计算系统基础提纲

南京大学软件学院
江苏省南京市鼓楼区汉口路 22 号费彝民楼

GitHub 地址点这里

致各位 NJUSEer:

这是你们的学长一代代流传并总结下来的“计算系统基础”课程相关资料，希望对你们能有所帮助，也欢迎你们加入完善这份资料的伟大事业中。

其他课程相关资料可以访问：

https://github.com/NJU-SE-15-share-review/professional-class

在此谨祝你们学业进步，万事胜意！

在此诚挚感谢每一位为本份材料的完善做出贡献的人！

计算系统基础

知识点总结

注：本文档整理自学长的笔记，基本上生成于 2016 年，希望那位热心人出来声明一下版权。

本文档中如有缺漏，欢迎随时指出以帮助后人。

2019 年 12 月 9 日星期一

第一部分 引言

第一章 C 语言程序设计简介

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
- 语言处理
  - 高级语言处理（编译器/解释器）
  - 汇编语言处理（汇编器）
- 操作处理
  - 系统调用
  - I/O 例程

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
- 高级语言
- 低级语言
  - 汇编语言
  - 机器语言

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。

第二部分 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
- C89 /* 注释内容 */
- C99 //注释内容

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 -2^{31}~$2^{31}-1$
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--⇔--1⇔i=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）

第三部分 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。

①if(condition)

action;

②if(condition)

action_if;

else

action_else;

③if(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

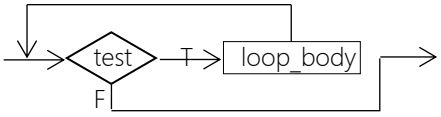
循环可以嵌套。

loop_body 是一条简单或符合语句

①while(test)

loop_body;

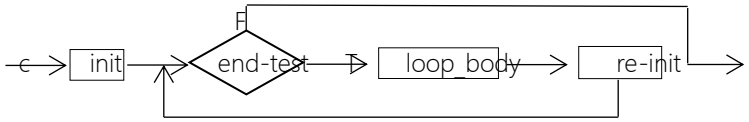
适用于标识控制循环，不知循环次数。



②for(init;end-test;re-init)

loop_body;

适用于计数器控制循环，已知重复次数。

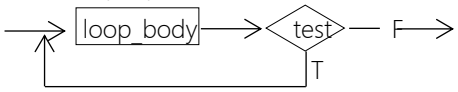


③do

loop_body;

while(test);

while(test)后有分号；。



4.其他控制结构

①switch(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。

②break； 直接跳出当前循环或 switch 结构。

③continue; 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.错误类型
- 语法错误：违反编程语言规则。（编译器能指出）
- 语义错误：程序员与编译器对语义理解不同。（运行结果与程序员期望不同）
- 逻辑错误：使用有缺陷的算法。（运行结果与程序员期望相同）

6.黑盒测试：检查程序是否满足其输入和输出规格说明。

只关心结果，不关心过程。

编写检查器程序的黑河测试员不允许查看正在测试的黑盒内代码，原因：防止被测程序与检查器程序有相同错误。

缺点：任何一行未测试的代码都可能错误。

7.白盒测试：保证每一行代码都经过某个级别的测试。

需要修改被测代码。

断言：在程序特定部位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，

核心操作集{控制程序执行。；程序执行过程中检查变量值等相关信息。

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。

①Step Into：进入函数内部并在函数中单步执行。

②Step 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。

③Step 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

第四部分 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 2^k 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 $A\cdot B$，或 $A+B$，非 $\overline{A}$，异或 $(\overline{A}\cdot B)+(A\cdot\overline{B})$

①恒等律：$A+0=A$，$A\cdot 1=A$

②0/1 律：$A+1=1$，$A\cdot 0=0$

③互补律：$A+\overline{A}=1$，$A\cdot\overline{A}=0$

④交换律：$A+B=B+A$，$A\cdot B=B\cdot A$

⑤结合律：$A+(B+C)=(A+B)+C$，$A\cdot(B\cdot C)=(A\cdot B)\cdot C$

⑥分配律：$A\cdot(B+C)=(A\cdot B)+(A\cdot C)$，$A+(B\cdot C)=(A+B)\cdot(A+C)$

⑦德摩根律：$\overline{A\cdot B}=\overline{A}+\overline{B}$

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 $0\sim 2^k-1$

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 $-2^{k-1}\sim 2^{k-1}-1$

1000...：绝对值最大的负数（最大正整数的相反数-1）

1111...：-1

0000...：0

2^0=1　2^4=16　2^8=256　2^{-1}=0.5
2^1=2　2^5=32　2^9=512　2^{-2}=0.25
2^2=4　2^6=64　2^{10}=1024　2^{-3}=0.125
2^3=8　2^7=128　2^{11}=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：$45.8125_d=0101101.1101_b=1.011011101X2^5{}_b$

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：$10^{\pm38}$

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）

$$\begin{cases}(-1)^s\times1.fraction\times2^{exponent-127}，\ 1(00000001)\le exponent\le254(11111110)\\(-1)^s\times0.fraction\times2^{-126}，\ exponent=0(00000000)\\(-1)^s\infty，\ exponent=255(11111111)，\ fration=0\\NaN(not\ a\ number)，\ exponent=255(11111111)，\ fration\neq0\end{cases}$$

22.双精度浮点数

范围：$10^{\pm306}$

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$\begin{cases}\text{黑白：}1\text{ 位，}0\text{ 白 }1\text{ 黑}\\ \text{彩色：}\mathrm{RGB(red,green,blue)}\text{各 }8\text{ 位}\end{cases}$

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

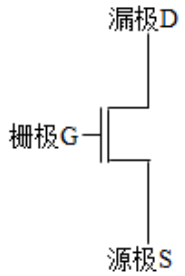
第七章 数字逻辑电路

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。

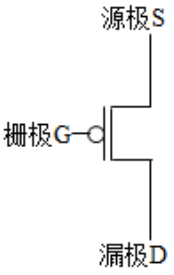
2.摩尔定律：集成电路上可容纳的晶体管数目约每隔 18 个月增加一倍。即，价格不变时，每一美元能买到的电脑性能每隔 18 个月翻两倍以上。

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。

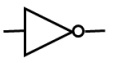


5. P 型 MOS 晶体管源极不能接地。栅极加 2.9V 电压，漏极和源极断开；栅极加 0V 电压，漏极和源极导通。

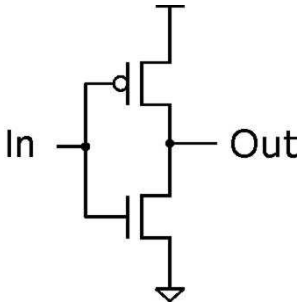


6.CMOS（互补金属氧化物半导体）电路：既包含 P 型晶体管，又包含 N 型晶体管。

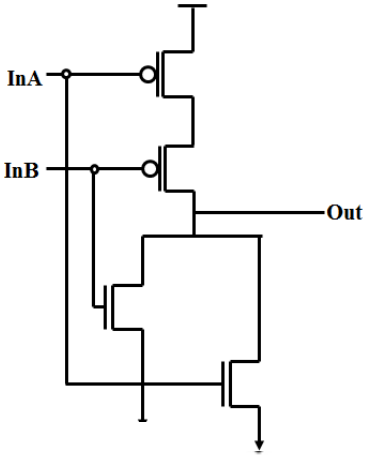
7.反相器（非门）



In	Out
1	0
0	1

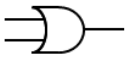


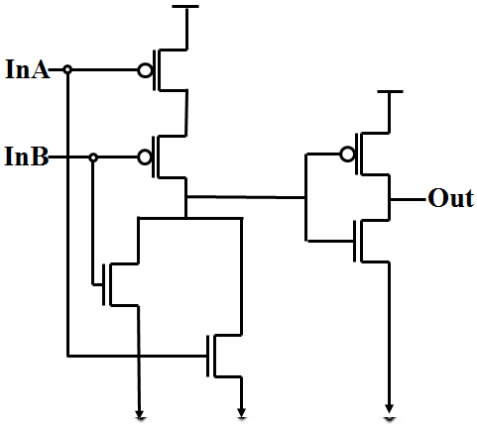
8.非或门



InA	InB	Out
1	1	0
1	0	0
0	1	0
0	0	1

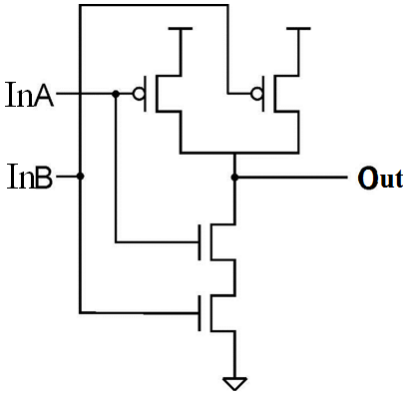
9.或门（非或门+反相器）





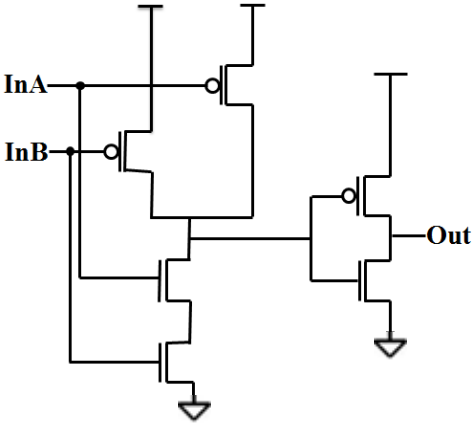
InA	InB	Out
1	1	1
1	0	1
0	1	1
0	0	0

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。

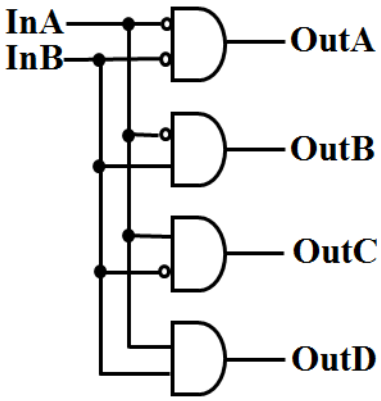
13.译码器

只有一个输出为 1，其他全为 0。若有 n 个输入，则有 2^n 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



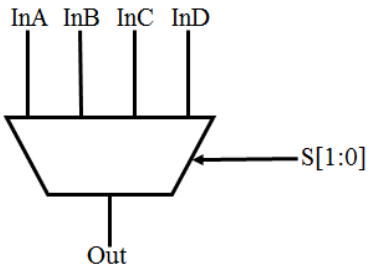
14.多路选择器

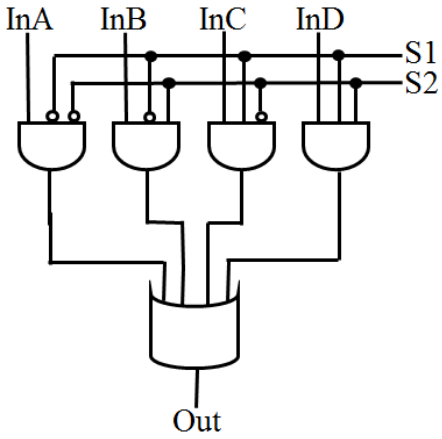
由选择信号决定哪个输入连接到输出。

n 条选择线，2^n 个输入，1 个输出。

n=2，4-1 多路选择器：

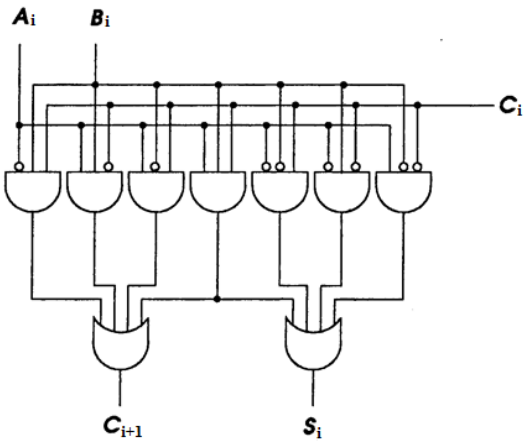
S1	S2	Out
1	1	InD
1	0	InC
0	1	InB
0	0	InA



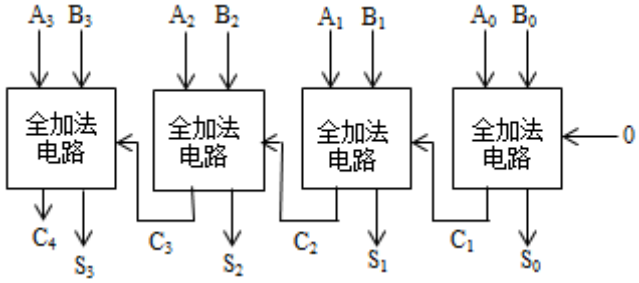


15.全加法电路

In			Out	
a_i	b_i	c_i	c_{i+1}	s_i
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：

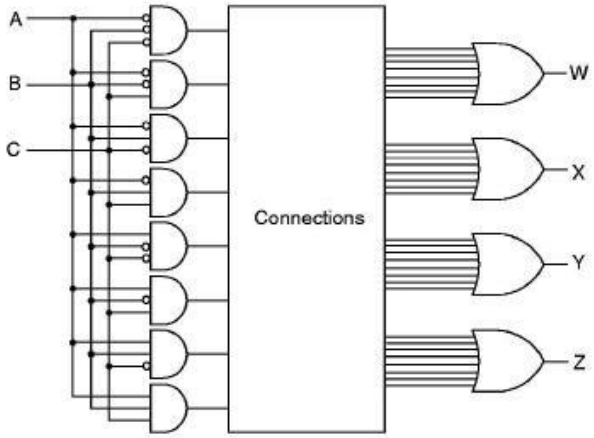


16.PLA（可编程逻辑阵列）：可以实现任意逻辑运算的通用组件。

与阵列：每个与门有 n 个输入，一共 2^n 个与门。

或阵列：输出。

n=3：

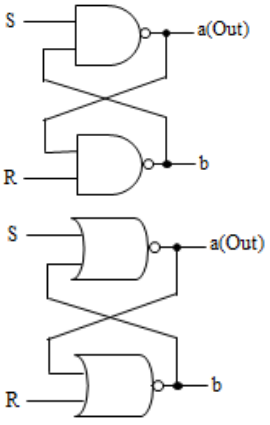


17.逻辑完备性：提供足够多的与或非门，可以实现任意逻辑运算。即，门集合{与、或、非}在逻辑上完备。

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）

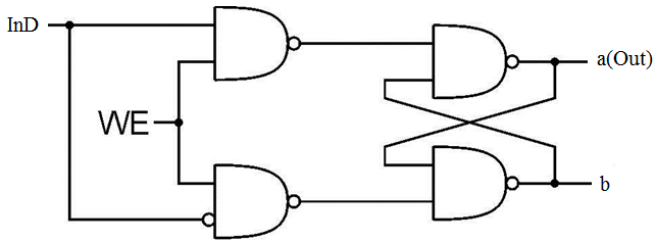
②非或门：

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

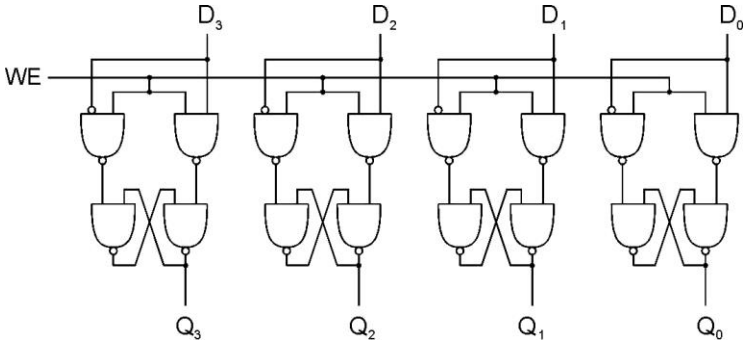
优点：对 R-S 锁存器何时设为 1 和 0 进行控制，避免 R=S=0 的情况。



22.寄存器

多个门控 D 锁存器存储多位，共享 WE。

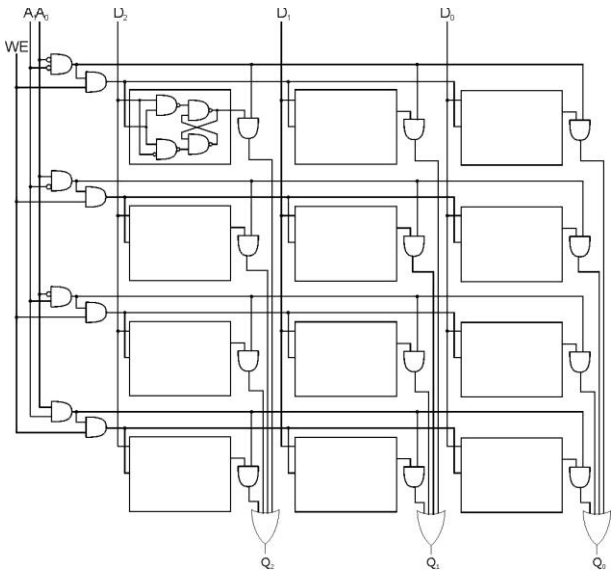
4 位寄存器 Q[3:0]：



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：$\left.\begin{matrix}2-4\text{ 译码器}\\ \text{写使能 WE}\end{matrix}\right\}$→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 2^2 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 2^n。

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=2^{10}，1M=2^{20}，1G=2^{30}，1T=2^{40}

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。

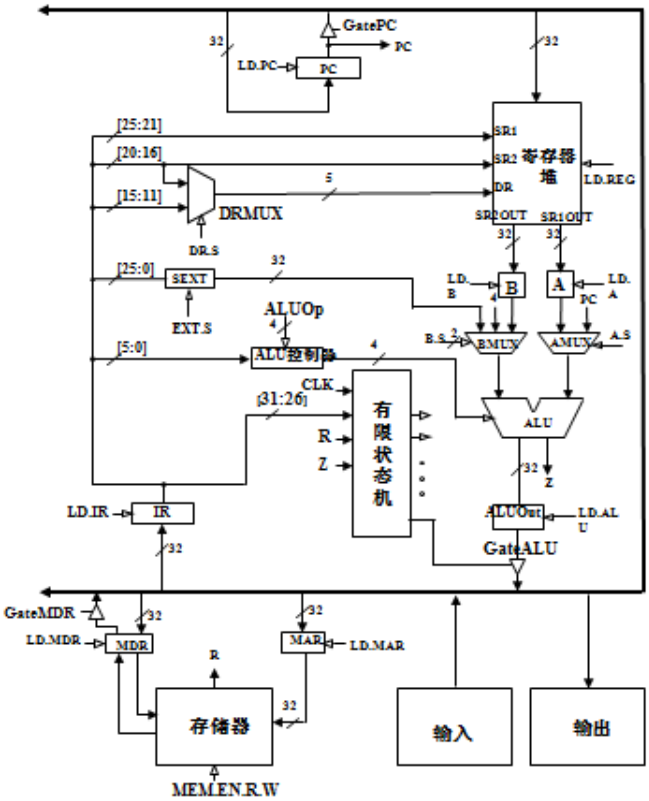
40.时钟周期：重复的时间间隔序列中的一个时间间隔。

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 $\begin{cases} \text{DRMUX}：\text{提供 5 位地址给寄存器堆。} \\ \left.\begin{matrix}\text{AMUX}\\ \text{BMUX}\end{matrix}\right\}\text{分别提供 32 位数值给 ALU。} \end{cases}$

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。

- 存储器
- 处理单元
- 控制单元
- 输入设备
- 输出设备

（输入设备、输出设备）外围设备

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 2^{32} 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。

- 执行程序时，跟踪存储器中的指令。
- 处理指令时，跟踪指令的处理阶段。

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

PC 中的内容加载到 MAR，PC 与 4 在 ALU 中做加法运算，并用结果改变 PC 的值。

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。

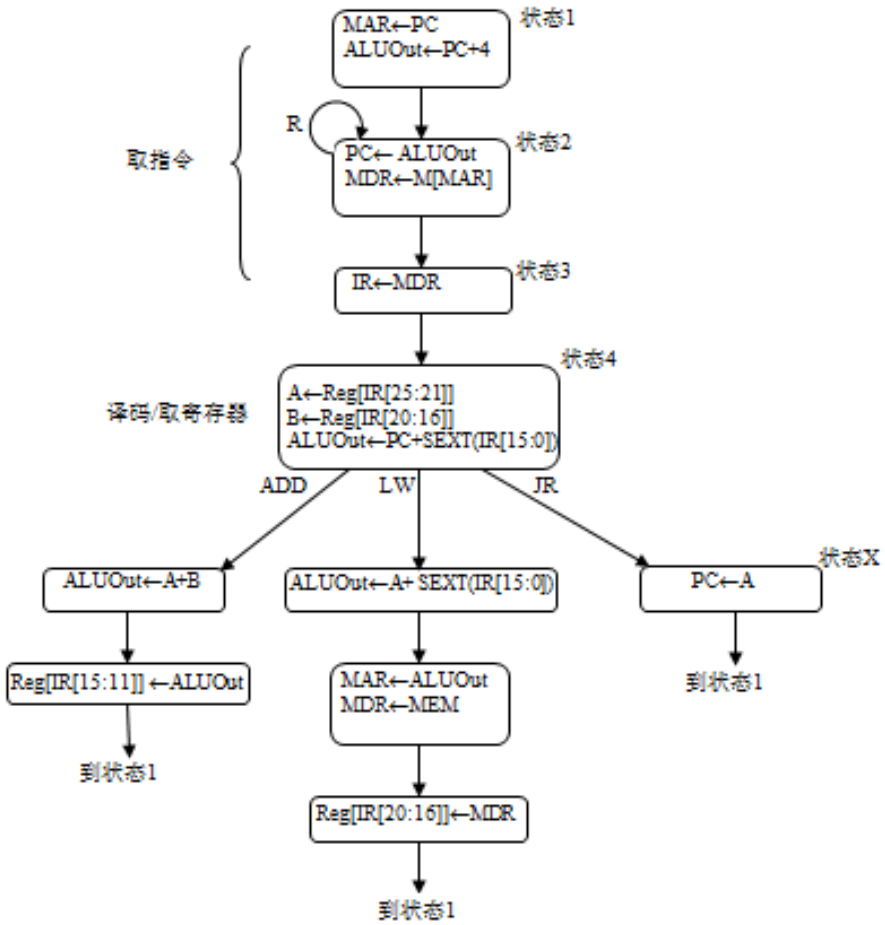
有效地址：计算出处理指令所需存储单元的地址。

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；

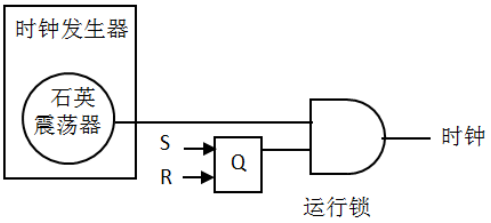
有限状态机使 EXT.S 设为 0，B.S 设为 00，选择 IR[15:0]符号扩展结果；

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 2^{32} 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
- CISC（复杂指令集计算机）：开发容易，执行效率低。
- RISC（精简指令集计算机）：开发不足，执行效率高。（DLX）

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
- 条件分支
- 无条件跳转
- 子例程（函数）
- TRAP
- 异常 /中断返回

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 $2^{15}+3$~$-2^{15}+4$ 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

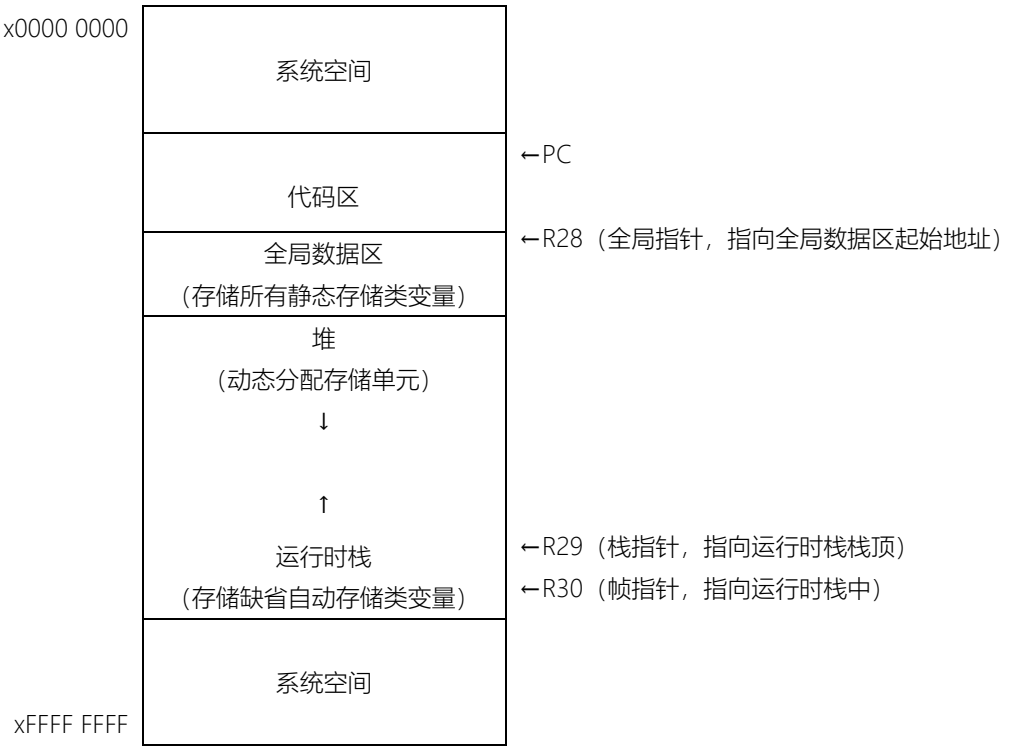
32.pop（出栈）：从栈中移出一个元素。

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



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
```

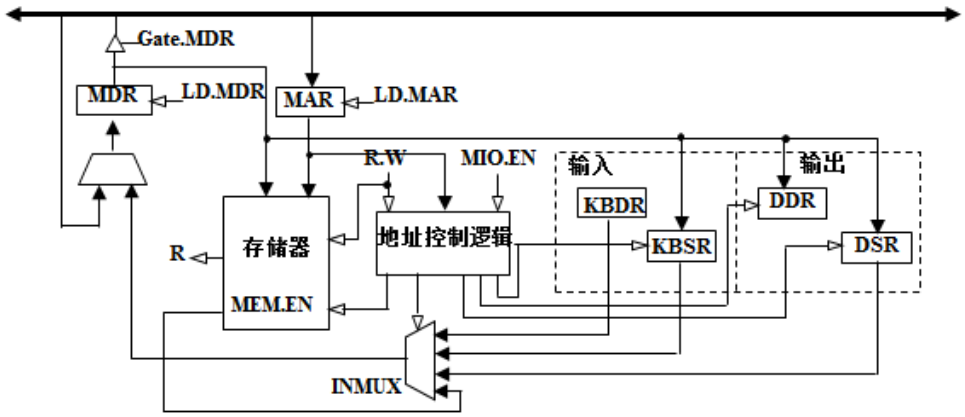
13.DLX 基本输出服务例程 OUT（轮询）:

```
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。

3 个块输入：
- **MIO. EN**：表示当前时钟周期里是否发生存储器或 I/O 数据传送操作。
- **MAR**：存储单元地址或 I/O 设备寄存器内存映射地址。
- **R. W**：表示加载或存储。



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 2^{10} 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$\log_2 n$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。

第五部分 附录

1、C 语言填空题

1. 当定义一个无返回值函数时，函数的返回值类型标应为__void__。

2. 若已有二维数组声明：

```
int a[ ][3]={ {1} , {2} , 3 , 4 , 5 , 6} ;
```

则该数组共有____12____个数组元素。

3. "int (*pa)( ); "是___指向函数的指针变量___的声明。

4. 当程序中需要调用库函数 strcmp 时，应当包含头文件__string . h__。

5. 当某程序准备从一个磁盘文件中读入数据， 需要有类似 " FILE *fp ; " 这样的声明，该声明中的 FILE 是___结构类型标识符___。

6. 若有如下数据类型定义及数组声明，则 p 数组在内存中占用的存储字节数为__40__。

```
struct {
        long   x ;
        union {   int   a ; char   b ; float   c ;} y ;
} p[5] ;
```

7. 在 C 语言源程序中需要建立或更新数据文件时，都要声明和使用 FILE 结构类型的指针变量，因此在源程序中必须包含头文件__stdio . h__。

8. fopen 函数允许的文件打开方式"r+"与"w+"的相同点是允许读/写、更新文件。但它们的一个区别是打开的文件是否必须存在，另一个区别则是 是否清除文件已有内容 。

9. 执行如下程序后，分别向文件 file1 和 file2 中存储了 8 和 4 字节的数据。

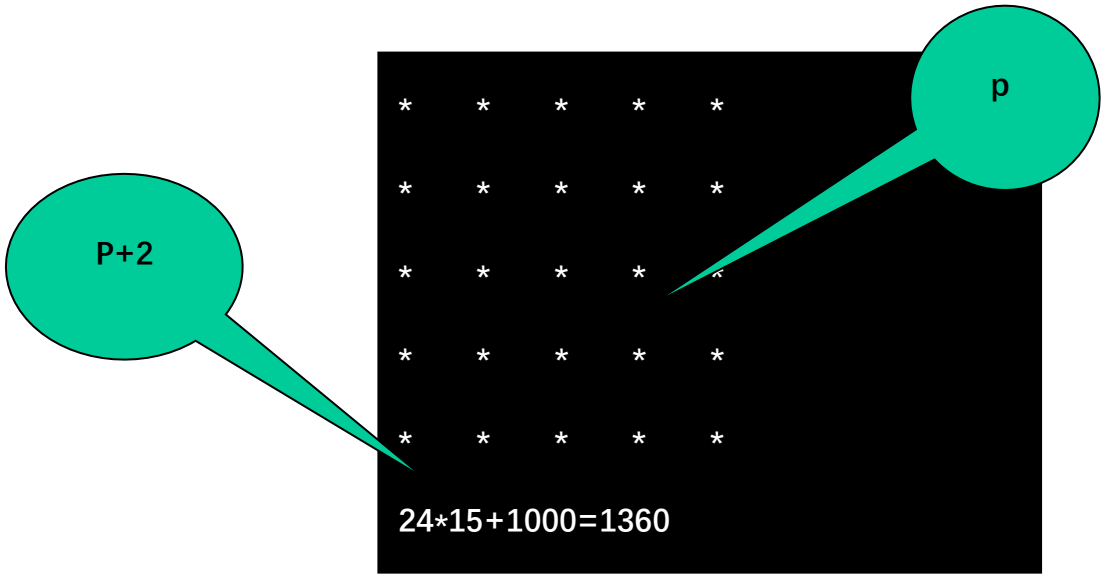
```
#include<stdio.h>
main( )
 {
    float num=1.5;
    FILE *f1 ,*f2;
    f1=fopen( "file1" , "w" );
    f2=fopen( "file2" , "wb" );
    fprintf( f1 , " %f " , num );
    fwrite( &num , siziof(float) , 1 , f2 );
}
```

10. 如果要限制一个函数只能在定义它的源程序文件中使用 ，而不能被该源程序的其他源程序文件使用，则必须声明这个函数的存储类别为 static 。

11. 已知有定义：

```
struct student{ char name[20];
                float score;
}s[5][5], *p=&s[2][3];
```

假设数组 s 的首地址为 1000,则表达式 p+2 的值为 1360 。



12. 如果要打开一个文本文件，若该文件已经存在，则清除其中的内容，然后可向该文件中添加新的内容,也可将添加的内容再读出来 ，如果添加的内容不正确，还可以进行修改，则使用 fopen 函数打开文件时，文件的打开方式应该是 w+ 。

13. 若一个函数不需要形式参数， 则在定义函数时，应使形式参数表为空或放置一个 void 。

14. 若有说明 int a[ ][4]={1,2,3,4,5,6,7,8,9}; 则数组 a 的第一维的大小是 3 。

15. 在用 fopen 函数打开一个已经存在的数据文件 abc 时，若要求读出文件 abc 中原来的内容，也可以用新的数据覆盖文件原来的数据，则调用 fopen 函数

时,使用的存取方式参数应当是 r+ 或 rb+ 。

16. 若有定义 ：

enum seq{ mouse,cat,dog,rabbit=0,sheep,cow=6,tiger } ;

则执行语句 printf (“%d”,cat+sheep+cow); 后输出的结果是 8 。

17. 若二维数组 a 有 m 列，则在存储该数组时，a[i][j] 之前有___i*m+j_____个数组元素。

18. 以下程序执行后建立文件 a.dat ,若另一个程序运行时要从该文件中读数据 ，则调用的读文件库函数名应当是 __fread__。

```
main( )
 { FILE *fp=fopen("a.dat", "wb");
     float x=25.0;
     fwrite(&x,sizeof(float),1,fp);
 }
```

19. 当程序中需要调用库函数 isdigit 时，应当包含头文件__ctype.h____。

20. C 预处理命令行总是以____回车换行符____结束

21. 一个 C 语言源程序文件由一个或若干个函数定义、若干个外部对象声明与说

明及_若干个预处理命令__组成.

22. 库函数 sqrt 的原型说明是：___double sqrt(double)____

23. 一般而言,递归程序采用__选择结构_ 程序结构是。

24. stdin 是_____FILE 指针__类型的数据对象。

致谢

向所有为本文档的写作与改进做出贡献的个人与集体致谢！

2019 年 12 月 9 日星期一

一些基础的C语言复习题

> 感谢前辈学长们的细心总结，页面重置：YDJSIR
>
> 祝各位期末考试取得优异成绩！

一. 完善程序题

1、以下程序中函数float fun(int n)的功能是：根据所给公式计算s的值并作为函数返回值，n 通过形参 传入，n的值大于等于0。

```
float  fun(int n)
{    float s=0.0,w,f=    -1.0     ;
        int i;
        for(i=0;i<=n;i++)
        {   f=-f ;
            w=   f/(2*i+1)    ;
            s+=w;
        }
         return s;
   }
main( )
{ int n=5;
  float s;
  s=   fun(n)   ;
 printf("%f",s);
}
```

2、以下程序实现对main函数内声明的局部数组a中的后10个元素做升序排序。排序算法为选择法。

```
      #include<stdio.h>
      void sort(int *x , int n);
       main( )
       {
          int a[12]={4,2,8,5,6,7,1,9,10,3} , k;
          sort(  &a[2]  , 10 );
       for(k=0 ; k<1 ; k++)
          printf("%d" , a[k]);
       }
void sort(int *x,int n)
{ int i,j,t,k;
      for(i=0;i<n-1;i++)
     {   k=i  ;
       for(j=i+1;j<n;j++)
       if(x[j]<x[k])k=j;
       if(k!=i){  t=x[i]; x[i]=x[k]; [ }
      }
}
```

3、以下程序中函数find的功能是：在a指向的一维数组中存储的m´n矩阵内查找数值最大的元素，将该元素所在的行、列的值分别保存到 pm和 pn指向的变量中。

```
void find(int a[],int m,int n,int *pm,int *pn)
{  int i,j,k,s;
k=  a[0]   ;*pm=0;*pn=0;
for(i=0;i<m;i++)
           for(j=0;j<n;j++)
           {  if(  a[i*m+j]  >k)
              { k=s; *pm=i;*pn=j;}
           }
}
main( )
{
     int b[3][4]={1,2,2,4,55,4,6,9,12,14,11,13}, pm, pn;
     find(&b[0][0], 3 , 4,    &pm , &pn     );
     printf(“the largest nember is located in  b[%d][%d]” , pm,pn);
}
```

4、 f1函数功能：建立一个链表，链表各节点的数据来源于a指向的一个指针数组中的前n个元素所指向的n个字符串，函数返回链表首节点的地址。 f2函数功能：对于h指向的链表中连续出现的多个name成员相同的那些结点，统计并保存这些相同结点的数量，保留一个结点，删除其余结点。

```
      #include<stdlib.h>
      #include<string.h>
      typedef struct p {
                    char name[20];
                    int count;
                   struct p *next;
      }PS;
PS *f1(char *a[],int n)
{ int i,c=1;char name[20];
       PS *head=0,*p,*tp;
for(i=0;i<n;i++)
{
  p=(PS*)malloc(  sizeof(PS) 或sizeof(struct p )   );
          if(tp==NULL){ puts("fail!");exit(0); }
          strcpy(tp->name,a[i]);tp->count=1;tp->next=NULL;
          if(head==0)head=p=tp;
          else
          {    p->next   =tp; p=tp;  }
        }
        return head;
}
void f2(PS *h)
{ while(h->next!=NULL)
      if(strcmp(h->name,h->next->name)==0)
      {  h->next=  h->next->next  ;   h->count++;  }
      else    h=h->next;
}
main( )
{ PS *head,*p;  char a[6]={"a","b","b","b","c","c"};
      head=f1(a,6);  f2 (head);
```

```
      for(p=head;p!=NULL; p=p->next)
      printf("%s:%d  ",p->name,p->count);
}
```

5、insert函数功能：将key中保存的一组数据（一个新记录）插入到a指向的一个有序结构数组中，已知该结构数组按name成员字典顺序存储，插入新记录后结构数组仍然按name成员的字典顺序存储。

```
 #include<stdio.h>
 #include<string.h>
     struct p{
                   int id;
                   char name[5];
     };
int  insert(struct p a[],int n,struct p key)
{  int i,j,k;
       for(i=0;i<n;i++)
       if(strcmp(a[i].name,key.name)>0 )  break  ;
       for(j=n;j>i;j--)
           a[j]=a[j-1]   ;
          a[i]   =key;
      return ++n;
}
main( )
{  int i,n=4;
     struct p x[5]={{1,"a"},{3,"c"},{4,"d"},{5,"e"}},y={2,"b"};
     n=insert(x,4,y);
     for(i=0;i<n;i++)
          printf("%d, %s \n",x[i].id,x[i].name);
}
```

6、find函数功能：在x指向的一个有序二维数组的前n行中查找值为key的数组元素，若找到则将该数组元素在二维数组中的行、列下标值分别保存到 row和col指向的变量中且函数返回1，若未找到则函数返回0。已知x指向的二维数组的每一行中元素的值均从小到大顺序存储，第i行中所有元素的值均小于第i+1行中所有元素的值（i=0,1,2,3...n-1）。

```
算法：用折半查找法定位值为key的数组元素所在行，
用线性查找法在已经定位的行中查找值为key的数组元素。
```

```
int find(int x[ ][5],int n,int key,int *row,int *col)
{  int i,low=0,high=n-1,mid;
   while(low<=high)
   {  mid=  (low+high)/2    ;
      if(key>=x[mid][0]&& key<=x[mid][4])break;
      if(key<x[mid][0])
          high=mid-1   ;
      else if(key>x[mid][4])
        low=mid+1;
   }
   for(i=0;i<5;i++)
       if(  key==x[mid][i]  )  { *row=mid; *col=i;  return 1;}
   return 0;
}
```

```
main( )
{ int a[5][5]={ {1,3,4,6,9},
                        {12,14,15,17,19},
                        {22,23,24,26,28},
                        {31,33,34,36,37},
                        {42,44,46,47,48}
                      } m, n, key;
     puts("input key:");  scanf("%d",&key);
     if(find(a,5,key,&m,&n))
         printf("%d is stored at a[%d][%d]",key,m,n);
     else
         printf("\n %d not found!",key);
}
```

7、函数change功能：将x指向的一个二维数组中存储的n´n矩阵变换为其转置矩阵转置矩阵的数学定义：把矩阵A的行换成同序数的列得到的一个新矩阵叫做A的转置矩阵。

```
#define  swap(a,b,c)  (c)=(a); (a)=(b); (b)=(c);
void change(int x[],int n)
    {  int i, j, k;
       for(i=0;i<n;i++)
           for(j=0;  j<i 或 j<=i  ;j++)
{  swap(x[i*n+j],x[j*n+i],k); }
     }                                                    }
main( )
    {
        int a[3][3]={1,1,1,2,2,2,3,3,3},i,j;
change(  a[0] 或 *a    , 3);
for(i=0;i<3;i++)
{ for(j=0;j<3;j++)
               printf("%d ",a[i][j]);
            putchar('\n');
          }
    }
```

8、下面程序完成的功能是：从键盘输入一行字符，反序后输出。

```
struct node{ char data; struct node *link; }*head;
main()
{ char ch; struct node *p;
      head=NULL;
      while((ch=getchar())!='\n'){
              p=(struct node *)malloc(sizeof(struct node)));
              p->data=ch; p->link=(  head  );  head=(_ p__);
      }
      (  p=head  );
      while(p!=NULL){ printf("%c",p->data); p=p->link; }
}
```

二. 程序阅读题

1、以下程序执行时输出结果为 *2.00, 3.50* 。

```
#include <stdio.h>
 #define p 3.5
    int min(int x,int y)
    {
    int z;
if(x<y)
 z=x;
else
 z=y;
return z;
}
void main( )
 {
int  ix=2.5;
double  dz;
        float  fx=2.5,fy=3,fz;
        fz=min(fx,fy);
        dz=p*(float)(ix/2);
printf("\n%.2f,%.2f",fz,dz);
     }
```

特别注意！传值的时候他传进去的是 `int` 型，也就是说进去的时候就进行了一次转换！注意到 `float` 和 `double` 变成整形的时候是直接把小数掐去，并非四舍五入！

2、以下程序运行后输出结果是 *Yellow* 。

```
enum color{ BLACK ,YELLOW ,BLUE ,GREEN , WHITE } ;
    main( )
    {
      char  *colorname[]={“Black”,“Yellow”,
                                “Blue” , “Green”,  “White” } ;
      enum color c1=GREEN , c2=BLUE ;
      printf(“%s” , colorname[c1-c2] ) ;
   }
```

3、以下程序运行后输出结果的第一行是 *1 1 2* ， 第二行是 *3 5 8* 。

```
int f(int n)
{
static int f1 , f2 , f ;
        if(n==0) return f1=1;
        if(n==1) return f2=1;
       f=f1+f2 ;
        f1=f2 ;
       f2=f ;
       return f ;
   }
main( )
{
      int i;
      for(i=0;i<6;i++){
          if(i%3==0)printf("\n");
          printf(“%d ” , f(i) );
      }
}
```

4、以下程序运行后输出结果的第一行是 *1010*， 第二行是 *2 2* 。

```
void change(int x, int m)
{ static char ch[ ]={'0','1','2','3','4','5','6','7','8','9'};
      int i=0,r;
      char b[80];
      while(x)
      { r=x%m; x/=m;
        b[i++]= ch[ r]   ;
       }
      for(--i ;i>=0;i--)
         printf("%c",b[i]);
}
main( )
{ int a,b;
      change(10,2);
      printf("\n");
      change(10,4);
}
```

5、以下程序运行后输出结果的第一行是 *1 2 3*， 第二行是 *0 0 0*，第三行是 *0 0 4* 。

```
main( )
{
      int m,n,x[3][3]={0},k=1;
      for(m=0;m<3;m++)
          for(n=0;n<3;n++)
          {
            if(m%2) break;
            if(m>n) continue;
            x[m][n]=k++;
          }
      for(m=0;m<3;m++){
          for(n=0;n<3;n++){
                    printf("%d ,", x[m][n]);
            printf("\n");
      }}
}
```

6、以下程序运行时输出b[2]的值是 *6*，b[4]的值是 *15* 。

```
#include<stdio.h>
void sum(int *pa,int *pb,int n)
   {  int i;
      if(n==1) *pb=*pa;
      else
      { for(i=0;i<n;i++)
            *(pb+n-1)+=*(pa+i);
        sum(pa,pb,n-1);
      }
    }
int main( )
{ int a[5]={1 , 2 , 3 , 4 , 5};
    int b[5]={0} , i;
```

```
    sum(a,b,5);
printf("%d, %d ",b[2], b[4]);
return 0;
  }
```

7、 以下程序运行后输出结果的第一行是 *A B C D*， 第二行是 *D C B A* ,第三行是 *D B A C* 。

```
#include<stdio.h>
   struct nd{
   char name[2];
   int age;
   struct nd *next;
   }a[4]={{"A",19},{"B",21},{"C",18},{"D",23}},*hd;
main()
{   int i,j,k;
         truct nd *p1,*p2,*p3;
        hd=&a[3];
        for(i=3;i>0;i--)
            a[i].next=&a[i-1];
        a[i].next=NULL;
        for(i=0;i<4;i++)
            printf("%s  ",a[i].name);
       printf("\n");
       p1=p3=hd;
       while(p1!=NULL)
       {  printf("%s  ",p1->name);
           p3=p1; p1=p1->next;
       }
printf("\n");
 p1=hd;p2=p1->next;
       p1->next=p2->next;
 p3->next=p2;
       p2->next=NULL;
       p1=hd;
       while(p1!=NULL) {
             printf("%s  ",
           p1->name);
             p1=p1->next;
       }
}
```

8、 下列程序运行后的输出是 *5* 。

```
int fun(int first , int second)
       {
          return first++*second++;
       }
        main( )
       {
int p=1 , r;
r=p+++fun(p++ , p++);
printf("%d\n" , r );
       }
```

9、若下列程序段中的变量a、z、和p分别存储在主存FFD2H、FFD4H、和FFD8H处，则执行表达式*p++ ->y 后，变量p的值为： *FFD8H* 。

```
    main()
{ int  a = 4 ;
struct {
int  x , *y ;
}z , *p=&z ;
z·x = 10 ;
z·y = &a ;
//……
}
```

10、 以下程序运行后输出结果是 *3* 。

```
     #include<stdio.h>
     int  f(int x,int y)
     {  if(x<y)
           return  x,y;
        else
           return y,x;
      }
      main()
      {
         printf("%d" , f(2,3) ) ;
      }
```

注意！取后面那个！（虽然说他同时返回了两个值）

11、以下程序运行后输出结果的第一行是 *2 3 4 1* 第二行是 *0* 。

```
#include<stdio.h>
     void f(int *a,int n, int times)
     {  int i,t=a[0];
        for(i=0;i<n;i++)a[i]=a[i+1];
        a[n-1]=t; times++;
     }
     main( )
     { int a[5]={1,2,3,4},i;
       f(a,4, a[4]);
       for(i=0;i<4;i++)printf("%d ",a[i]);
       printf("\n%d",a[4]);
     }
```

12、以下程序运行后输出结果的第一行是 *6* 第二行是 *15* 。

```
#include<stdio.h>
     int f()
     {  int i,s=0;
        for(i=1;i<=3;i++)
       { static int i=1;
          s=s+i;   i++;
       }
       return s;
     }
```

```
    main( )
{  int i;
       for(i=0;i<2;i++)
           printf("%d\n" , f( ) );
    }
```

13、以下程序运行后输出结果的第一行是 *2* 第二行是 *12* 。

```
#include<stdio.h>
        int f(int n,int x)
        {  if(n==1)
              return  x;
           else   return f(n-1,x+2)+x;
         }
        main( )
        {
           printf("%d\n",f(1,2));  printf("%d",f(3,2));
        }
```

14、以下程序运行后输出结果的第一行是 1 4 9， 第二行是 2 3 8， 第三行是 5 6 7 。

```
#include<stdio.h>
 int f(int (*a)[5],int m,int n)
{ int i,j,k;
      a[0][0]=m;
      for(k=1;k<n;k++)
      {  i=k; j=0;
         a[i][j]=++m;
         while(j<i)a[i][++j]=++m;
         while(i>0)a[--i][j]=++m;
      }
}
main( )
{  int i,j,x[5][5]={0};
       f(x,1,3);
       for(i=0;i<3;i++)
       { for(j=0;j<3;j++)
             printf("%d ",x[i][j]);
         printf("\n");
       }
}
```

15、以下程序运行后输出结果的第一行是 C 第二行是 this, VC,C 。

```
        #include<string.h>
        #include<stdio.h>
        void ss(char a[][10] , int *m , char b[ ][10] , int *n)
        {  int i , j , k;
           for(i=0 ; i<*m ; i++) {
                for(j=0;j<*n;j++)
                       if(strcmp(a[i],b[j])==0) {
                          for(k=i;k<*m-1;k++)
                                 strcpy(a[k],a[k+1]);
```

```
                        *m=*m-1;  i--;   break; }
            }
            for(i=0;i<*m;i++)
                 strcpy(b[i+*n],a[i]);
            *n=*n+*m;
         }
main( )
{  char a[][10]={"this","C"};
       char b[10][10]={"this","VC"};
       int i,m=2, n=2;
       ss(a,&m,b,&n);
       for(i=0;i<m;i++)
           printf("%s ",a[i]);
       printf("\n");
       for(i=0;i<n;i++)
           printf("%s ",b[i]);
}
```

16、下列程序运行后的输出是　9　。

```
   main( )
    {
       union{
                int i[2];
                long k;
                char c[4];
      }r={071,070},*s=&r;
      printf(“%c” ,s->c[0]);
    }
```

17、下列程序运行后的输出是　10，20 。

```
void swap(int *x,int *y)
{
      int temp;
      temp=*x;
      *x=*y;
      *y=temp;
}
void SWAP(int x,int y)
{
      swap(&x,&y);
}
main( )
{
      int a=10,b=20;
      SWAP(a,b);
      printf(“%d , %d\n”,a,b);
}
```

18、下列程序运行后的输出是　14　　。

```
        main( )
        {
          int f1(int),f2(int) sum(int (*f)(int),int ,int );
```

```
          printf("%d \n",sum(f1,1,2)+sum(f2,1,2));
        }
        int sum(int (*f)(int),int m,int n)
        {
          int k,sum=0;
          for(k=m;k<=n;k++)
                sum+=(*f)(k)*(*f)(k);
          return sum;
        }
int f1(int x)
{
          return x+1;
}
int f2(int x)
{
          return x-1;
}
```

19、 执行以下程序后，输出结果的第一行是 77，第二行是 30，最后一行是 101

```
           int main( )
           {   int x=30;
               {
                   int x=77;
                   printf(" %d\n",x);
               }
               printf(" %d\n",x);
               while(x++<33){
                           int x=100;
                           x++;
                           printf(" %d\n",x);
               }
           }
```

20、运行以下程序后，输出结果的第一行是 her，

```
第二行是  teac  ,第三行是  the   。
```

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
    struct node{
                char info[5];
                struct node *link;
};
main( )
{ struct node *create(char *);
      void print(struct node *);
      struct node *head=NULL;
      char c[ ]="the teacher";
      head=create(c);
      print(head);
}
     struct node *create(char *s)
{ int k;
```

```
      struct node  *h,*p;
      h=NULL;
      while(*s){
            k=0;
            p=(struct node *)malloc(sizeof(struct node ));
            while(k<4&&*s)
                        p->info[k++]=*s++;
            p->info[k]='\0';
            p->link=h;
            h=p;
    }
    return h;
}
void print(struct node *head)
    {
      struct node *p=head;
      while(p!=NULL){
                    puts(p->info);
                    p=p->link;
      }
    }
```

21、运行以下程序后，输出结果的第一行是 7 8 1，

```
第二行是    6 9 2     ,第三行是    5 4 3
```

```
main( )
    { int i=0, j, k=1, n=3, arr[20][20];
            while(i<n/2){
                      for(j=i; j<n-i;j++) arr[j][n-i-1]=k++;
                      for(j=n-i-2; j>=i;j--) arr[n-i-1][j]=k++;
                      for(j=n-i-2; j>=i;j--) arr[j][i]=k++;
                      for(j=i+1; j<=n-i-2;j++) arr[i][j]=k++;
                      i++; }
            if(n%2) arr[i][i]=k;
            for(i=0; i<n;i++){
                   for(j=0; i<n;j++) printf("%4d", arr[i][j]);
                   printf("\n");
            }
         }
```

22、运行以下程序后，输出结果的第一行是 1010，第二行是 12 。

```
void change(int x, int m)
{
ch[]={'0','1','2','3','4','5','6','7','8','9'}, b[80];
       int i=0; r;
       while(x){
                 r=x%m; x/=m;
                 b[i++]=ch[r];
       }
       for(--i;i>=0;i--)
           printf("%c",b[i]);
}
main( )
```

```
{ int a,b;
      change(10,2);
      printf("\n");
      change(10,8);
}
```

23、下面程序的输出结果是 3,4,2,4,

```
#include<stdio.h>
int a[][2]={1,2,3,4};
void main()
{ int (*p)[2],i;
      p=a;
      printf("%d, %d,",p[1][0],(*(p+1))[1]);
      for(i=0;i<2;i++) printf("%d,",*(*(p+i)+1));
}
```

24、下面程序的输出结果是 1

```
#include<stdio.h>
int f( int x,int y)
{ return (y-x);}
main()
{ int a=5,b=6,c=2;
      int (*g)(int,int)=f;
      c=(*g)(a,b);
      printf("%d\n",c);
}
```

25、下面程序的输出结果是 abcd,bcd,efgh,mnpq

```
#include<stdio.h>
#define T "%s,%s\n"void main()
{ char *x[]={"abcd","efgh","mnpq","rstu"};
      char **y;
      y=x;
      ++y;c
      printf(T,*x,*x+1);
      printf(T,*y,y[1]);
}
```

26、下面程序运行的正确结果是 3

```
#include<stdio.h>
#define N 5
int fun(char *s,char a,int n)
{ int j;
      *s=a;
      j=n;
      while(*s<s[j]) j--;
      return j;
}
void main()
{  char c[N+1];
       int b;
```

```
        for(b=1;b<=N;b++)    *(c+b)='A'+b+1;
        printf("%d\n",fun(c,'E',N));
}
```

27、下面程序运行的正确结果是 1,y

```
      struct tree
      {  int x;  char c;
       } t ;
      func(struct tree t)
      {  t.x=10;  t.c='x';
      }
      void main()
      {  t.x=1;
         t.c='y';
         func(t);
         printf("%d,%c\n", t.x, t,c);
      }
```

28、下面程序运行的正确结果是 MNQ

```
struct str1
       { char c[5];  char *s;  };
       void main()
       {   struct str1 s1[2]={"ABCD", "EFGH", "IJK", "LMN"};
           struct str2
          { struct str1 sr;  int d;
           } s2={"OPQ", "RST", 32767};
           struct str1 *p[2];
           p[0]=&s1[0];  p[1]=&s1[1];
           printf("%s", ++p[1]->s);
           printf("%c", s2.sr.c[2]);
       }
```

29、以下程序运行时输出结果为 3 。

```
#include<stdio.h>
#define Y 2
#define  M(y)  ((y)+(y)/Y)
main()
{ printf("%d",M(Y));
}
30、以下程序运行时输出结果为    1  2  6     。
#include <stdio.h>
int fun(int n)
{   static int j=1;
      j*=n++;
      return j;
}
void main(void)
{ int k;
    for(k=1; k<4; k++)
    printf("%4d",fun(k));
}
```

31、以下程序运行时输出结果是 2468 .

```
#include <stdio.h>
int f(int n)
{ if(n<1)return;
    f(n-2);
    printf("%d",n);
}
main()
{ f(8); }
```

32、以下程序运行时输出结果第一行是 6,5 第二行是 5,6 。

```
#include<stdio.h>
#define f(x,y,z)  z=x,x=y,y=z
void g(int x,int y)
{ int z; z=x,x=y,y=z; }
main()
{
     int a=6,b=5,c;
      g(a,b);
      printf("%d,%d",a,b);
      a=6;b=5;
      f(a,b,c);
      printf("\n%d,%d",a,b);
}
```

33、以下程序运行时输出结果第一行是 2 3 1 第二行是 1 2 3 第三行是 3 1 2 。

```
#include<stdio.h>
void exchang(int x[][3],int n,int m)
{ int i,k;
     for(i=0;i<3;i++)
     { k=x[n][i];x[n][i]=x[m][i];x[m][i]=k; }
}
main()
{ int a[3][3]={3,1,2,1,2,3,2,3,1},*p[3],i,j;
     for(i=0;i<3;i++)    p[i]=a[i];
     for(i=0;i<2;i++)
         for(j=0;j<2-i;j++)
              if(p[j][1]<p[j+1][1])exchang(a,j,j+1);
     for(i=0;i<3;i++)
     {    for(j=0;j<3;j++) printf("%3d",a[i][j]);
          printf("\n");
     }
}
```

34、执行以下程序段后，输出的第一行是_12345 _,输出的第二行是1817

```
#include <stdio.h>
void func(int m, int n)
{   if(m<n) printf("%d",m);
       else{ func(m/n,n); printf("%d",m%n); }
}
void main()
{   void (*p)(int,int);
       p=func;
       (*p)(12345,10);   putchar('\n');
       (*p)(377,20);     putchar('\n');
}
```

35、执行以下程序段后，在main函数返回之前，全局变量x的值是_*0,40,80,120*,main函数中局部变量y的值是*20,60,100,140*.

```
#include <stdio.h>
int x;
int func(int n)
{   static char m=1;
       m=m*8;            x+=m*n--;
       return(--n);
}
void main()
{   int i,y;
       for(i=2;i<8;i+=2)    y=func(i);
}
```

36、执行以下程序段后，输出的第一行是_*2,1*,输出的第三行是_3,3 _;

```
#include <stdio.h>
void main()
{
       int a[16],i,j,t;
      for(i=0;i<16;i++) *(a+i)=i*10;
     for(i=0;i<4;i++)
         for(j=0;j<i;j++)
         {  t=*(a+i*4+j);
         *(a+i*4+j)=*(a+j*4+i);
         *(a+j*4+i)=t;  }
     for(i=1;i<=16;i++)
     {  printf("%4d,",a[i-1]);
        if(i%4==0) printf("\n");
     }
}
```

37、执行以下程序段后，a[0]的值是_*12,4* ,a[9]的值是 30 ;

```
void main()
{   int a[10]={104,4,19,7,23,56,49,97,33,35};
    int i=0,j=9,k,flag,tmp;

    while(i<=j)
    {   flag=1;
        while(flag)
        {   for(k=2;k<a[i];k++)
```

```
                if(a[i]%k==0) break;
    if(k<a[i]) flag=0;
            else i++;
            if(i>j) flag=0;
        }
        flag=1;
        while(flag)
        {   for(k=2;k<a[j];k++)
                if(a[j]%k==0) break;
            if(k<a[j]) j--;
            else flag=0;
            if(i>j) flag=0;
        }
        if(i<j) {tmp=a[i]; a[i]=a[j]; a[j]=tmp;}
    }
}
```

38、执行以下程序段后，输出的第一行是_ GooD__；

```
#include <stdio.h>
#include <string.h>
void func(char *a, int n)
{   int i,t;
    char *p=a;
    for(i=0;i<n;i++)
    {   if(*p>='a' && *p<='z' )
        {   t=(*p-'a'+3)%26;
            *p='A'+t;
        }
        else if(*p>='A' && *p<='Z' )
            {   t=(*p-'A'+3)%26;
                *p='a'+t;
            }
        p++;
    }
}
void main()
{   char s[][100]={"dLLa","iRZh:)"};
    func(s[0],strlen(s[0])); puts(s[0]);
    func(s[1],strlen(s[1])); puts(s[1]);
}
```

DLX 指令集-YDJSIR 重制版

操作的含义	DLX 汇编语言中的对应结构				31 26	25 21	20 16	15 11	10 6	5 0
加	ADD	DR,	SR1,	SR2	000000	SR1	SR2	DR	00000	000001
加立即数	ADDI	DR,	SR1,	Imm/LABEL	000001	SR1	DR	Imm16		
减	SUB	DR,	SR1,	SR2	000000	SR1	SR2	DR	00000	000011
减立即数	SUBI	DR,	SR1,	Imm/LABEL	000011	SR1	DR	Imm16		
按位与	AND	DR,	SR1,	SR2	000000	SR1	SR2	DR	00000	001001
按位与立即数	ANDI	DR,	SR1,	Imm/LABEL	001001	SR1	DR	Imm16		
按位或	OR	DR,	SR1,	SR2	000000	SR1	SR2	DR	00000	001010
按位或	ORI	DR,	SR1,	Imm/LABEL	001010	SR1	DR	Imm16		
按位异或	XOR	DR,	SR1,	SR2	000000	SR1	SR2	DR	00000	001011
按位异或立即数	XORI	DR,	SR1,	Imm/LABEL	001011	SR1	DR	Imm16		
加载高位立即数	LHI	DR,	Imm		001100	00000	DR	Imm16		
左移	SLL	DR,	SR1,	SR2	000000	SR1	SR2	DR	00000	001101
左移立即数	SLLI	DR,	SR1,	Imm/LABEL	001101	SR1	DR	Imm16		
逻辑右移	SRL	DR,	SR1,	SR2	000000	SR1	SR2	DR	00000	001110
逻辑右移立即数	SRLI	DR,	SR1,	Imm/LABEL	001110	SR1	DR	Imm16		
算数右移	SRA	DR,	SR1,	SR2	000000	SR1	SR2	DR	00000	001111
算数右移立即数	SRAI	DR,	SR1,	Imm/LABEL	001111	SR1	DR	Imm16		
小于	SLT	DR,	SR1,	SR2	000000	SR1	SR2	DR	00000	010000
小于立即数	SLTI	DR,	SR1,	Imm/LABEL	010000	SR1	DR	Imm16		
小于等于	SLE	DR,	SR1,	SR2	000000	SR1	SR2	DR	00000	010010
小于等于立即数	SLEI	DR,	SR1,	Imm/LABEL	010010	SR1	DR	Imm16		
等于	SEQ	DR,	SR1,	SR2	000000	SR1	SR2	DR	00000	010100
等于立即数	SEQI	DR,	SR1,	Imm/LABEL	010100	SR1	DR	Imm16		
加载字节	LB	DR,	Imm/LABEL(SR1)		010110	SR1	DR	Imm16		
存储字节	SB	Imm/LABEL(SR1),	DR		010111	SR1	DR	Imm16		
加载字	LW	DR,	Imm/LABEL(SR1)		011100	SR1	DR	Imm16		
存储字	SW	Imm/LABEL(SR1),	DR		011101	SR1	DR	Imm16		
(授权)存储字	MOVI2S	GPR	NUMBER		100010	00000	GPR	NUMBER	000 0000 0000	
(授权)加载字	MOVS2I	GPR	NUMBER		100011	00000	GPR	NUMBER	000 0000 0000	
条件假跳转	BEQZ	SR1,	LABEL		101000	SR1	00000	Imm16		
条件真跳转	BNEZ	SR1,	LABEL		101001	SR1	00000	Imm16		
跳转-偏移量	J	LABEL			101100	PCOffset26				
跳转-寄存器	JR	SR1			101101	SR1	0 0000 0000 0000 0000 0000			
从 TRAP 返回	RET	(JR R31)			101101	11111	0 0000 0000 0000 0000 0000			
链接-偏移量	JAL	LABEL			101110	PCOffset26				
链接-寄存器	JALR	SR1			101111	SR1	0 0000 0000 0000 0000 0000			
自陷 GETC	TRAP	x06			110000	00 0000 0000 0000 0000 0000 0110				
自陷 OUT	TRAP	x07			110000	00 0000 0000 0000 0000 0000 0111				
自陷 PUTS	TRAP	x08			110000	00 0000 0000 0000 0000 0000 1000				
自陷 IN	TRAP	x09			110000	00 0000 0000 0000 0000 0000 1001				
自陷 GETS	TRAP	x0A			110000	00 0000 0000 0000 0000 0000 1010				
自陷 HALT	TRAP	x00			110000	00 0000 0000 0000 0000 0000 0000				
(授权)恢复 PC	RFE				110001	00 0000 0000 0000 0000 0000 0000				

图例以及拓展解释在下一页。

常见缩写

黄色是 I 型指令，蓝色与 IO 相关，绿色与逻辑相关，红色与控制相关。

1、R 型指令 31-26 位必定全为 0；与系统相关的 31 位必为 1，其余为 0；未用的位一定是 0；

2、TRAP x00 会导致数据丢失，但是能让程序停止运行；

3、关于算术左/右移与逻辑左/右移：①如果是无符号数，不管是左移还是右移都是"逻辑移位"；②如果是有符号数，则做左移运算，即做的是"逻辑移位"，同 ①中无符号数的左移。而做右移运算，那么做的是"算术移位"。

特征：算术移动是进行符号拓展（TRAP 向量里面也是），逻辑移动一概在空位补 0，以此类推；

用途：左移 n 位等同于 x 乘以 2 的 n 次方，算术右移 n 位等同于除以 2 的 n 次方并舍去余数；这可以极好地取代运算极为缓慢的%取余命令！

4、关于 AND，OR 与 XORI：它们可以十分有效地用于格式化特定位的内容，或者取反，常用于各种逻辑判定的条件中，使用起来相当灵活；

5、注意 DLX 里面都是比小于或者小于等于，没有大于！为什么？节约宝贵的指令分配资源！

Smaller or Less Than = SLT

Smaller or Less than or Equal to = SLE

6、所有与比较相关的内容，DR 中均以 1 为真，0 位假。

详细内容请参看对应章节的 PPT！

DLX 指令集附加

伪操作

.data address	DLX 数据存放在数据区地址 address，必须以此作为 DLX 代码的开头！
.align n	将下面的数据或代码加载到二进制以 n 个 0 结尾的地址中，用于对齐
.word word1,word2 …	将字 1、字 2、…连续存储到存储单元中。(32 位)
.ascii "string1", "string2" …	将字符串 1、字符串 2、…连续存储到存储单元中。
.asciiz "string1", "string2" …	分别在在字符串 1、字符串 2、…末尾添加 x00 后连续存储到存储单元中。
.space size	在数据区留出 size 个连续存储单元，其前面常有一个 Label
.text address	DLX 指令存放在地址 address 中。
.global label	使标记 label 全局（使以后加载的其他文件也可以使用此标记）。 必须有.global main，且执行从标记 main 的指令开始。
.extern label	使用其他文件的全局标记 label，如引用其他库函数。
.byte byte1,byte2 …	将字节 1、字节 2、…连续存储到存储单元中。(8 位)

注：没有 x00，使用 TRAP x09 输出可能会导致异常！

设备寄存器映射地址

MCR	机器控制寄存器	xFFFF 00F8
KBSR	键盘状态寄存器	xFFFF 0000
KBDR	键盘数据寄存器	xFFFF 0004
DSR	显示器状态寄存器	xFFFF 0008
DDR	显示器数据寄存器	xFFFF 000C

特殊寄存器编号

SR	12	x0C	中断屏蔽，中断允许
CAUSE	13	x0D	未决中断位
EPC	14	x0E	PC

EPC 仅在中断里面保存返回地址，TRAP x06-x09，子例程-函数之类需要返回的请使用 R31！系统中断优先级那部分请参阅课本，其中给出两个特殊的指令。

31 26 25 21 20 16 15 11 10 0

MOVI2S	100010	00000			00000 00000 0
MOVS2I	100011	00000			00000 00000 0

未用 GPR 特殊寄存器 未用

DLX 寄存器

GPR 通用寄存器

I/O 寄存器：R4

局部变量寄存器：R16、R17、R18、R19、R20、R21、R22、R23

临时寄存器：R8、R9、R10、R11、R12、R13、R14、R15、R24、R25

全局指针：R28

栈指针：R29

帧指针（框架指针）：R30

PC 寄存器：R31

HALT 影响寄存器：R1、R2、R4、R31

不可改写（值为 0）：R0

参数传递寄存器：R4、R5、R6、R7

返回值寄存器：R2、R3

MCR 机器控制寄存器

KBSR 键盘状态寄存器：提供键盘键入字符的状态信息

KBDR 键盘数据寄存器：保存由键盘键入的字符的 ASCII 码

DSR 显示器状态寄存器：提供显示器显示的状态信息

DDR 显示器数据寄存器：保存将被显示在显示器上的内容的 ASCII 码

PC 程序计数器：指向下一条指令的地址

IR 指令寄存器：保存正在处理的指令

MAR 地址寄存器：用于寻址

MDR 数据寄存器：用于存取单元内容

CAUSE 原因寄存器：记录哪些设备发出中断信号（特权模式允许访问）

SR 状态寄存器：决定谁能中断处理器（特权模式允许访问）

EPC 中断时保存 PC

DLX 模拟器

1.配置：dlx 文件夹所在目录配置到环境变量 path 中。

2.汇编：dlxassembler *.dlx 输出*.link

3.链接：dlxlinker *.link *.bin 输出*.bin，由模拟器运行。

DLX 汇编语法：英语分号；作为注释，；后该行内内容作为注释； . 开头的都是伪操作；数字和寄存器用英语逗号，作分隔；偏移量可以是数字或者是 label，其基准的寄存器用括号括起来；

其他具体使用操作请参看配套的讲解视频。

YDJSIR心中的计基重点

> 鉴于细节上的内容已经在前面的文档中总结得非常详细了，这里来说的话就把重点或者说在实践中要注意的点特别地总结出来放在这里，供大家参考。

C语言

1. 冒泡排序

这种早已结构化的东西，想必各位已经烂熟于心了吧。

```
//降序排列
for(int i=1;i<n;i++){
        for(int j=n-1;j>=i;j--){
            if(a[j]>a[j-1]){
            int temp=a[j-1];
            a[j-1]=a[j];
            a[j]=temp;}
        }
    }
//升序排列
for(int i=1;i<n;i++){
        for(int j=n-1;j>=i;j--){
            if(a[j]<a[j-1]){
            int temp=a[j-1];
            a[j-1]=a[j];
            a[j]=temp;}
        }
    }
```

2. 快速排序（调用库函数`qsort`）

```
//这是qsort调用时所需要的函数，qsort自身在stdlib里面就有，没有违反规定；
//注意要自己写一个调用时比较中使用的函数，用于告诉qsort调换顺序的触发条件；
int comp(const void *a, const void *b)
{
  return -(*(int *)a - *(int *)b);//这是期望降序时的输入；
  //return (*(int *)a - *(int *)b);//这是期望升序时的输入；
}
```

```
qsort (values, num, sizeof(int), comp);
//values是指向数组等结构开头的指针，第二个数是需要比较的数的数量；
//第三个是元素的类型对应大小，最后一个是刚才写的比较函数的指针；
```

快速排序的具体原理这里不展开详述，各位有兴趣可以自行了解。

3. 各类数据的IO输入与输出

目前来看我们做过的最复杂的应该就是上次那个矩阵了吧，涉及到了多处判断（进去哪个数组哪一行，如何判断是否开始或者是否结束，这都需要特别留神！下面选取的是老师提供的标准答案！

```c
#include <stdio.h>
int main(){
long long a[505][505];
long long b[505][505];
int n,p,m;
long long pp = 1000000007;
scanf("%d %d %d",&n,&p,&m);
for(int i=0;i<n;i++){
for(int j=0;j<p;j++){
scanf("%lld",&a[i][j]);//思考一下，这里要是没有提供n,p,m，又该怎么办呢？
}
}
//小诀窍：选中页面中输入样例，假如行末数字后还有个空白，那大概率是个\n，其实一般也都是\n;
for(int i=0;i<p;i++){
for(int j=0;j<m;j++){
scanf("%lld",&b[i][j]);
}
}
for(int i=0;i<n;i++){
for(int j=0;j<m;j++){
long long sum=0;
for(int k=0;k<p;k++){
long long temp=((a[i][k]))*((b[k][j]))%pp;
sum= (sum + temp)%pp;
}
printf("%lld ",(sum+pp)%pp);
}
printf("\n");
}
}
```

还记得之前那个可恶的EOF结尾的数组吗?

```c
for (i = 0;i<100 ; i++)
{
    scanf("%d", &b[i]);
    n++;
    if (getchar() == EOF)//见证奇迹发生的时刻
        break;
```

`getchar`的在获知输入流是否结束方面有奇效！结合动态分配内存的数组更香！

`scanf`的强大与局限性各位想必已经在翻转字符串那个题目里面体验到了。要一次获取一行，请使用以下命令。`scanf`的最大好处，莫过于可以自动实现类型转换了。

```c
fgets(a, 50, stdin);//stdin是我们默认的输入流;
gets(a);
//a是char型数组名;
```

这里展示一个手写的类似gets函数。和DLX里面一样，结尾的\0不要省掉！

```c
void getstring(){
    int i=0;

    for(i=0;i<100;i++){
```

```
        binary[i]=getchar();
        if(binary[i]=='\n'){
            binary[i]=='\0';
            length=i;
            //printf("Done!");
            printf("length=%d",length);
            break;
        }
    }
}
```

总而言之，我们要根据题目的实际情况作出考量，从而巧妙地高效地实现数据流的录入。

这方面可以多看下www.cplusplus.com 上对各类函数声明的详细讲解。

4. 数据类型的转换

实际上，C语言的`<stdlib>`内很贴心地为我们提供了字符串和数字之间进行相互转换的函数，这是标准库里面的内容，没有超出要求，当然进制转换什么的当然还是自己动手吧，反正考试时间很充足的。不过貌似MOOCCODE上会出问题。

- 字符串→数字（亲测可用）

```
atoll(const char * str)//转成long long
atoi(const char * str)//转成int
atof(const char * str)//转成double
```

反过来亦可！把`to`前后的内容交换即可。下面是一个综合的示例。

```
int main(int argc, char *argv[]) {
    int i=0;
    int result=0;
    char buffer[100];
    scanf ("%d",&i);
    itoa (i,buffer,10);
    printf("%s\n",buffer);
    result=atoi (buffer);
    printf("%d\n",result);
    return 0;
}
```

注意到其中的`itoa`函数，后面的那个数可以直接实现进制的转化，支持2-36进制（26个字母用完了没办法）。

当然，你显然可以不辞劳苦自己写一次，我对你表示佩服。

所以当然也要有一个例子。以二进制转换十进制为例。这里的难点时要特别注意循环进入与跳出的条件。可以结合后面的DLX一起理解？这是我一个项目中的一个片段。

千万别忘了你对字符0-9运算的时候要考虑到这是当作ASCII码用的！

```
void binarytodecimal(){
    int i=0;
    int j=0;
    long long decimal=0;
    long long temp=1;
    for(i=0;i<length;i++){
```

```
        int temp=1;
        for(j=0;j<length-i-1;j++){//本人在这一步被坑了很久……length-i-1，注意！
            temp=2*temp;
        }
        decimal=decimal+(binary[i]-48)*temp;
    }
    printf("\n%d\n",decimal);
}
```

提到二进制，我们又不得不提到C语言作为中级语言的高明之处。C语言可以直接对位进行操作。所以在MOOCCODE上的一条题里面，我们看到了位运算的一个绝妙的应用题。除2取余用位运算比%快得多了！下面还是举这个例子。其实这个例子在DLX的习题里面也出现过。

```
int NumOf1(int num){
    int result=0;
    int one=1;
    while(num>0){
        if(num%2==1){
            num-=1;
            result++;
        }
        num=num>>1;
        //位运算大显身手
    }
    return result;
}
```

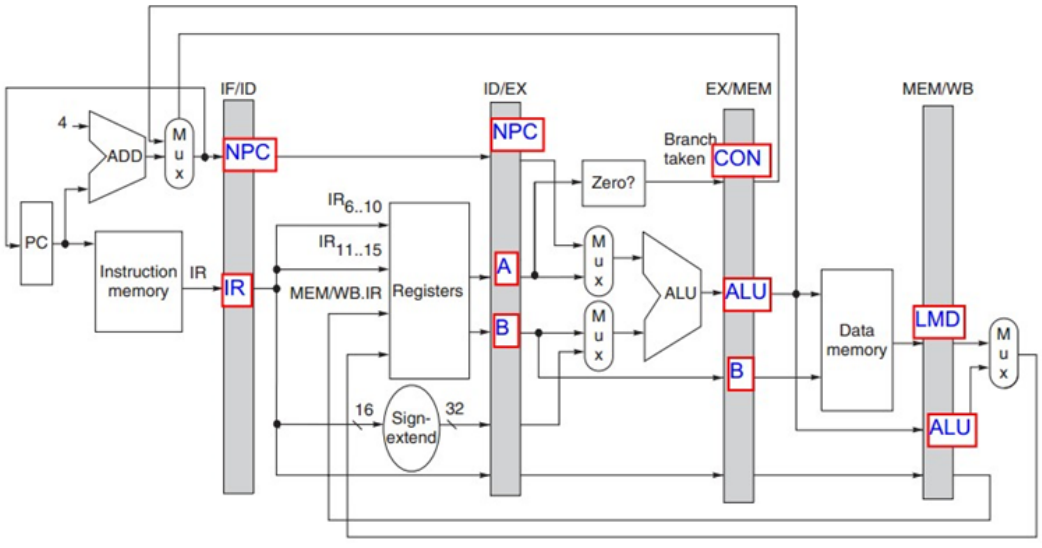
位运算的详细用法这里就不展开了。

DLX

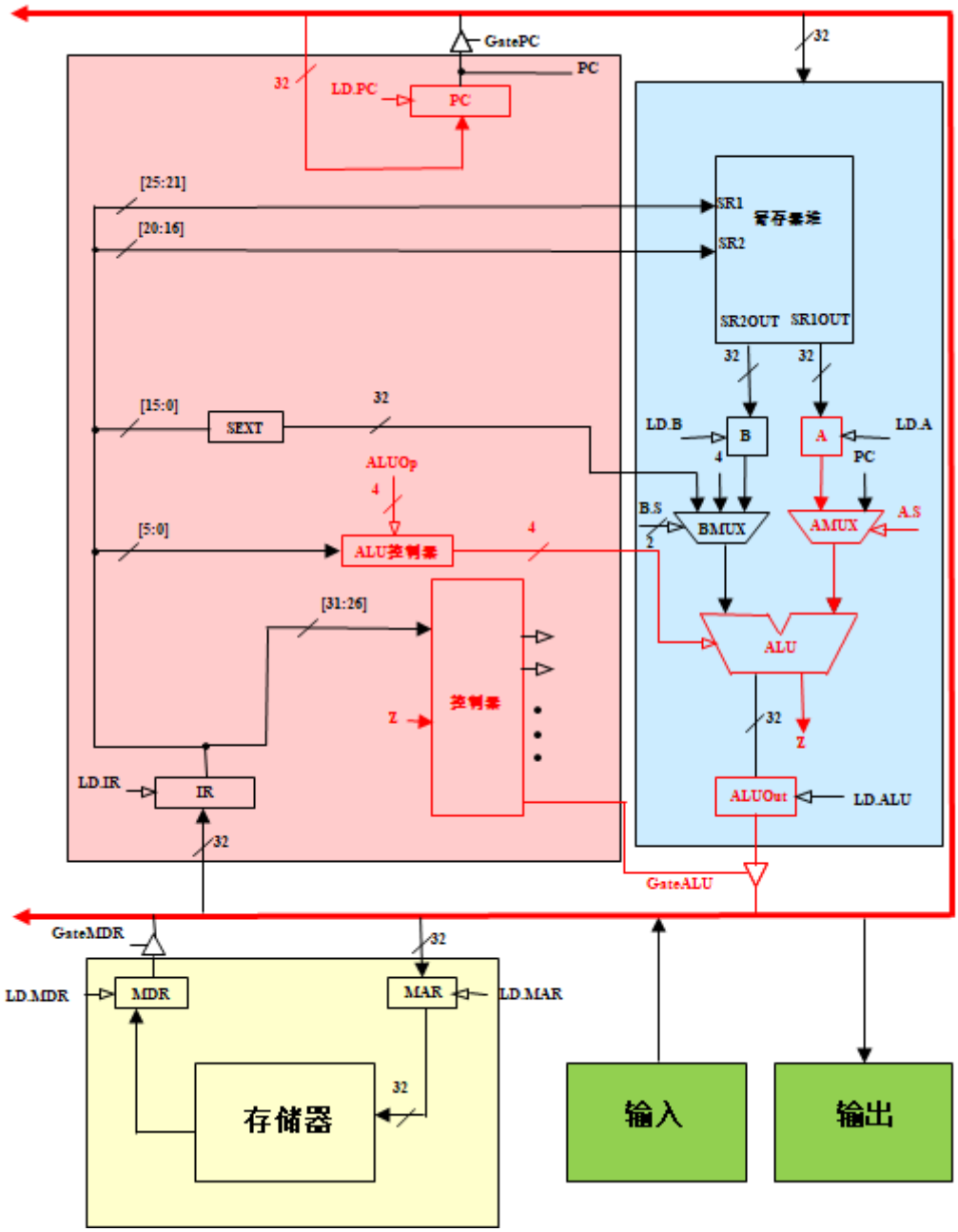
1. 具体指令部分

这部分我已经在`EX DLX指令集`详解那部分中介绍得非常透彻，相关的注意事项也会更新到`EX DLX指令集`部分中，因而在此不再赘述。

至于硬件结构部分，这个还是扎扎实实看书吧，理解透彻，毕竟问这些的难度和切入点有限。



还有那个完整的大图……



至于2的多少次方是多少，我想2^{16}以内的大家应该都要熟记吧。

当然，实际开发环境中Windows自带的计算器的程序员模式也是很香的。

2. 具体编程部分

课上老师说过，其实后面的子例程，系统中断以及衍生的栈结构、递归，本质上都是各种各样的子例程。因此下面将着重讲子例程。

下面是汇编语言（DLX）相较机器语言的部分优点（这个应该不会考吧……）

— **目的**

—1. **程序设计的用户友好性比机器语言强**

—2. **精确控制计算机能够执行的指令**

—3.**便于记忆的符号**

- **操作码，例如 ADD 和 AND**
- **存储单元，例如 SUM 和 LOOP**

- **符号地址**
- **标记**

—在符号表的帮助下，再次遍历汇编语言程序

—汇编语言指令被翻译成机器语言指令

—其实编译器比你 想象的聪明！ 编译器会做出很多优化，只是你暂时看不到（这里的编译器是教学用仅做示范）

—详细：编译原理

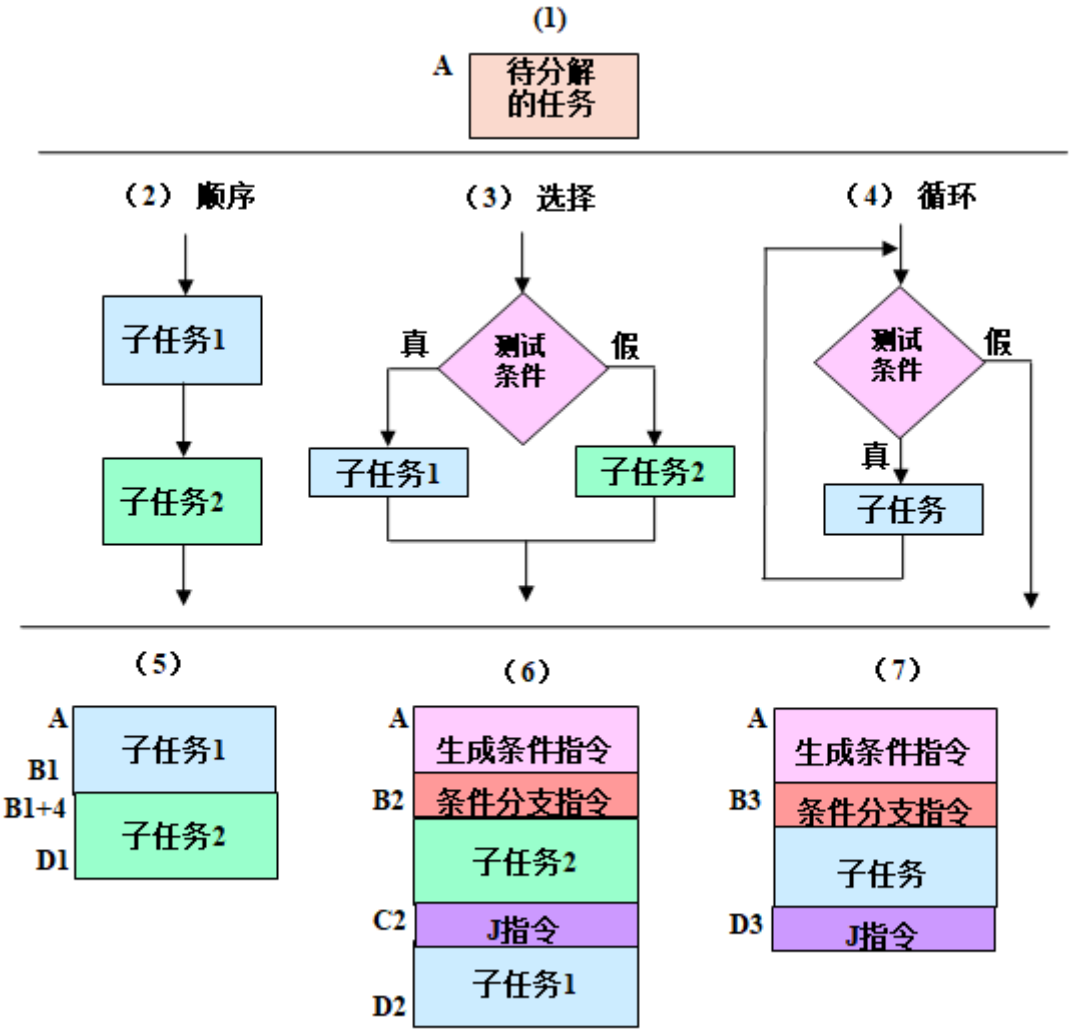
—鸡生蛋还是蛋生鸡的问题不存在：高级语言编译器肯定是用低级的语言弄的；

注意到DLX单语句只能做些相当基础的事情：

- 整数加减（本课程不涉及浮点，但是你可以知道有32个32位浮点寄存器）；
- 两数是否相等/小于/小于等于的判断；

  特别注意：
- 条件跳转与无条件跳转；

所以我们要回到任何计算机程序的三种最基本的结构。



此前我们的开发更多是面向过程，而现在我们要更深入计算机运算的底层，把原来经过抽象后的简单还原为真实的复杂。这也许正是本课程的目的吧。如果觉得实在困难，不如先构思C的写法然后尝试脑力编译模拟？

特别提示：不要忘了下条指令PC自动+4那个设计，所以图上J指令该条多少大家自己想清楚。

生成条件指令一般涉及三个寄存器，两个做比较一个作判断用的寄存器，存储一个bool值。

附加：寄存器分配的一些原则

R0	**0**
R1	**汇编器保留**
R2、R3	**返回值**
R4~R7	**参数**
R8~R15	**临时值**
R16~R23	**局部变量**
R24、R25	**临时值**
R26、R27	**操作系统保留**
R28	**全局指针**
R29	**栈指针**
R30	**帧指针**
R31	**返回地址**

常见问题列举

1. 如何将R1的值设为*x3000 000C*?

立即数超出范围？注意，此处符号表的变化要特别注意！

0D **main:** addi r1, r0, *numbers* ; numbers=x3000000C，一个label的地址

翻译为两条指令-编译器其实会考虑到这个问题

LHI R1, x3000 ; R1=x3000 0000
ADDI R1, R1, x000C ; R1=x3000 000C

2. for循环的实现？while呢？简单点的，if-else呢？

```
    ;for循环一例
    ;摘取自判断一段连续的存储单元内是否包含一个5的例子
again:  beqz    r3, exit;生成判断条件

        seqi    r5, r2, #5          ;如果是5就把R1这个bool变量变成1并跳出当前循环，理解为C中的break
        bnez    r5, setR1
        addi    r4, r4, #4           ;下一个整数地址

        subi    r3, r3, #1

        lw  r2, 0(r4)                   ;加载下一个整数到R4中

        j   again               ;循环往复
;
```

```
    ;while循环一例
    ;while R2>0的实例
again:  beqz    r2, exit
        lw  r4, 0(r1)
        add r3, r3, r4
        addi    r1, r1, #4          ;R1跟踪下一个整数地址
        subi    r2, r2, #1
        j   again
   ;do while呢？
   ;把判断条件跳转放后面就好！
   ;
   ;下面是if-else一例，其实反映到图上也很简单

MAIN :          ADDI    R1, R0, #0
            ADDI    R2, R0, #0
            LHI     R5, X3000
            LW      R5, 0(R3)
LOOP :          ANDI    R6, R5, #1
            BEQZ    R6, EVEN
            ADDI    R1, R1, #1
            J       NEXT
EVEN :          ADDI    R2, R2, #1
NEXT :          ADDI    R3, R3, #4
            LW      R5, 0(R3)
            SEQI    R6, R5, #-1
            BEQZ    R6, LOOP
            TRAP    X00
   ;下面是switch-case一例，switch的那个对象放在R3里面,这个其实课本上有一个类似练习。
SWITCH:      ;生成R3的值
                ADDI R2, R2, #0
                ;
CASE_01:        SEQI R2, R3, x01
            BEQZ R2, CASE_02
                ;Operation-x01
                J NEXT_TASK
CASE_02:        SEQI R2, R3, x02
            BEQZ R2, CASE_03
                ;Operation-x02
                J NEXT_TASK
                ;
                ;
                ;
                J NEXT_TASK
DEFAULT:        ;Operation-Default
              J NEXT_TASK
```

3. 栈的综合性运用

A. 保存过程中产生的值又保存递归调用时要保存的返回地址-R31

下面先以阶乘为例。这里用栈保存的不仅仅是R31的返回值，还有f(n)的数据。

```
            .DATA   x30000000
STACK :     .SPACE  40
            .TEXT   x40000000
```

```
            .GLOBAL  MAIN
MAIN :      ADDI     R29, R0, STACK
            ADDI     R29, R29, #40
            ADDI     R1, R0, #3       ;n=3
            JAL      FACTORIAL        ;f(n)
            ;TRAP     x00



FACTORIAL : SUBI     R29, R29, #4         ; 压栈保存R1
            SW       0(R29), R1
            ; R3，R4，R5压栈保存
            SUBI     R29, R29, #4         ; 压栈保存R31
            SW       0(R29), R31
            SEQI     R5, R1, #1       ;n==1?
            BNEZ     R5, EXIT1
            SUBI     R1, R1, #1       ;n--
            JAL      FACTORIAL        ;f(n-1)
            ADDI     R3, R2, #0       ; f(n-1)
            ADDI     R4, R1, #1       ;n，这里貌似就体现了n--而非--n的作用
            ANDI     R2, R2, #0
LOOP :      BEQZ     R4, EXIT2        ;计算n * f(n-1)
            ADD      R2, R2, R3
            SUBI     R4, R4, #1
            J        LOOP
EXIT1 :     ADDI     R2, R0, #1       ;f(1)=1
EXIT2 :     LW       R31, 0(R29)      ; 出栈恢复R31
            ADDI     R29, R29, #4
            LW       R1, 0(R29)       ; 出栈恢复R1
            ADDI     R29, R29, #4
            JR       R31

```

B. 课本上子函数那一节包含了R30框架指针的改变与复原的例子。

和课本一样，这里会做C和DLX的对比（不是说会有这样的题吗？）

```
;这是一个标准开头……貌似课本上C编译成DLX时所有变量定义的时候都是默认初始化为0
main:    ......
    addi     r4, r16, #0     ; 变元valueA
    addi     r5, r17, #0     ; 变元valueB
    jal Swap
    ......

```

```
//这是错误的SWAP示范-C
void Swap (int firstVal, int secondVal)  // R4: firstVal, R5: secondVal
{
    int tempVal;                         //R16
    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
}
```

```
;究其原因?
Swap:    subi    r29, r29, #4
         sw      0(r29), r16    ; 压入R16（寄存器的保存）

         addi    r16, r4, #0    ; tempVal = firstVal;
         addi    r4, r5, #0     ; firstVal = secondVal;
         addi    r5, r16, #0    ; secondVal = tempVal;

         lw      r16, 0(r29)    ; R16出栈
         addi    r29, r29, #4
         jr      r31
```

是的，你完成了一个“交换”。但是从这个子例程返回，R16就释放了，R4，R5都恢复原样了，实际上什么也没有发生。这与C语言向子函数传的一般是值有关。下面有正确示范。

```
//这是正确的示范-NewSwap
void NewSwap (int *firstVal, int *secondVal) // R4: firstVal, R5: secondVal
{
    int tempVal;                             // R16
    tempVal = *firstVal;
    *firstVal = *secondVal;
    *secondVal = tempVal;
}
```

在DLX中体现如下:

```
NewSwap:     subi   r29, r29, #4
             sw     0(r29), r16    ; 压入R16（寄存器的保存）

             lw     r8, 0(r4)      ; *firstVal
             addi   r16, r8, #0    ; tempVal = *firstVal;
             lw     r9, 0(r5)      ; *secondVal
             sw     0(r4), r9      ; *firstVal = *secondVal;
             sw     0(r5), r16     ; *secondVal = tempVal;

             lw     r16, 0(r29)    ; R16出栈
             addi   r29, r29, #4
             jr r31
             ;也许你觉得为什么不直接把R4和R5缓过来就好？这里编译器没有作优化。*号传值进去之后就是这样用
```

注意到以下示例中R30的使用。R30让你能在不乱动R29的情况下迅速找到你所需要的量。

```
addi    r8,r0,#4    ; R8 = 4
sw      -12(r30),r8 ; object = 4;
subi    r16,r30,#12 ; ptr = &object;
```

```
//鲜明的对比
int object;  //栈
int *ptr;    //R16

object = 4;
ptr = &object;
```

PPT中的那个图很明显了，这里体现的是&的作用在此我不再赘述。

4. 几个原型例程

特别是那几个IO的例程，还有之前那几个课本习题！

二进制乘法加速乘法运算：下面的案例演示了10×5的情形。

```
            .DATA
            .TEXT
            .GLOBAL  MAIN
MAIN :      ADDI    R8, R8, #5
            ADDI    R9, R9, #10  ;这两行初始化两个乘数
            ADDI    R10, R0, #0
            ADDI    R1, R0, #0   ;R1初始化
            ADDI    R2, R0, #0   ;R2初始化
            ADDI    R3, R0, #1   ;R3初始化
LOOP :      SLL     R1, R3, R2
            AND     R1, R8, R1  ;判断R8的末位，然后此后亦同
            BEQZ    R1, NEXT    ;跳转到下一位
            SLL     R1, R9, R2   ;将R9移动对应位数然后逐次相加
            ADD     R10, R10, R1  ;结果存放在R10中，R1是每次的加数
NEXT :      ADDI    R2, R2, #1
            SLTI    R1, R2, #31  ;0-31位运算，从第一位开始算起
            BNEZ    R1, LOOP
            TRAP    X00
```

斐波拉契数列：需要的那项的下标放在R4中，结果输出到R1，f(0)=1,f(1)=1；

下面的做法是一个很典型的迭代做法，迭代的效率明显要比递归要高！

```
            .DATA
            .TEXT
            .GLOBAL  MAIN
MAIN :      ADDI    R1, R0, #1
            ADDI    R2, R0, #1  ;f(n-1)
            ADDI    R3, R0, #1  ;f(n-2)
            ;
            ;ADDI    R4, R4, #10
            ;
            SUBI    R4, R4, X31
LOOP :      BEQZ    R4, EXIT
            ADD     R1, R3, R2
            ADDI    R3, R2, #0
            ADDI    R2, R1, #0
            SUBI    R4, R4, #1
            J       LOOP
EXIT :      TRAP    X00

```

汉诺塔/猴子吃桃（其实他们很容易就可以改造一下实现互相转换）

以递归版本的汉诺塔为例。

当然如果你想用循环体，$H_{(n)}=2H_{(n-1)}+1$, $H_{(1)}=1$，也很简单。

```
        ;递归部分
HANNUO :    SUBI    R29, R29, #4
            SW      0(R29), R31
            SEQI    R5, R1, #4
            BNEZ    R5, LASTDAY
            ADDI    R1, R1, #1
            JAL     HANNUO
            ADD     R2, R2, R2    ;除了最后一个以外的 要现在一个柱子上中转，往返共2次
            ADDI    R2, R2, #1    ;最后一个要先过去
            J       EXIT
LASTONE :   ADDI    R2, R0, #1    ;只有一个个就直接移过去了；
EXIT :      LW      R31, 0(R29)
            ADDI    R29, R29, #4
            JR      R31
```

冒泡排序-升序（极好地综合了以上所有的点）

```
            ;这个题其实是课本练习题14.8
            .DATA   x30000000
DATA :      .WORD   #3, #14, #35, #47, #5, #20, #12, #14, #6, #22
SAVER31 :   .SPACE  #4
        ;
            .TEXT   x40000000
            .GLOBAL  MAIN
MAIN :      ADDI    R1, R0, DATA
            ADDI    R2, R0, #9
OUTLOOP :   BEQZ    R2, EXIT
            ADDI    R3, R2, #0
INNERLOOP : LW      R4, 0(R1)
            LW      R5, 4(R1)
            JAL     CMP
            ADDI    R1, R1, #4
            SUBI    R3, R3, #1
            BNEZ    R3, INNERLOOP
            SLLI    R6, R2, #2
            SUB     R1, R1, R6
            SUBI    R2, R2, #1
            J       OUTLOOP
EXIT :      TRAP    x00
        ;
CMP :       SW      SAVER31(R0), R31
            SLT     R6, R4, R5
            BNEZ    R6, RETURN
            JAL     SWAP
RETURN :    LW      R31, SAVER31(R0)
            RET
        ;
SWAP :      SW      4(R1), R4
            SW      0(R1), R5
            RET

```

> 特别鸣谢：助教的课后答案

5. 我认为有意思的DLX课后习题

这里选取的题目都是我们曾经做过的。

立即数超范围

（强烈抗议这个DLX模拟器不说明真正原因，下附图）

> 存在语法错误:
> ->错误出现在第6行36列：指令不完整！需要标记。
> 共1个错误!

```
            ;这是课本练习题11.3
            .DATA   x30000000
NUM :       .SPACE  4
        ;
            .TEXT   x40000000
            .GLOBAL  MAIN
MAIN :      ADDI    R1, R0, #100000
            SW      NUM(R0), R1
            TRAP    x00

```

2^{15}=32678，道理显而易见了。

栈的一个形象化例子

PUSH A	PUSH B	PUSH C	POP	PUSH D	PUSH E	POP	POP
					A		
		A		A	B	A	
	A	B	A	B	D	B	A
A	B	C	B	D	E	D	B

此处涉及到的操作如下：

```
PUSH A
PUSH B
PUSH C
POP
PUSH D
PUSH E
POP
POP
```

I/O状态寄存器与数据寄存器之间的美妙结合

```
    ;这是课本练习题12.2
```

```
    ;输入例程
    A:      .word   xFFFF0010   ; IOSR的起始地址
    B:      ;.word  xFFFF0004   ; KBDR的起始地址
;……
            lw      r1, A(r0)
    START:  lw      r2, 0(r1)   ; 测试是否有字符被输入
            andi    r3, r2, #2
            beqz    r3, START
            lw      r1, B(r0)
            lw      r4, 0(r1)
            j       NEXT_TASK   ; 执行下一个任务
    ;输出例程—
    A:      .word   xFFFF0010   ; IOSR的起始地址
    B:      .word   xFFFF000C   ; DDR的起始地址
;……
            lw      r1, A(r0)
    START:  lw      r2, 0(r1)   ; 测试输出寄存器是否就绪
            andi    r3, r2, #1
            beqz    r3, START
            lw      r1, B(r0)
            sw      0(r1), r4
            j       NEXT_TASK   ; 执行下一个任务
```

诸如什么`KBDSR`，`DDSR`之类的玩意儿都可以以此类推。

字符串结尾要插x00！

这个就不啰嗦了，前面机器语言那节已经提到了这个问题。毕竟一堆电子运动的方式和结果是我们人去设计并去理解执行（也许是借助机器理解执行的。

I/O综合应用

```
            .data   x30000000
    STACK:  .space  #100
    PROMPT: .asciiz "Please enter your string: "
    ;
            .text   x40000000
            .global main
    main:   addi    r4, r0, PROMPT
            trap    x08
            addi    r3, r0, STACK
            addi    r5, r3, #100
            addi    r29, r5, #0
    Input:  seq     r2, r29, r3
            bnez    r2, Output
            trap    x06
            trap    x07
            seqi    r2, r4, #10
            bnez    r2, Output
            jal PUSH
            j   Input
    ;
    Output: seq     r2, r29, r5
            bnez    r2, DONE
            jal POP
            trap    x07
            j   Output
;
```

```
    DONE:   trap    x00
    ;
    PUSH:   subi    r29, r29, #4
             sw      0(r29), r4
            ret
    ;
    POP:    lw  r4, 0(r29)
            addi    r29, r29, #4
            ret
```

这里涉及了栈的性质（后进先出，在反序输出的过程中得到体现），输入字符串的长度检查（不能超过25个），多次看看，很有意义！

> 特别鸣谢：助教的课后答案
>
> 看看课后作业中的伪代码也挺有意思的，毕竟来说伪代码也是我们以后特别是搞算法的所需要的。

6. 还在想……

有什么建议欢迎大家及时补充！

3. 其他附加建议

> 真诚建议大家用好二进制转字符串输出那部分代码，TRAP输入可能会出问题，TRAP x00停机会改变数据的情况下，用好这套东西直接把你的结果输出到屏幕上……这里我做了一个例子。这显然是为了你自己方便调试。毕竟TRAP x00会直接改写数据（课本上给出了他的例程），而单步调试比较烦。

> 特别鸣谢：许礼孟同学

```
        ;
        ;以下为二进制数转成字符串输出到屏幕上，原二进制数存放于R2中
        ;
MOVE :      SW       SAVER1(R0), R1
            SW      SAVER2(R0), R2
            SW      SAVER3(R0), R3
            SW      SAVER4(R0), R4
            ADDI    R4, R0, ASCIIBUF
        ;
BEGIN100 :  ADDI    R1, R0, X30
LOOP100 :   SLTI    R3, R2, #100
            BNEZ    R3, END100
            ADDI    R1, R1, #1
            SUBI    R2, R2, #100
            J       LOOP100
        ;
END100 :          SB      0(R4), R1
        ;
        ;判断十位数
BEGIN10 :   ADDI    R1, R0, X30
LOOP10 :    SLTI    R3, R2, #10
            BNEZ    R3, END10
            ADDI    R1, R1, #1
            SUBI    R2, R2, #10
            J       LOOP10
```

```
        ;
END10 :     SB      1(R4), R1
        ;
        ;判断个位数并输出
BEGIN1 :    ADDI    R1, R2, X30
            SB      2(R4), R1
            ADDI    R1, R0, ASCIIBUF
            LB      R4, 0(R1)
            TRAP    X07
            LB      R4, 1(R1)
            TRAP    X07
            LB      R4, 2(R1)
            TRAP    X07
            LB      R4, 3(R1)
            TRAP    X07
            ;为什么不直接输出字符串？那样注意结尾要自己加个/n!
            J       RECOVER
        ;
        ;以下为恢复寄存器并退出程序
        ;
RECOVER :   LW      R1, SAVER1(R0)
            LW      R2, SAVER2(R0)
            LW      R3, SAVER3(R0)
            LW      R4, SAVER4(R0)
            TRAP    X00
```

这个实例同样可以用来理解`调用者/被调用者保存机制`。这个实例没有采用课本上查表的方法，效率上可能不如课本。课本上的想法相当新奇，也可以作为参考。

> 未完待续.....

计算系统基础
考前复习提纲

1 月 1 日

南京大学软件学院

感谢所有对本资料整理做出贡献的人