电子器件与电路（二）
嵌入式部分

工程科学学院2017级

电子信息与通信学院
钟国辉
zhonggh@hust.edu.cn

内容安排

- 课程概述
- 课程要求
- 课程内容安排
- 计算机系统概述
- 嵌入式系统概述
- 单片机技术基础

内容概述

- 目的
  - 掌握计算机系统的理论基础
  - 以微型单片计算机为模型掌握计算机系统的开发应用
  - 掌握快速学习一种全新计算机/嵌入式系统的方法
- 先修课程
  - 计算机基础、C语言

课程概述

- 知识
  - 计算机组成原理基础
  - 嵌入式系统基本概念
  - Microchip PIC系列单片机系统应用知识
- 工具
  - PIC汇编语言程序开发
  - 嵌入式C语言程序开发
  - 小型嵌入式系统设计（软、硬件）过程及工具
- 方法
  - 掌握快速学习一种全新计算机/嵌入式系统的方法
  - 嵌入式系统软硬件设计、分析、调试方法

课程要求

- 教学过程要求
- 课后收尾工作
- 考核方式

教学过程要求

- 工作日志（每人）
  - 每天**22:00**前提交到QQ群当中
  - 记录每天遇到的问题、观察到的现象、分析过程和解决途径

教学过程要求

- 工作日志（每人）
  - 工作日志命名要求：
    - **编号-姓名-工作日志-日期.doc**
    - **如01-李想-工作日志-0907.doc**

1	李想	11	何忠毅	21	张力
2	王鸿波	12	赵擎宇	22	周政宏
3	袁雪楠	13	陈伟煌	23	田思成
4	马明慧	14	陈子康	24	吴沁忆
5	周瀛	15	鲁高科	25	杨雨薇
6	孟兆阳	16	劳国彦	26	李怡然
7	汪能志	17	卢博文	27	王柄稀
8	李昊	18	王星	28	刘继林
9	王晓瑞	19	包宇坤		
10	叶开	20	邹晓阳		

教学过程要求

- 课程和实验环节（分组）
  - **请随时提问**
  - 组内和组间鼓励相互讨论
  - 每次实验后安排汇报、讨论
    - **要求每组每次派出不同的同学进行汇报**
  - 请各组详细记录本组实验过程和问题解决过程
  - 按时间要求提交实验报告
  - 实验报告命名规则
    - **组别编号-姓名-实验报告-实验编号.doc**

教学过程要求

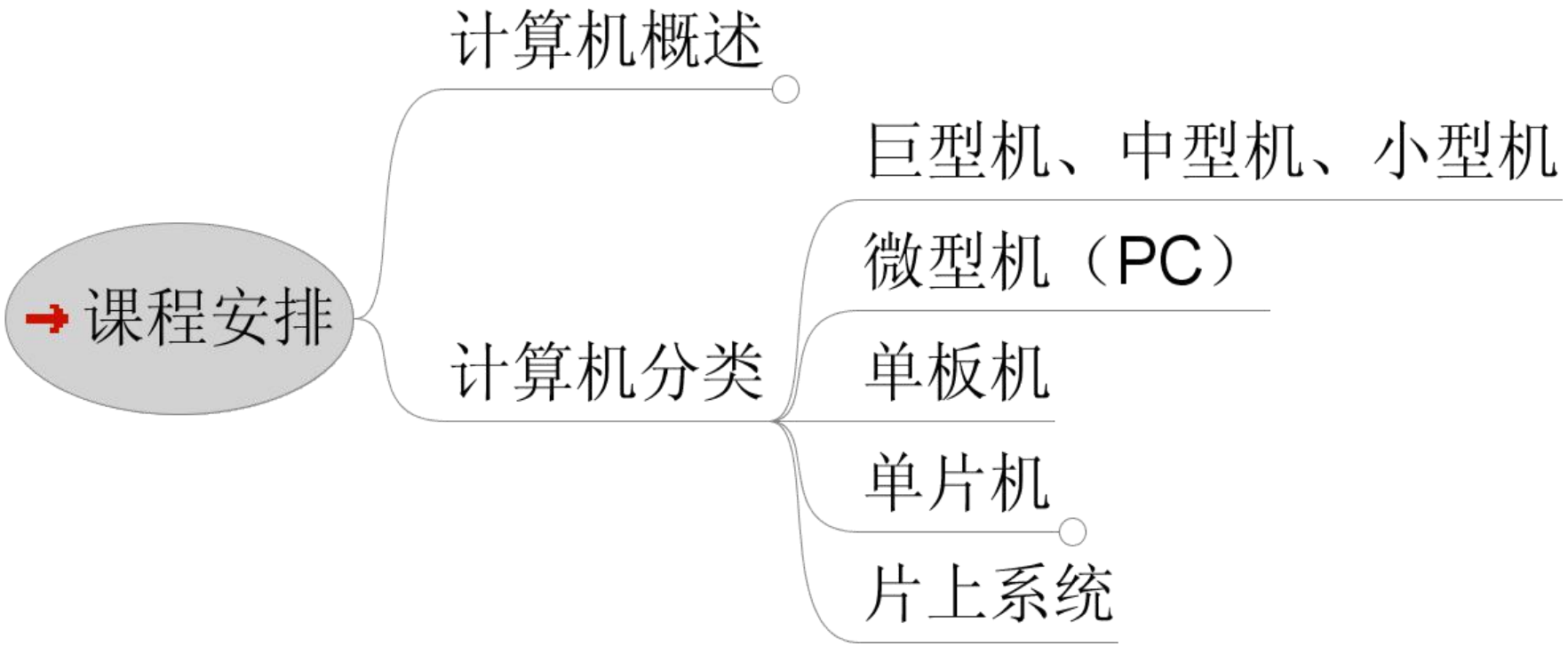
- 分组安排

1	李想	8	李昊
	刘继林		张力
2	王鸿波	9	王晓瑞
	王柄稀		邹晓阳
3	袁雪楠	10	叶开
	赵擎宇		包宇坤
4	劳国彦	11	何忠毅
	杨雨薇		王星
5	周瀛	12	李怡然
	吴沁忆		卢博文
6	孟兆阳	13	陈伟煌
	田思成		马明慧
7	汪能志	14	陈子康
	周政宏		鲁高科

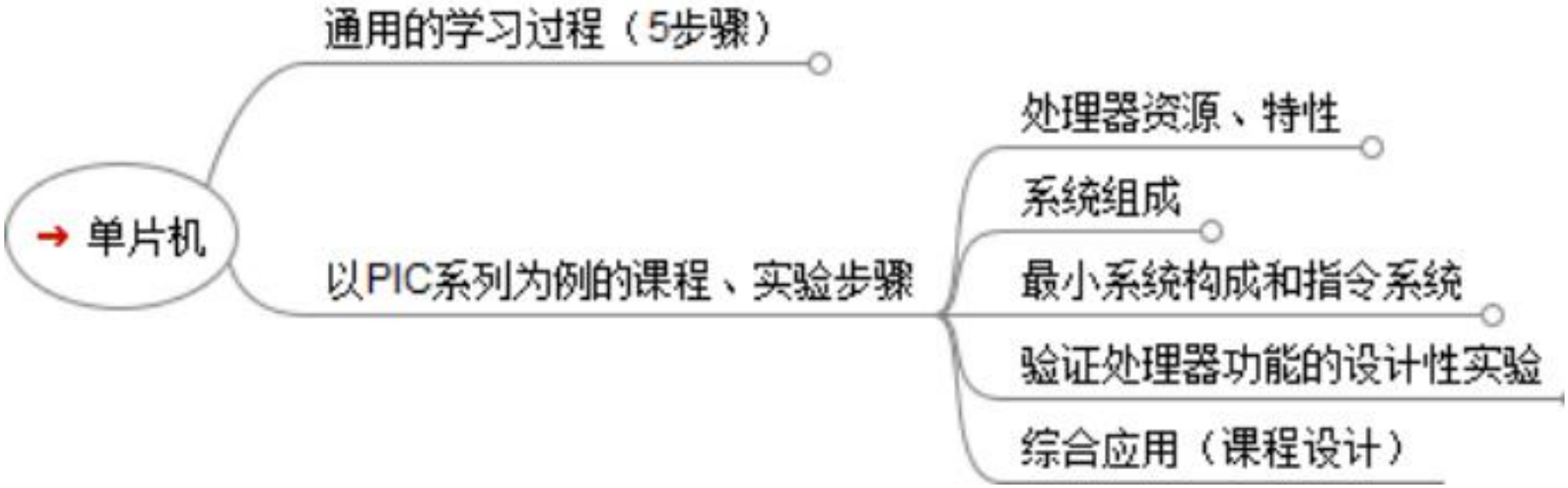
课程要求

- 课后收尾工作
  - 归还下载器和元器件
- 考核方式
  - 教学过程
    - 日常表现、工作日志、实验报告（共50%）
    - 汇报（50%）

课程内容安排



课程内容安排

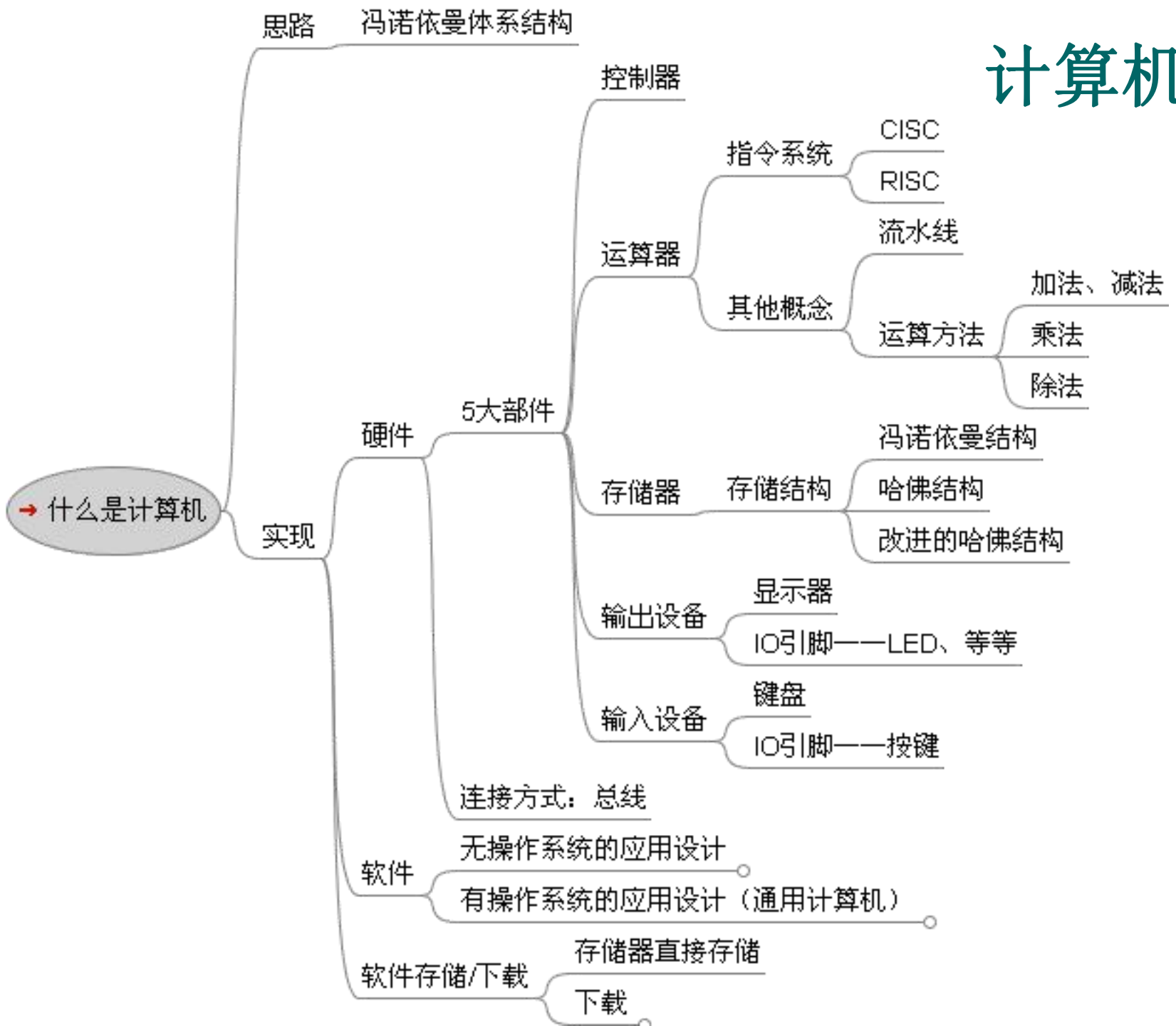

单片机
通用的学习过程（5步骤）
以PIC系列为例的课程、实验步骤
处理器资源、特性
系统组成
最小系统构成和指令系统
验证处理器功能的设计性实验
综合应用（课程设计）

实验安排

- 共安排五个小实验
  - 实验一：闪灯实验（Hello world）
  - 实验二：采用查询定时器的闪灯实验
  - 实验三：采用定时器中断的闪灯实验
  - 实验四：走马灯动态显示实验
  - 实验五：按键实验

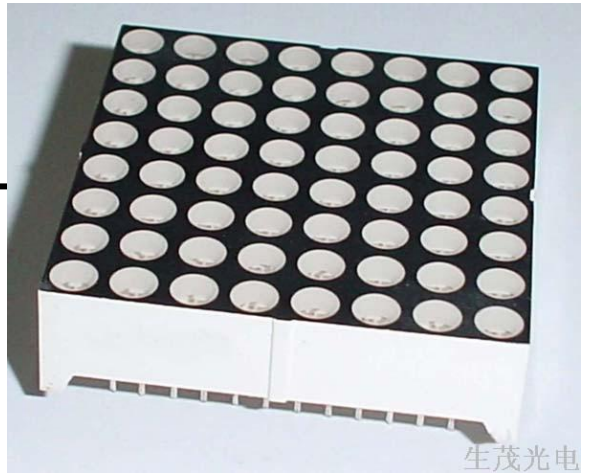
计算机系统概述

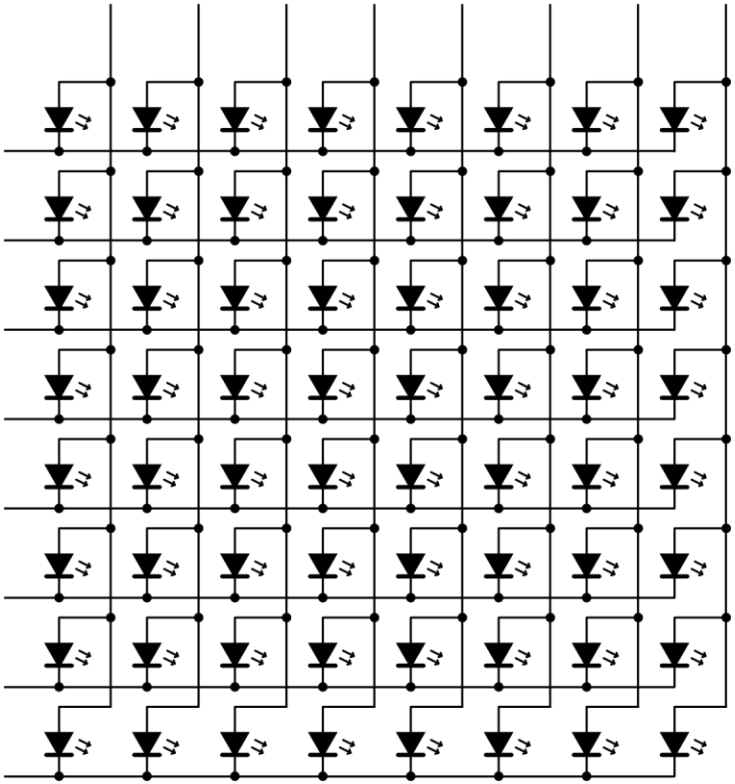
计算机概述

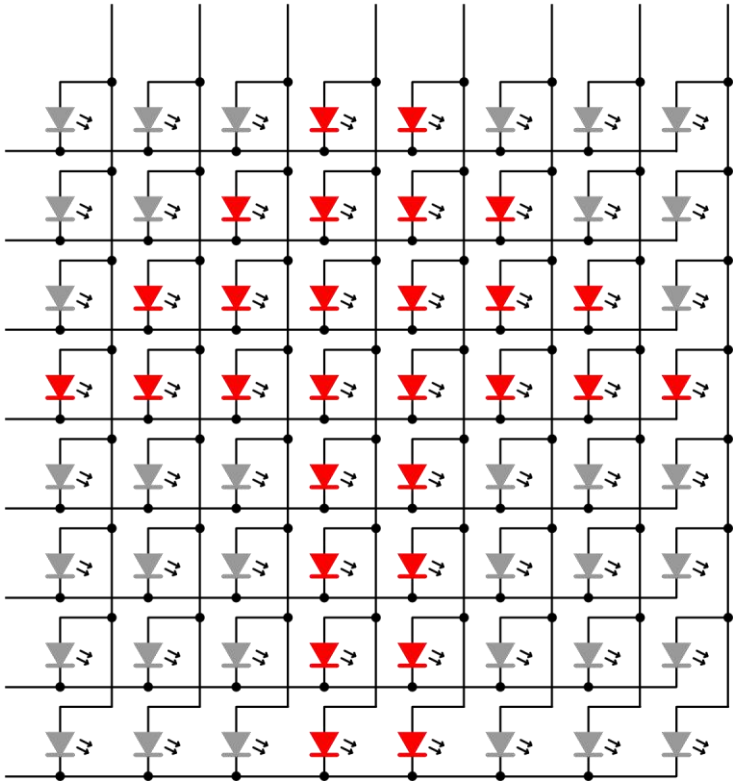

什么是计算机
思路
冯诺依曼体系结构
实现
硬件
5大部件
控制器
运算器
指令系统
CISC
RISC
其他概念
流水线
运算方法
加法、减法
乘法
除法
存储器
存储结构
冯诺依曼结构
哈佛结构
改进的哈佛结构
输出设备
显示器
IO引脚——LED、等等
输入设备
键盘
IO引脚——按键
连接方式：总线
软件
无操作系统的应用设计
有操作系统的应用设计（通用计算机）
软件存储/下载
存储器直接存储
下载

引言

- 数字电路课程——存储器的应用
  - 用8x8LED阵列实现字符显示

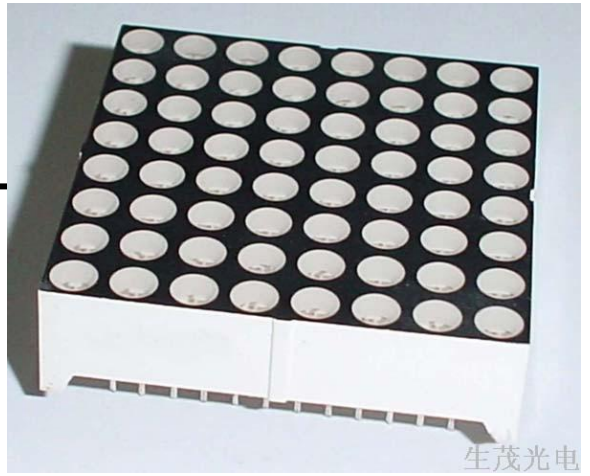


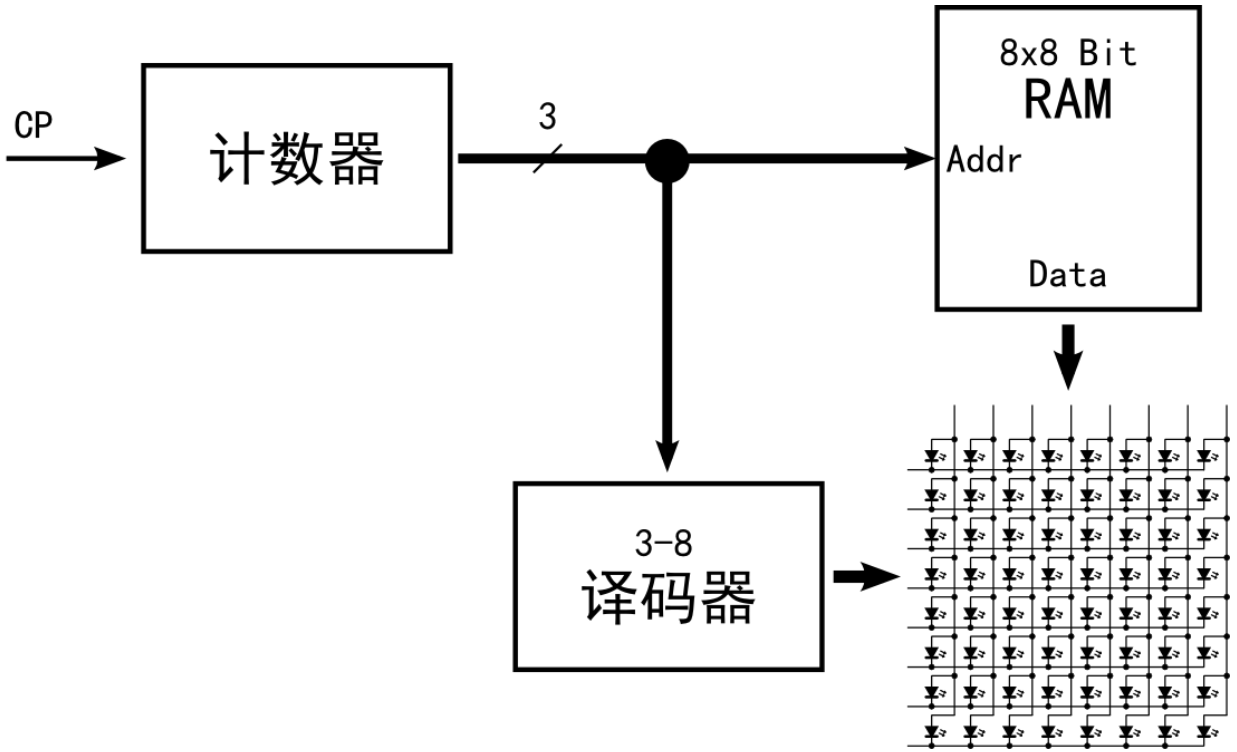




引言

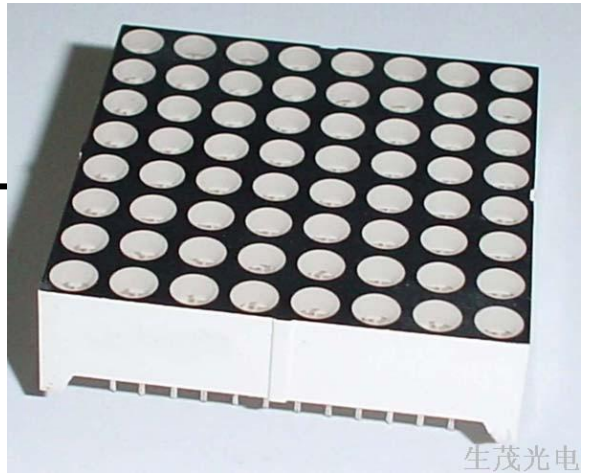
- 数字电路课程——存储器的应用
  - 用8x8LED阵列实现字符显示

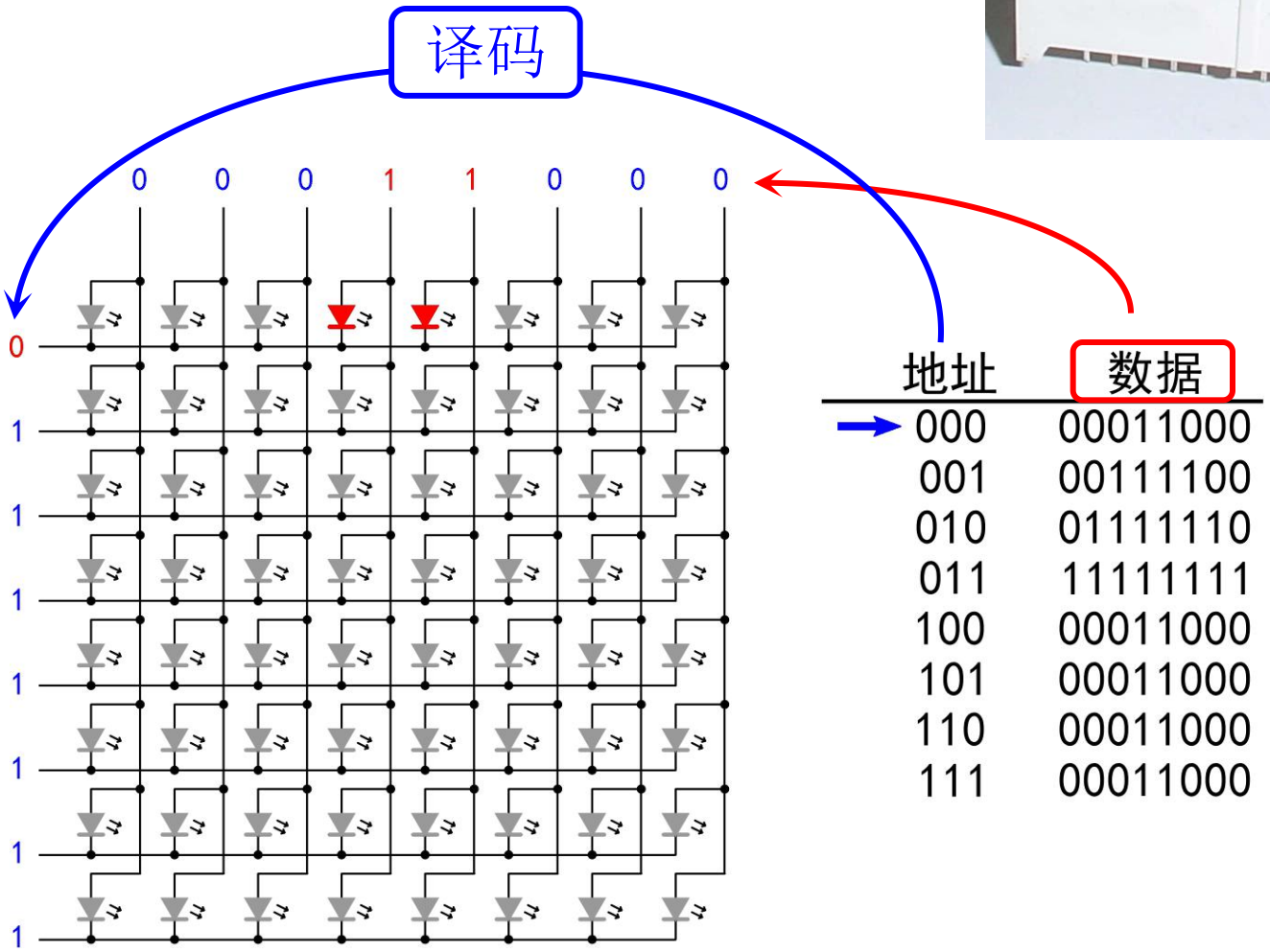




引言

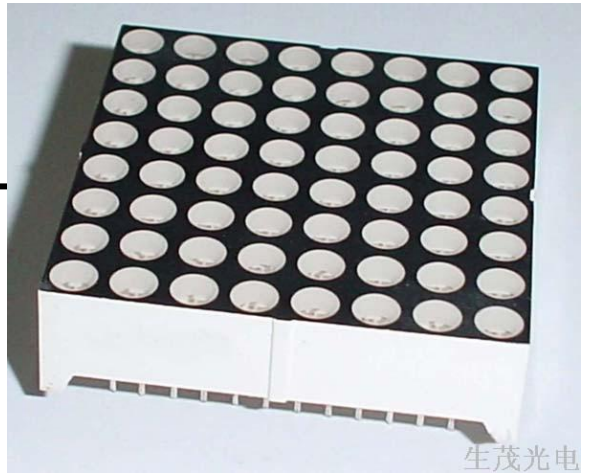
- 操作过程

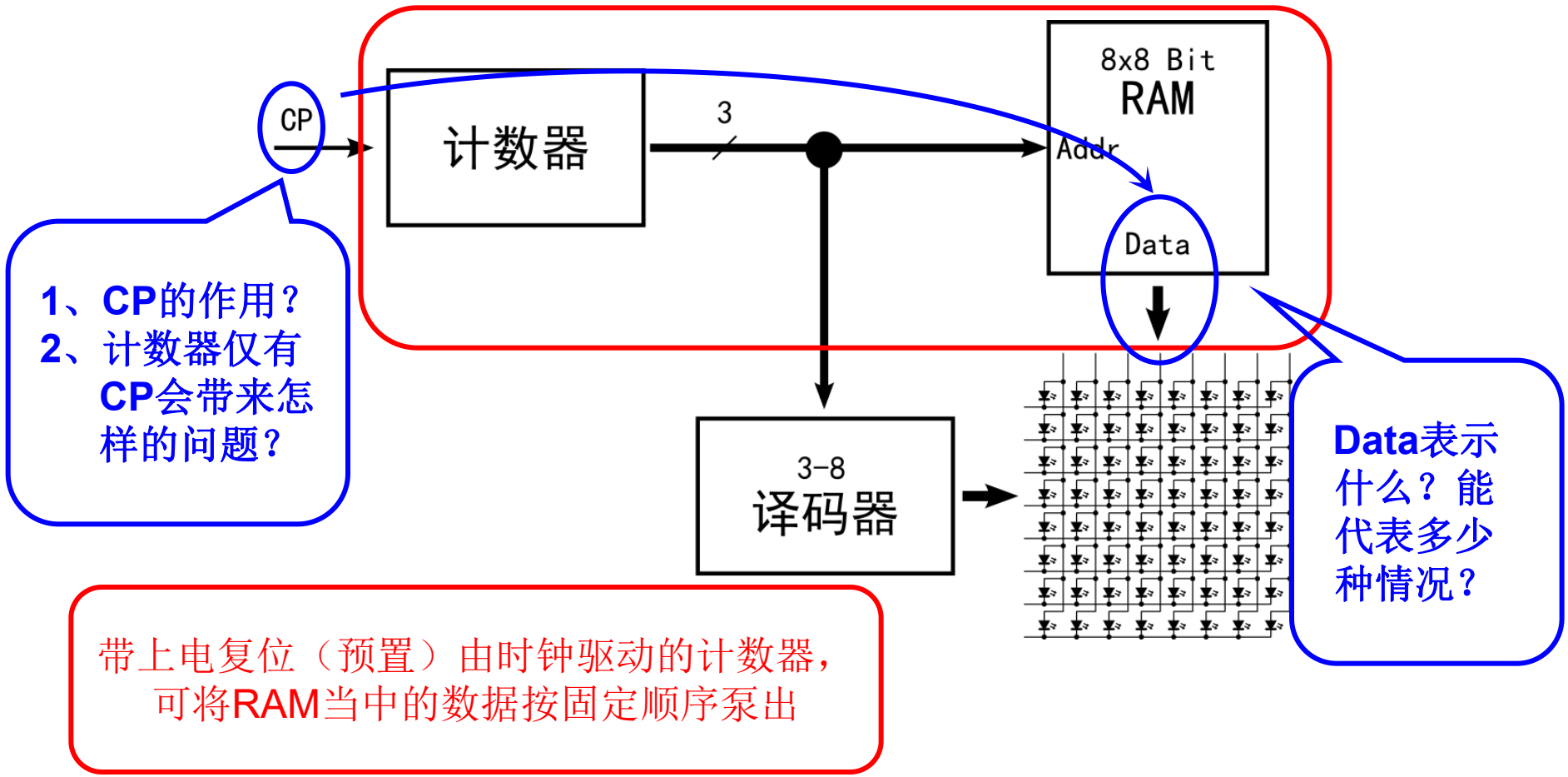

生茂光电


译码
0 0 0 1 1 0 0 0
0
1
1
1
1
1
1
1
地址 数据
000 00011000
001 00111100
010 01111110
011 11111111
100 00011000
101 00011000
110 00011000
111 00011000

引言

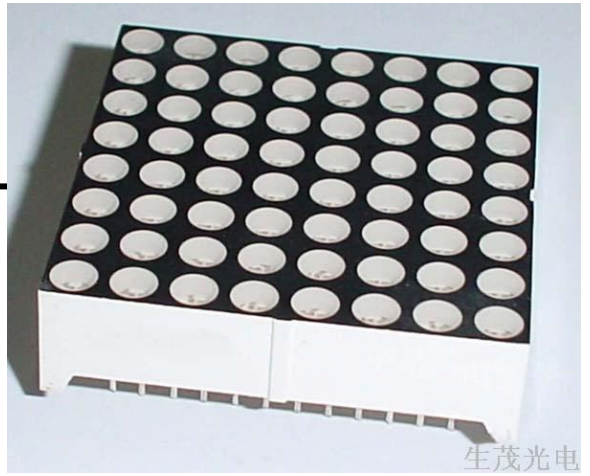
- 数字电路课程——存储器的应用
  - 用8x8LED阵列实现字符显示

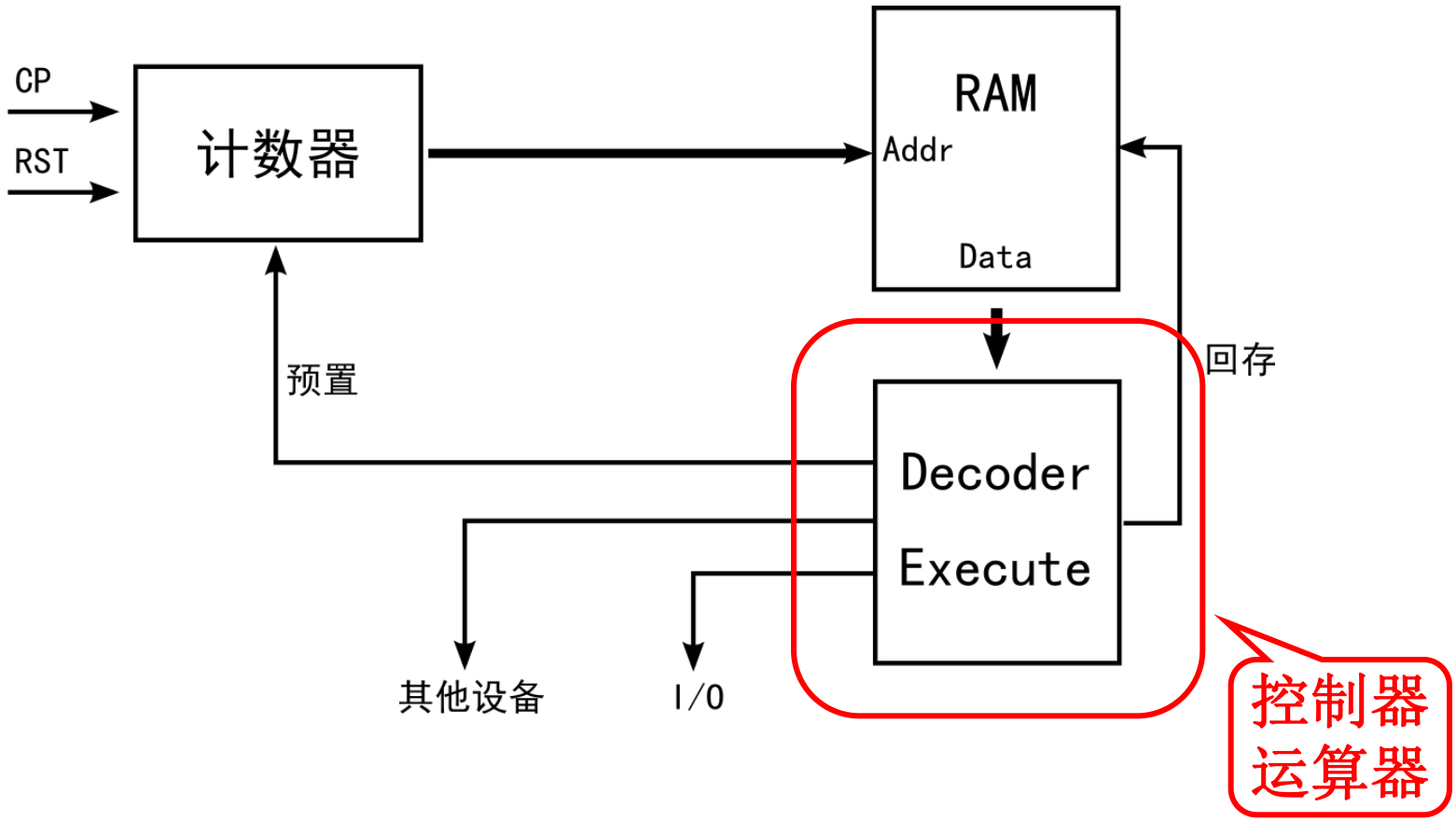




引言

- 数字电路课程——存储器的应用
  - 用8x8LED阵列实现字符显示



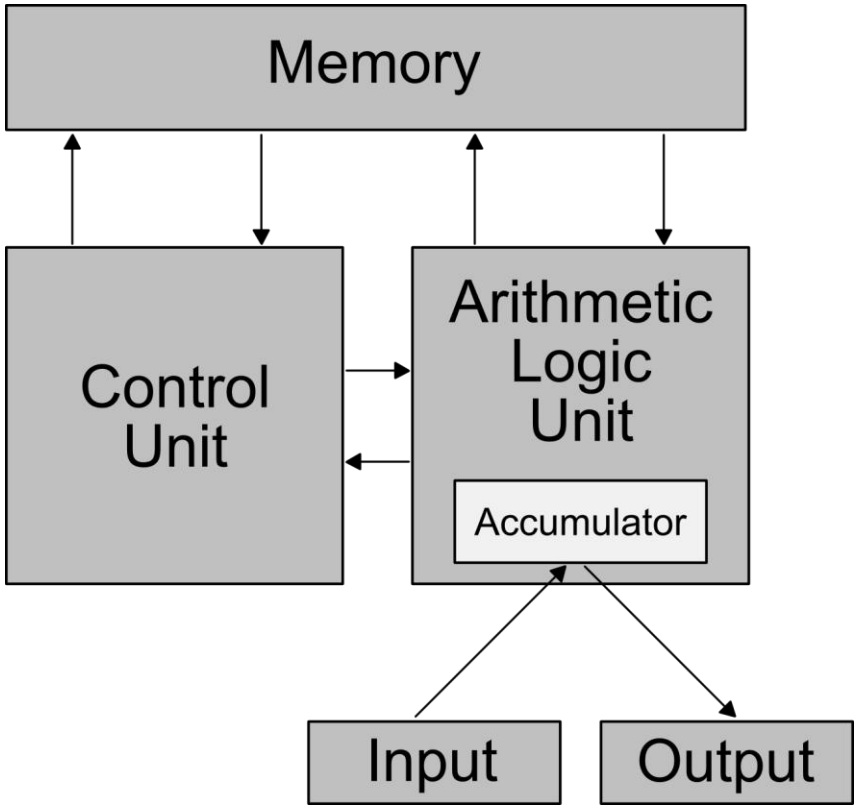


计算机概述

什么是（电子）计算机？

- 计算机是一个系统
- 现在计算机最为普遍的应用形式却是嵌入式（来源：维基百科）

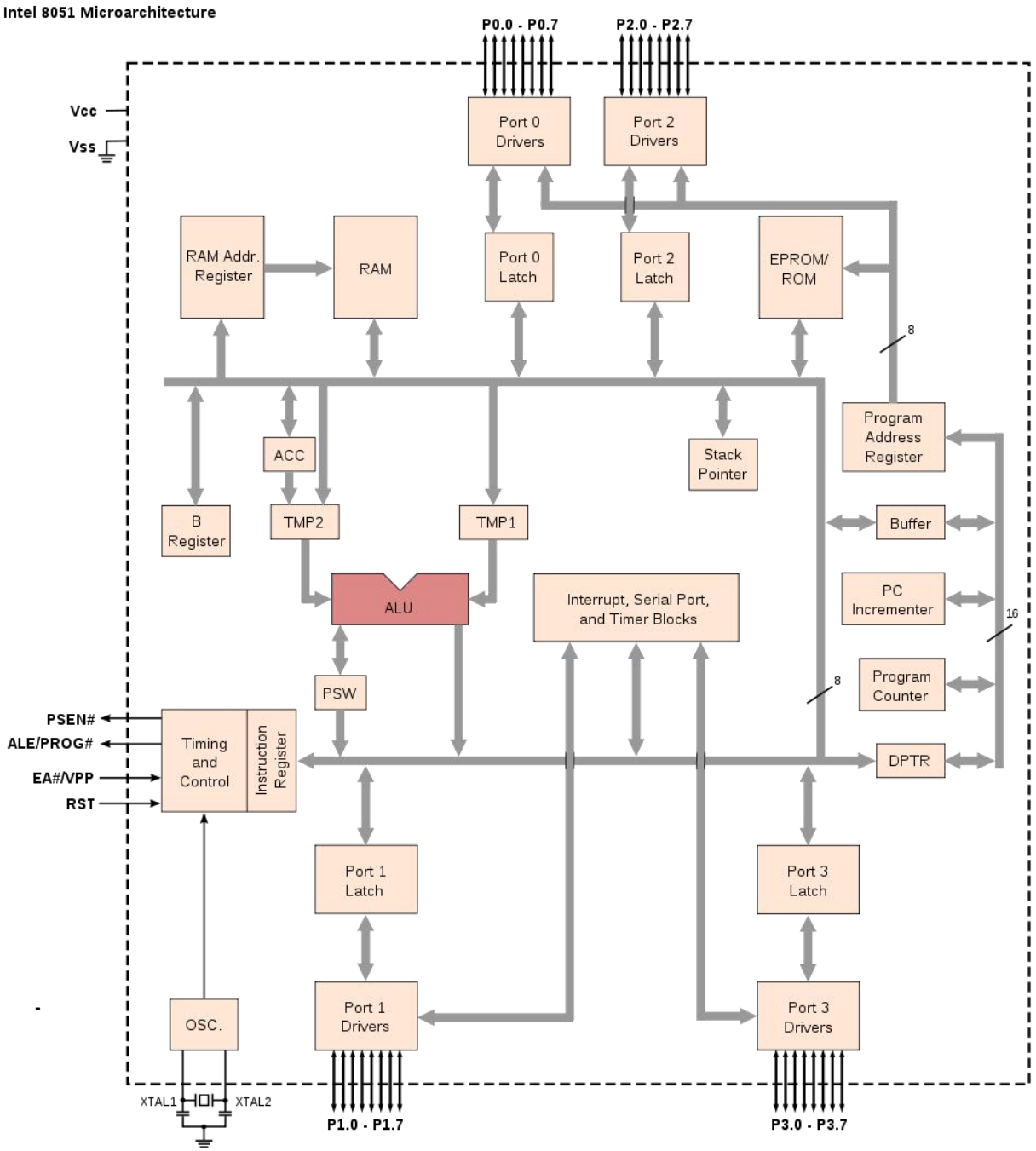
- 计算机发展史（参见维基百科）
- 计算机体系结构知识当中最重要的内容
  - Von Neumann体系结构



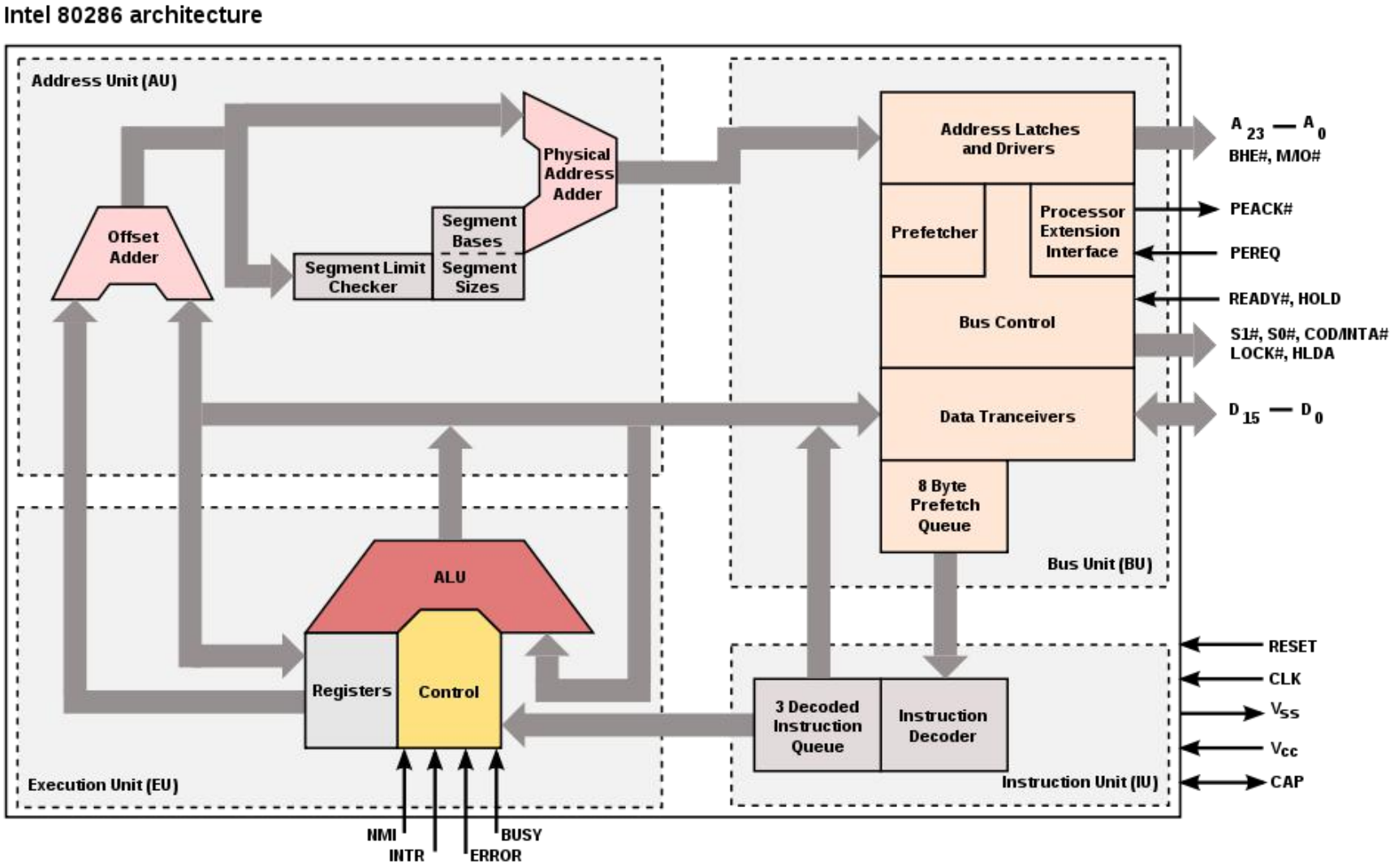
计算机体系结构

- 冯氏体系结构特点：程序存储、顺序执行
  - 存储器是**字长固定的、顺序线性编址**的唯一结构；
  - 存储器提供可按地址访问的一级地址空间，每个地址是唯一定义的；
  - 由指令形式的低级机器语言驱动；
  - **指令的执行是顺序的**，即一般按照指令在存储器中存放的顺序执行，**程序分支由转移指令实现**；
  - 机器以运算器为中心，输入输出设备与存储器之间的数据传送都途径运算器，运算器、存储器、输入输出设备的操作以及它们之间的联系都由控制器集中控制

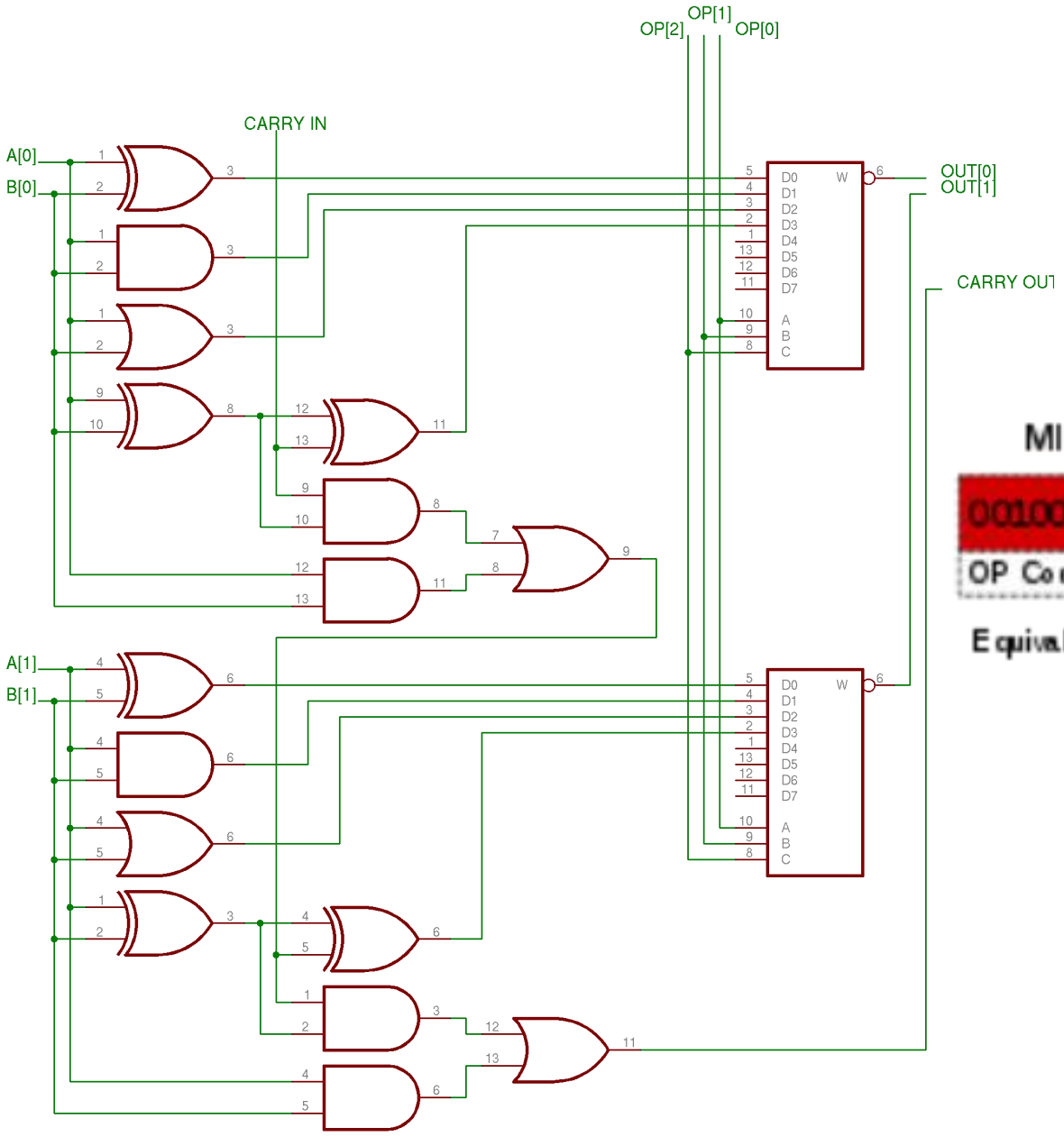
CPU结构示例

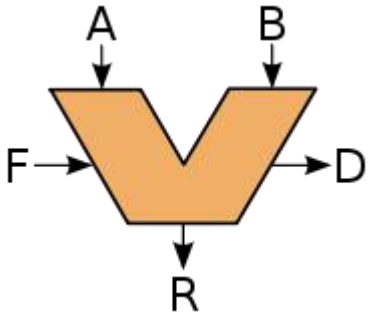

Intel 8051 Microarchitecture
P0.0 - P0.7
P2.0 - P2.7
Vcc
Vss
Port 0 Drivers
Port 2 Drivers
RAM Addr. Register
RAM
Port 0 Latch
Port 2 Latch
EPROM/ ROM
8
Program Address Register
ACC
Stack Pointer
B Register
TMP2
TMP1
Buffer
ALU
Interrupt, Serial Port, and Timer Blocks
PC Incrementer
16
PSW
8
Program Counter
PSEN#
ALE/PROG#
EA#/VPP
RST
Timing and Control
Instruction Register
DPTR
Port 1 Latch
Port 3 Latch
OSC.
Port 1 Drivers
Port 3 Drivers
XTAL1
XTAL2
P1.0 - P1.7
P3.0 - P3.7

计算机CPU的结构


Intel 80286 architecture
Address Unit (AU)
Physical Address Adder
Offset Adder
Segment Bases
Segment Limit Checker
Segment Sizes
Address Latches and Drivers
A 23 — A 0
BHE#, M/IO#
Prefetcher
Processor Extension Interface
PEACK#
PEREQ
READY#, HOLD
Bus Control
S1#, S0#, COD/INTA#
LOCK#, HLDA
Data Tranceivers
D 15 — D 0
8 Byte Prefetch Queue
Bus Unit (BU)
ALU
Registers
Control
3 Decoded Instruction Queue
Instruction Decoder
Execution Unit (EU)
Instruction Unit (IU)
RESET
CLK
Vss
Vcc
CAP
NMI
INTR
ERROR
BUSY

从ALU到指令系统





MIPS32 Add Immediate Instruction

001000	00001	00010	0000000101011110
OP Code	Addr 1	Addr 2	Immediate value

Equivalent mnemonic: addi $r1, $r2, 350

指令系统的分类

- CISC (Complex Instruction Set Computer)
  - 指令种类丰富
  - 强调ALU硬件解码（OPCode）能力
  - 代码密度高（程序占用空间小）
  - 指令长度不固定
  - 指令执行周期不固定
  - 例：x86指令集

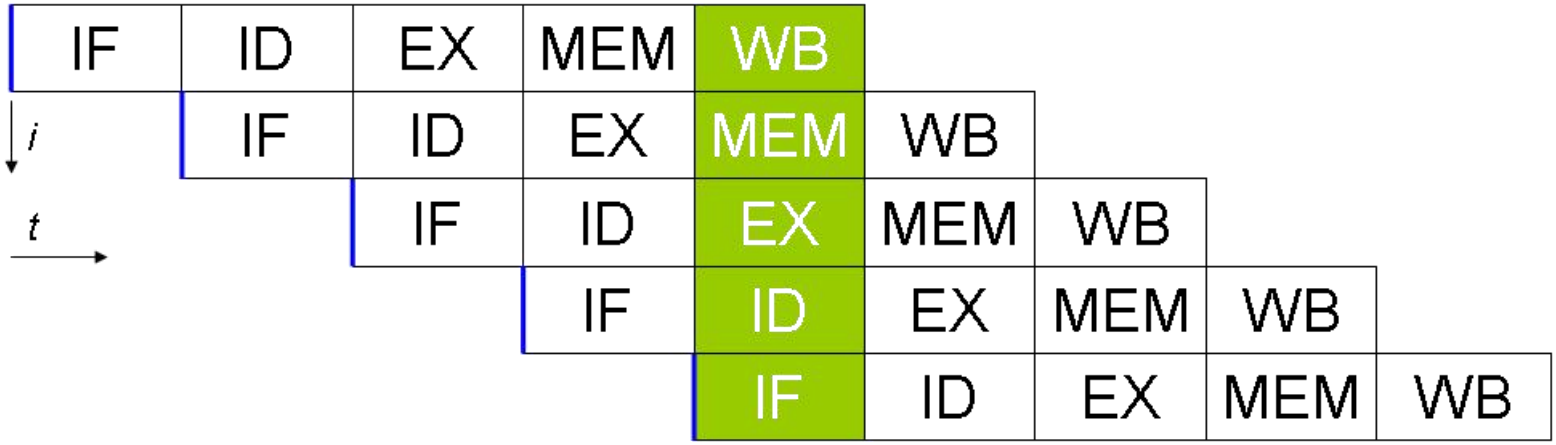
指令系统的分类

- RISC (Reduced Instruction Set Computer)
  - 大多数指令在单周期内完成
  - 采用LOAD/STORE结构，凡是CPU执行部件中所需要的操作数都来自于通用寄存器中，运算结果也只放在通用寄存器
  - 固定的指令格式，使得译码过程简单
  - 注重软件编译器的优化
  - 减少指令和寻址方式的种类
  - 代码密度相对低（程序占用空间相对大）

其他的一些概念

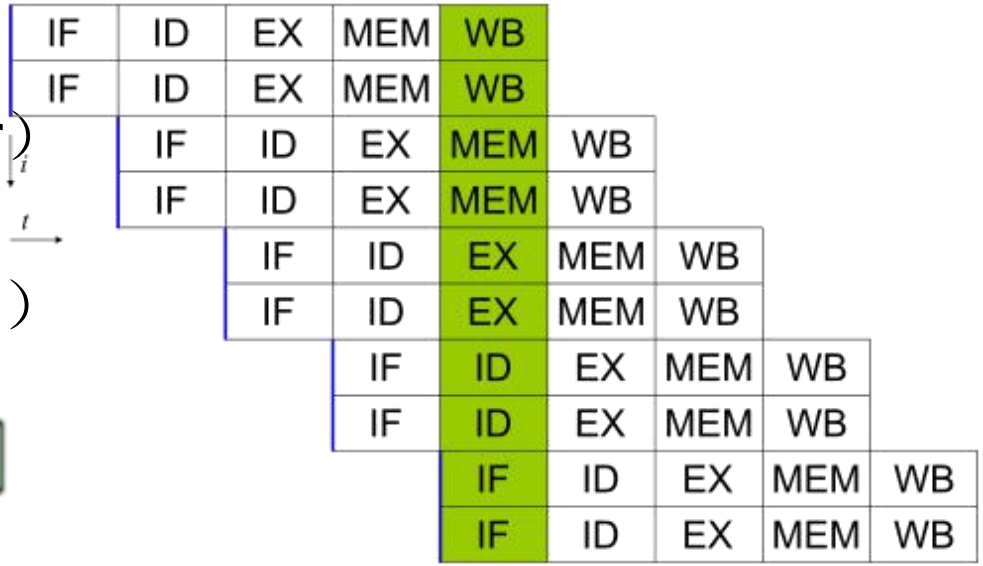
- 流水线（**Pipeline**）
  - 常规的CPU执行过程

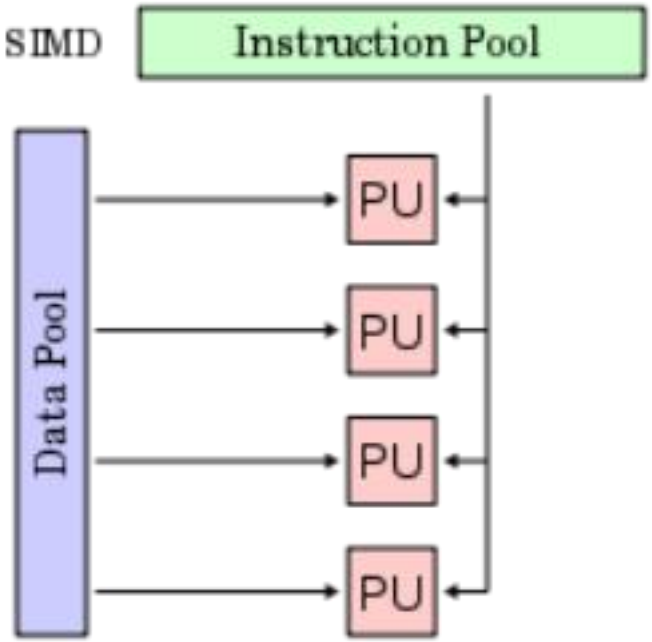
  - 流水线工作方式



其他的一些概念

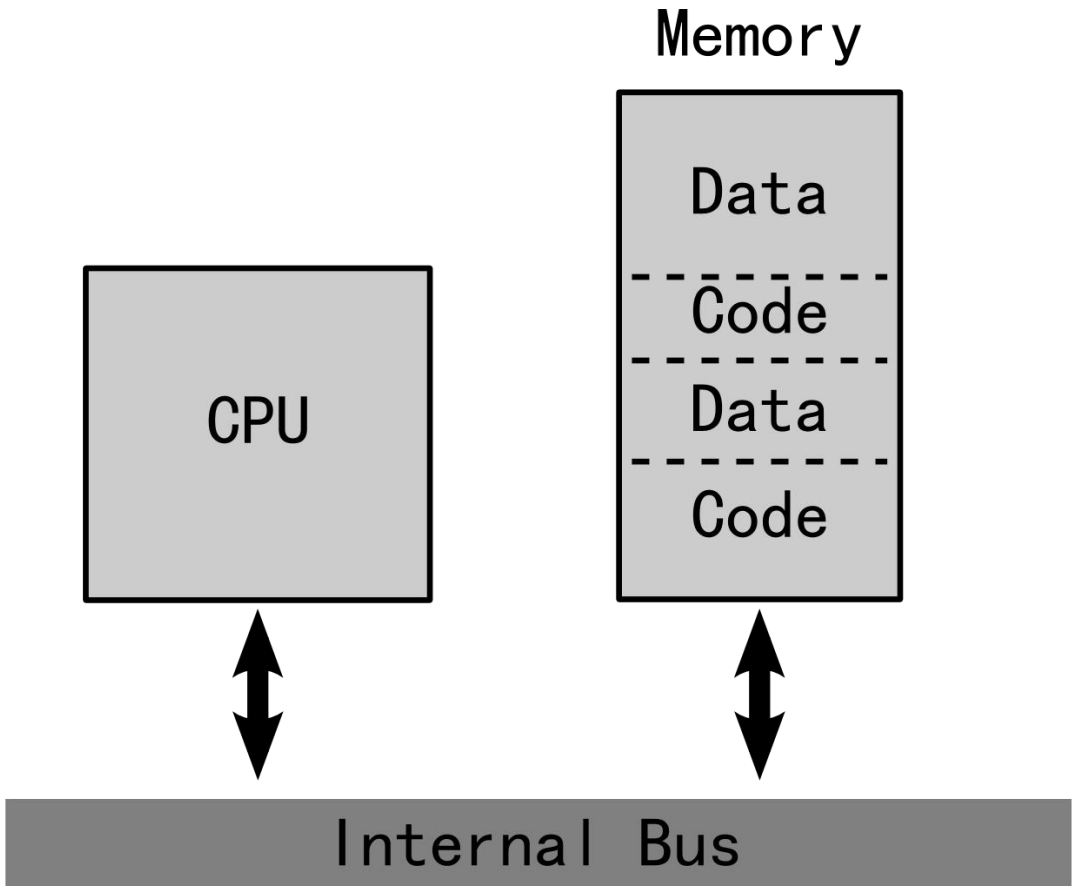
- 超标量和SIMD
  - 超标量（superscalar）
    - 需要分支预测
  - SIMD（单指令多数据）





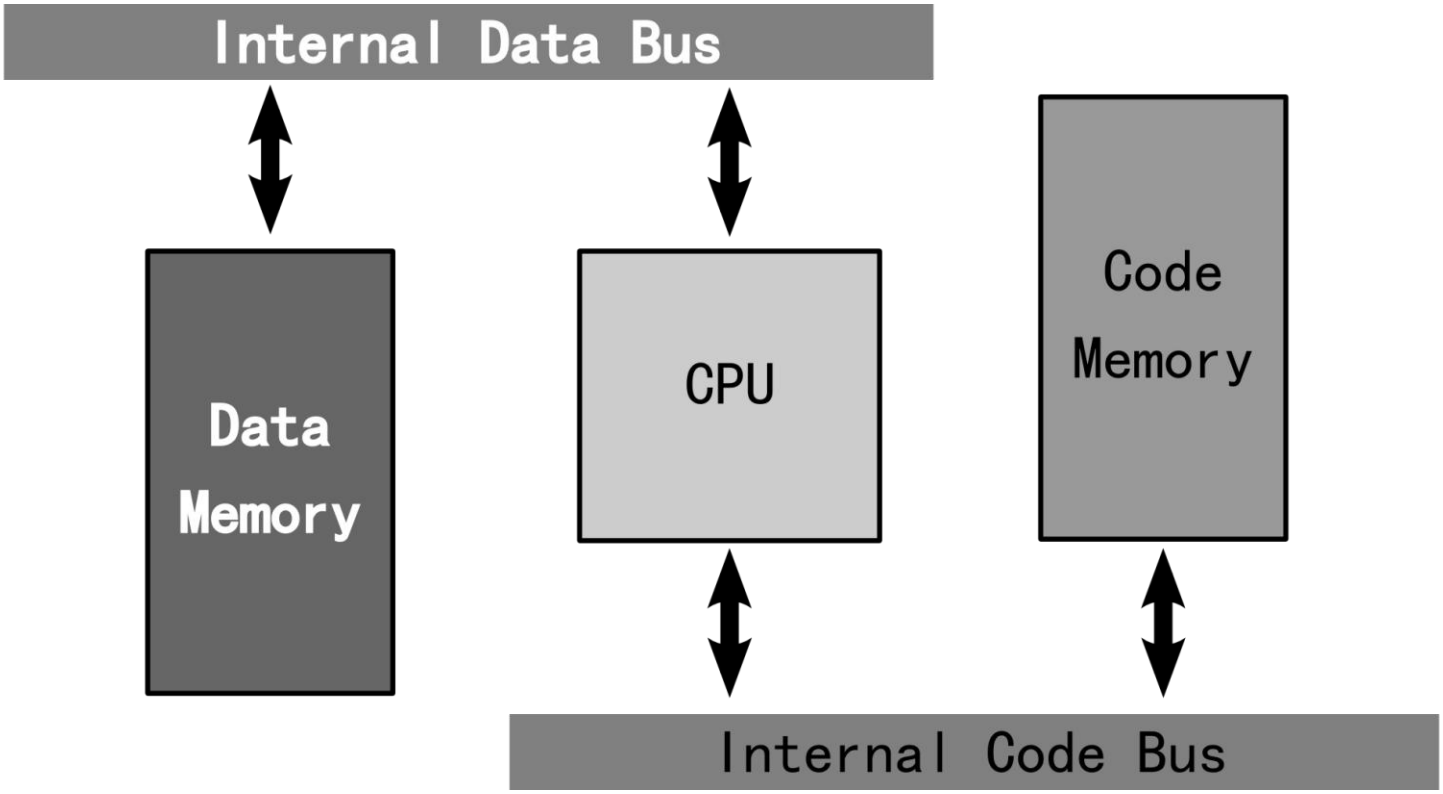
计算机的存储结构

- 冯诺依曼结构



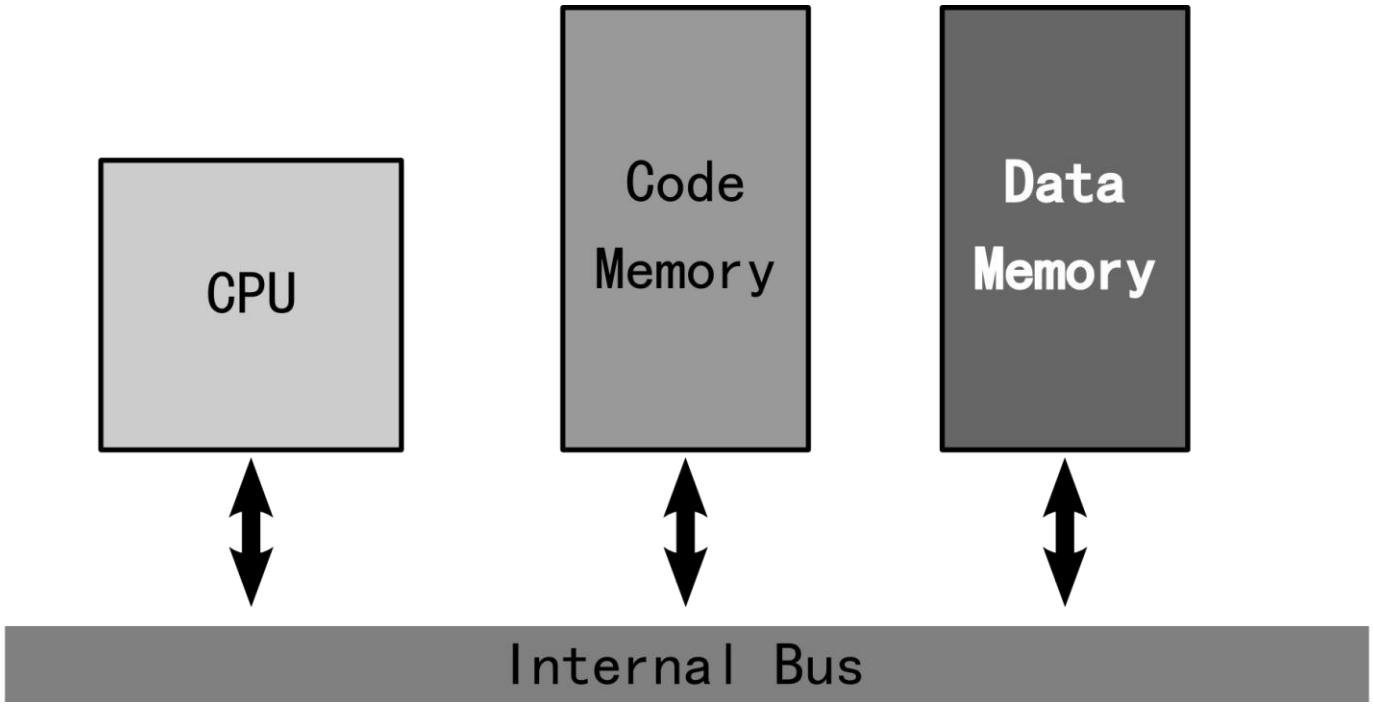
计算机的存储结构

- 冯诺依曼结构
- 哈佛(Harvard Architecture)结构



计算机的存储结构

- 冯诺依曼结构
- 哈佛(Harvard Architecture)结构
- 改进的哈佛结构



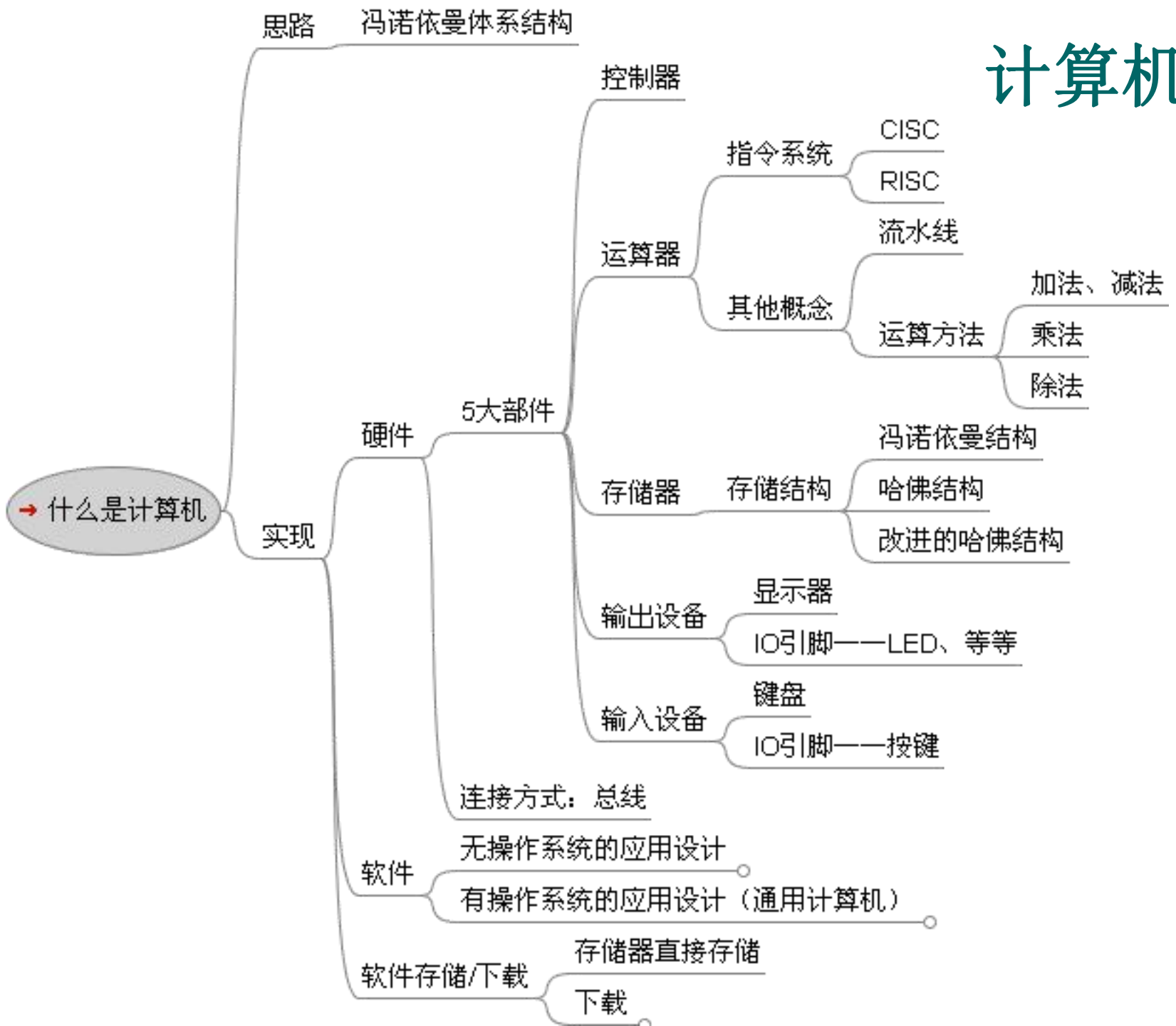
计算机的存储结构

- 冯诺依曼结构
- 哈佛(Harvard Architecture)结构
- 改进的哈佛结构
- 讨论：
  - 它们各自的特点带来了什么优缺点？

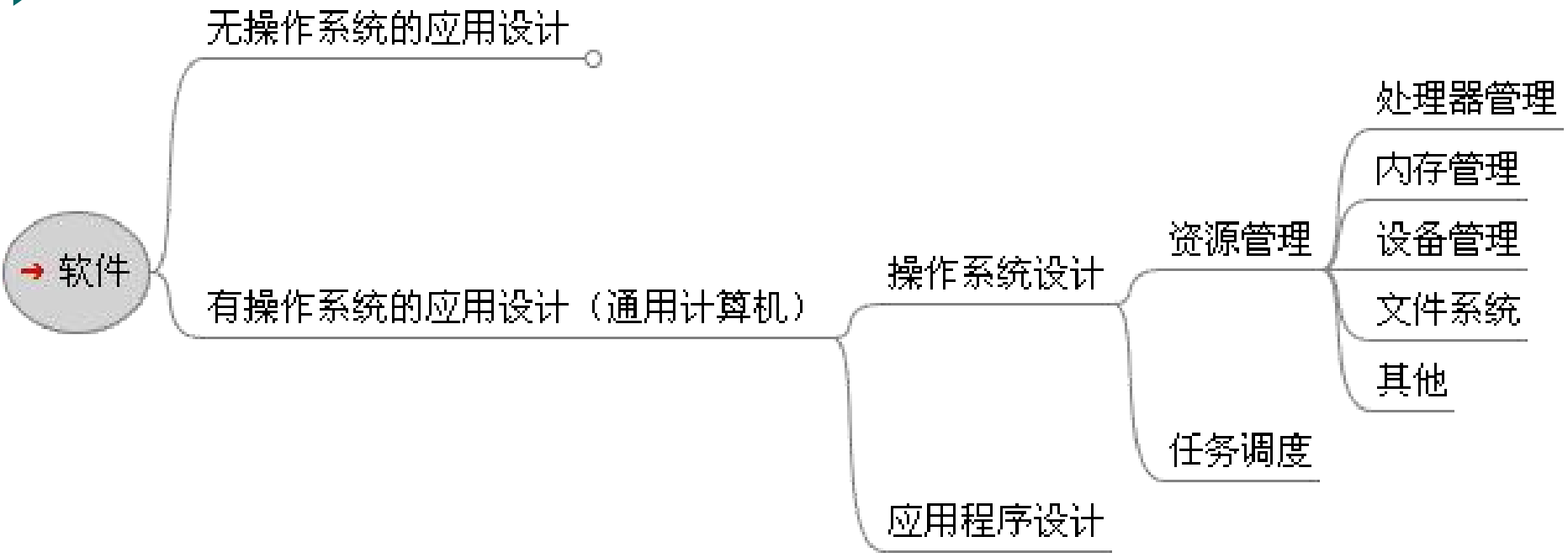
计算机的输入输出设备

- 输入设备
  - 键盘
  - 鼠标
  - IO引脚
    - 按键、触摸屏
- 输出设备
  - 显示器
  - IO引脚
    - LED、数码管、液晶模块等等

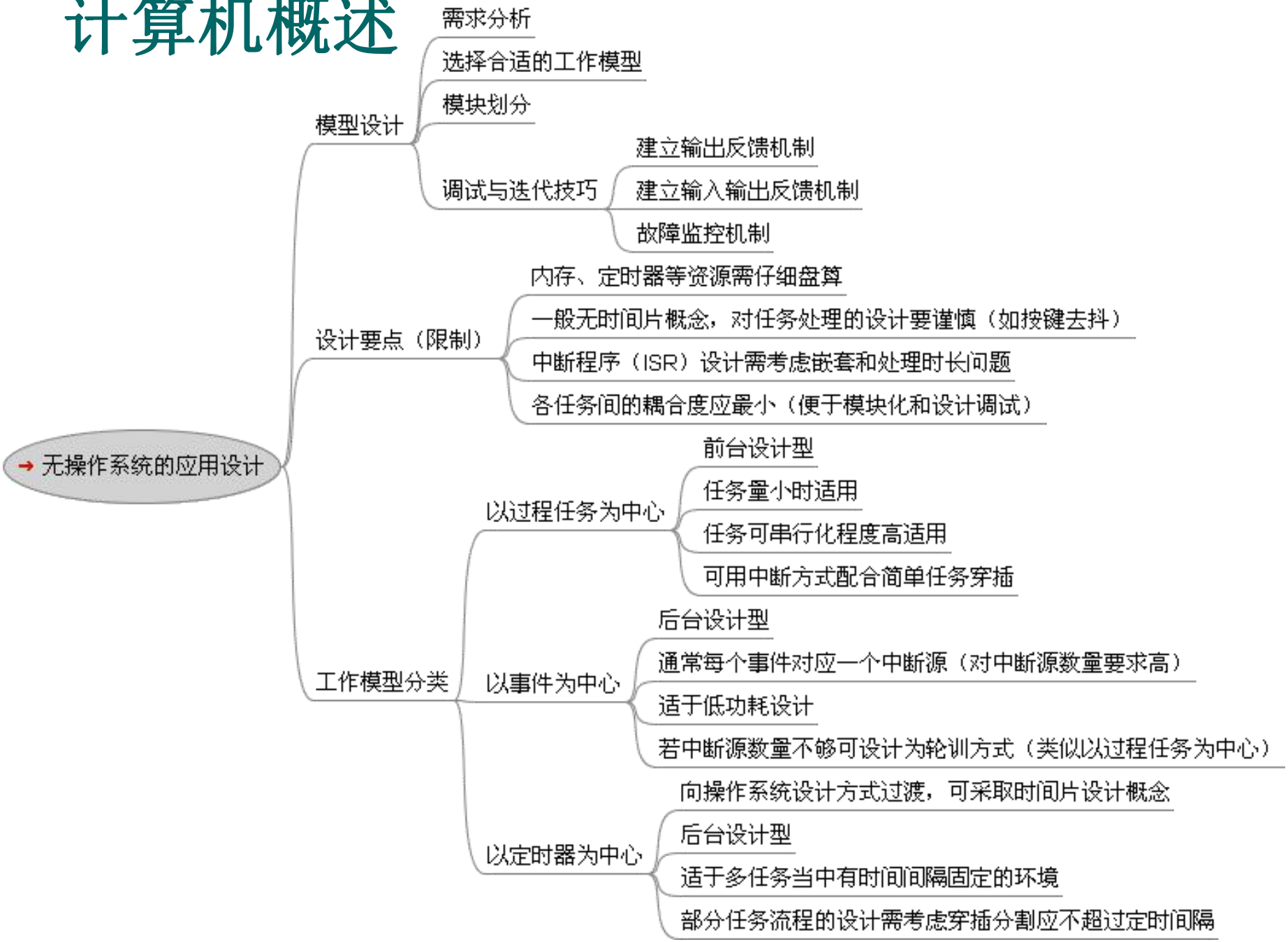
计算机概述


什么是计算机
思路
冯诺依曼体系结构
实现
硬件
5大部件
控制器
运算器
指令系统
CISC
RISC
其他概念
流水线
运算方法
加法、减法
乘法
除法
存储器
存储结构
冯诺依曼结构
哈佛结构
改进的哈佛结构
输出设备
显示器
IO引脚——LED、等等
输入设备
键盘
IO引脚——按键
连接方式：总线
软件
无操作系统的应用设计
有操作系统的应用设计（通用计算机）
软件存储/下载
存储器直接存储
下载

计算机概述


软件
无操作系统的应用设计
有操作系统的应用设计（通用计算机）
操作系统设计
资源管理
处理器管理
内存管理
设备管理
文件系统
其他
任务调度
应用程序设计

计算机概述

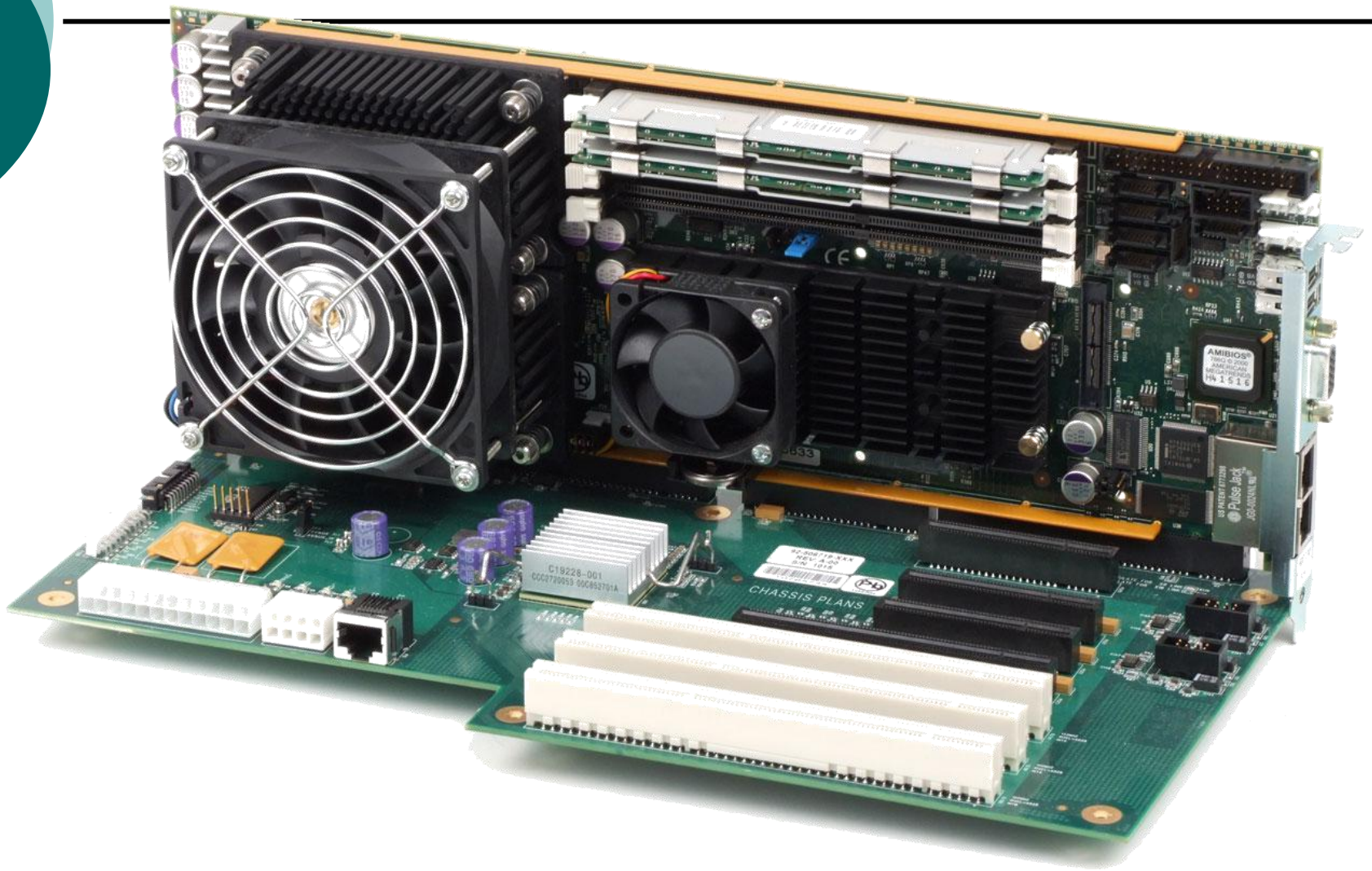

无操作系统的应用设计
模型设计
需求分析
选择合适的工作模型
模块划分
调试与迭代技巧
建立输出反馈机制
建立输入输出反馈机制
故障监控机制
设计要点（限制）
内存、定时器等资源需仔细盘算
一般无时间片概念，对任务处理的设计要谨慎（如按键去抖）
中断程序（ISR）设计需考虑嵌套和处理时长问题
各任务间的耦合度应最小（便于模块化和设计调试）
工作模型分类
以过程任务为中心
前台设计型
任务量小时适用
任务可串行化程度高适用
可用中断方式配合简单任务穿插
以事件为中心
后台设计型
通常每个事件对应一个中断源（对中断源数量要求高）
适于低功耗设计
若中断源数量不够可设计为轮训方式（类似以过程任务为中心）
以定时器为中心
向操作系统设计方式过渡，可采取时间片设计概念
后台设计型
适于多任务当中有时间间隔固定的环境
部分任务流程的设计需考虑穿插分割应不超过定时间隔

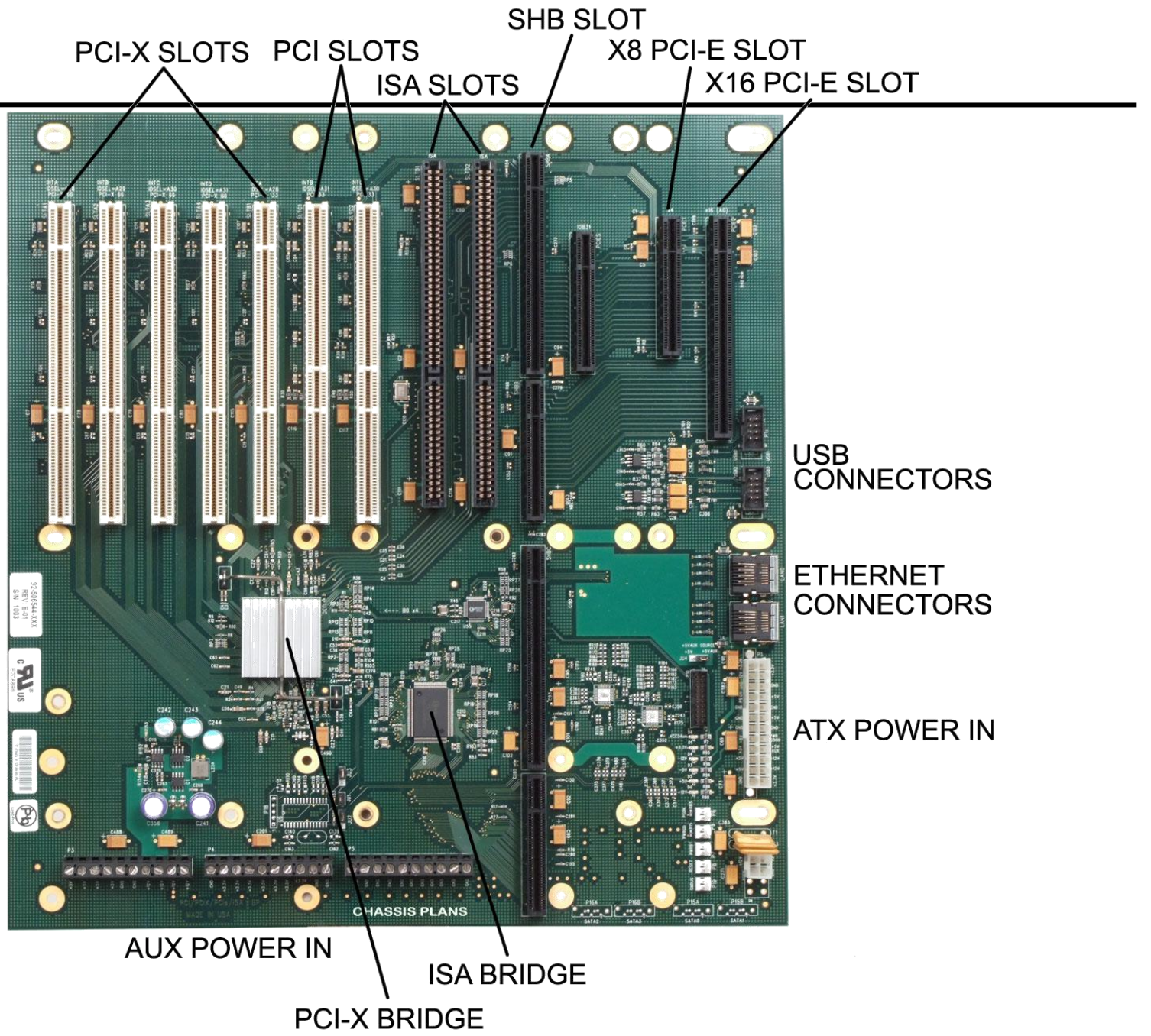
计算机概述

- 软件存储
  - 通用存储器存储
    - 硬盘、软盘、U盘、SSD
  - 专用存储器存储
    - FLASH、DOM
- 软件下载（针对嵌入式应用）
  - 专用编程器
  - JTAG接口
  - ISP在系统编程

计算机分类

- 巨型机、大型机、中型机、小型机
- 微机（PC）
- 单板机


AMIBIOS
C19228-001
CHASSIS PLANS


PCI-X SLOTS
PCI SLOTS
ISA SLOTS
SHB SLOT
X8 PCI-E SLOT
X16 PCI-E SLOT
USB CONNECTORS
ETHERNET CONNECTORS
ATX POWER IN
AUX POWER IN
ISA BRIDGE
PCI-X BRIDGE
CHASSIS PLANS

计算机分类

- 巨型机、大型机、中型机、小型机
- 微机（PC）
- 单板机
- 单片机（Microcontroller）
- 片上系统（SOC）

嵌入式系统概述

丰富多彩的嵌入式产品

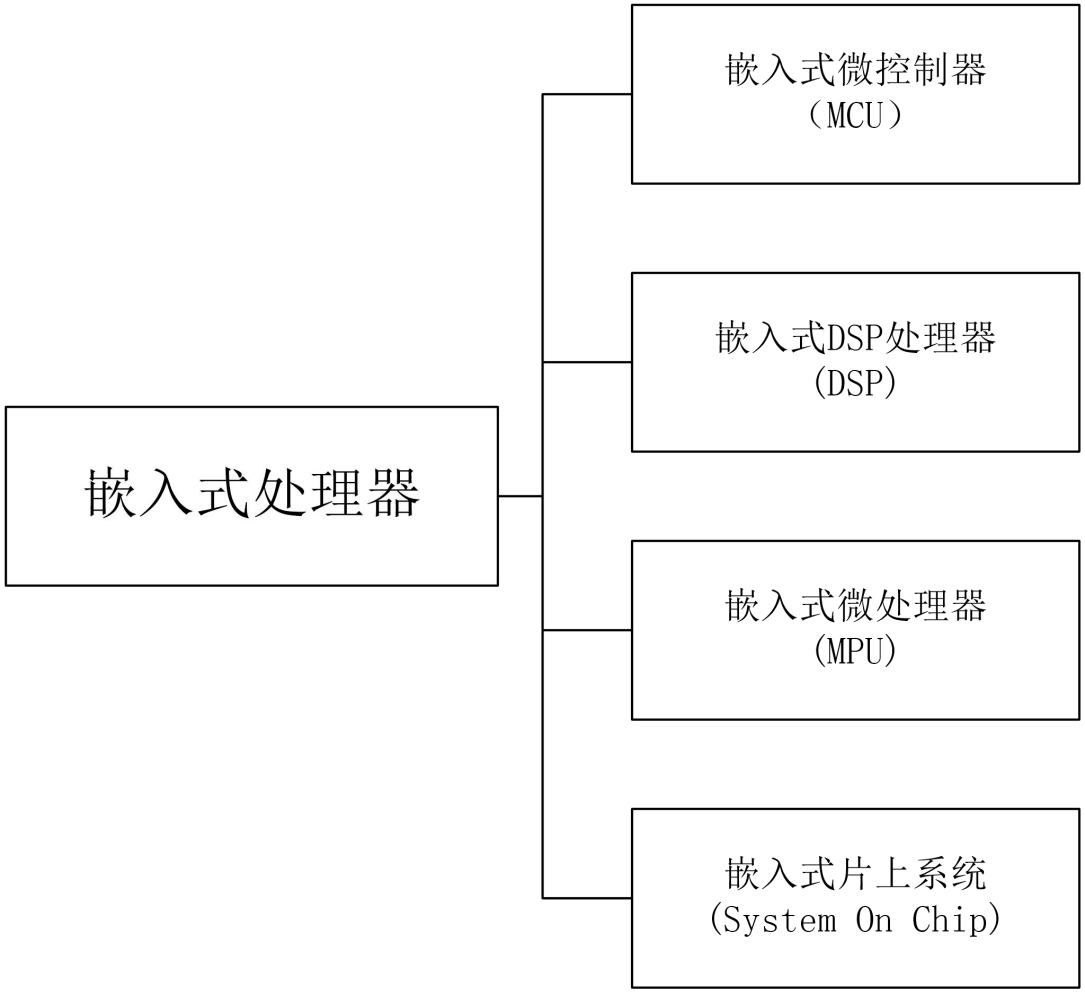
嵌入式系统的概念

- IEEE（电气和电子工程师协会）定义嵌入式系统：devices used to control, monitor, or assist the operation of equipment, machinery or plants—**控制、监视或者辅助操作机器和设备运行的装置**
- 国内一般认为**嵌入式系统是以应用为中心**，以计算机技术为基础，软件、硬件可裁剪，功能、可靠性、成本、体积、功耗严格要求的**专用计算机系统**

嵌入式系统的构架

- 嵌入式系统的构架可以分成四个部分：**处理器、存储器、输入输出（I/O）和软件**
- 多数嵌入式设备的应用软件和操作系统都是紧密结合

嵌入式处理器分类


嵌入式处理器
嵌入式微控制器
（MCU）
嵌入式DSP处理器
(DSP)
嵌入式微处理器
(MPU)
嵌入式片上系统
(System On Chip)

单片机技术基础

单片机技术基础

- 单片机特点
  - 体积小
  - 成本低
  - 功耗低
  - 面向应用为主，特别是嵌入式应用
  - 运算能力弱，控制能力较强

单片机技术基础

- 如何去学习单片机？
  - Step 1:了解单片机的资源
  - Step 2:了解单片机系统组成
  - Step 3:学习单片机最小系统及指令
  - Step 4:实验单片机的各种功能
  - Step 5:综合应用

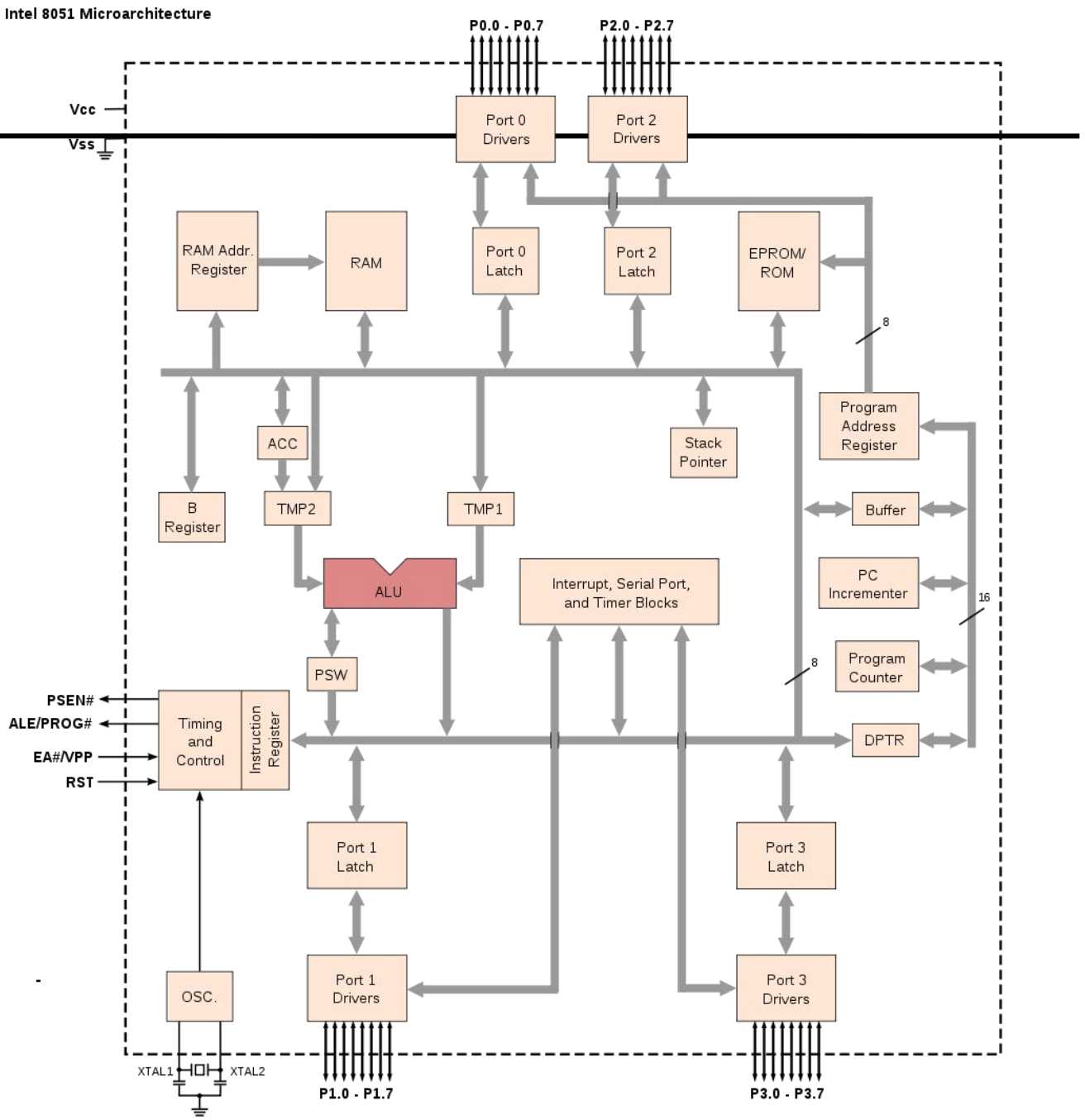
Step 1:Features and Resources

- 对一台全新的计算机，你会关注什么？
  - CPU、内存、硬盘.....
- 对一块单片机，你又会关注什么？
  - 从数据手册（Datasheet）说开去
  - Focus on:
    - Voltage and Current
    - I/O Resources (pin number/footprint)
    - Memory Resources (internal/external)
    - Working Speed (frequency)
  - Brief Description

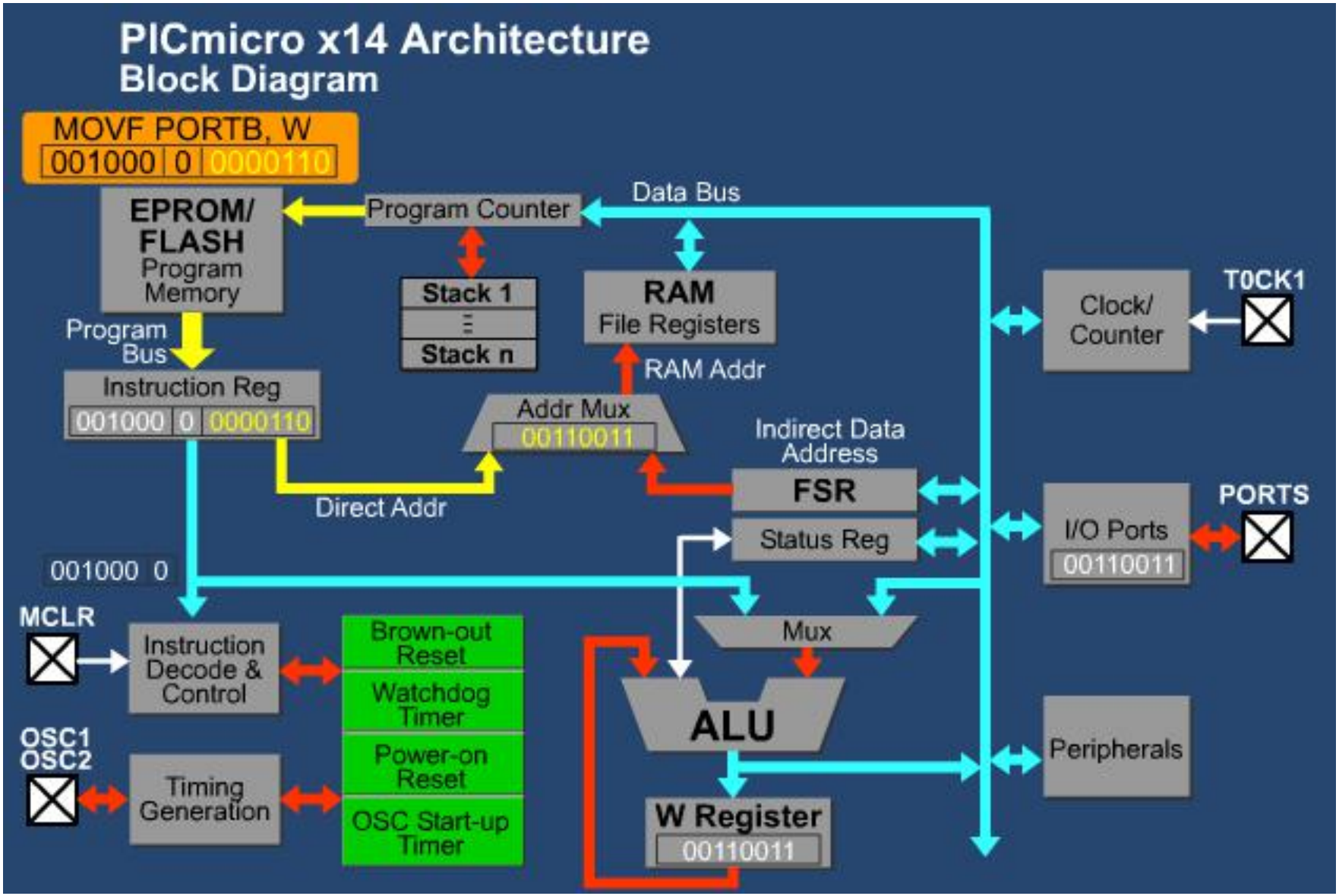
Step 2: MCU Architecture

- Read Datasheet
- Look for Components
- Focus on:
  - Input/Output Configuration
  - Control Signal connection

8051 Architecture



PIC Mid-Range Architecture

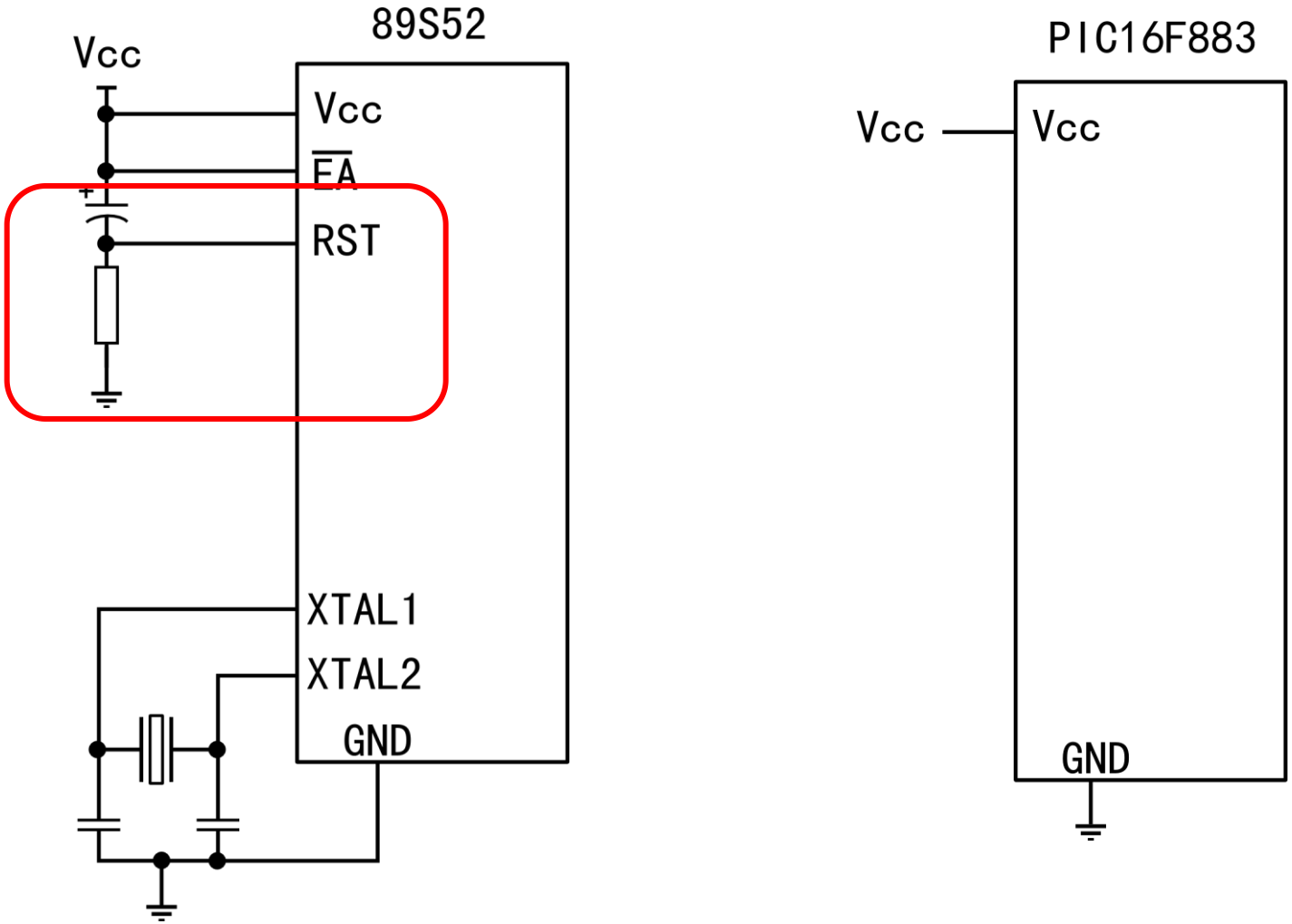


Step 3: How can we make it run?

- The requirement on MCU development
  - Hardware Connection
    - Power Supply
    - Oscillator
    - Reset
  - Software Development
    - Instruction
    - Development Environment
  - Firmware Download
    - Download tool
  - Focus on:
    - The minimal hardware requirement
    - The minimal instruction requirement
    - The development (firmware DL) requirement

Step 3: How can we make it run?

- The minimal hardware requirement



Step 4: MCU resources

- Focus on:
  - Completed Instruction Sets
  - How to control resources
    - List all corresponding registers to each resource
  - **Design testing code** on resources

Step 5: Integrated Application

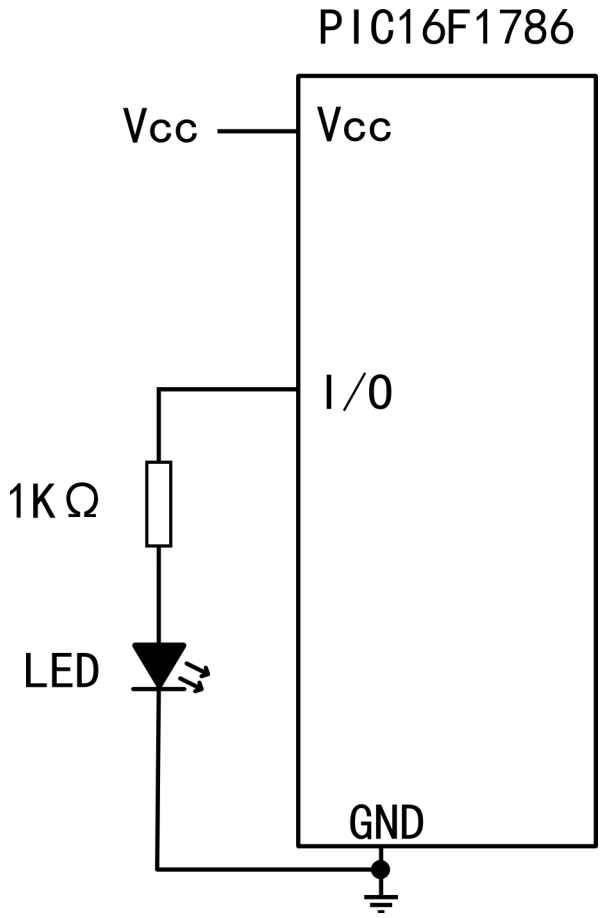
- Could you apply all the resources in one application?
- How to **keep balance** in handling several resources?
- Focus on:
  - Hardware/Software assignment
  - Software Implementation

MCU Development Env.

- MPLAB X IDE
- PIC16F18854 w/ Bootloader

实验一：闪灯实验

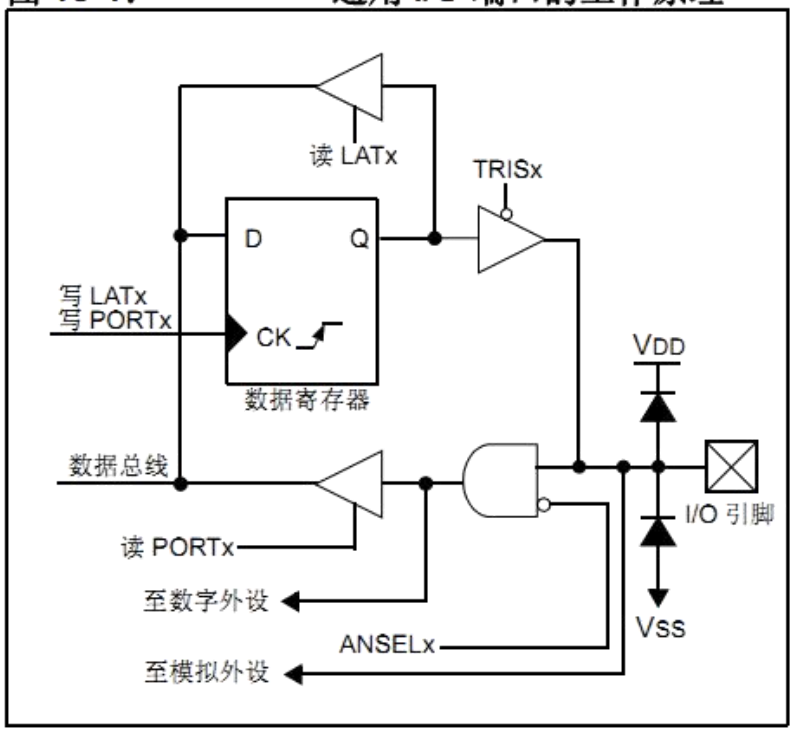
- **Hello World**
- 硬件原理

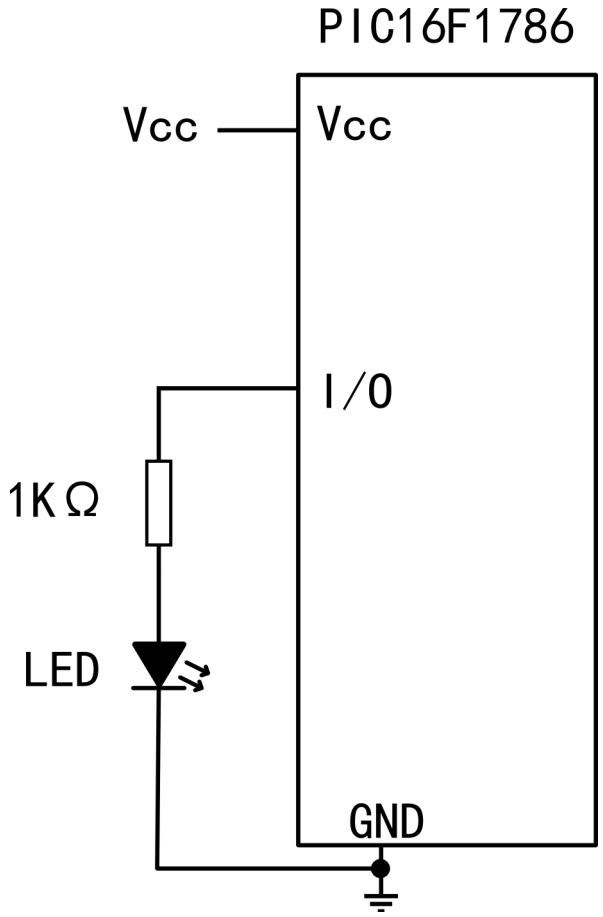


实验一：闪灯实验

- **Hello World**
- 硬件原理
- 软件设计
  - 如何控制LED亮灭？

图 13-1：　　通用 I/O 端口的工作原理





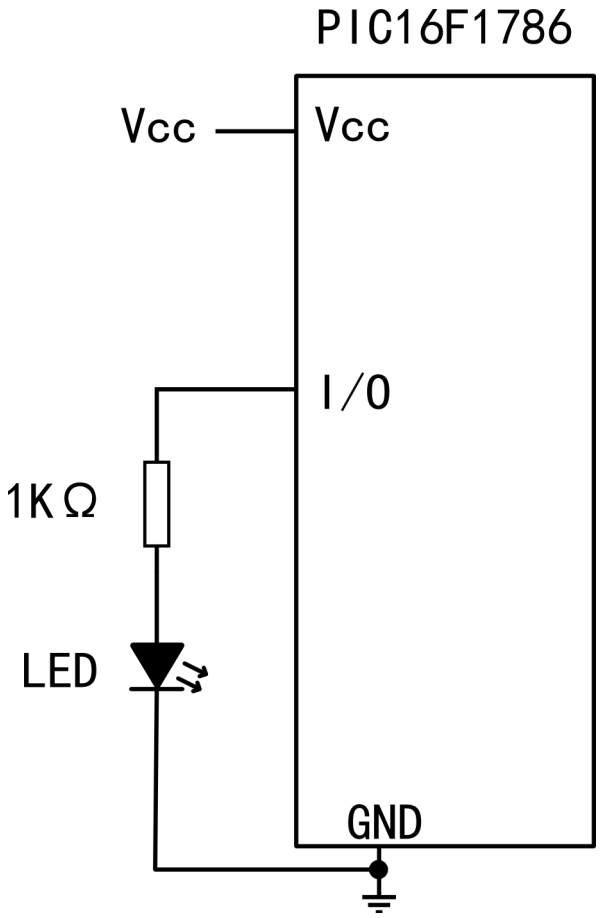
实验一：闪灯实验

- **Hello World**
- 硬件原理
- 软件设计
  - 如何控制LED亮灭？

例 13-1：　　初始化 PORTA

```
; This code example illustrates
; initializing the PORTA register.The
; other ports are initialized in the same
; manner.

BANKSEL  PORTA        ;
CLRF     PORTA        ;Init PORTA
BANKSEL  LATA         ;Data Latch
CLRF     LATA         ;
BANKSEL  ANSELA       ;
CLRF     ANSELA       ;digital I/O
BANKSEL  TRISA        ;
MOVLW    B'00111000'  ;Set RA<5:3> as inputs
MOVWF    TRISA        ;and set RA<2:0> as
                      ;outputs
```



实验一：闪灯实验

- **Hello World**
- 硬件原理
- 软件设计
  - 如何控制LED亮灭？
  - 完成闪烁需要哪些语句？（指令？）
  - 最简单的汇编结构是？

```
org 0x0
...
instructions
...
end
```

实验一：闪灯实验

- 前期准备
  - 安装MPLAB X IDE （版本v5.20+）
  - 安装XC8编译器 （版本v2.00+，实验二之后使用）
  - 快速跳跃阅读PIC16F18854 Datasheet
    - 主要关注：振荡器、**IO**端口、指令集部分
  - 参考阅读PIC Mid-Range MCU Family Reference Manual
  - 学习PICmicro x14 Basic Training:
    - 自学PICmicro x14 Architecture章节

实验一：闪灯实验

- 任务要求
  - 在口袋板上完成第一个单片机实验
  - 用汇编语言完成
  - 下载前先进行软件仿真
  - 记录每次下载时遇到的问题

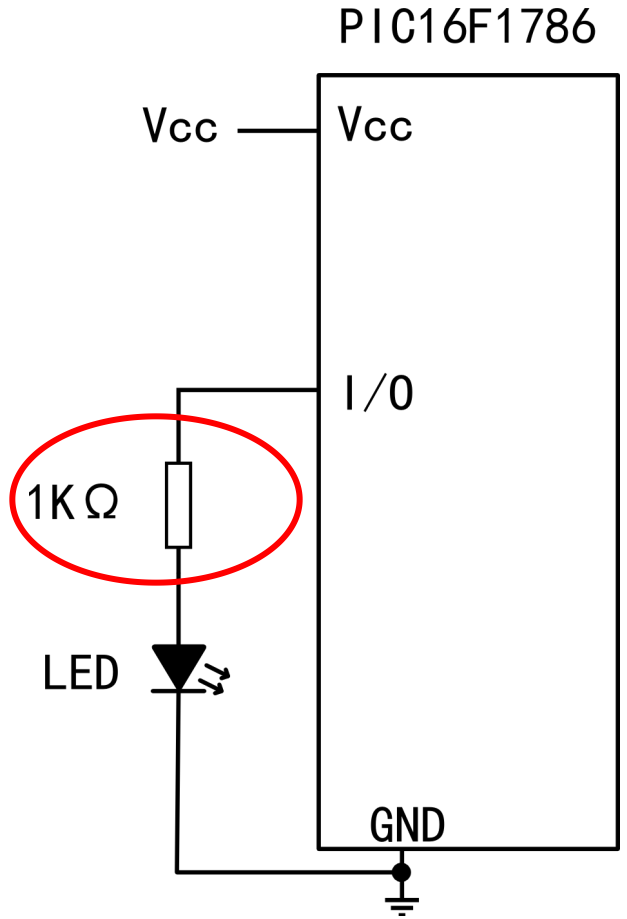
实验一时间

实验一分析

- 前提讨论：
  - 为什么设定实验一的内容为闪灯实验？
- 分析
  - 目标
    - 证实软件对硬件的可控性（软硬件操作的完备检验）
  - 先验知识
    - 几乎为零——风险很高
  - 方法（步骤）
    - 硬件——最简（最小系统）
    - 软件——最简（尽量减少对器件资源的依赖）

实验一分析

- 硬件连接
  - LED与电阻



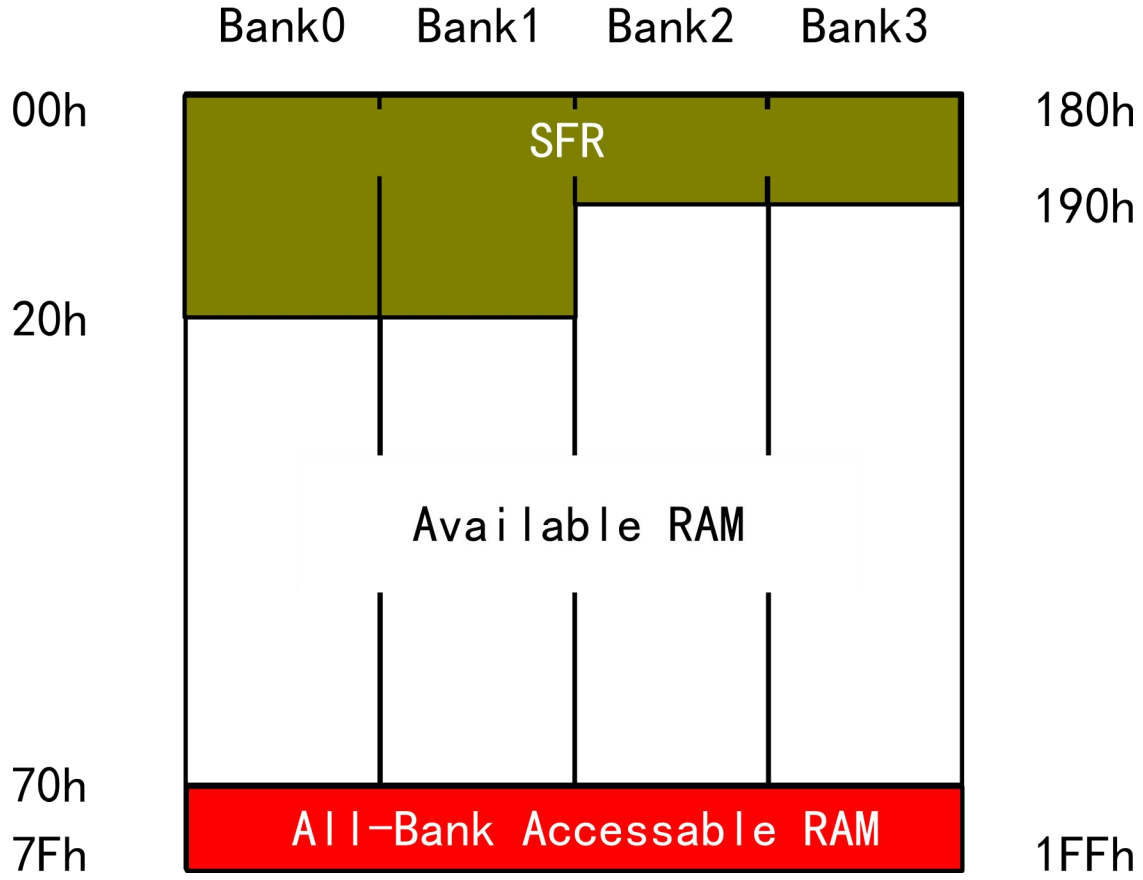
实验一分析

- 硬件连接
  - LED与电阻
- 单片机软件编写
  - 端口操作
  - 延时程序段

如何操作寄存器/内存？

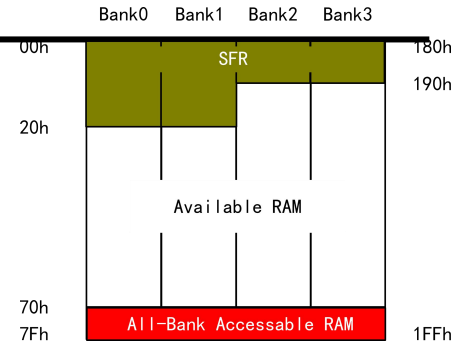
实验一分析

- PIC单片机的内存组织形式（PIC16F886）



1. 为什么需要分区（bank）进行存储器的组织？
2. 如何指定不同的bank进行访问？

实验一分析



- 访问不同bank的方法
  - 4个bank至少需要2bit进行区分

寄存器 2-1：　　STATUS：状态寄存器 （PIC16F886）

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC(1)	C(1)
bit 7							bit 0

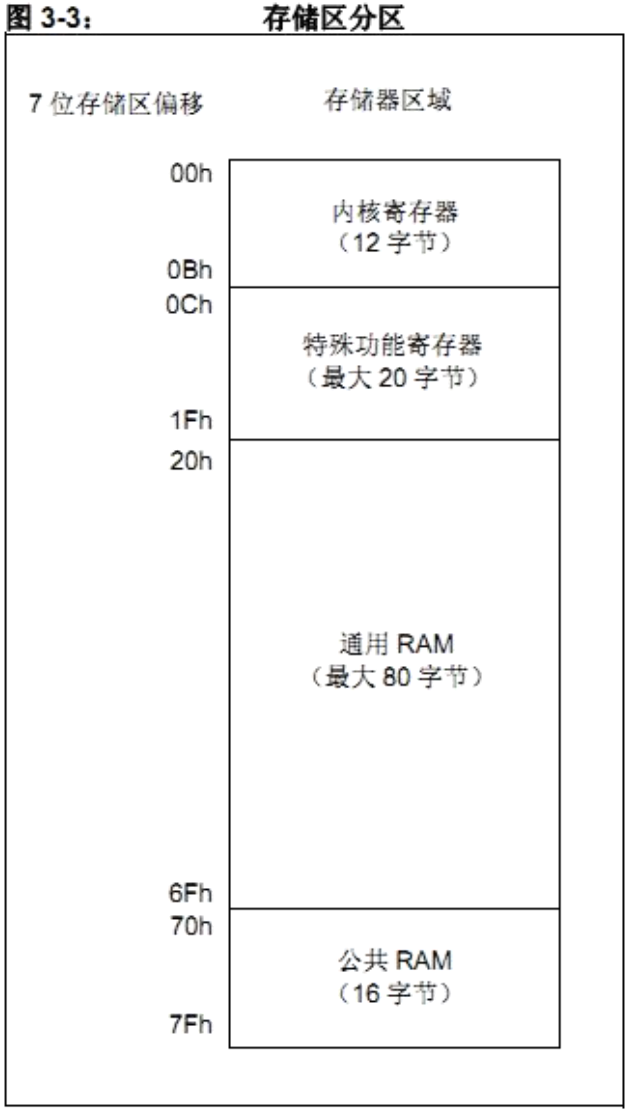
bit 6-5　**RP<1: 0>：**寄存器存储区选择位 （用于直接寻址）

00 = Bank 0 （00h-7Fh）
01 = Bank 1 （80h-FFh）
10 = Bank 2 （100h-17Fh）
11 = Bank 3 （180h-1FFh）

  - 通过直接设置RP1、RP0位选择不同的bank
  - 通过banksel宏汇编指令，由编译器自动完成寄存器切换选择工作
    - banksel PORTC ->
      - bcf　　STATUS, RP1
      - bcf　　STATUS, RP0

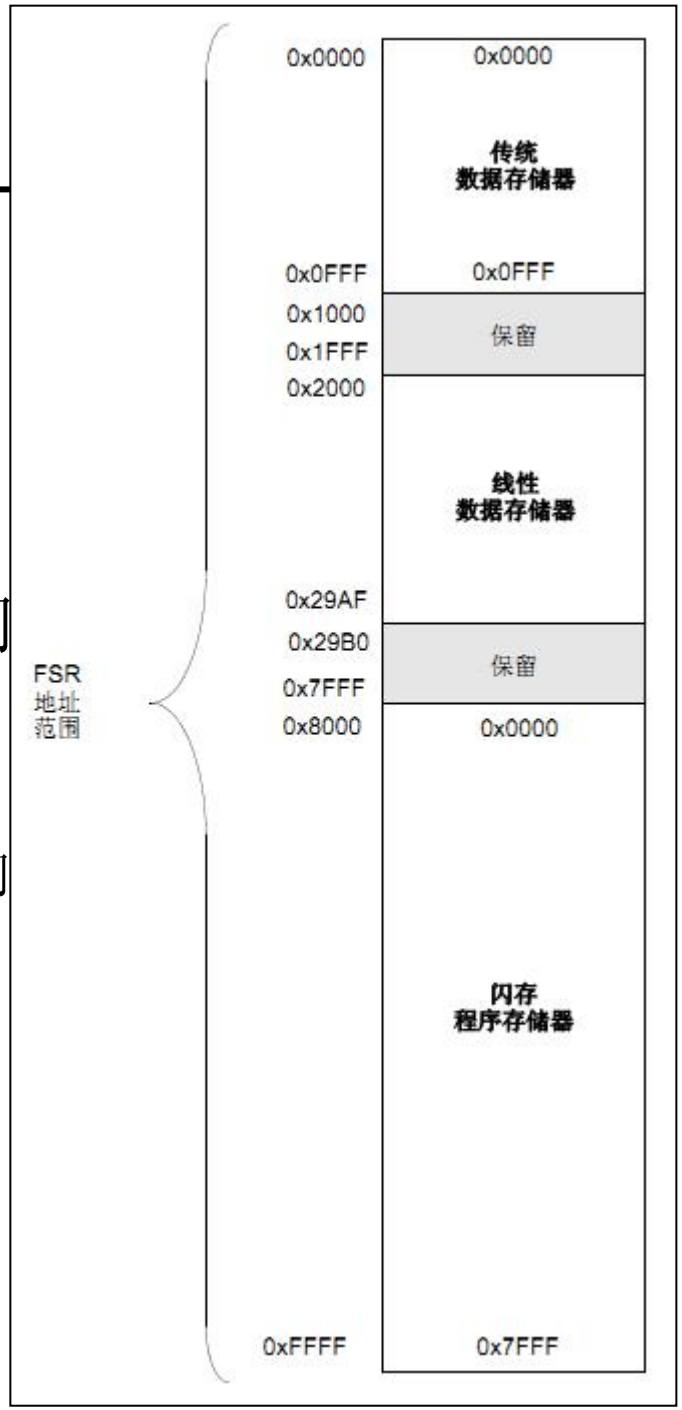
实验一分析

- PIC单片机的内存组织形式（PIC16F1x）
  - PIC16F1x系列最多可有4096字节RAM
  - 内存分区机制不能再依赖RP0、RP1两位
  - bank分区共有32个——通过BSR寄存器进行选择
  - 各bank内功能分区较为固定
    - 内核寄存器（core reg）
    - 特殊功能寄存器（SFR）
    - 通用RAM（GPRAM）
    - 各bank共用RAM（16字节）



实验一分析

- PIC单片机的内存组织形式（PIC16F1786）
- PIC16F1x系列支持连续间接寻址访问
  - 通过FSR（16bit）指定地址
  - 通过INDF可对FLASH、RAM进行访问
    - 对RAM支持传统访问和统一线性连续访问
    - 详见图3-10、图3-11



实验一分析

- 延时程序段
  - C语言的参考范例

```
for ( i = 0; i < 10000; i++ ) ;
```

```
unsigned char i, j;
while(++i)
        while(++j) ;
```

  - 汇编当中如何实现？

实验一分析

- PIC的汇编程序应该怎么写？
  - 汇编程序的本质
  - 机器指令的助记表达
  - 编译器支持下的难点简化
- 实验一程序的困难点
  - 变量存放在哪里？
  - 判断跳转指令用哪些来实现？
- 其他的一些汇编编写技巧
  - incf xxx, **w**
  - incf xxx, **f**

```
org 0x0
...
instructions
...
end
```

知识补充：寻址方式

- 立即数寻址
  - 操作数直接放在指令当中
  - 例：movlw 0x55
- 直接寻址
  - 操作数地址直接在指令当中
  - 例：movwf 0x55
- 寄存器寻址
  - 操作数存放在寄存器当中
- 间接寻址（参见数据手册3.5节）
  - 操作数的地址放在寄存器当中

知识补充：可重定位代码与绝对地址代码

- 绝对地址代码
  - 可用预定义语句辅助编程

```
delayvar1 EQU 0x70
delayvar2 EQU 0x71
```

- 可重定位代码
  - 由链接器帮助空间分配
  - 引入“段”的概念
    - 代码段
    - 数据段
    - 堆栈段
    - .....

```
org 0
main
init
        banksel TRISC
        movlw   b'01111111'
        movwf   TRISC
loop
        banksel PORTC
        movlw   0x80
        xorwf   PORTC, f
delayloop
        incfsz  0x70, f
        goto delayloop
        incfsz  0x71, f
        goto delayloop
        goto loop
end
```

知识补充：可重定位代码与绝对地址代码

- 可重定位代码示例

```
        udata_shr
delayvar1       res 1h
delayvar2       res 1h
```

  - code
  - udata_shr……
- More details in
  - Help->Help Contents
    - ->Language Tool
  - ->MPASMX Assembler

```
RST     code 0x0
        pagesel  main
        goto main

        code
main
        movlw   b'01111111'
        banksel  TRISC
        movwf   TRISC
loop
        banksel PORTC
        movlw   0x80
        xorwf   PORTC, f
delayloop
        decfsz  delayvar1, f
        goto    delayloop
        decfsz  delayvar2, f
        goto    delayloop
        goto    loop
        end
```

实验一小结

- 完成了第一个嵌入式系统的实验
  - 尝试了交叉编译
  - 尝试了程序下载
- 验证了实验系统的可用性
- 建立了软件到硬件的反馈途径
- 不足：
  - 闪烁的时间控制并不精确

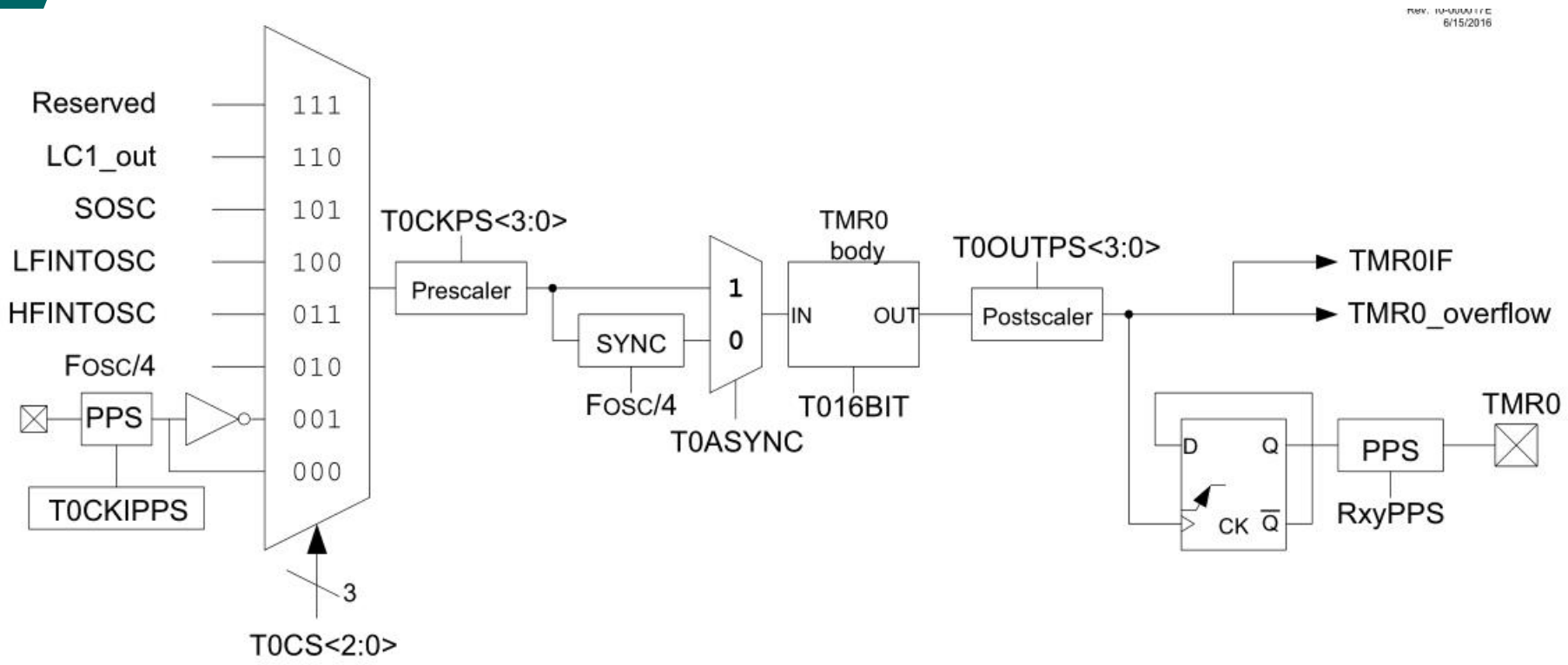
深挖实验一

- 代码所占用的程序存储区是多大？
- 通过软件仿真得到的闪烁间隔是多少？
  - 这个数值与实际硬件的表现有多大差异？
    - 如果能明显感觉到差异，问题在哪？

实验二：定时器查询闪灯实验

- 定时器Timer0、**Timer1**、Timer2
- 阅读Timer**X**模块框图

实验二：定时器查询闪灯实验



实验二：定时器查询闪灯实验

- 定时器Timer0、**Timer1**、Timer2
- 阅读Timer模块框图
  - 思考如何使用、控制定时器
- 任务要求
  - 用**可重定位汇编代码**完成1s为周期的闪灯实验
  - 画出程序流程图
  - 验证程序的正确性
  - 撰写实验报告，**描述计算过程**
  - 明天上午进行分组汇报，每组5分钟

实验二时间

实验二汇报、分析和讨论

- 分组汇报

实验二汇报、分析和讨论

- 精确定时？
  - 如何获得准确的1秒周期(500ms半周期)
- 分析
  - 检查溢出标志所需要的时间

```
test
        btfss   PIR4, TMR1IF
        goto    test
```

  - 减少对TMRx寄存器赋值的时间差
    1. 在TMRx溢出之后，TMRx仍然在保持计数状态
    2. 对TMRx使用计算值赋值越晚，则遗漏的时间越长
    3. 若一个长周期内赋值次数越少，则遗漏时间可能性越小
    4. 建立计数补偿机制有助于减少时间差

实验二汇报、分析和讨论

- 获得长周期定时的方法
  1. 预分频 × Timer计数 × **软件计数**
     - 例如：1:8预分频，8 bit Timer计数 200，软件计数10
     - 则一个周期为8×200×10 = 16000 us (4MHz)
  2. 预分频×[(后分频-1)×全Timer周期+Timer计数]
     - 例如：1:8预分频，8 bit Timer计数 200，后分频10
     - 则一个周期为8×(9×256+200)=20032 us(4MHz)
     - 还可以再辅以软件计数增加长度
- 它们的**区别**：
  - 在一个长周期内，是否每一次定时器溢出都要设置初始值

实验二汇报、分析和讨论

- 示例代码

```
TIMERCNT EQU .50000
TIMERMUL EQU .10
TMRVALUE EQU (.65536 - TIMERCNT )
TMRHVALUE EQU HIGH (TMRVALUE)
TMRLVALUE EQU LOW (TMRVALUE)
```

```
test    btfss   PIR4, TMR1IF
        goto    test
label1
        bcf     T1CON, TMR1ON
        movlw   TMRLVALUE
        addwf   TMR1L, f
        btfsc   STATUS, C
        incf    TMR1H
        movlw   (label2 - label1)
        addwf   TMR1L, f
        movlw   TMRHVALUE
        addwfc  TMR1H, f
label2
        bsf     T1CON, TMR1ON
        bcf     PIR4, TMR1IF
```

实验二小结

- 完成了单片机系统重要部件——定时器的测试
- 进一步了解**指令执行时间对任务的影响**
- 理解可重定位代码的编写规则
- 不足之处：
  - 单片机的大部分时间在空等
  - 如果穿插执行其他任务将影响对标志位的检查时效

知识补充：如何复现问题

- 为什么要复现问题？
  - 请阅读“如何有效地报告bug”
- 复现问题的手段
  - 使用相同的硬件
  - 使用相同的软件（包括版本）
  - **使用相同的操作步骤（明确、具体、无歧义）**
- 在我们的课程里复现问题
  - 使用相同的源码？
  - 应使用相同的工程设置（workspace）

单片机C语言

- 为什么用C语言？
  - 可读性和可维护性更好
  - 改进的内存分配策略
    - 进行自动分析，对内存可进行重用（overlap）
  - 不足之处：
    - 精细化控制能力较弱
    - 代码效率不高

单片机C语言

- XC8 PIC单片机C语言环境
  - 遵循ANSI C标准
  - 差异部分：增加了单片机平台特有的部分
    - #include <xc.h>，全局头文件
    - 特殊功能寄存器：定义了STATUS等SFR
    - 中断函数的声明方式
      - **void __interrupt()** irs_routine(**void**)
    - 查表的数据存放方式
      - 将const修饰的变量放置在程序存储区当中
      - const unsigned char buf[] = {"Hello"};
  - 隐藏的技术细节
    - main之前发生了什么？
      - 看看反汇编的代码（Window->Debuging->Disassembly）
      - XC8 linker选项：Keep generated startup.as
      - 看看程序存储区内容（Window->PIC Memory Views->Program Memory）

知识补充：C语言的那些事儿

- 代码风格与运行效率

```
#include <xc.h>

void main(void)
{
    unsigned char i;
    while(1)
    {
        while(i--);
        PORTB ^= 0xff;
    }
}
```

```
#include <xc.h>

void main(void)
{
    unsigned char i;
    while(1)
    {
        while(--i);
        PORTB ^= 0xff;
    }
}
```

知识补充：C语言的那些事儿

- 代码风格与运行效率

while(--i)

```
main  DECFSZ main@i, F
      GOTO main
      MOVLW 0xff
      XORWF PORTB, F
      GOTO main
```

while(i--)

```
main  DECF main@i, F
      MOVF main@i,W
      XORLW 0xff
      BTFSS STATUS, Z
      GOTO main
      MOVLW 0xff
      XORWF PORTB, F
      GOTO main
```

实验三：中断闪灯实验

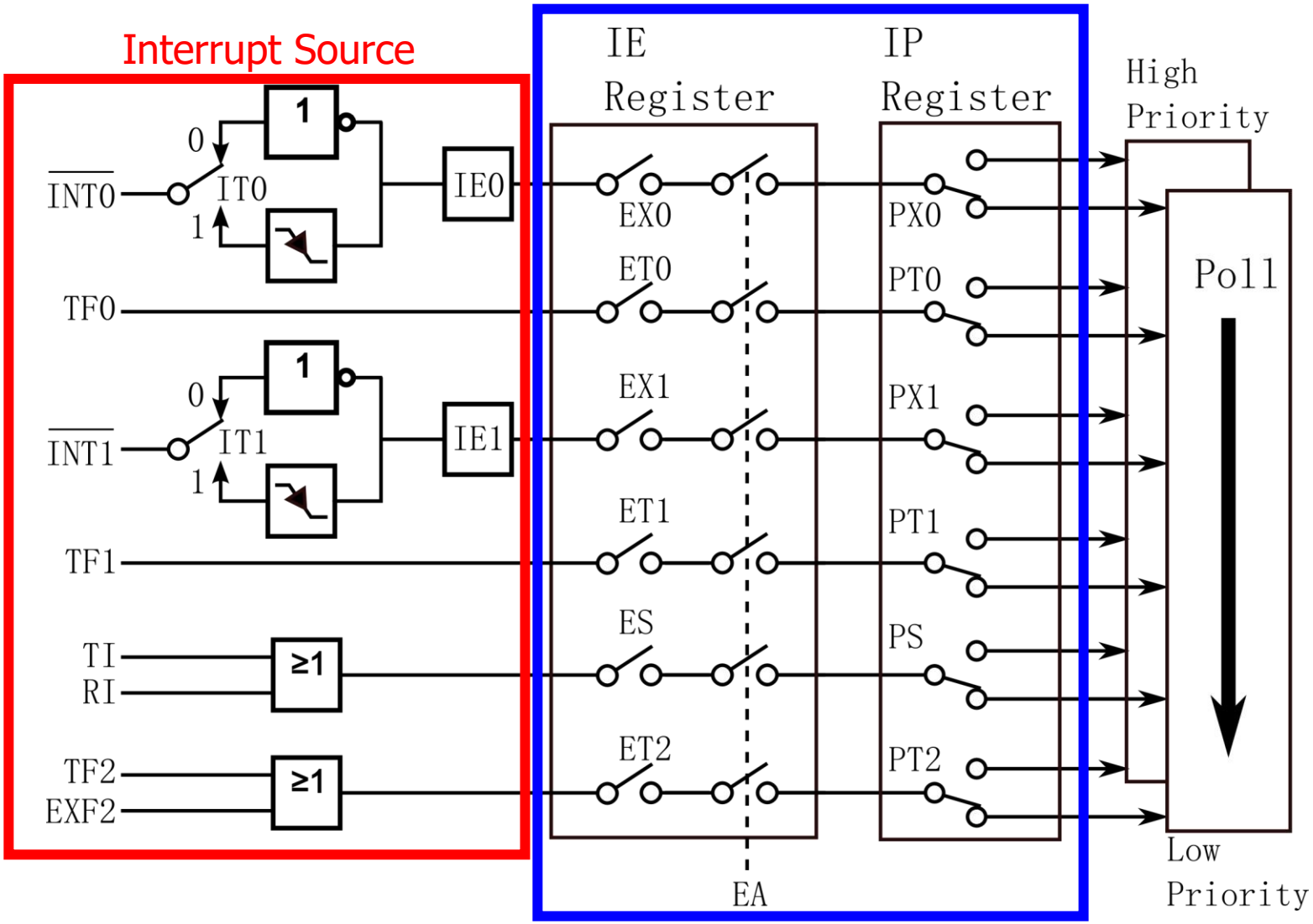
- 什么是中断
  - 中断是一种可打断控制器正常处理流程的异步事件
  - 或者是由软件产生的改变处理流程的同步事件
- 中断系统的构成
  - 中断源（Interrupt Source）
  - 中断控制器（Interrupt Controller）
  - 中断服务程序（Interrupt Service Routine）

实验三：中断闪灯实验

- PIC单片机的中断源
  - 多达十几种中断源，包括定时器、引脚中断等等
- PIC单片机的中断控制器
  - 功能简单，中断入口仅一个（中断逻辑见数据手册）
  - 需要中断服务程序自行判断中断源
  - 没有中断优先级概念

实验三：中断闪灯实验

- 对比51的中断系统



实验三：中断闪灯实验

标志出现到中断服务程序开始执行的时间称为中断响应延时

中断事件发生后如何处理？

- 硬件在每个机器周期轮询中断标志
- 如果中断标志出现，硬件将自动产生一条函数调用语句并指向事先约定的服务程序入口
  - 例：PIC单片机所有中断服务入口均为0004H
  - 例：89s52中断服务程序地址因中断源不同而不同
- 此后CPU将执行从服务程序入口开始代码，直到遇见函数返回语句
  - 注：中断服务程序当中的函数返回语句在不同的平台上可能会有特殊约定
  - 例：PIC系列中断返回用**RETFIE**而非RETURN
  - 例：51系列中断返回用RETI而非RET

实验三：中断闪灯实验

- **ISR与普通函数的区别**
  - 入口地址固定
  - 被调用的时刻不确定
  - 需要保存上下文环境
    - **context saving**
  - 返回指令不同

```
rst     code 0x0
    pagesel main
    goto main

isr     code 0x04
        ……
        retfie

        code
main
        ……
        end
```

实验三：中断闪灯实验

- 上下文保护（context saving）
  - CALL之后发生了什么？
    - 返回地址记录在哪里？
  - ISR应保护所有可能改变，但非ISR专用的寄存器
    - PIC16F1x具备有限自动现场保护功能

8.5　自动现场保护

进入中断时， PC 的返回地址被保存在堆栈中。此外，以下寄存器会被自动保存到影子寄存器中：

- W 寄存器
- STATUS 寄存器 （$\overline{TO}$ 和 $\overline{PD}$ 除外）
- BSR 寄存器
- FSR 寄存器
- PCLATH 寄存器

在退出中断服务程序时，将会自动恢复这些寄存器。在 ISR 期间对这些寄存器进行的任何修改都会丢失。如果需要修改其中的任意寄存器，则应修改相应的影子寄存器，该值在退出 ISR 时将会被恢复。影子寄存器位于 Bank 31 中，它们是可读写寄存器。根据用户的应用，可能还需要保存其他寄存器。

中断中用到的其他非ISR专用寄存器需要进行现场保护

实验三：中断闪灯实验

- 上下文保护（context saving）
  - CALL之后发生了什么？
    - 返回地址记录在哪里？
  - ISR应保护所有可能改变，但非ISR专用的寄存器
    - PIC16F1x具备有限自动现场保护功能
    - PIC16F886保护参考
  - 例如：W，STATUS

```
context_saving  udata_shr
        w_tmp   res 1
        s_tmp   res 1
```

为什么不用
movf STATUS, w

```
ISR     code    0x4
        movwf   w_tmp
        swapf   STATUS, W
        movwf   s_tmp
        ......
        swapf   s_tmp, w
        movwf   STATUS
        swapf   w_tmp, f
        swapf   w_tmp, w
        retfie
```

实验三：中断闪灯实验

- 任务要求：
  - **阅读数据手册7章，中断部分**
  - **阅读数据手册各Timer模块的中断章节**
  - 用**定时器中断方式**完成闪灯实验
  - 采用C语言实现

实验三时间