# 计算系统基础知识点纲要

**注：**本文档整理自学长的笔记，基本上生成于2016年，内容上经我审阅，基本上与本学期的学习内容相接近， 照顾到了很多细节。我自己也将本文档打印出来备用，也欢迎和大家共享。如果那位学长出来声明一下版权，我也很乐意写上他的名字。

本文档中如有缺漏，欢迎随时指出以帮助后人。

**南京大学软件学院，余东骏**

**2019年12月9日星期一，于南京大学仙林校区**

# 引言

1.现代计算机：**通用电子数字计算机**。

2.计算机是**通用计算设备**

①只要有足够的时间和存储器，所有计算机可以做相同计算（计算速度差别）。

②做新的计算只需安装软件，无需更换计算机。

3.**图灵机**：有限状态自动机（不是通用的）。

4.**电子元件**是计算机主体和硬件实现的物理基础。

5.**数字**是计算机的基本特征和通用性基础。

6.计算机核心处理部件：**CPU**（**中央处理器**）

①指挥信息处理

②执行信息的实际处理

7.**指令**是计算机程序中规定的可执行的最小工作单位。

8.**冯·诺依曼**提出“**存储程序控制原理**”的思想。

9.**微处理器**：CPU中的半导体集成电路。（Pentium，80486,80586）

10.**计算系统抽象层次**（每一层对上一层隐藏自己的技术细节）

①**问题**

②**算法**：有限性，确定性，有效可计算性

③**程序**

④**指令集结构**：软硬件接口

⑤**微处理器**

⑥**逻辑电路**

⑦**元件**：CMOS（互补金属氧化物半导体）

11.**计算机语言**

12.**指令集结构（ISA）**：编写的程序和执行程序的底层计算机之间的接口的完整定义。（MIPS，PowerPC，IA-64），13.DLX指令集是MIPS指令集的简化版。

# 一、C语言程序设计基础

## 第二章 C语言程序设计简介

1.高级语言翻译技术

①**解释**：程序由解释程序执行，易开发和调试，执行慢。

②**编译**：输出可执行映像，直接在硬件上执行，执行快。

2.**main函数：程序从main函数开始执行。每个C程序有且仅有一个main函数。**

3.C99规定**main函数必须声明为返回一个整数值**。返回值可以省略。

int main(){

return 0; /\*可以省略\*/

}

4.C89中可以声明

main(){

}

5.**语句和声明以分号;结束。**

6.编程风格：程序中单词间和行间的空格数量不改变程序的意义，采用**缩进格式**。

7.**注释**

8.**预处理指令**

以#开头。**不可以以分号；结尾。**

#include指令被文件内容代替。

头文件stdio.h中定义与I/0函数相关的信息。

#include<stdio.h> 预定义目录中查找

#include”MyProgram.h” 源文件相同目录中查找后在系统库中查找

7.**输入和输出**（I/O）

**格式化输出函数printf**：输出到标准输出设备（显示器）

printf(“2+3=5”);

printf(“2+3=%d”,5);

printf(“2+3=%d”,2+3);

格式说明数目=格式用字符串后的数值数目

**格式化输入函数scanf**：从标准输入设备输入（键盘）

scanf(“%d”,&x);

特别注意不能加“\n”!

8.格式说明

**%d** 十进制整数格式说明

**“ “** 中的内容是格式用字符串

**\n** 换行

%f 单精度

%lf 双精度

%c 字符

%s 字符串

## 第三章 类型和变量

1.变量特征：**标识符**、**类型**、**作用域**、**存储类**。

2.**变量类型**

int：整数 -231~231-1

char：字符

float：单精度浮点数，有效数字7位

double：双精度浮点数，有效数字16位

3.标识符命名原则

①字母、数字、下划线组成

②不能以数字开头

③大小写敏感

④长度由编译器决定

⑤不以关键字命名

4.匈牙利命名法：首字符为小写的类型的首字母，单词首字母大写。

例：cInChar iReturnValue

5. **所有变量必须在使用前声明。**C89规定必须在块开头声明，C99规定使用前声明。

6.可以一次声明多个相同类型的变量，例：int a=1,b=2,c=3;

7.**浮点型常量e或E后的指数必须为整数。（表示10的幂次）**

8.格式说明%f 单精度浮点数，小数点后显示6位

9.**局部变量**：在程序块开头声明，只能在该程序块中访问。

10.在不同程序块中可以使用相同名字的不同局部变量。

11.**全局变量**：在程序块外部声明，在整个程序中访问。

局部变量和全局变量同名，全局变量在MAIN函数中失效。

12.用运算符把变量和字面常量结合形成C表达式。

13.表达式后加上分号；形成一条语句。

14.语句表达了一个被计算机执行的完整工作单元。

15.**空语句** **；**（不执行任何运算）

16.**复合语句**：一条或多条简单语句包围在大括号{ }内形成的块。

17.一条复合语句等价于一条简单语句。

18.**运算符**

①**优先级**：运算符被计算的顺序。

②**结合型**：优先级相同的运算符被计算的顺序。

19.用圆括号( )保证运算正确，其优先级最高，用于增强代码可读性。

20.**赋值运算符** **=**

先计算右值，再将右值赋给左值对象。

**赋值表达式的值等于所赋的值。**

**赋值时变量类型保持不变。**浮点数赋给整数时去小数部分取整。

21.**算术运算符 + - \* / %**

%：取模运算（整数除法求余数）

用C执行两个整数的除法运算，小数部分被忽略。

22.**关系运算符 == <= >= !=**

结果若真则值为1，结果若假则值为0。

23.**逻辑运算符 ！ && ||**

①!x 逻辑非：x若不等于0则值为0，x若等于0则值为1。

②x&&y 逻辑与：x、y若均不等于0则值为1，否则值为0。

③x||y 逻辑或：x、y若均等于0则值为0，否则为1。

24.**自增/自减运算符 ++ --**

i++i++i=i+1

i----1i=i-1

**前缀++i和- -i值为i+1和i-1的值。**

**后缀i++和i- -值为i的值。**

25.**特殊赋值运算符 += -= \*= /= %=**

26.**条件表达式**

x?y:z x若不等于0，则表达式的值等于y的值；x若等于0，则表达式的值等于z的值。

27**常量**：程序在执行中不会改变的值。

① 字面常量：从字面上出现在源代码中无名子的数值。

② 预处理指令#define创建宏代替

#define X Y 用Y代替X

编程风格：所有宏名用大写字母

③ 限定修饰词const：使变量在程序执行过程中不会改变数值。

**声明时必须初始化。**

28.**存储类**：指明当前包含这个变量的程序块执行完毕时该变量是否失去值。

①**静态变量**调用之间保留其值。

全局变量、static修饰的局部变量。

开始执行程序时被初始化为0。

②**自动变量**块结束后值丢失。

缺省情况下的局部变量。

不被初始化。（具有未知的值）

29.**系统分解法**：**自顶向下，逐步求精**

30.**混合类型表达式运算须要类型提升。**（整型→浮点型）

# 二、程序设计思想

## 第四章 程序化结构设计和控制结构

1.**顺序结构**

2.**条件结构**

**action是一条简单或符合语句。**

①**if(condition)**

**action;**

②**if(condition)**

**action\_if;**

**else**

**action\_else;**

③**if(condition\_1)**

**action\_1 ;**

**else if(condition\_2)**

**action\_2;**

**else if(condition\_3)**

**action\_3;**

**…**

**else**

**action\_else;**

**用条件表达式 ? : 效率更高。**

3.**重复结构**

循环可以嵌套。

**loop\_body**是一条简单或符合语句

①**while(test)**

**loop\_body;**

适用于标识控制循环，不知循环次数。

test T loop\_body

F

②**for(init;end-test;re-init)**

**loop\_body;**

适用于计数器控制循环，已知重复次数。

F

c init end-test T loop\_body re-init

③**do**

**loop\_body;**

**while(test);**

**while(test)后有分号；。**

loop\_body test F

T

4.其他控制结构

①**switch(expression){**

**case const\_1:**

**action\_1; break;**

**case const\_2:**

**action\_2; break;**

**case const\_3:**

**action\_3; break;**

**…**

**default:**

**action\_default;**

**}**

使编译器器跳过一些测试以优化代码。

代替级联的if-else。

break;可选，无break；时继续执行下一个case。

**expression必须为整型（int或char）**

**case后必须是互不相同的常量表达式。**

②**break；** 直接跳出当前循环或switch结构。

③**continue;** 停止当前循环，并直接开始下一次循环。

## 第五章 测试与调试

1.软件开发过程：软件需求分析，软件设计，程序编码，软件测试，软件维护。

2.**缺陷**（**bug**）：程序的错误。

3.**测试**（**test**）:找出缺陷的过程。

4.**调试**（**debug**）：去除错误的过程。

5.

6.**黑盒测试**：检查程序是否满足其输入和输出规格说明。

7.**白盒测试**：保证每一行代码都经过某个级别的测试。

插入的用来检测错误的代码。

8.源代码级别调试器：允许程序在可控环境下执行的工具，

9.**断点**：在程序执行过程中被指定的临时停止点。

10.**条件断点**：特定条件成立才暂停于某行。（隔离被怀疑执行有误的情况）

11.**观察点**：在任意特定条件为真的地方停止的点 。

12.**单步**：**从断点开始**，一次处理一条语句。

①**Step Into**：进入函数内部并在函数中单步执行。

②**Step Over**：不进入函数内部，将函数调用作为一步执行完，用于跳过被认为不包含错误的函数。

③**Step Out**：在函数内部一步完成函数调用。

13.防御性程序设计

①初始化所有变量

②写注释

③避免全局变量

④对齐左右大括号

⑤避免假设

⑥注意编译器警告信息

14.**常见低级错误**

(1)错误：if(x**=**’3’) i++; **//等于与赋值**

正确：if(x**==**’3’) i++;

(2)错误：if(**0<=n<=2**) i++; **//连续不等式**

正确：if(**0<=n&&n<=2**) i++;

(3)错误：if(**year==’7’||’2’**) i++; **//多项或**

正确：if(**year==’7’||year==’2’**) i++;

(4)错误：int i=0,j=0;

for(;i<=5;i++){

for(;j<=5;j++){

printf(“%d%d”,i,j);

}

}

正确：int i=0,j=0;

for(;i<=5;i++){

for(;j<=5;j++){

printf(“%d%d”,i,j);

}

**j=0; //内层循环再次初始化**

}

(5)错误：double b=**4**/**3**\*2.1;

正确：double b=**4.0**/**3.0**\*2.1; **//常量计算类型转换**

(6)错误：int main(){

int x[10];

function(**x[10]**); **//传递数组时要用地址**

}

void function(int cox[]){

cox[1]=0;

}

正确：int main(){

int x[10];

function(**x**);

}

void function(int cox[]){

cox[1]=0;

}

(7)错误：int main(){

char string[3];

printf(“%s\n”,**function(string)**);

}

**char \*function(char \*string)**{

char str[3]={1,2,3}

**return str**; **//不可return局部变量指针**

}

正确：int main(){

char string[3];

**function(string);**

printf(“%s\n”,**string**);

}

**void function(char \*string)**{

**string[0]=1; string[1]=2; string[2]=3;**

}

(8)错误：#include<stido.h>

**#define STRING abc**

int main(){

**char str[4]=”STRING”; //宏替换不替换字符串中内容**

}

正确：#include<stido.h>

**#define STRING “abc”**

int main(){

**char str[4]= STRING;**

}

(9)错误：char string[**3**]=”123”; **//要为空字符留一个位置**

正确：char string[**4**]=”123”;

(10)错误：void function(int);

int main(){

int x=2;

**void function(x);**  **//函数引用不用指明类型**

}

void function(int i){

return ++i;

}

正确：void function(int);

int main(){

int x=2;

**function(x);**

}

void function(int i){

return ++i;

}

注：这部分内容可以看我总结的题目，另附于本文档之外。

此前学长上机系列源码也可以参考参考。 ——余东骏

# 三、机器语言与汇编语言部分

## 第六章 数据的机器级表示

1.**比特**(**bit**)：信息最小单位（每一个0或1），也叫**位**或**二进制位**。接近0的电压视为0，远离0的电压

视为1。

2.位组合：有k位，则区分2k个值。

3.按位运算

①**按位与运算 &**

**位屏蔽**（**掩码**）：屏蔽位0，凸显位1。

②**按位或运算 |** （同或运算，包含或运算）

③**按位非运算 ~** （补运算）

按位取反

④**按位异或运算 ^**

**相异为1，相同为0**

按位取反：取反位1，保持位0

判断相同：两个数运算结果每一位都等于0则相同。

⑤**左移 <<**

右侧空位用0补充

⑥算术**右移 >>**

左侧空位符号扩展

逻辑右移：左侧空位用0补充

4.**布尔代数**：二进制逻辑运算。

与A·B，或A+B，非，异或(·B)+(A·)

①恒等律：A+0=A，A·1=A

②0/1律：A+1=1，A·0=0

③互补律：A+=1，A·=0

④交换律：A+B=B+A，A·B=B·A

⑤结合律：A+(B+C)=(A+B)+C，A·(B·C)=(A·B)·C

⑥分配律：A·(B+C)=(A·B)+(A·C)，A+(B·C)=(A+B)·(A+C)

⑦德摩根律：·=

5.**数据类型**：计算机对信息的编码表示法。

6.**无符号整数**：有k位，表示0~2k-1

7.**有符号整数**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 原码 | 反码 | 补码 |
| 正数（首位0） | 不变 | 不变 | 不变 |
| 负数（首位1） | 首位变1 | 按位取反 | 按位取反后加1 / 右侧若干0和第一个1不变，其他位取反 |

8.**计算机采用二进制补码表示有符号整数**。有k位，能表示-2k-1~2k-1-1

***1000…：绝对值最大的负数（最大正整数的相反数-1）***

1111…：-1

0000…：0

|  |  |  |  |
| --- | --- | --- | --- |
| 20=1 | 24=16 | 28=256 | 2-1=0.5 |
| 21=2 | 25=32 | 29=512 | 2-2=0.25 |
| 22=4 | 26=64 | 210=1024 | 2-3=0.125 |
| 23=8 | 27=128 | 211=2048 | 2-4=0.0625 |

9.**ALU**：**算数逻辑单元**

10.保证ALU正确做加法的运算原则

①绝对值相同符号相反的两个整数相加得0。

②每个数加1得到正确结果。（**首位的进位总是被忽略**）

11.二进制转化为十进制：按各位权重相加。

12.十进制转化为二进制

整数部分：除2取余，由下到上。

小数部分：乘2取整，依次排列。

例：45.8125d=0101101.1101b=1.011011101X25b

13.算数进位

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 源A+源B+ carry in | 0 | 1 | 2 | 3 |
| 结果位 | 0 | 1 | 0 | 1 |
| carry out | 0 | 0 | 1 | 1 |

14.**符号扩展**（**SEXT**）

正数以0向左扩展

负数以1向左扩展。

**不同长度补码整数作加法运算需先符号扩展。**

15.**溢出**：加法结果超出表示范围。

表现形式：两个正数相加得负数，两个负数相加得正数。

**一个正数与一个负数相加永不溢出。**

16.**ASCII码**：**美国标准信息交换码**，所有计算机处理单元和输入输出设备转换字符的标准码。

键盘上每个键被唯一个ASCII码识别，8个二进制位表示，首位为0。

0-32号、127号：通讯专用字符或控制字符。

33-126号：可见字符。

17.**定点小数**小数点对齐作加法。

18.**浮点数**用来表示精度少、位数多的小数。

19.**IEEE**：**国际电气和电子工程师协会**

20.浮点数算数运算标准：IEEE-754浮点数。

21.**单精度浮点数**

范围：10±38

精度：7位

符号s（1位，0正1负）+指数域exponent（8位无符号整数）+分数域fraction（23位）

22.**双精度浮点数**

范围：10±306

精度：15位

符号s（1位，0正1负）+指数域exponent（11位无符号整数）+分数域fraction（52位）

23.浮点数运算步骤

①使指数相等

②分数运算

③结果规格化

④丢失位舍入

⑤判断指数溢出

24.**十六进制**和**八进制**方便人们手工处理二进制位。

25.一个十六进制位对应四个二进制位

|  |  |  |  |
| --- | --- | --- | --- |
| 0000=0 | 0100=4 | 1000=8 | 1100=C |
| 0001=1 | 0101=5 | 1001=9 | 1101=D |
| 0010=2 | 0110=6 | 1010=A | 1110=E |
| 0011=3 | 0111=7 | 1011=B | 1111=F |

26.十六进制字面常量书写时使用前缀0x或x，在C语言中十六进制字面常量使用前缀0X或0x。

27.一个**八进制**位对应三个二进制位

|  |  |  |  |
| --- | --- | --- | --- |
| 000=0 | 010=2 | 100=4 | 110=6 |
| 001=1 | 011=3 | 101=5 | 111=7 |

28.字符串：字符序列，以NUL（0）结束

29.图像：像素矩阵

30.声音：定点数序列

31.C语言中的数据类型采用位数与计算机指令集结构和编译器有关。

int：二进制补码整型

char：ASCII码（进行运算时与对应ASCII码等值）

double：双精度浮点数

32.格式说明

%d 十进制补码整数

%x 十六进制

%o 八进制

%c 字符

%f 单精度浮点数

%lf 双精度浮点数

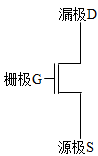
## 第七章 数字逻辑电路

1.**集成电路**：用工艺将电路中的元件及布线互连，固定在半导体晶片或介质基片上，封装在管壳内，称为具有所需电路功能的微型结构。

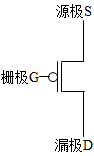
2.**摩尔定律**：集成电路上可容纳的晶体管数目约每隔18个月增加一倍。即，价格不变时，每一美元能买到的电脑性能每隔18个月翻两倍以上。

3.**微处理器由MOS**（**金属氧化物半导体**）**晶体管组成。**

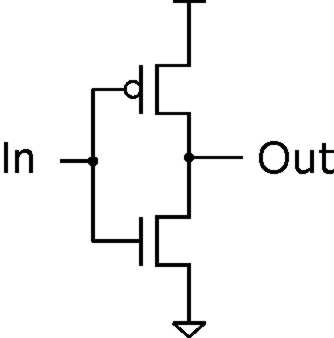
4.**N型MOS晶体管源极不能接电源正极**。栅极加2.9V电压，漏极和源极导通；栅极加0V电压，漏极和源极断开。

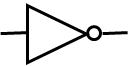


5. **P型MOS晶体管源极不能接地。**栅极加2.9V电压，漏极和源极断开；栅极加0V电压，漏极和源极导通。

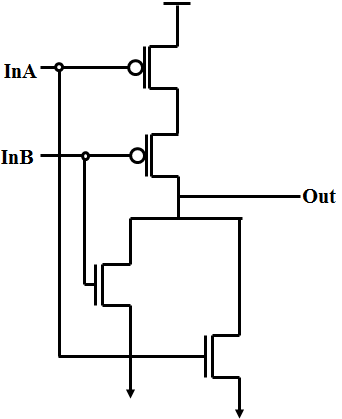


6.**CMOS**（**互补金属氧化物半导体**）电路：既包含P型晶体管，又包含N型晶体管。

7.**反相器**（**非门**）



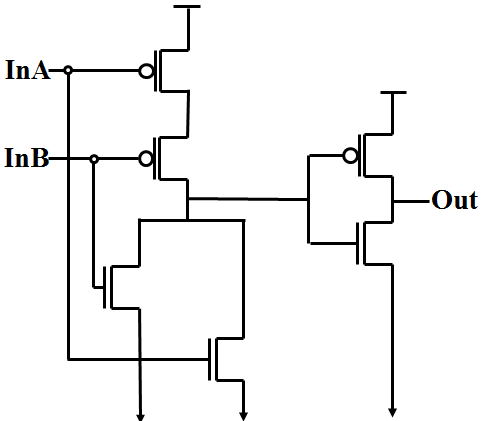
|  |  |
| --- | --- |
| In | Out |
| 1 | 0 |
| 0 | 1 |

8.**非或门**





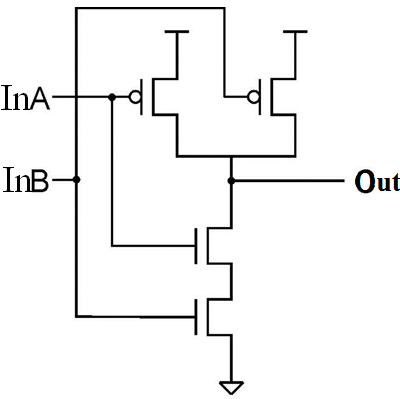
|  |  |  |
| --- | --- | --- |
| InA | InB | Out |
| 1 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

9.**或门**（非或门+反相器）



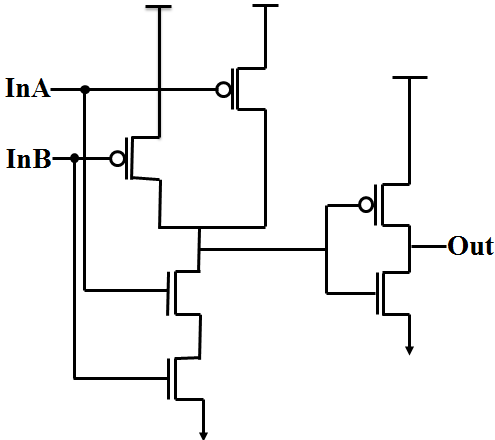
|  |  |  |
| --- | --- | --- |
| InA | InB | Out |
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |



10.**非与门**



|  |  |  |
| --- | --- | --- |
| InA | InB | Out |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |



11.**与门**（非与门+反相器）



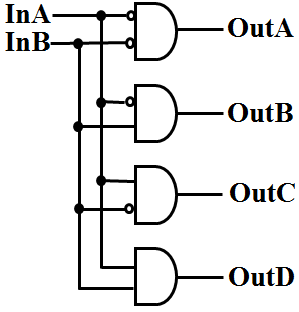


|  |  |  |
| --- | --- | --- |
| InA | InB | Out |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

12.**组合逻辑电路**（**判定元件**）不存储信息，输出值只由当前输入值决定。包括**译码器**、**多路选择器**、**全加法器**。

13.**译码器**

只有一个输出为1，其他全为0。若有n个输入，则有2n个输出。

作用：判断某个位组合。

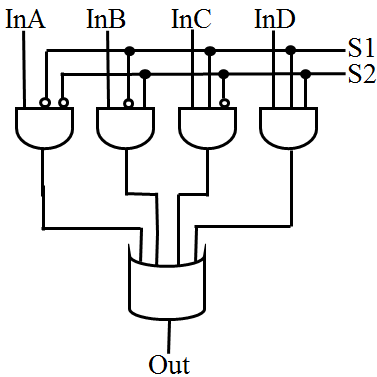
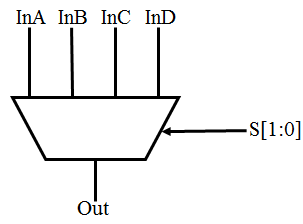
n=2，2-4译码器：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| InA | InB | OutA | OutB | OutC | OutD |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |

14.**多路选择器**

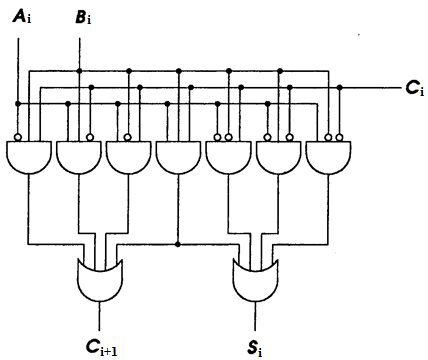
由选择信号决定哪个输入连接到输出。

n条选择线，2n个输入，1个输出。

n=2，4-1多路选择器：

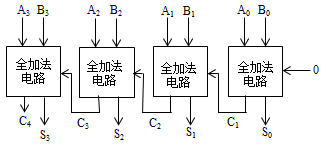
|  |  |  |
| --- | --- | --- |
| S1 | S2 | Out |
| 1 | 1 | InD |
| 1 | 0 | InC |
| 0 | 1 | InB |
| 0 | 0 | InA |

15.**全加法电路**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| In | | | Out | |
| ai | bi | ci | ci+1 | si |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

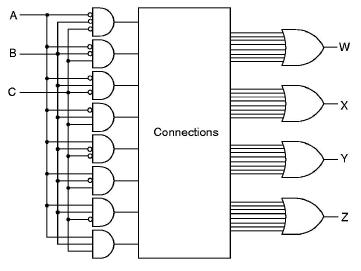
4位加法电路：



16.**PLA**（**可编程逻辑阵列**）：可以实现任意逻辑运算的通用组件。

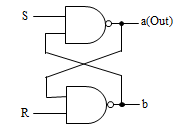
与阵列：每个与门有n个输入，一共2n个与门。

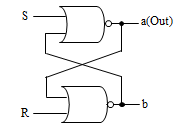
或阵列：输出。

n=3：

17.**逻辑完备性**：提供足够多的与或非门，可以实现任意逻辑运算。即，门集合{与、或、非}在逻辑上完备。18.任意逻辑运算可以通过PLA实现。

19.**基本存储元件**：能够存储信息的元件。包括**R-S锁存器**、**门控D锁存器**、**寄存器**。

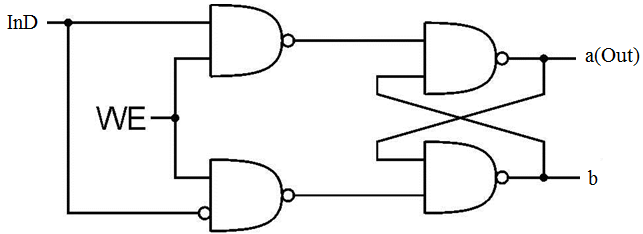
20.**R-S锁存器**

①非与门：

|  |  |  |
| --- | --- | --- |
| S | R | 状态 |
| 1 | 1 | 静止状态， a与b相异，锁存a值 |
| 1 | 0 | 存储0（清空），a=0，b=1 |
| 0 | 1 | 存储1，a=1，b=0 |
| 0 | 0 | 取决于晶体管电子特性（禁止此操作） |

②非或门：

|  |  |  |
| --- | --- | --- |
| S | R | 状态 |
| 1 | 1 | 静止状态， a与b相异，锁存a值 |
| 1 | 0 | 存储1，a=1，b=0 |
| 0 | 1 | 存储0（清空），a=0，b=1 |
| 0 | 0 | 取决于晶体管电子特性（禁止此操作） |

21.**门控D锁存器**

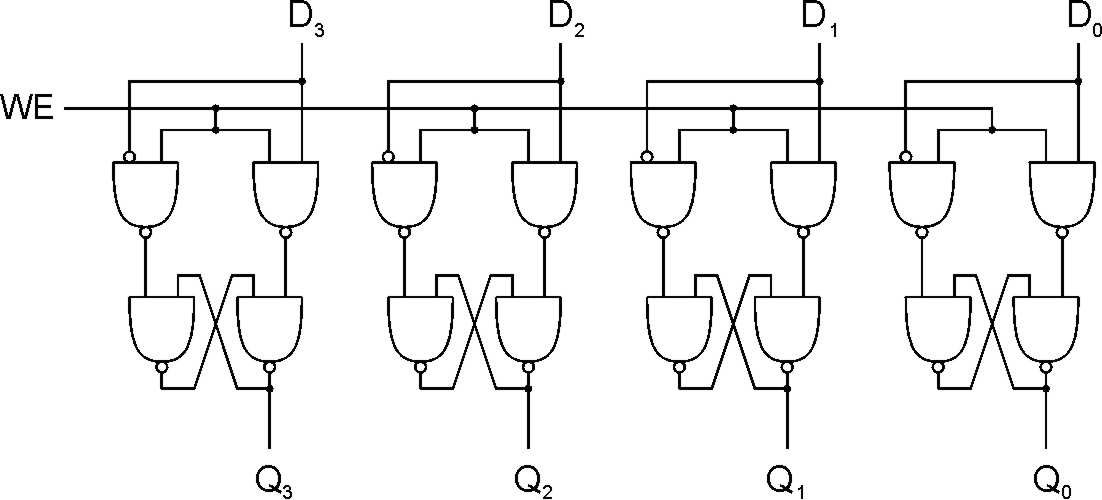
|  |  |  |
| --- | --- | --- |
| InD | WE | 状态 |
| 1/0 | 0 | 静止状态， a与b相异，锁存a值 |
| 1 | 1 | 存储1，a=1，b=0 |
| 0 | 1 | 存储0（清空），a=0，b=1 |

优点：对R-S锁存器何时设为1和0进行控制，避免R=S=0的情况。

22.**寄存器**

多个门控D锁存器存储多位，共享WE。

4位寄存器Q[3:0]：



23.**存储器**由一定数量的单元组成，每个单元可被唯一识别，都有存储一个数值的能力。

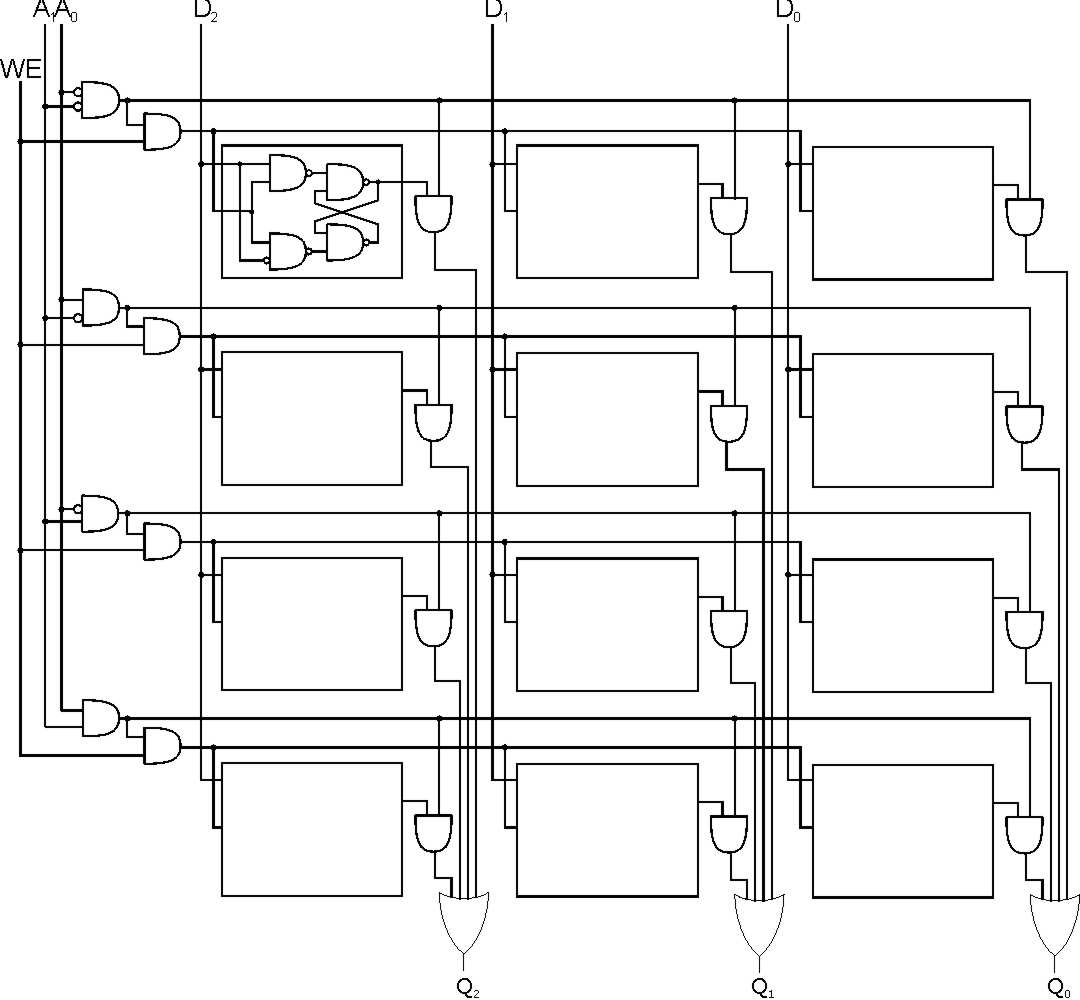
4×3存储器：

结构：→门控D锁存器→4-1多路选择器→门控D锁存器→4-1多路选择器→门控D锁存器→4-1多路选择器

有22个存储单元，3位寻址能力。

2根地址线[1:0]：译码器

3根数据线[2:0]：门控D锁存器。



24.**字段**：位组合的子单元。

25.**地址**：和每个单元联系在一起的唯一标识符。

26.**地址空间**：唯一可被识别的单元总数。

n位地址，则地址空间为2n。

27.**寻址能力**：存储在每个单元中的位数。

大多数存储器字节可寻址。

28.存储单元单位：1K=210，1M=220，1G=230，1T=240

29.**字节**（**byte**）：1B=8bit

30.**单元组**（**nibble**）：4bit

31.4GB存储容量表示有2^32个存储单元，字节可寻址。

32.译码器某一位地址输出的1是被寻址的**字线**。

33.**SRAM**（**静态随机访问存储器**）

静态：只要供电，内部数据就不会丢失。

随机：任意顺序访问而不关心前次访问的单元。

34.**时序逻辑电路**既能存储信息，又能处理信息。输出值由当前输入和存储元件中的值共同决定。

35.**状态**：某一特定时刻，系统内所有相关部分的一个瞬态图。

36.状态数目必须有限，原因是存储元件容量有限。

37.**有限状态机**组成元素

①有限数目的状态。

②有限数目的外部输入。

③有限数目的外部输出。

④明确定义的状态转换函数。

⑤明确定义的外部输出函数。

38.**状态图**中圆对应一个状态，弧线与箭头确定一个状态的转换。

39.**时钟电路**：触发状态从一个向下一个转换的机制。

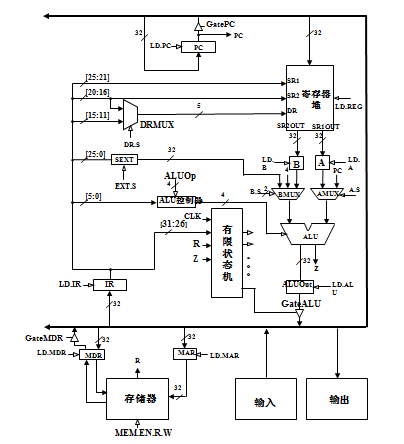
40.**时钟周期**：重复的时间间隔序列中的一个时间间隔。

41.存储元件使用**主从触发器**，即2个门控D锁存器。

## 第八章 冯·诺依曼模型

1.**数据通路**：计算机内部用于处理信息的所有元件总和。

2.**DLX子集数据通路**



3.**PC**：**程序计数器**（**指令指针**），32位，存储下一条指令所在地址的寄存器。

4.**IR**：**指令寄存器**，32位。保存正在处理的指令。

5.多路选择器

6.DLX数据通路采用**总线结构**，**多时钟周期**实现方案。

7.**总线**：存储器和处理器，处理器和I/O设备间的通信。用黑色箭头框表示。

优点：功能多成本低。

缺点：产生通信瓶颈（总线上一次只传输一个值）。

8.**三态设备**（三角形）：使计算机的控制逻辑一次只允许一个提供者向总线提供信息。位于每一个提供数据给总线的组件的输入箭头处。

9.**LD.**X（**加载使能**）：使信息允许加载时为1，不允许加载时为0。

10.**Gate**.X：使数据与总线相连。

11.实心箭头：通信流动数据元素。

12.空心箭头：控制数据元素处理的控制信号。

13.**冯诺依曼模型主要思想**：把程序和数据都作为一个二进制序列存储在存储器里，在控制单元的引导下一次执行一条指令。

14.由指令组成的程序和程序所需的数据位于存储器中。

15.**MAR**（**地址寄存器**）：32位（DLX有232个存储单元）。

16.**MDR**（**数据寄存器**）：32位，存储连续4个单元的数据，或单元数据符号扩展的结果。

17.**处理单元执行信息的实际处理。**DLX只包含ALU和寄存器堆。

18.**ALU**：**算数逻辑单元**。

19.**字长**：ALU正常处理的信息量大小。ALU有32位字长。

20.**字**：信息中的每一个元素。**1个字=4个字节**，32位。

21.**寄存器堆**：靠近ALU提供少量存储空间用于临时存取数据的一组寄存器。

避免对存储器不必要的长时间访问。

每个寄存器包含一个字。

共32个通用寄存器（R0，R1，…，R31），32个浮点寄存器（F0，F1，…，F31）。

22.**控制单元指挥信息的处理。**

23.指令包括操作码和操作数。

24.寻址模式：计算机将要读取/存储的存储单元的地址的机制。

25.**多周期实现方案**

①**取指令**

查询存储器，把指令放进MDR。

把MDR的内容加载入IR。

②**译码**：识别指令而确定下一步。

**取寄存器**：为后面的阶段进行获取操作数的操作。（有的操作结果后面不会用到，但不浪费时间，原因：操作同时进行）

③**执行**：对上一阶段得到的寄存器的值执行算数/逻辑运算。

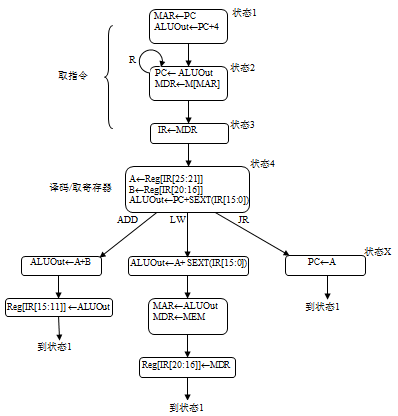
**有效地址**：计算出处理指令所需存储单元的地址。

**完成分支**：完成分支跳转。

④**访问内存**：获取内存中的数据。

⑤**存储结果**：将结果写入指定目标。

26.DLX状态图（需要多个时钟周期，一个状态内的操作同时执行）



**状态①**

**MAR←PC**

有限状态机使Gate.PC和LD.MAR设为1，PC与总线相连，当前时钟周期结束时将总线内容写入MAR。

**ALUOut←PC+4**

有限状态机使A.S（AMUX选择信号）设为1，选择来自PC的输入；

有限状态机使B.S（BMUX选择信号）设为01，选择输入4；

有限状态机使ALUOp设为0001，在ALU中执行加法；

有限状态机使LD.ALU设为1，结果存储与ALUOut中。

**状态②**

**PC←ALUOut**

有限状态机使Gate.ALU和LD.PC设为1，将ALUOut的值写入PC。

**MDR←M[MAR]**

有限状态机使MEM.EN.R.W（写使能）设为0，LD.MDR设为1，读存储器。

**R**

读存储器需要多个时钟周期，读取时就绪信号R设为0，读取结束时设为1。

**状态③**

**IR←MDR**

R为1时，有限状态机使Gate.MDR和LD.IR设为1，当前时钟周期结束时，IR被写入。

**状态④**

**A←Reg[IR[25:21]]**，**B←Reg[IR[20:16]]**

有限状态机使LD.A和LD.B设为1，从IR[25:21]和IR[20:16]分别获得源操作数传给ALU的A、B寄存器。

**ALUOut←PC+SEXT(IR[15:0])**

有限状态机使A.S设为1，选择来自PC的输入；

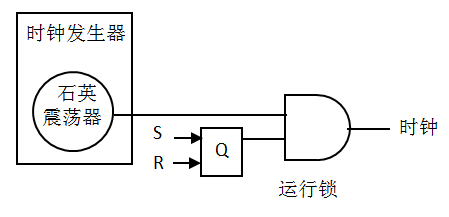
有限状态机使EXT.S设为0，B.S设为00，选择IR[15:0]符号扩展结果；

有限状态机使ALUOp设为0001，在ALU中执行加法；

有限状态机使LD.ALU设为1，结果存储于ALUOut中。

27.**时钟电路**

|  |  |
| --- | --- |
| Q=1 | 时钟电路输出与时钟发生器输出一样 |
| Q=0（清0） | 时钟电路输出为0，即停止指令运行。（HALT） |



## 第九章 指令集结构

1. **ISA**（**指令集结**构）指明了在一台机器上编写软件时所要注意的全部信息。

2.DLX存储器组织有232个地址空间单元，8位寻址能力（字节可寻址）。

3.存储顺序：**高位优先**（字的高位字节放在内存低地址端，低位字节放在高地址端）。

4.**边界对齐：字的起始地址必须是4的倍数（二进制末尾为00）。**

5.单精度浮点数要1个浮点寄存器，双精度浮点数要2个浮点寄存器。

6.指令集

7.DLX数据类型支持二进制补码整数（8位，16位，32位），单、双精度浮点数。

8.**DLX指令可定义127条（操作码和函数），但DLXISA只定义了91条，未定义的被保留。**

9.**I-类型**（**立即数操作**）：[31:26]操作码，[25:21]源寄存器，[20:16]目标寄存器，[15:0]16位立即数。

10.**R-类型**（**寄存器操作**）：[31:26]000000，[25:21]源寄存器1，[20:16]源寄存器2，[15:11]目标寄存器，[10:6]00000（未用），[5:0]函数。

11.**J-类型**（**跳转操作**）：[31:26]操作码，[25:0]26位地址偏移量。

12.指令类型

①算术/逻辑运算：处理整数。

②数据传送：在存储器和寄存器、寄存器和I/O设备间传送数据。

③控制：改变指令执行顺序。

④浮点：处理浮点数。

13.DLX只支持**“基址+偏移量”寻址模式**。

14.同一条指令，同一个寄存器可以同时作为源寄存器和目标寄存器。

15.001011 SR SR 1111 1111 1111 1111 (xori sr,sr,xFFFF) 对SR按位取反。

16.逻辑运算时，结果为真设目标寄存器值为1，结果为假设目标寄存器值为0。

17.**srai按位算术右移，左侧空位符号扩展。每右移一次表示除以2一次。**

18.**srli按位逻辑右移，左侧空位补0。**

19.**slli按位左移，右侧空位补0。每左移一次表示乘以2一次。**

20.**lhi加载高位立即数，将imm16左移16位，结果保存于DR中。与addi指令一起使用将较大的立即数赋给寄存器。**

21.**加载**：将数据从存储器移动到寄存器的过程。

22.**存储**：将数据从寄存器移动到存储器的过程。

23.lb和sb：加载和存储8位字节，一个存储单元和一个寄存器之间传送数据。

24.lw和sw：加载和存储32位子，四个连续的存储单元和一个寄存器之间传送数据。

25.lb，sb，lw，sw将16位偏移量符号扩展至32位，与基址寄存器相加，获得存储器起始地址。

26.lw和sw计算所得的存储器起始地址必须是4的倍数。

27.**绝对地址**：当基址寄存器为r0（或值为0）时，Imm16就是访问存储器的地址。

28.控制指令

29.**beqz条件假跳转，若SR1值为0，则从PC+SEXT[Imm16]+4开始执行；若SR1值不为0，则PC+4开始执行。**

30.**bnez条件真跳转，若SR1值不为0，则从PC+SEXT[Imm16]+4开始执行；若SR1值为0，则PC+4开始执行。**

31.**地址限制**：只能跳转至立即数足够表示的范围内（当前215+3~-215+4的单元）。

32.在beqz中使SR1=R0，成为无条件分支。

33.**jr无条件跳转**，跳转至SR1值作为地址的指令处。

34.**j无条件跳转，跳转至PC+SEXT[PCOffset26]+4处。**

35.无条件跳转比条件分支范围更广。

36.trap指令，调用一个操作系统服务例程，[25:0]是TRAP向量。

37.C语言修饰符**long**和**short**可以扩大或缩小数据类型位数，**仅当特定系统支持才有效**。

38.unsigned int 表示无符号整数。

39.类型提升的本质只能从较短的类型转换成较长的类型。

## 第十章 机器语言程序设计

1.**顺序结构**

2.**条件结构**

生成条件指令→条件分支指令→子任务2→J指令→子任务1

条件分支指令偏移量为（子任务2指令数+1）×4。

J指令偏移量为子任务1指令数×4。

3.**重复结构**

生成条件指令→条件分支指令→子任务→J指令

条件分支指令偏移量为（子任务指令数+1）×4。

J指令可以指向生成条件指令或条件分支指令。

4.**设置值**：不考虑之前运行的其他部分而直接设置数据。

5.**运行**：在TRAPx00或断点处停止。

6.**按步运行**：运行一定数量的指令后停止。

7.**设置断点**：在特定指令下停止。

8.**Step Over**：单步执行但将子例程当做一步来执行。

9.**Step In**：进入子例程并在子例程内部单步执行。

10.**Step Out**：已进入子例程时一步完成子例程。

11.**输入数字字符时保存了其ASCII码，必须先转换再计算，数值=ASCII码—x30。**

12.**编写子例程时若还调用了其他例程，应当先保存r31。**

13.**子例程占用寄存器时需要保存并恢复。**

## 第十一章 汇编语言

1.**符号地址**：使用符号表示存储单元的地址。

2.高级语言编程对用户更加友好，但放弃了对指令的精确控制。

3.程序执行前将汇编语言翻译成适合于期望执行计算机的ISA程序。翻译程序是**汇编器**，翻译过程是**汇编**。

4.指令不区分大小写。

5.自由格式，单词间和行间空格不改变程序意义。

6.寄存器共32个，R0 、R1…R31。

7.**寄存器限制**

①**输入输出：R4**

②**局部变量**：**R16、R17、R18、R19、R20、R21、R22、R23**

③**临时值**：**R8、R9、R10、R11、R12、R13、R14、R15、R24、R25**

④**全局指针：R28**

⑤**栈指针：R29**

⑥**帧指针**（**框架指针**）**：R30**

8.**立即数基符号**：**# 十进制**；**x 十六进制**；**b 二进制**

9.**算数/逻辑运算指令**

**I-类型**：OPCODE DR SR1 Imm/LABEL

**R-类型**：OPCODE DR SR1 SR2

**LHI指令**：LHI DR Imm/LABEL

10.**数据传送指令**

**加载指令：**LW/LB DR Imm/LABEL(SR1)

**存储指令：**SW/SB Imm/LABEL(SR1) DR

**LW和SW指令基址寄存器+偏移量得到的地址必须是4的倍数。**

11.**控制指令**

**条件分支指令：**BEQZ/BNEZ SR1 LABEL

**J指令：**J LABEL

**JR指令：**JR SR1

**TRAP指令：**TRAP Imm（**trap向量**）

12.**标记（LABEL）**：用来明确标识存储单元的符号名。

13.标记由**字母**、**数字**、**下划线**组成，**以字母、下划线、$开头**，**以冒号：结尾**。

14.**指令操作码不可以作为标记名**。

15.**注释以分号；开头**，至行末的内容被忽略。

16.使用空格将程序对齐。

17.注释目的：提高可读性。

18.**伪操作**有助于汇编器把输入的字符串作为使用DLX汇编语言写的计算机程序，并将其翻译为DLX的ISA程序。汇编完成即被抛弃，程序执行过程中的伪操作不执行。

19.伪操作**以点.开头。第一行必须为.data address。必须有.global main**20.**汇编过程**至少扫描两趟。

第一趟：构建符号表。

第二趟：把汇编语言指令翻译成机器语言指令。

21.**LC**：**地址计数器**

22.一条汇编语言指令可能翻译成多条机器语言指令。（大立即数由LHI实现）

23.**可执行映像**：当计算机开始一个程序的执行时的被执行实体。

24.链接过程由**链接器**完成。链接器为每个模块重新分配存储空间，因此.text address和.data address中的address是可省略的。

25.**栈**是一种抽象数据类型。

26.**栈协议**：**后进先出**（**LIFO**）

27.**栈顶**（**TOP**）：数据只能经栈顶处访问。

28.**硬件栈**：push时数据向上移动，pop时数据向下移动，栈顶固定在底层。

29.**栈指针**（**R29**）：存储栈顶的单元地址。push时数据向上添加，栈顶上移一格，R2减4；pop时数据不进行物理移动，栈顶下移一格，R29加4，**原数据不删除**。

30.**push**（**压栈**）：把一个元素插入栈。

31.push: subi r29,r29,#4

sw 0(r29),SR

32.**pop**（**出栈**）：从栈中移出一个元素。

33.pop: lw DR,0(r29)

addi r29,r29,#4

34.当变量个数多于寄存器数目，需要使用存储器。**存储时边界对齐**。

35.DLX存储器组织

|  |  |  |
| --- | --- | --- |
| x0000 0000 | 系统空间 |  |
|  | 代码区 | ←PC  ←R28（全局指针，指向全局数据区起始地址） |
|  | 全局数据区  （存储所有静态存储类变量） |
|  | 堆  （动态分配存储单元） |  |
|  | ↓  ↑ | ←R29（栈指针，指向运行时栈栈顶）  ←R30（帧指针，指向运行时栈中） |
|  | 运行时栈  （存储缺省自动存储类变量） |
| xFFFF FFFF | 系统空间 |  |

## 第十二章 输入和输出

1.I/O设备至少包含两个设备寄存器。

①保存计算机和设备之间进行传输的数据。

②保存设备的状态信息。

2.访问I/O设备寄存器的两种机制。

①I/O指令。（Intel x86指令集，使用IN和OUT指令）

②用于在通用寄存器和存储器之间传送数据的数据传送指令。（DLX）

3.**内存映射**：每一个I/O设备寄存器都被分配一个存储器地址空间中的地址。

4.**DLX设备寄存器**（**每个I/O寄存器映射4个存储器存储单元**）

**xFFFF 0000：键盘状态存储器（KBSR）**

**xFFFF 0004：键盘数据寄存器（KBDR）**

**xFFFF 0008：显示器状态寄存器（DSR）**

**xFFFF 000C：显示器数据寄存器（DDR）**

**xFFFF 00F8：机器控制寄存器（MCR）**

5.**被映射的存储单元不能作为存储单元使用，加载数据时直接获得I/O数据寄存器中的数据。**

6.**异步**：I/O设备与微处理器不一致。

7.处理异步问题的**协议**或**握手机制**：用1位状态寄存器标识**就绪位**。

8.打印机状态寄存器需要2位表示打印是否完成和打印机是否出错。

9.**轮询**：通过处理器周期性检查状态位来判断是否执行I/O操作。

**由处理器完全控制和执行通信工作。**

缺点：浪费大量处理时间。

10.**中断驱动的I/O**：处理器做自己的工作直到被信号打断。

**由I/O设备控制交互。**

11.KBDR和DDR[7:0]存放数据，[31:8]都为0；KBSR和DSR[0]存放就绪位，[31:1]另有用途。

12.**DLX基本输入服务例程In**（轮询）：

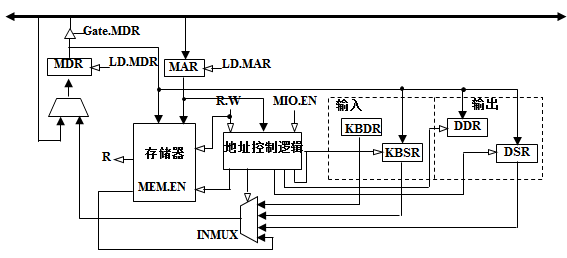
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01 |  | .data | x00003000 |  |
| 02 | SaveR1: | .space | 4 | ;保存寄存器的存储单元 |
| 03 | SaveR2: | .space | 4 |  |
| 04 | SaveR3: | .space | 4 |  |
| 05 | SaveR5: | .space | 4 |  |
| 06 | KBSR: | .word | xFFFF0000 |  |
| 07 | KBDR: | .word | xFFFF0004 |  |
| 08 | DSR: | .word | xFFFF0008 |  |
| 09 | DDR: | .word | xFFFF000C |  |
| 0A | Newline: | .byte | x0A | ;换行的ASCII码 |
| 0B | Prompt: | .asciiz | “ Input a character>“ | ;提示符字符串 |
| 0C | ;保存例程使用的寄存器 | | | |
| 0D |  | .text | x00003100 |  |
| 0E |  | sw | SaveR1(r0),r1 | ;保存例程所需寄存器 |
| 0F |  | sw | SaveR2(r0),r2 |  |
| 10 |  | sw | SaveR3(r0),r3 |  |
| 11 |  | sw | SaveR5(r0),r5 |  |
| 12 |  | lb | r2,Newline(r0) |  |
| 13 |  | lw | r5,DSR(r0) | ;测试输出寄存器是否就绪 |
| 14 | L1: | lw | r3,0(r5) |  |
| 15 |  | andi | r3,r3,#1 |  |
| 16 |  | beqz | r3,L1 | ;循环直到显示器就绪 |
| 17 |  | lw | r5,DDR(r0) |  |
| 18 |  | sw | 0(r5),r2 | ;光标移到新行 |
| 19 | ;输出提示符 | | | |
| 1A |  | addi | r1,r0,Prompt | ;提示符字符串起始地址 |
| 1B | LOOP: | lb | r2,0(r1) | ;输出提示符 |
| 1C |  | beqz | r2,Input | ;提示符字符串结束 |
| 1D |  | lw | r5,DSR(r0) |  |
| 1E | L2: | lw | r3,0(r5) |  |
| 1F |  | andi | r3,r3,#1 |  |
| 20 |  | beqz | r3,L2 | ;循环直到显示器就绪 |
| 21 |  | lw | r5,DDR(r0) |  |
| 22 |  | sw | 0(r5),r2 | ;输出下一个提示符 |
| 23 |  | addi | r1,r1,#1 | ;提示符指针加1 |
| 24 |  | j | LOOP | ;获取下一个提示符 |
| 25 | ;输入回显 | | | |
| 26 | Input: | lw | r5,KBSR(r0) |  |
| 27 | L3: | lw | r3,0(r5) |  |
| 28 |  | andi | r3,r3,#1 |  |
| 29 |  | beqz | r3,L3 | ;轮询直到一个字符被键入 |
| 2A |  | lw | r5,KBDR(r0) |  |
| 2B |  | lw | r4,0(r5) | ;将输入的字符加载到R4 |
| 2C |  | lw | r5,DSR(r0) |  |
| 2D | L4: | lw | r3,0(r5) |  |
| 2E |  | andi | r3,r3,#1 |  |
| 2F |  | beqz | r3,L4 | ;循环直到显示器就绪 |
| 30 |  | lw | r5,DDR(r0) |  |
| 31 |  | sw | 0(r5),r4 | ;将输入字符回显 |
| 32 | ;输出新行 | | | |
| 33 |  | lb | r2,Newline(r0) |  |
| 34 |  | lw | r5,DSR(r0) |  |
| 35 | L5: | lw | r3,0(r5) |  |
| 36 |  | andi | r3,r3,#1 |  |
| 37 |  | beqz | r3,L5 | ;循环直到显示器就绪 |
| 38 |  | lw | r5,DDR(r0) |  |
| 39 |  | sw | 0(r5),r2 | ;移动光标到新行 |
| 3A | ;恢复例程使用的寄存器 | | | |
| 3B |  | lw | r1,SaveR1(r0) | ;将寄存器恢复为原先的值 |
| 3C |  | lw | r2,SaveR2(r0) |  |
| 3D |  | lw | r3,SaveR3(r0) |  |
| 3E |  | lw | r5,SaveR5(r0) |  |
| 3F |  | jr | r31 | ;从TRAP返回 |

13.**DLX基本输出服务例程OUT**（轮询）：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01 |  | .data | x00002800 |  |
| 02 | SaveR1: | .space | 4 | ;保存寄存器的存储单元 |
| 03 | SaveR2: | .space | 4 |  |
| 04 | DSR: | .word | xFFFF0008 |  |
| 05 | DDR: | .word | xFFFF000C |  |
| 06 | ; |  |  |  |
| 07 |  | .text | x00002900 |  |
| 08 |  | sw | SaveR1(r0),r1 | ;保存例程所需寄存器 |
| 09 |  | sw | SaveR2(r0),r2 |  |
| 0A |  | lw | r1,DSR(r0) |  |
| 0B | START: | lw | r2,0(r1) | ;测试输出寄存器是否就绪 |
| 0C |  | andi | r2,r2,#1 |  |
| 0D |  | beqz | r2,START |  |
| 0E |  | lw | r1,DDR(r0) |  |
| 0F |  | sw | 0(r1),r4 |  |
| 10 |  | lw | r1,SaveR1(r0) | ;将寄存器恢复为原先的值 |
| 11 |  | lw | r2,SaveR2(r0) |  |
| 12 |  | jr | r31 | ;从TRAP返回 |

14.内存映射I/O数据通路

地址控制逻辑控制输入输出操作。



## 第十三章 自陷例程和中断

1.**硬件寄存器有特权，不拥有适当特权级别的程序不能访问。**

2.输入输出时使用TRAP指令使操作系统完成，用户程序不必知道实现细节，其他用户程序被保护，避免用户程序员不恰当行为的后果。

3.**服务调用**或**系统调用**：用户程序请求操作系统完成任务。

4.DLX有256个服务例程，用TRAP机制调用。

|  |  |  |
| --- | --- | --- |
| x06 | GETC | 从键盘读取一个字符，将其ASCII码复制到R4[7:0] |
| x07 | OUT | 将R4[7:0]中的字符输出到显示器 |
| x08 | PUTS | 将R4所指的地址开头的一个字符串输出到显示器，每个字符占用一个存储单元，字符串以x00终止 |
| x09 | IN | 输出“Input a character>”到显示器，从键盘读取一个字符后回显到显示器上，并将其ASCII码复制到R4[7:0] |
| x0A | GETS | 两个参数R4（字符串起始地址）和R5（长度n）,从键盘读取n-1个字符，如果输入小于n-1,，则至回车结束，读入缓冲区，并在末尾加上x00 |
| X00 | HALT | 输出“Halting the machine.”，并停止程序 |

5.**TRAP向量表**或**系统控制块**

①存储在**x00000000到x000003FF中**。

②**包括256个服务例程的起始地址**，每个地址占用4个存储单元，每个服务例程占用210个存储单元（数据区占x100，服务例程占x300，共x400）。

③**每个服务例程的数据段起始地址均为其代码段起始地址前x100个单元的位置。**

④部分TRAP向量表

|  |  |  |  |
| --- | --- | --- | --- |
| TRAP向量 | 向量表地址 | 服务例程地址 | 数据区地址 |
| x06 GETC | x0000 0018 | x0000 2500 | x0000 2400 |
| x07 OUT | x0000 001C | x0000 2900 | x0000 2800 |
| x08 PUTS | x0000 0020 | x0000 2D00 | x0000 2C00 |
| x09 IN | x0000 0024 | x0000 3100 | x0000 3000 |
| x0A GETS | x0000 0028 | x0000 3500 | x0000 3400 |
| x00 HALT | x0000 0000 | xFFFE 0100 | xFFFE 0000 |

6.TRAP指令执行步骤

①26位TRAP向量符号扩展到32位，再左移2位（乘以4）形成地址，该地址加载到MAR中。

②加载到MAR中的地址中的记录被读取，并将其加载到MDR中。

③通用寄存器R31被加载为PC当前内容（已经被加4）。

④MDR内容被加载到PC中，并完成指令的执行。

⑤返回用户程序。

7.返回用户程序时使用指令**JR R31**，用助记符**RET**表示，**调用TRAP指令后R31中的数据被破坏**。

8.**MCR（机器控制寄存器）的[0]位存储运行锁。**

9.**HALT服务例程**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01 |  | .data | xFFFE0000 |  |
| 02 | SaveR1: | .space | 4 | ;保存寄存器的存储单元 |
| 03 | SaveR2: | .space | 4 |  |
| 04 | SaveR4: | .space | 4 |  |
| 05 | SaveR31: | .space | 4 |  |
| 06 | Newline: | .byte | x0A |  |
| 07 | Message: | .asciiz | “Halting the machine. ” | |
| 08 |  | .align | 2 |  |
| 09 | MCR: | .word | xFFFF00F8 |  |
| 0A | ; |  |  |  |
| 0B |  | .text | xFFFE0100 |  |
| 0C |  | sw | SaveR1(r0),r1 | ;保存例程所需寄存器 |
| 0D |  | sw | SaveR2(r0),r1 |  |
| 0E |  | sw | SaveR4(r0),r1 |  |
| 0F |  | sw | SaveR31(r0),r31 |  |
| 10 | ; |  |  |  |
| 11 | ;输出停机消息 | | | |
| 12 | ; |  |  |  |
| 13 |  | lb | r4,Newline(r0) |  |
| 14 |  | trap | x07 |  |
| 15 |  | addi | r4,r0,Message |  |
| 16 |  | trap | x08 |  |
| 17 |  | lb | r4,Newline(r0) |  |
| 18 |  | trap | x07 |  |
| 19 | ; |  |  |  |
| 1A | ;清空xFFFF 00F8的0位，停机 | | | |
| 1B | ; |  |  |  |
| 1C |  | lw | r1,MCR(r0) |  |
| 1D |  | lw | r2,0(r1) | ;加载MCR值到R2中 |
| 1E |  | andi | r2,r2,#-2 | ;清空MCR[0] |
| 1F |  | sw | 0(r1),r2 | ;将R2值存储到MCR中 |
| 20 | ;**至此已经停机，若未重置以下指令不会执行** | | | |
| 21 | ;从HALT例程返回 | | | |
| 22 | ; |  |  |  |
| 23 |  | lw | r1,SaveR1(r0) | ;将寄存器恢复为原先的值 |
| 24 |  | lw | r2,SaveR2(r0) |  |
| 25 |  | lw | r4,SaveR4(r0) |  |
| 26 |  | lw | r31,SaveR31(r0) |  |
| 27 |  | jr | r31 | ;从TRAP返回 |

10.**HALT指令使用了R1、R2、R4、R31且停机时数据被破坏。**

11.**PUTS服务例程**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01 |  | .data | x00002C00 |  |
| 02 | SaveR2: | .space | 4 | ;保存寄存器的存储单元 |
| 03 | SaveR3: | .space | 4 |  |
| 04 | SaveR4: | .space | 4 |  |
| 05 | SaveR5: | .space | 4 |  |
| 06 | DSR: | .word | xFFFF0008 |  |
| 07 | DDR: | .word | xFFFF000C |  |
| 08 | ; |  |  |  |
| 09 | ; |  |  |  |
| 0A |  | .text | x00002D00 |  |
| 0B |  | sw | SaveR2(r0),r2 | ;保存此例程所需寄存器 |
| 0C |  | sw | SaveR3(r0),r3 |  |
| 0D |  | sw | SaveR4(r0),r4 |  |
| 0E |  | sw | SaveR5(r0),r5 |  |
| 0F | ; |  |  |  |
| 10 | ;对字符串中的每一个字符进行循环 | | | |
| 11 | ; |  |  |  |
| 12 | LOOP: | lb | r2,0(r4) | ;取得字符 |
| 13 |  | beqz | r2,Return | ;如果是0，字符串结束 |
| 14 |  | lw | r5,DSR(r0) |  |
| 15 | L2: | lw | r3,0(r5) | ;测试输出寄存器是否就绪 |
| 16 |  | andi | r3,r3,#1 |  |
| 17 |  | beqz | r3,L2 | ;循环直到显示器就绪 |
| 18 |  | lw | r5,DDR(r0) | ; |
| 19 |  | sw | 0(r5),r2 | ;显示字符 |
| 1A |  | addi | r4,r4,#1 | ;指针加1 |
| 1B |  | j | LOOP | ;获取下一个字符 |
| 1C | ; |  |  |  |
| 1D | ;从服务调用请求返回 | | | |
| 1E | Return: | lw | r2,SaveR2(r0) | ;将寄存器恢复为原先的值 |
| 1F |  | lw | r3,SaveR3(r0) |  |
| 20 |  | lw | r4,SaveR4(r0) |  |
| 21 |  | lw | r5,SaveR5(r0) |  |
| 22 |  | jr | r31 | ;从TRAP返回 |

12.**caller-save**（**调用者保存**）：调用程序保存占用的寄存器。

13.**callee-save**（**被调用者保存**）：被调用程序保存占用的寄存器。

14.**中断驱动的I/O**的本质是I/O设备能够

①强制程序停止。

②让处理器执行I/O设备的请求。

③让停止的程序继续执行。

15.某个I/O设备能够生成中断请求，必须具备

①I/O设备需要服务。

②设备有权请求服务。

16.**IE（中断允许位）在设备状态寄存器（KBSR和DSR）的[1]位，1表示允许中断，0表示不允许中断。**

17.**中断请求信号**（**IRQ**）是IE位与就绪位逻辑与运算的结果。

18.**原因寄存器**（**CAUSE**）记录哪些设备发出中断信号，只有特权模式（操作系统）下才能访问。

[15:8]为中断未决位，[15:10]为硬件中断未决位，[9:8]为软件中断未决位。

**CAUSE[11]代表键盘中断未决位，CAUSE[10]代表显示器中断未决位。**

按照优先级高低从左至右排列。

未决中断等到相应SR中断屏蔽位为1时才能引起处理器处理。

19.**SR**（**状态寄存器**）

[0]位可以改写所有设备的IE位，只有特权模式（操作系统）下才能访问。

[1]位表示正在运行的程序处于特权（管理员或内核）模式时为0，处于用户模式时为1。

[2]位中断发生时保存SR[0]的值。

[3]位中断发生时保存SR[1]的值。

[15:8]位是**中断屏蔽位**（**中断掩码位**），给出**中断阻塞方案**，[15:10]为硬件中断屏蔽位，[9:8]为软件中断屏蔽位。优先级高低从左至右排列。1表示允许中断，0表示屏蔽中断。

20.**程序状态**：程序影响的所有资源所包含的内容的瞬态图。

21.**EPC**：保存中断发生时PC的值，只有特权模式（操作系统）下才能访问。

22.进入中断服务例程时，应屏蔽所有中断，原因：避免受到来自其他设备的中断信号干扰。

23.**优先级**：执行的紧急程度。**请求的优先级必须高于希望中断的程序**。

24.DLX硬件优先级从低到高分为PL0、PL1、PL2、PL3、PL4、PL5，速度越高的I/O设备优先级越高。

25.**键盘优先级为1，显示器优先级为0。**

26.**中断服务例程**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01 |  | .data | x80000000 |  |
| 02 | SaveR1: | .space | 4 | ;保存寄存器的存储单元 |
| 03 | SaveR2: | .space | 4 |  |
| 04 | SaveR3: | .space | 4 |  |
| 05 | SaveR5: | .space | 4 |  |
| 06 | SaveR6: | .space | 4 |  |
| 07 | SaveR7: | .space | 4 |  |
| 08 | KBDR: | .word | xFFFF0004 |  |
| 09 | DDR: | .word | xFFFF000C |  |
| 0A | ; |  |  |  |
| 0B |  | .text | x80001000 |  |
| 0C | ;保存中断服务例程需要的寄存器 | | | |
| 0D |  | sw | SaveR1(r0),r1 |  |
| 0E |  | sw | SaveR2(r0),r2 |  |
| 0F |  | sw | SaveR3(r0),r3 |  |
| 10 |  | sw | SaveR5(r0),r5 |  |
| 11 |  | sw | SaveR6(r0),r6 |  |
| 12 |  | sw | SaveR7(r0),r7 |  |
| 13 | ;是否有允许的中断 | | | |
| 14 |  | movs2i | r1,x0D | ;CAUSE |
| 15 |  | movs2i | r2,x0C | ;SR |
| 16 |  | movs2i | r6,x0E | ;EPC |
| 17 |  | andi | r3,r1,xFF00 | ;CAUSE[15:8] |
| 18 |  | andi | r5,r2,xFF00 | ;SR[15:8] |
| 19 |  | and | r3,r3,r5 |  |
| 1A |  | beqz | r3,DONE | ;没有允许的中断，返回 |
| 1B | ;按照优先级顺序，依次处理 | | | |
| 1C | TEST5: | slli | r3,r3,#16 |  |
| 1D |  | slti | r5,r3,#0 |  |
| 1E |  | addi | r7,r0,TEST4 |  |
| 1F |  | bnez | r5,DEV5 | ;优先级为PL5的设备 |
| 20 | TEST4: | slli | r3,r3,#1 |  |
| 21 |  | slti | r5,r3,#0 |  |
| 22 |  | addi | r7,r0,TEST3 |  |
| 23 |  | bnez | r5,DEV4 | ;优先级为PL4的设备 |
| 24 | TEST3: | slli | r3,r3,#1 |  |
| 25 |  | slti | r5,r3,#0 |  |
| 26 |  | addi | r7,r0,TEST2 |  |
| 27 |  | bnez | r5,DEV3 | ;优先级为PL3的设备 |
| 28 | TEST2: | slli | r3,r3,#1 |  |
| 29 |  | slti | r5,r3,#0 |  |
| 2A |  | addi | r7,r0,TEST1 |  |
| 2B |  | bnez | r5,DEV2 | ;优先级为PL2的设备 |
| 2C | TEST1: | slli | r3,r3,#1 |  |
| 2D |  | slti | r5,r3,#0 |  |
| 2E |  | addi | r7,r0,TEST0 |  |
| 2F |  | bnez | r5,DEV1 | ;优先级为PL1的键盘 |
| 30 | TEST0: | slli | r3,r3,#1 |  |
| 31 |  | slti | r5,r3,#0 |  |
| 32 |  | bnez | r5,DEV0 | ;优先级为PL0的显示器 |
| 33 | ;…（软件中断测试） | | | |
| 34 |  | j | DONE |  |
| 35 | ;处理中断 | | | |
| 36 | ;…（其它设备） | | | |
| 37 | DEV1: | lw | r1,KBDR(r0) |  |
| 38 |  | lw | r4,0(r1) | ;输入字符加载到R4 |
| 39 |  | jr | r7 |  |
| 3A | DEV0: | lw | r1,DDR(r0) |  |
| 3B |  | sw | 0(r1),r4 | ;R4中的字符输出到显示器 |
| 3C | ;将寄存器恢复为原先的值 | | | |
| 3D | DONE: | lw | r1,SaveR1(r0) |  |
| 3E |  | lw | r2,SaveR2(r0) |  |
| 3F |  | lw | r3,SaveR3(r0) |  |
| 40 |  | lw | r5,SaveR5(r0) |  |
| 41 |  | lw | r6,SaveR6(r0) |  |
| 42 |  | lw | r7,SaveR7(r0) |  |
| 43 |  | movi2s | r0,x0D | ;清空CAUSE |
| 44 |  | rfe |  |  |

27.**中断嵌套**：中断服务例程执行时允许被比其优先级高的设备中断。

①保存SR和EPC。

②修改SR[15:8]，屏蔽比该设备优先级低（或相等）的其他设备的中断，允许优先级高的设备中断。

③将SR[0]设为1，允许中断。

④结束中断前设SR[0]为0（恢复SR和EPC时不允许被中断）

⑤恢复SR和EPC。

28.中断嵌套时保存程序状态使用栈。

29.C语言程序输入输出通过I/O库函数执行。

30.所有基于字符的输入和输出都是对**流**执行的。

31.当一个字符被键入，它被添加到输入流的结尾处。读取键盘的输入，总是从输入流开头处读取。

32.程序要打印的ASCII码字符序列被添加到输出流结尾处，输出时总是从输出流开头处输出，**每次流中的一个字符使用后被消耗，未被使用的字符将保留在流中**。

33.使用流使输入和输出以其各自速率操作而不用等待另一个就绪。

34.**stdin**（**标准输入流**）缺省映射到键盘，**stdout**（**标准输出流**）缺省映射到显示器。

35.**putchar()**输出一个字符，**getchar()**输入一个字符（**从输入流中读取一个字符，当输入流为空时等待**）。二者必须包含stdio.h头文件。

36.**I/O流缓冲**：每个键盘上的输入都被底层操作系统软件捕获，并被保存在一个小的数组缓冲区里，**直到用户按下回车键，缓冲区才被释放到输入流中，且回车键本身也作为换行字符加入输入流**。

优点：使用户能使用退格键删除以便编辑输入的内容，并按下回车键确认其输入。

37.**使用printf输出%使用序列”%%”。**

38.**空白字符**：空格、水平制表符、新行、回车、垂直制表符、换页。

39.**scanf输入时抛弃所有空白字符（但不会连接空白字符两边的内容），并依次匹配符合的格式，一旦遇到不符的格式即终止输入，以回车确认输入。此时未匹配的变量未被赋值而保持初值。**

40.scanf函数返回成功转换的参数个数，printf函数返回输出的字符串字符数（转义序列和格式说明计一个，不包括’\0’），putchar函数和getchar函数分别返回输出和输入的字符ACSCII码值。

## 第十四章 子例程

1.**子例程**：在程序内不必在每次需要时均说明原程序段全部细节，而是通过多次调用来实现的程序片段。

优点：可以由不同程序员分别实现需要程序片段的程序和程序片段。

2.**库**：提供的程序片段的集合。

3.**调用/返回机制**：实现子例程的机制。

①调用机制计算子例程起始地址，加载到PC，**保存返回地址于R31中**。

②返回机制使用返回地址加载PC。

4.TRAP服务例程包括操作系统资源，需要访问计算机底层硬件的特权，由管理计算机资源的系统程序员编写。

5.**JAL和JALR在R31中保存子例程需要返回的地址，返回地址为当前PC+4，并跳转至子例程。**

6.**JAL**链接：[31:26]101110，[25:0]26位地址偏移量，子例程目标地址为PC+4+SEXT[PCOffset26]。

7.**JALR**链接：[31:26]101111，[25:21]源寄存器，[20:0]0 0000 0000 0000 0000 0000（未用），PC直接加载SR1中的地址。

8.在子例程中以jr r31或ret结束子例程并返回。

9.**变元**（**arguments**）：传给子例程的值。

10.**返回值**（**return values**）：从子例程传出的值。

11.库函数由编译器和操作系统设计者提供。

12.**库例程标记在库中被标记为.global，用户程序调用时需要指出.extern。**

## 第十五章 函数

1.C语言中子程序称为**函数**。

2.C语言是面向函数的，C程序本质上是函数的集合，每条语句属于并仅属于一个函数。

3.C语言程序总是从main函数开始，在main函数中调用其他函数，这些函数也可以依次调用更多函数，控制最终会返回main函数，main函数结束时程序结束。

4.**函数声明**称为**函数原型**，包括函数名称、返回值类型、输入值列表，以分号;结束。

5.**没有返回值的函数返回值类型为void**。

6.函数名称按标识符命名规则。

7.**帕斯卡命名法：**单词首字母大写 ，其余部分小写。

8.**骆驼式命令法：**单词首字母大写，其余部分小写，但第一个单词首字母小写。

9.**下划线法：**函数名中的每一个逻辑断点都有一个下划线来标记。

10.函数声明括号中描述函数需要输入的参数的类型和顺序，**可以不指明参数名而只写参数类型**，参数名也可以与函数定义时的形式参数不同。不需要参数时括号里为空。

10.在调用者内部被传给被调用者的值被称为**变元**。

11.函数定义括号中的是**形式参数列表**，变元与形式参数类型和顺序匹配。

12.**任何调用者的局部变量对被调用函数不可见。**

13.用**return**指明返回的值，必须与声明返回的类型匹配。

14.**若有返回值的函数没有使用return语句，则最后一条语句的值作为返回值返回给调用者。**

15.函数定义在函数调用前，可以不使用函数声明。

16.变元运算顺序自右向左。例：function(i,i++);。

17.一个函数能被任何一个函数调用（包括其本身）。

18.C函数在DLX底层实现时，参数个数若多于4个将使用存储器，且使用运行时栈机制，这段存储空间称为函数的**栈框架**或**活动记录**。

19.编译器为每一次函数调用，在存储器中分配一个活动记录；当函数返回时，它的活动记录将被回收，以便分配给后面的函数；每一次函数调用都会在存储器中为其局部数值获得它自己的空间。优点：允许函数递归。

20.R29栈指针指向栈顶，R30框架指针指向活动记录底。

21.**动态链接**：为调用者的框架指针制作的副本。

22.**叶函数**：在执行过程中没有调用其他函数的函数，且没有改变R31。

23.头文件包含函数声明，预处理宏，但不包含库函数的源代码，即头文件已编译。

24.数学库函数使用头文件math.h。

25.**printf函数格式说明少于参数时，将使用期望在存储器指定位置的垃圾值。**

26.**scanf函数返回值为该函数在输入流中成功扫描的格式说明的个数。**

27.**将C语言函数调用翻译为DLX汇编语言**

①通过栈指针R29对局部变量R16-R23、临时寄存器R8-R15、R24、R25、参数传递寄存器R2、R3压栈（只对将要修改但还需使用的寄存器压栈）。

subi r29 r29 #4

sw 0(r29) SR1

subi r29 r29 #4

sw 0(r29) SR2

……

subi r29 r29 #4

sw 0(r29) SRn

②参数由R4-R7 传递。

③返回值由R2和R3传递。

④通过栈指针R29将压栈的寄存器出栈。

lw SRn 0(r29)

addi r29 r29 #4

……

lw SR2 0(r29)

addi r29 r29 #4

lw SR1 0(r29)

addi r29 r29 #4

ret

## 第十六章 指针和数组

1.**指针**：一个存储对象的**地址**。

2.**数组**：存储器中被连续排列的一列数据。

3.**变元总是以值的形式从主调函数传递到被调用函数。**

4.声明指针变量：**类型 \*变量名;** 。星号\*前后空格可有可无，指针变量类型视为类型\*。指向指针的指针使用多个星号\*。

5.**地址运算符&**：生成它操作数的存储地址。

6.**间接运算符\***：间接操作存储对象里的值。

7.赋值运算符右侧的间接运算符\*生成lw指令，取出存储的值；赋值运算符左侧的间接运算符\*生成sw指令，将右值存储到存储单元中。

8.**scanf函数格式用字符串后的参数必须使用指针。**

9.**主调函数的局部变量可以在被调用函数中通过指针间接修改。**

10.指针变量可以赋值**NULL**，表示不指向任何变量的**空指针**。NULL是特别定义的预处理宏，等于0，因为没有一个有效的存储对象可以存储在单元0中，所以空指针不指向任何变量。

11.声明数组：**类型 数组名[数组大小];** 。数组名[0]被分配到最低存储地址，**数组名[n-1]**被分配到最高存储地址。

12.访问数组通过**基址+偏移量**访问。**数组的基址是数组名[0]的地址。**

13.**数组的大小使用预处理宏有时比较恰当。**

14.**格式%Nd表示十进制补码整数占N位，如果不足N位，则左边补空格。**

15.**格式%-Nd表示十进制补码整数占N位，如果不足N位，则右边补空格。**

16.原因：不会在调用函数时把每个元素从一个活动记录复制到另一个活动记录中而花费大量时间。

17.**一维数组的名字就是该数组的基址，可以直接使用但不可以被修改，类型是声明时的类型。**例如：int x[10];，则x，&x[0]，&x三者等价，类型都为int\*。

18.**函数声明和函数定义时的参数若为一维数组，数组大小可以留空，形如“类型 数组名[]”；也可以使用指针，形如“类型 \*数组名”。**例如：void function(**int x[]**); 或void function(**int \*x**);。

19.**函数引用时，参数若为数组必须使用地址（一般直接写数组名）。参数传递后被调函数对应的数组形式参数名被赋值为传递的数组参数基址。**

例如：int main(){

int x[10];

function(**x**);

}

void function(int cox[]){

cox[1]=0;

}

则cox被赋值为x（&x[0]）。

20.**主调函数的数组会在被调用函数中被直接修改并可见，包括使用“形式参数数组”或者引用“基址+偏移量指针”。偏移量是数组号而不用考虑所占存储单元数。**

例如：void function(int cox[]){

**cox[1]=0;** 或**\*(cox+1)=0;**

}

21.**字符串**：表示文本的字符序列，字符数组。

22.字符串用双引号” “包括。

23.字符串赋初值时编译器在末尾自动加上0（**’\0’**），不足位会补0。

24.**以空结尾的字符串’\0’占用一个数组元素。**

25.格式序列%s在printf函数中，打印以参数表示的地址开始的字符串，以’\0’结尾。

26.格式序列%s在scanf函数中，在输入流中读入从第一个非空字符开始，到下一个空白字符之间的字符，并在末尾自动加上一个’\0’，存储于以参数表示的地址为首地址的存储器中。最后的空白字符以及其后的内容保留在输入流中。

27.**scanf函数不检查数组大小，当输入超过数组大小时，将强制修改数组后面单元中的内容。**

28.字符串处理函数使用头文件string.h。

29.**C语言不提供防止超出数组大小（或边界）的保护措施，访问超出数组大小的存储单元会修改其中的内容，因此需要防御性编程。**

30.**C89规定编译程序必须知道数组大小，即声明时数组大小不能使用变量。**

31.二维数组声明：**类型 数组名[行数][列数];** 。分配空间时先按行后按列分配。

32.**二维数组名是该数组的基址，数组名[行数]是数组中该行的首地址。**例如：char a[5][5]; ，则a，&a[0][0]，a[0]，&a，&a[0]五者等价；a[1]，&a[1][0]，&a[1]三者等价。

33.**strcmp函数**比较两个字符串，原型为**int strcmp(char \*string1,char \*string2);** 。若string1大于string2，则返回1；若string1小于string2则返回-1；若string1等于string2，则返回0。比较方法为ASCII码字典序。

34.**strcpy函数**复制字符串，**strcpy(char \*string1,const char \*string2);** 。将string2复制到string1中。

35.**函数声明和定义时若参数为二维数组，一定要注意参数使用指针或地址。**

**①若在被调函数内对整个数组操作，则参数是整个数组，可以省略行数，函数引用参数是数组基址。**

例如：有二维数组int x[5][4]，则void function(int [5][4])或void function(int [][4])或void function(int (\*x)[4])中可以对整个数组x操作，函数引用为function(x);。

**②若在被调函数内对某行操作，则参数是指针。被调函数的形式参数名已经表示原数组的该行地址。**

例如：有二维数组int x[5][4]，若函数引用为function(x[2]);，则void function(int \*x)中只能操作第三行，x[3]表示x[2][3]。

## 第十七章 递归

1.**递归思想**：一个递归的函数通过在一个更小的子任务中调用它本身来解决某个任务。

2.所有的递归都可以用重复实现，编译器未优化递归时比重复运行慢。

3.二分法查找运行时间与成正比，n 为数组大小。

4.顺序查找运行时间与n成正比，n为数组大小。

5.递归函数翻译为DLX需要保存参数寄存器。

6.递归分解：将一个任务分成两部分，初始步骤和更小规模的不变型步骤。

# 致谢

向所有为本文档的写作与改进作出贡献的个人与集体致谢！

**南京大学软件学院，余东骏**

**2019年12月9日星期一，于南京大学仙林校区**