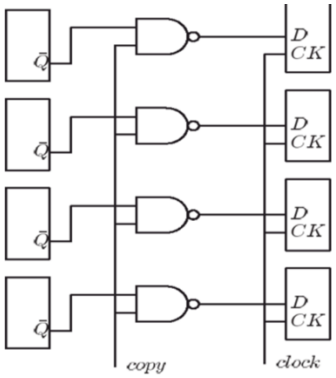


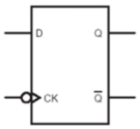

| T \| Q |
|---|
| 0 \| $Q_{prev}$ |
| 1 \| $\bar{Q}_{prev}$ |

| D \| Q |
|---|
| 0 \| 0 |
| 1 \| 1 |

| J K \| Q |
|---|
| 0 0 \| $Q_{prev}$ |
| 0 1 \| 0 |
| 1 0 \| 1 |
| 1 1 \| $\bar{Q}_{prev}$ |

8. D 触发器：如果预设 copy=0，那么所有输出结果都是 1，没有实现 copy。如果预设 copy=1，则会把-Q 所有相反数输出，也就是 Q，实现了把 Q 从左边复制到右边 。




D flip-flop

| D \| Q |
|---|
| 0 \| 0 |
| 1 \| 1 |

**11.( )** Assume a block of 256 data bytes has to be stored. Which of the following solutions is NOT sufficient?

- □ a) 8bit system with memory locations 0000 to 00FF
- □ b) 24bit system with memory locations 0000 to 0055
- □ c) 16bit system with memory locations 0000 to 007E
- □ d) 32bit system with memory locations 0000 to 005E
- □ e) 64bit system with memory locations 0000 to 0022

a. 8(15*16+15+1)/8

b. 24(5*16+5+1)/8

Assume Process A needs 5 pages of memory. When the CPU runs the process, it requests data from each of the 5 pages with equal probability. Assume that the average time to read a word of data from main memory is 5 ns. Assume the average time to read/write a page from hard disk from/into main memory is 5000ns. Furthermore, assume that a page must be swapped out to make room for the incoming page. Assume no caching is used. What is the average access time to read a word of data if 1 page of process A is stored in main memory at one time while the content of the other 4 pages are on hard disk?

□ a) 5 ns □ b) 5005 ns □ c) 7505 ns □ d) 8005 ns □ e) 10005 ns

内存的 page：读取 5

硬盘的 4page：一进一出 5000*2，读取 5

BCD：二进制编码的十进制，每四个表示一个十进制

Stereo 双声道 *2

第一位用作表示正负之后，后面就不用对半了

Complementary 补码

Hexadecimal 十六进制

Java 中间形式 byte code

除法也会 overflow，-128/（-1）=128

Sequential storage 顺序存储