微型计算机的组成

程晨闻

东南大学电气工程学院

➢ 冯·诺依曼体系结构

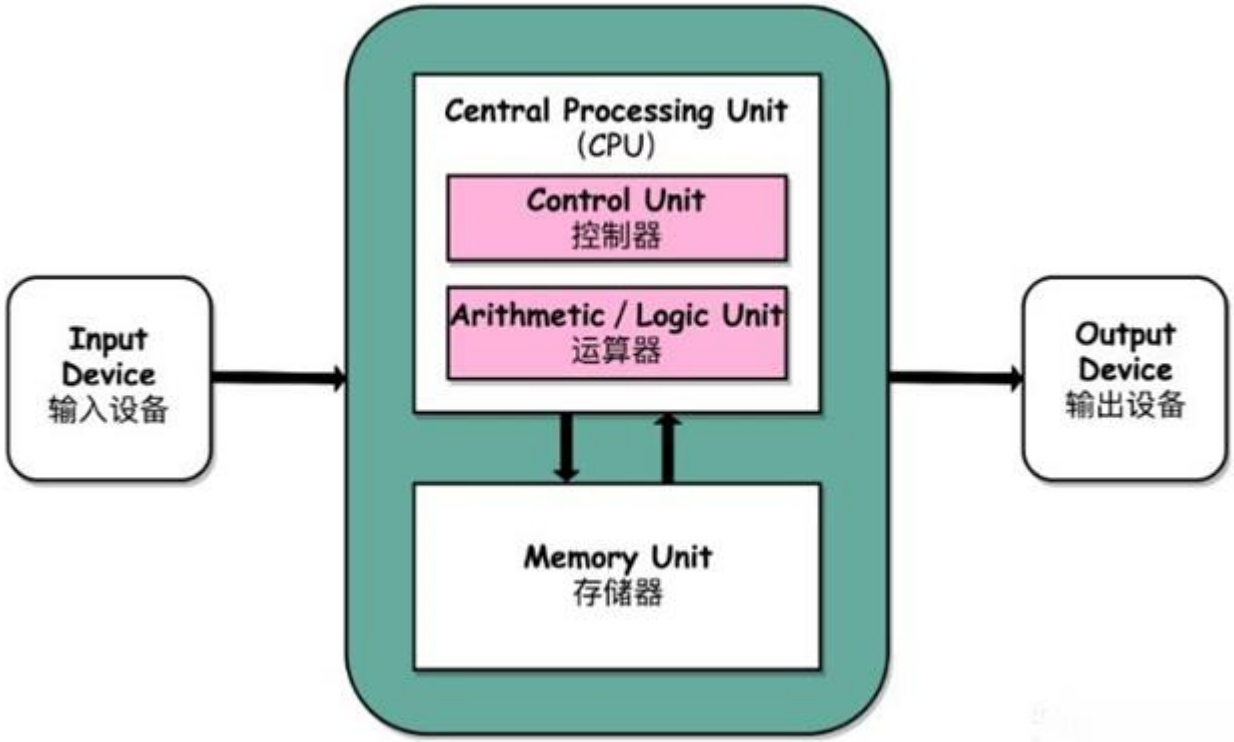
– 计算机处理的数据和指令用**二进制**数表示；

– 采用**存储程序**方式，指令和数据存储在存储器中；

– **顺序执行**程序的每一条指令；

– 由**存储器**、**运算器**、**控制器**、**输入设备**和**输出设备**五大部件组成计算机系统，并规定了这五部分的基本功能。

➢ 冯·诺依曼体系结构

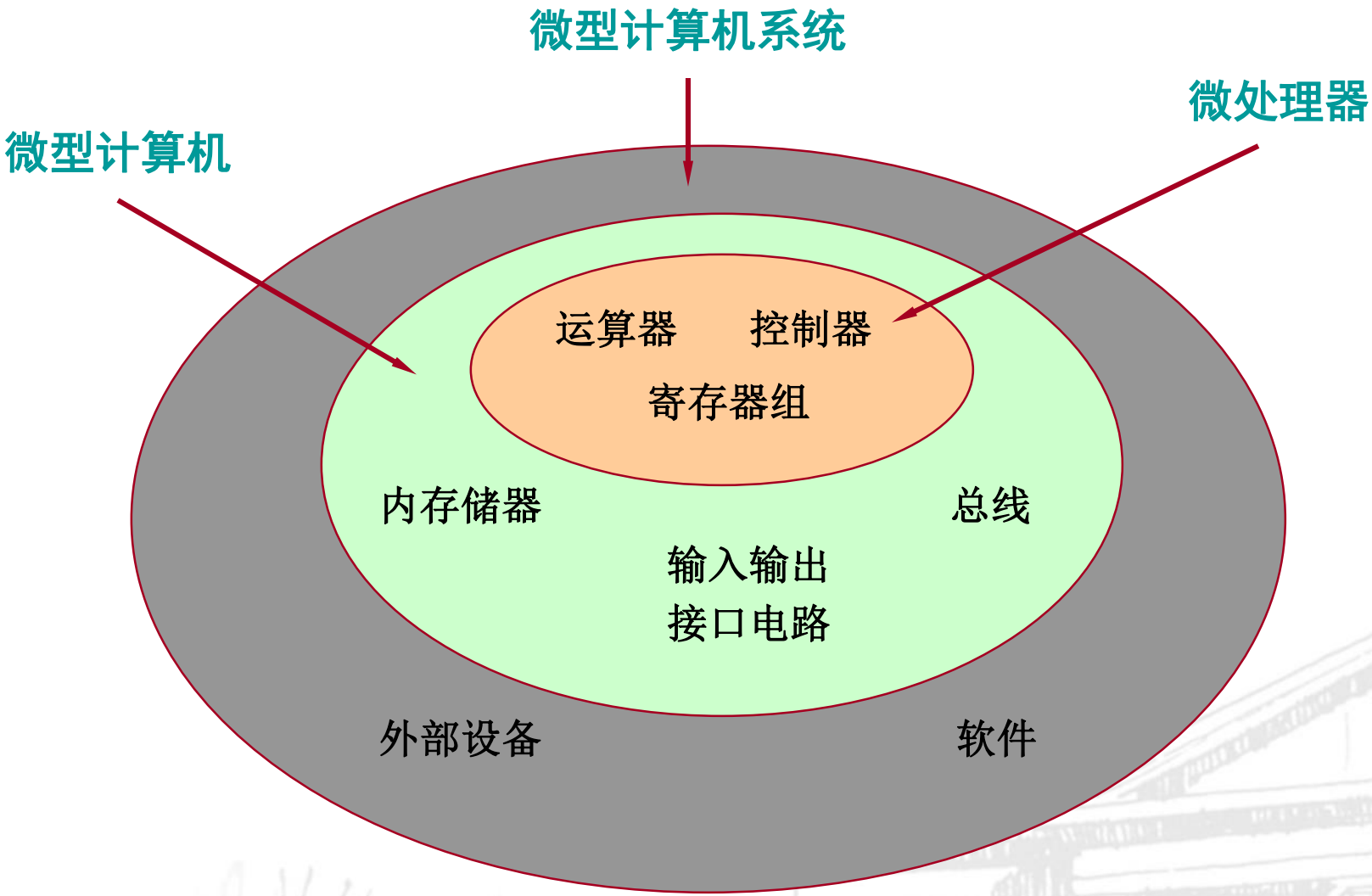

Central Processing Unit
(CPU)
Control Unit
控制器
Arithmetic / Logic Unit
运算器
Input
Device
输入设备
Output
Device
输出设备
Memory Unit
存储器

冯·诺依曼体系结构

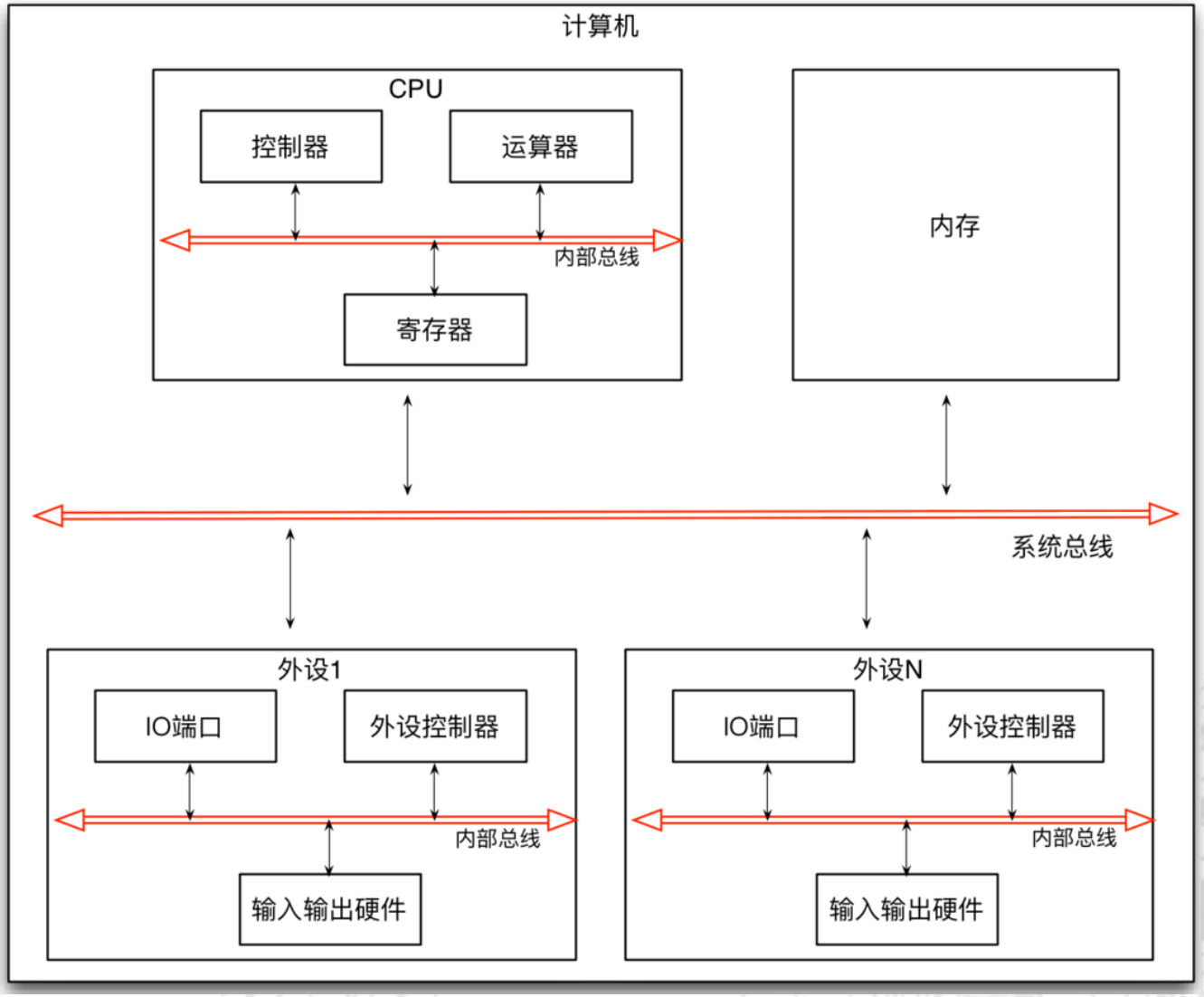
- **存储器**：用来存放数据和程序
- **运算器**：主要运行算数运算和逻辑运算，并将中间结果暂存到运算器中
- **控制器**：主要用来控制和指挥程序和数据的输入运行，以及处理运算结果
- **输入设备**：用来将人们熟悉的信息形式转换为机器能够识别的信息形式，常见的有键盘，鼠标等
- **输出设备**：可以将机器运算结果转换为人们熟悉的信息形式，如打印机输出，显示器输出等

运算器和控制器合为处理器（CPU），输入输出设备合为I/O设备。

微型计算机的组成



采用总线结构连接各个功能部件



- 总线是指传递信息的一组公用导线
- 总线是传送信息的公共通道
- 任一时刻，在总线上只能传递一种信息，只有有一个部件在发送信息，但可以有多个部件在接收信息
- 系统总线信号可分成三组
  - 地址总线AB：传送地址信息
  - 数据总线DB：传送数据信息
  - 控制总线CB：传送控制信息

系统总线

– 地址总线AB

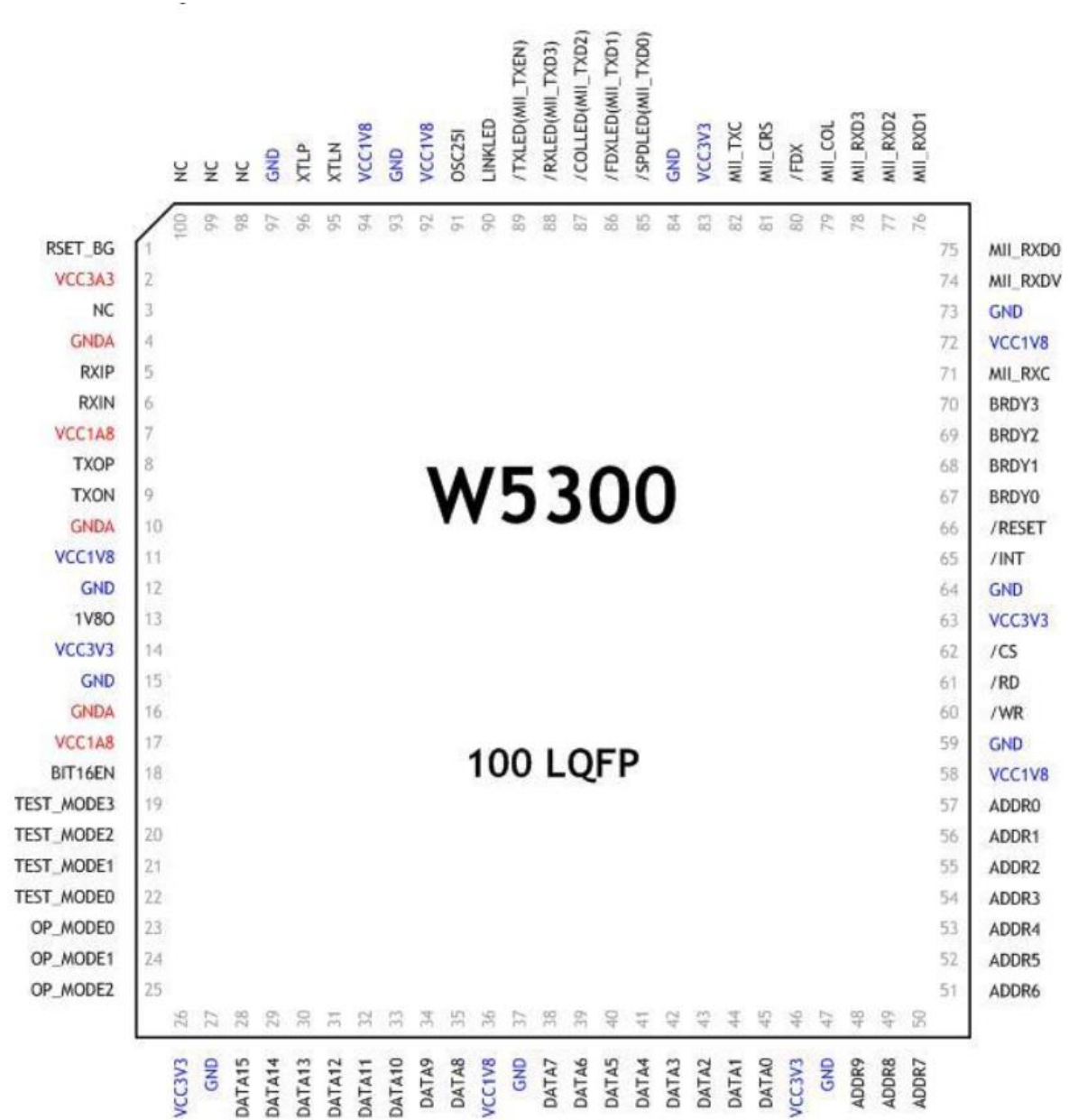
- 输入/输出将要访问的内存单元或I/O端口的地址
- 地址线的多少决定了系统直接寻址存储器的范围

– 数据总线DB

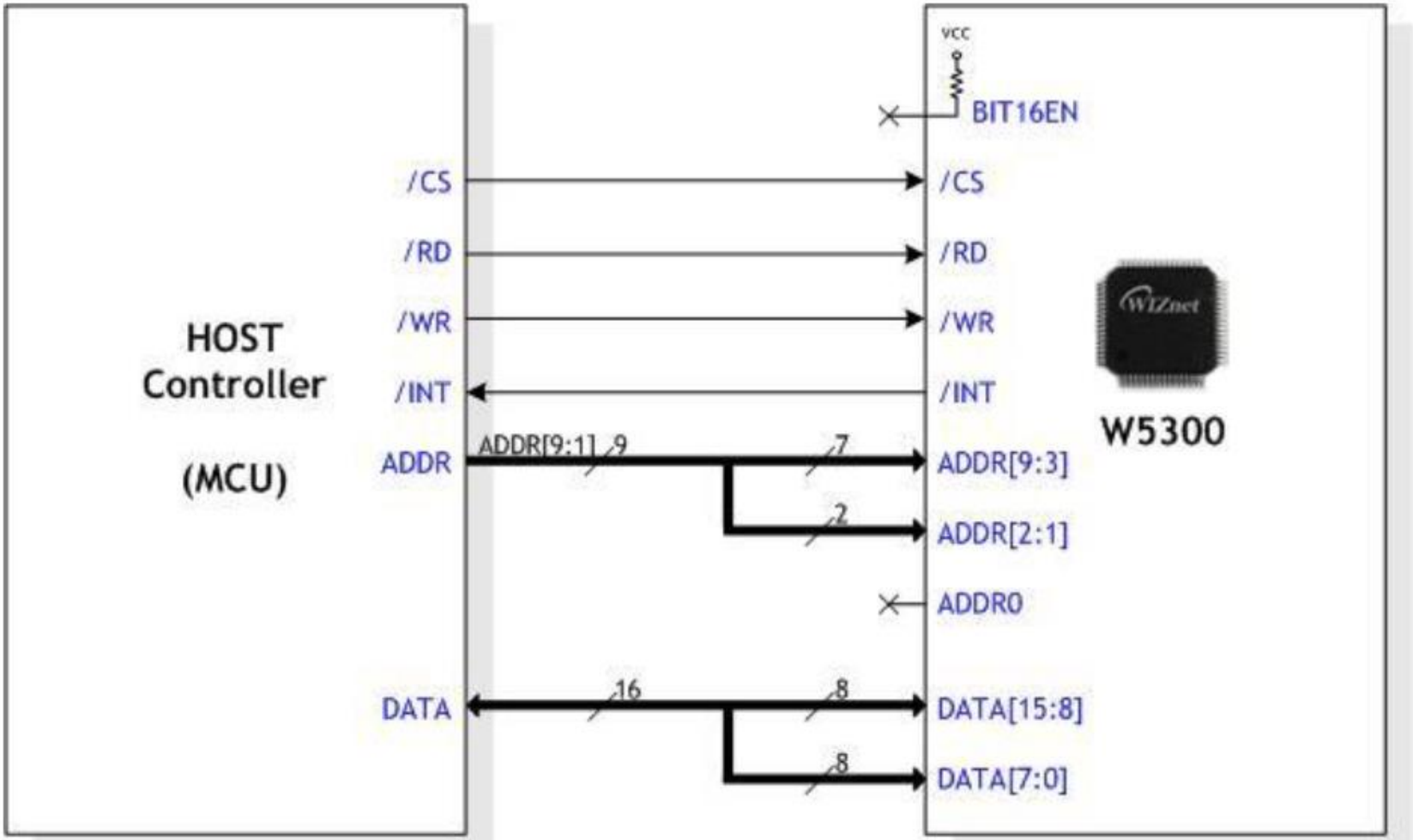
- CPU读操作时，外部数据通过数据总线送往CPU
- CPU写操作时，CPU数据通过数据总线送往外部
- 数据线的多少决定了一次能够传送数据的位数

– 控制总线CB

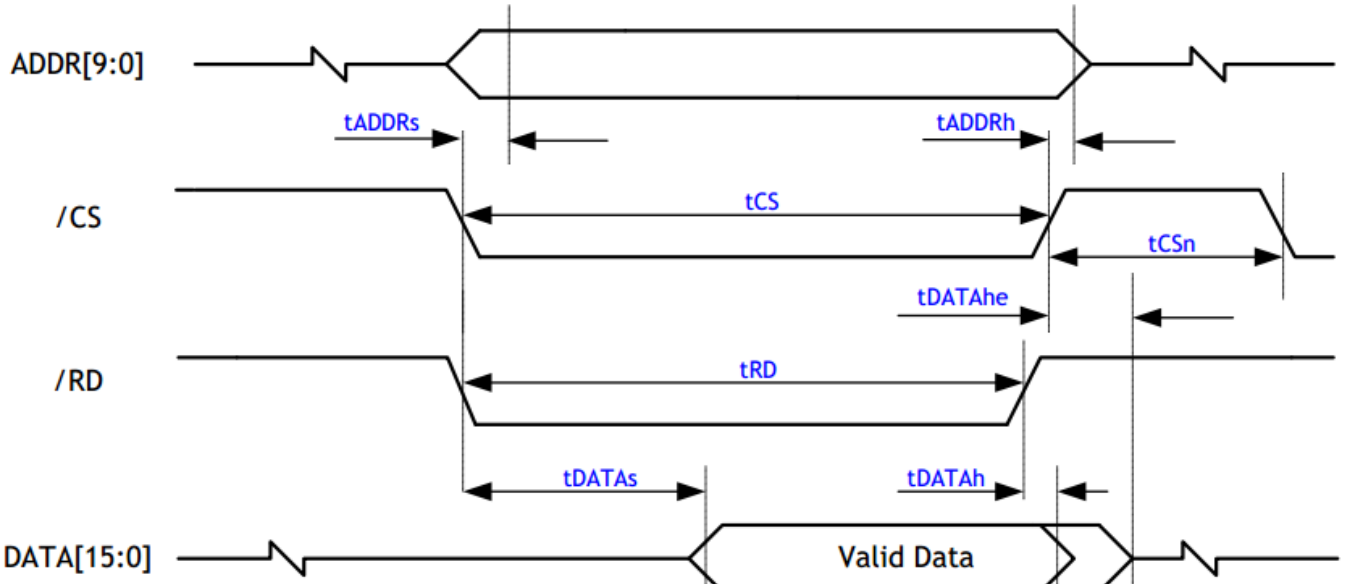
- 协调系统中各部件的操作，有输出控制、输入状态等信号
- 控制总线决定了系统总线的特点，例如功能、适应性等


W5300
100 LQFP
1 RSET_BG
2 VCC3A3
3 NC
4 GNDA
5 RXIP
6 RXIN
7 VCC1A8
8 TXOP
9 TXON
10 GNDA
11 VCC1V8
12 GND
13 1V8O
14 VCC3V3
15 GND
16 GNDA
17 VCC1A8
18 BIT16EN
19 TEST_MODE3
20 TEST_MODE2
21 TEST_MODE1
22 TEST_MODE0
23 OP_MODE0
24 OP_MODE1
25 OP_MODE2
26 VCC3V3
27 GND
28 DATA15
29 DATA14
30 DATA13
31 DATA12
32 DATA11
33 DATA10
34 DATA9
35 DATA8
36 VCC1V8
37 GND
38 DATA7
39 DATA6
40 DATA5
41 DATA4
42 DATA3
43 DATA2
44 DATA1
45 DATA0
46 VCC3V3
47 GND
48 ADDR9
49 ADDR8
50 ADDR7
51 ADDR6
52 ADDR5
53 ADDR4
54 ADDR3
55 ADDR2
56 ADDR1
57 ADDR0
58 VCC1V8
59 GND
60 /WR
61 /RD
62 /CS
63 VCC3V3
64 GND
65 /INT
66 /RESET
67 BRDY0
68 BRDY1
69 BRDY2
70 BRDY3
71 MII_RXC
72 VCC1V8
73 GND
74 MII_RXDV
75 MII_RXD0
76 MII_RXD1
77 MII_RXD2
78 MII_RXD3
79 MII_COL
80 /FDX
81 MII_CRS
82 MII_TXC
83 VCC3V3
84 GND
85 /SPDLED(MII_TXD0)
86 /FDXLED(MII_TXD1)
87 /COLLED(MII_TXD2)
88 /RXLED(MII_TXD3)
89 /TXLED(MII_TXEN)
90 LINKLED
91 OSC25I
92 VCC1V8
93 GND
94 VCC1V8
95 XTLN
96 XTLP
97 GND
98 NC
99 NC
100 NC

系统总线


HOST
Controller
(MCU)
/CS
/RD
/WR
/INT
ADDR
DATA
ADDR[9:1]
9
7
2
16
8
8
VCC
BIT16EN
/CS
/RD
/WR
/INT
ADDR[9:3]
ADDR[2:1]
ADDR0
DATA[15:8]
DATA[7:0]
W5300

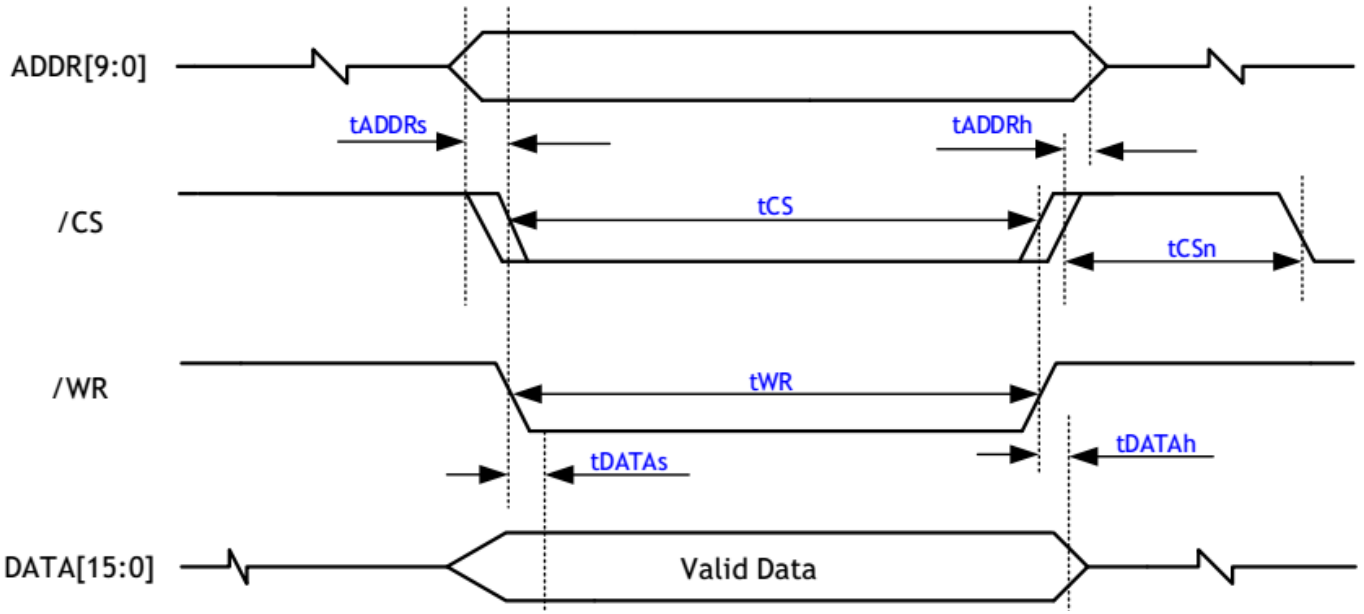
Register READ Timing



Description		Min	Max
tADDRs	Address Setup Time after /CS and /RD low	-	7 ns
tADDRh	Address Hold Time after /CS or /RD high	-	-
tCS	/CS Low Time	65 ns	-
tCSn	/CS Next Assert Time	28 ns	-
tRD	/RC Low Time	65 ns	-
tDATAs	DATA Setup Time after /RD low	-	42 ns
tDATAh	DATA Hold Time after /RD and /CS high	-	7 ns
tDATAhe	DATA Hold Extension Time after /CS high	-	2XPLL_CLK

<Note> 'tDATAhe' is the data holding time when MR(RDH) is '1'. During this time, data bus is driven during 2XPLL_CLK after /CS is de-asserted high. So, be careful of data bus collision.

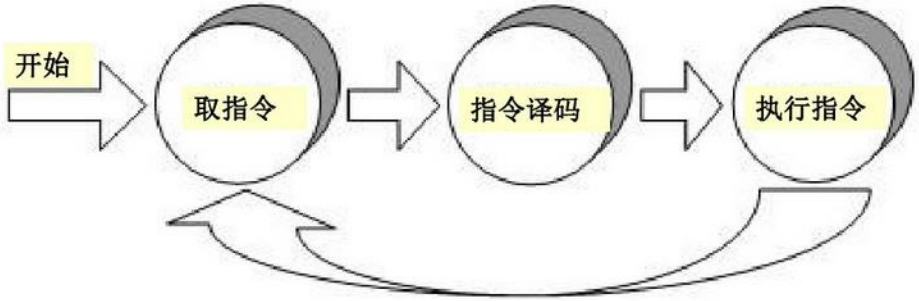
Register WRITE Timing



Description		Min	Max
tADDRs	Address Setup Time after /CS and /WR low	-	7 ns
tADDRh	Address Hold Time after /CS or /RD high	-	-
tCS	/CS low Time	50 ns	-
tCSn	/CS next Assert Time	28 ns	
tWR	/WR low time	50 ns	
tDATAs	Data Setup Time after /WR low	7 ns	-
tDATAh	Data Hold Time after /WR high	7 ns	-

➢ CPU的工作原理

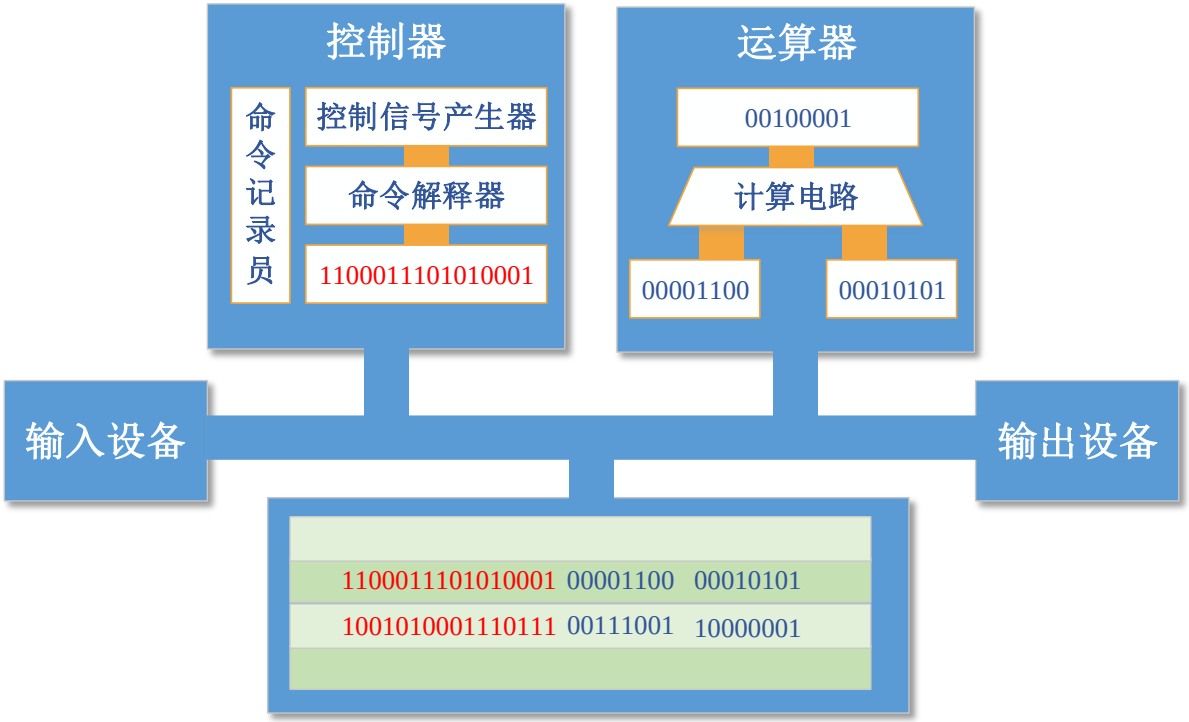
– **程序的执行过程实际上是不断地取出指令、分析指令、执行指令的过程。**



（1）预先把指挥计算机如何进行操作的指令序列（就是程序）和原始数据输入到计算机内存中，每条指令中明确规定了计算机从哪个地址取数，进行什么操作，然后送到什么地方去等步骤。

（2）计算机在执行时，先从内存中取出第一条指令，通过控制器的译码器接收指令的要求，再从存储器中取出数据进行指定的运算和逻辑操作等，然后再按地址把结果送到内存中，如果需要向硬盘等存储设备存储数据，还需要将内存中的该数据存储到硬盘中。接下来取出第2条指令，在控制器的指挥下完成规定操作，依次进行下去，直到遇到停止指令。

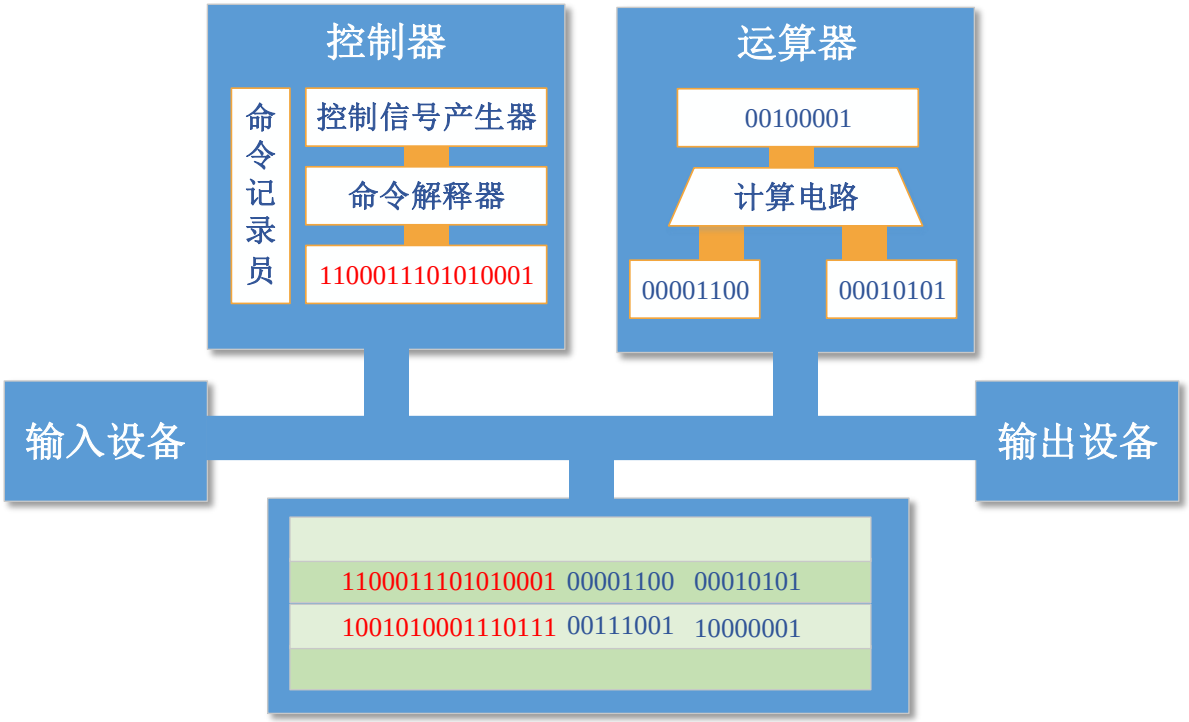
（3）计算机中基本上有两股信息在流动，一种是**数据**，即各种原始数据、中间结果和程序等，另一种信息是**控制信息**，它控制机器的各种部件执行指令规定的各种操作。



（1）通过命令记录员找到当前执行到的命令，并将命令提取出来放到命令控制器中的指令暂存处

（2）接着控制器中的命令解释器对命令进行解释，并产生相应的控制信号

（3）在控制器的控制下，将两个数再从存储器中提取出来，分别放到运算器的两个数据缓存区中



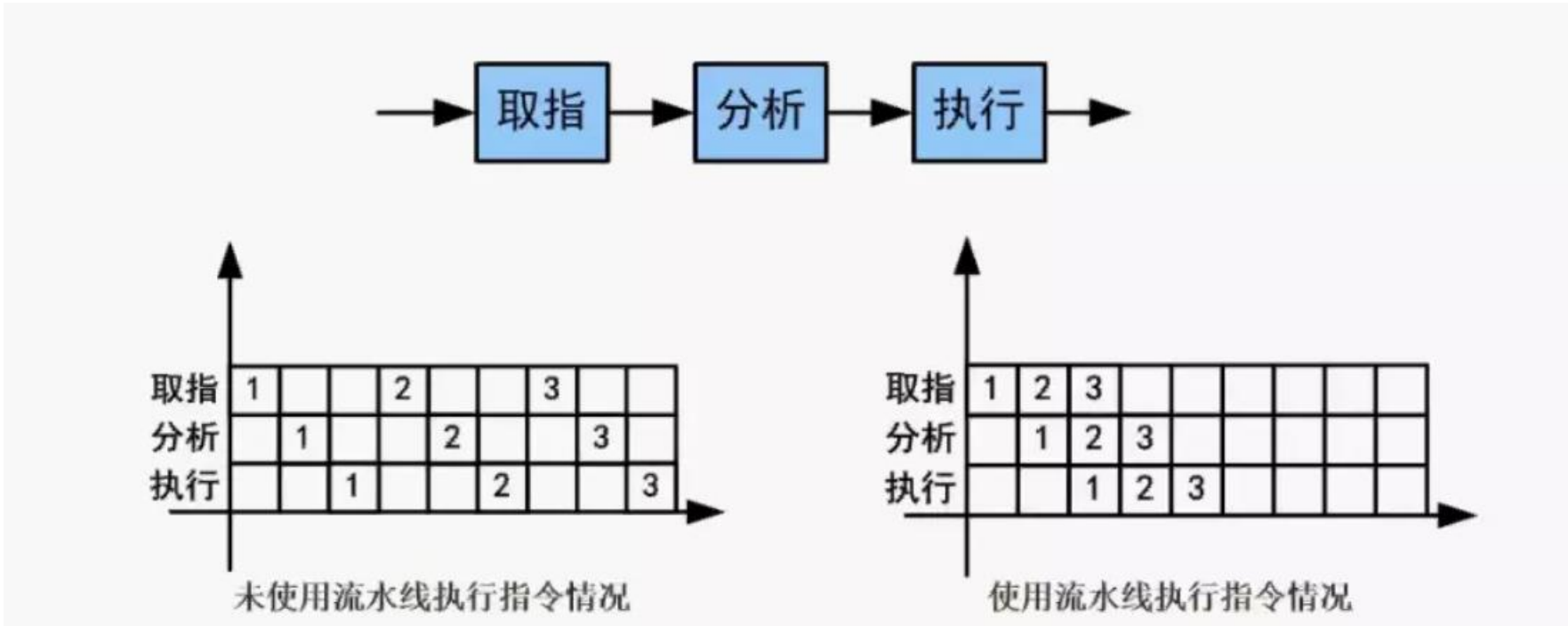
（4）接着控制器产生一个控制信号告诉计算电路做这两个数的加法，相加得到运算结果.

（5）把运算结果运算结果写回存储器指定位置。

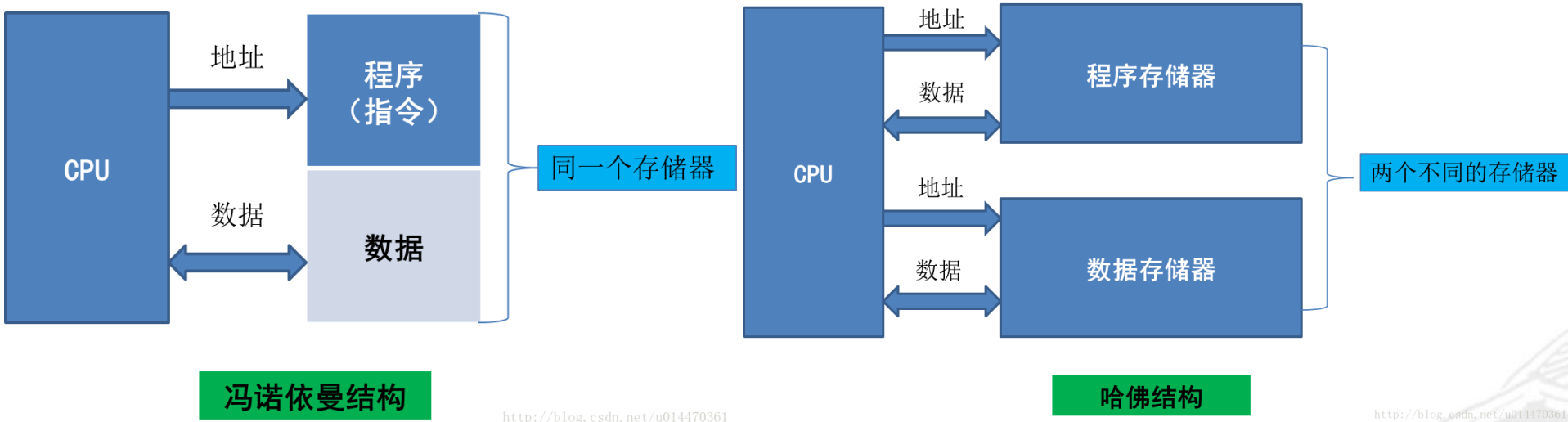
（6）完这条命令后，转到下一条命令继续执行.

➢ 流水线技术

流水线是指在程序执行时**多条指令重叠**进行操作的一种准并行处理实现技术。



➢ 冯诺依曼结构、哈佛结构、改进型哈佛结构



不能同时取指令和取操作数

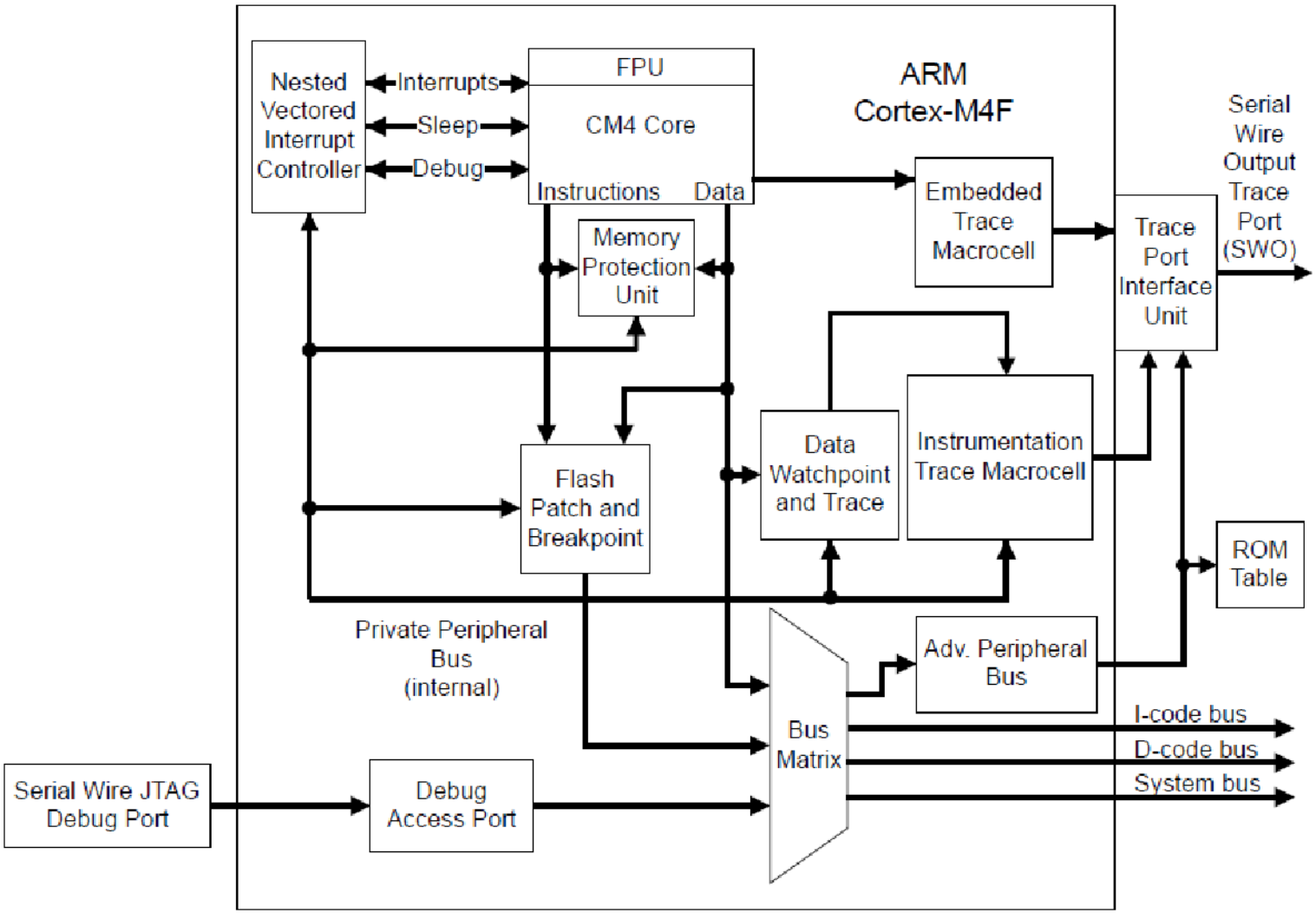
程序指令和数据的**宽度相同**

将程序指令存储和数据**存储分开**,具有较高的执行效率

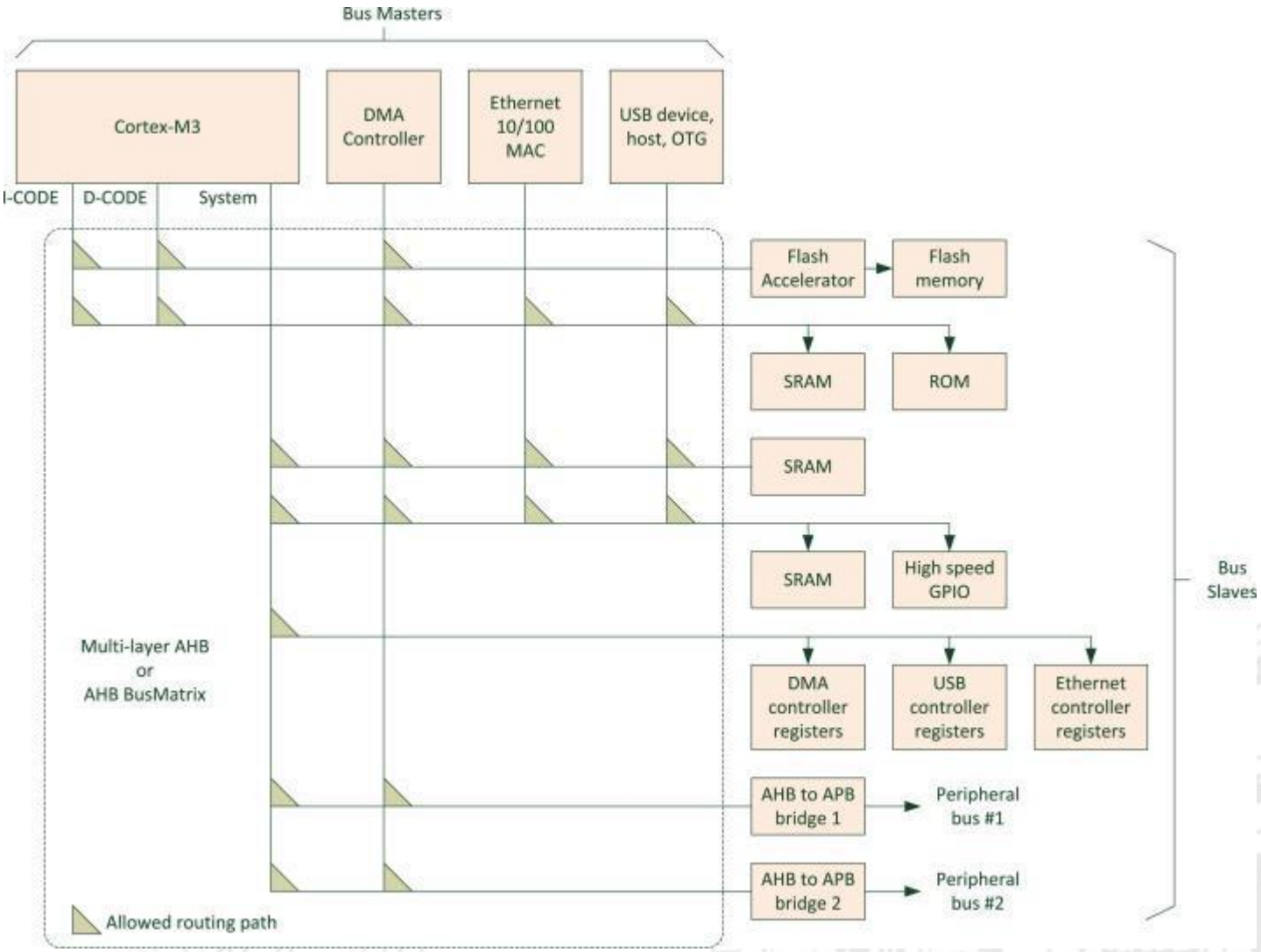
数据和指令的储存可以**同时进行**，可以使指令和数据有不同的数据宽度

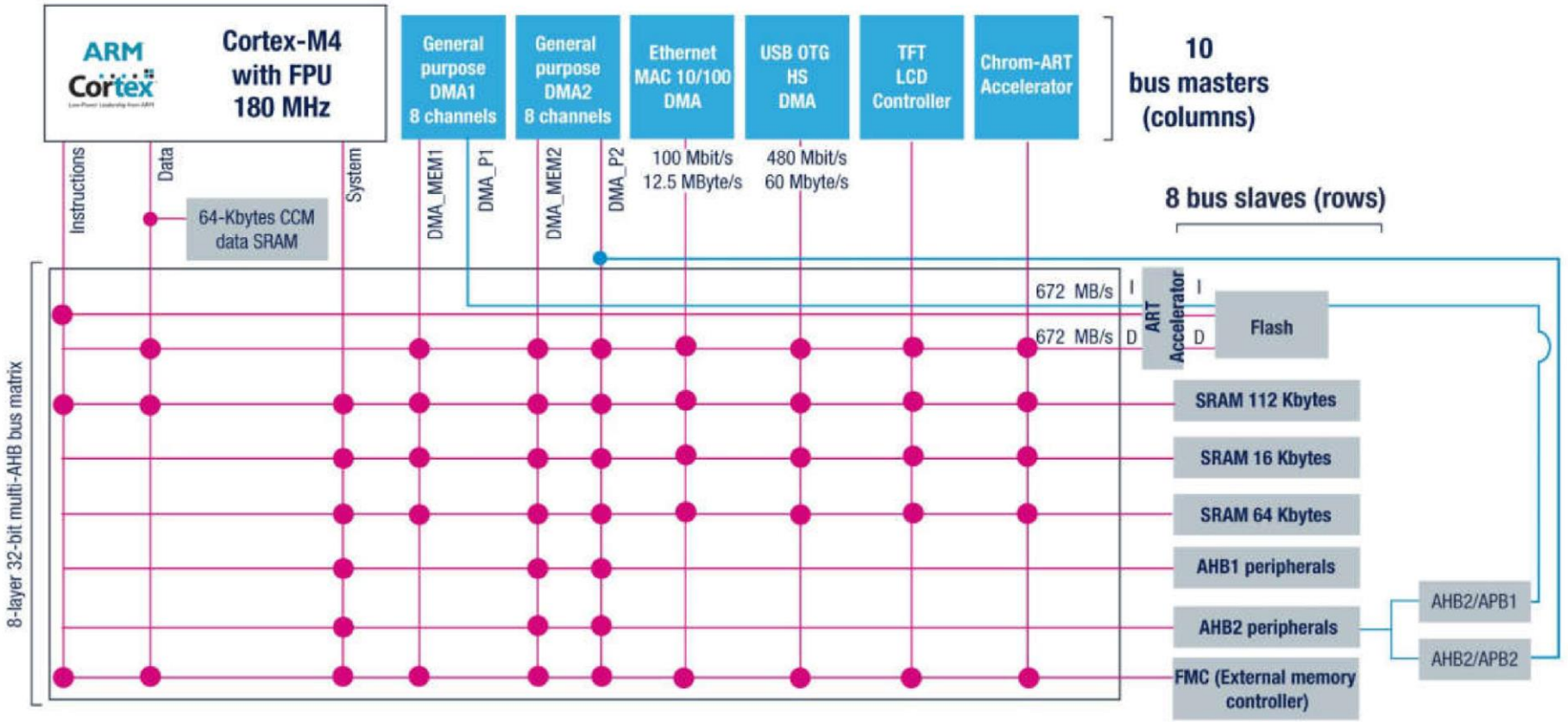
总线过多，复杂，成本高

➢ ARM Cortex-M4F具有多条总线

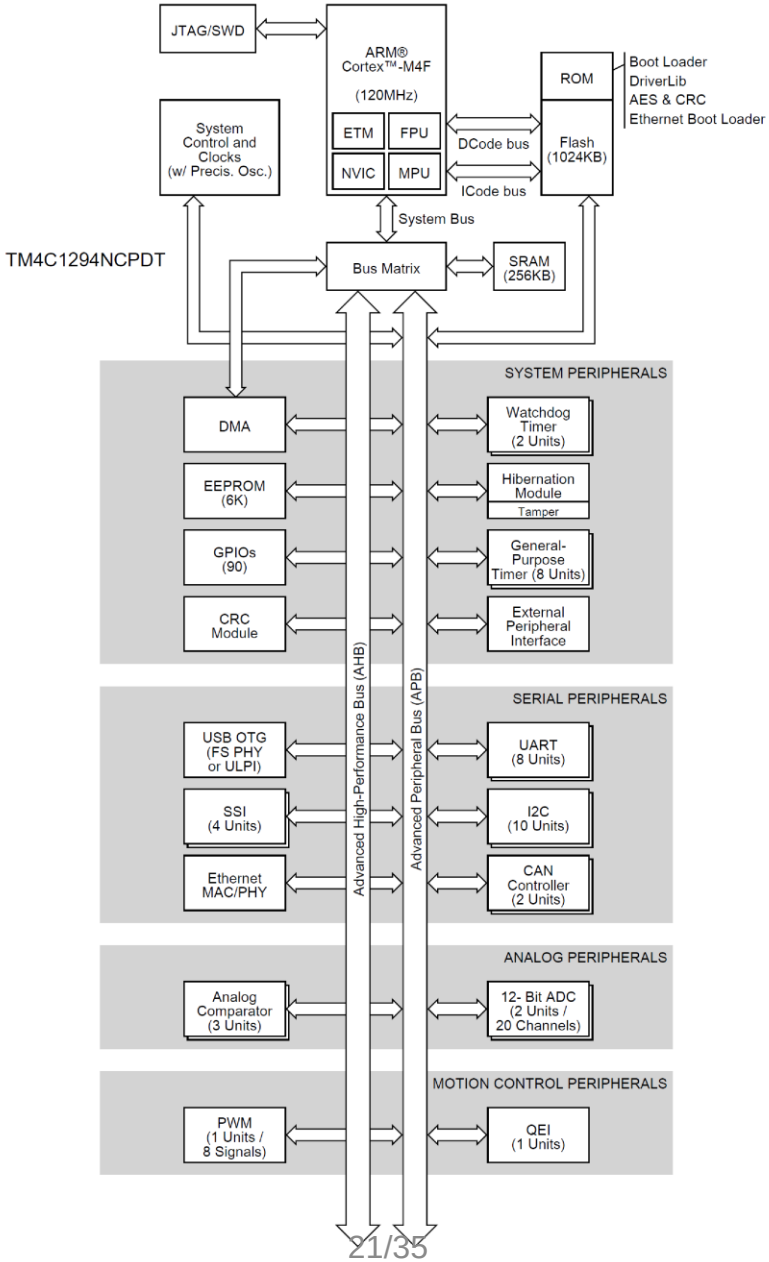


➢ ARM Cortex处理器的总线矩阵




ARM
Cortex
Cortex-M4
with FPU
180 MHz
General purpose DMA1 8 channels
General purpose DMA2 8 channels
Ethernet MAC 10/100 DMA
USB OTG HS DMA
TFT LCD Controller
Chrom-ART Accelerator
10 bus masters (columns)
Instructions
Data
System
DMA_MEM1
DMA_P1
DMA_MEM2
DMA_P2
100 Mbit/s
12.5 MByte/s
480 Mbit/s
60 Mbyte/s
8 bus slaves (rows)
64-Kbytes CCM data SRAM
672 MB/s
672 MB/s
I
D
ART Accelerator
Flash
SRAM 112 Kbytes
SRAM 16 Kbytes
SRAM 64 Kbytes
AHB1 peripherals
AHB2 peripherals
FMC (External memory controller)
AHB2/APB1
AHB2/APB2
8-layer 32-bit multi-AHB bus matrix

微型计算机的组成


JTAG/SWD
ARM®
Cortex™-M4F
(120MHz)
ETM
FPU
NVIC
MPU
ROM
Boot Loader
DriverLib
AES & CRC
Ethernet Boot Loader
Flash
(1024KB)
DCode bus
ICode bus
System
Control and
Clocks
(w/ Precis. Osc.)
System Bus
TM4C1294NCPDT
Bus Matrix
SRAM
(256KB)
SYSTEM PERIPHERALS
DMA
EEPROM
(6K)
GPIOs
(90)
CRC
Module
Watchdog
Timer
(2 Units)
Hibernation
Module
Tamper
General-
Purpose
Timer (8 Units)
External
Peripheral
Interface
SERIAL PERIPHERALS
USB OTG
(FS PHY
or ULPI)
SSI
(4 Units)
Ethernet
MAC/PHY
UART
(8 Units)
I2C
(10 Units)
CAN
Controller
(2 Units)
Advanced High-Performance Bus (AHB)
Advanced Peripheral Bus (APB)
ANALOG PERIPHERALS
Analog
Comparator
(3 Units)
12- Bit ADC
(2 Units /
20 Channels)
MOTION CONTROL PERIPHERALS
PWM
(1 Units /
8 Signals)
QEI
(1 Units)

➢ TM4C1294NCPDT微控制器系统的硬件组成：

- **System Timer (SysTick) 系统时钟**
- **Nested Vectored Interrupt Controller (NVIC) 嵌套向量中断控制器**
- **System Control Block (SCB)系统控制模块**
- **Memory Protection Unit (MPU)内存保护单元**
- **Floating-Point Unit (FPU)浮点运算单元**
- **On-Chip Memory片上存储器**
  - 256 KB single-cycle SRAM
  - 1024 KB Flash memory
  - 6KB EEPROM
  - Internal ROM
- **External Peripheral Interface外设接口**
- **Cyclical Redundancy Check (CRC)循环冗余校验**

– **Serial Communications Peripherals**
串行通信外设
  - **Ethernet MAC and PHY以太网**
  - **Controller Area Network (CAN)**
  - **Universal Serial Bus (USB)**
  - **UART异步串行通信**
  - **I2C**
  - **QSSI同步串行通信**

- Direct Memory Access直接存储器控制
- System Control and Clocks系统控制与时钟
- Programmable Timers可编程定时器
- Capture Compare PWM pins (CCP)捕获比较强
- Hibernation Module (HIB)休眠模块
- Watchdog Timers看门狗定时器
- Programmable GPIOs可编程输入输出模块
- Advanced Motion Control先进运动控制
  - PWM脉冲宽度调制
  - QEI正交编码器接口
- Analog模拟器件
  - ADC模拟数字转换器
  - Analog Comparators模拟比较强
- JTAG and ARM Serial Wire Debug调试接口

```
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
```

包含头文件

```
void main(void)
{
    // Enable the GPIO module.
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    ROM_SysCtlDelay(1);

    // Configure PA1 as an output.
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_1);
    // Loop forever.
    //
    while(1)
    {
        // Set the GPIO high.
        ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_1, GPIO_PIN_1);
        // Delay for a while.
        ROM_SysCtlDelay(1000000);
        // Set the GPIO low.
        ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_1, 0);
        // Delay for a while.
        ROM_SysCtlDelay(1000000);
    }
}
```

由main函数开始执行
包含一个while无限循环

C语言的语法回顾

关键字：

1). 数据类型(常用char, short, int, long, unsigned, float, double)

2). 运算和表达式(=, +, -, *, while, do-while, if, goto, switch-case)

3). 数据存储(auto, static, extern, const, register, volatile, restricted),

4). 结构(struct, enum, union,typedef),

5). 位操作和逻辑运算(<<, >>, &, |, ~, ^, &&),

6). 预处理(#define, #include, #error, #if...#elif...#else...#endif等),

7). 平台扩展关键字(__asm, __inline, __syscall)

C语言的语法回顾

数据类型：

C语言提供的typedef就是用于处理这种情况的关键字，在大部分支持跨平台的软件项目中被采用，典型的如下:

```
typedef unsigned char uint8_t;
typedef unsigned short uint16_t;
typedef unsigned int uint32_t;
......
typedef signed int int32_t;
```

➢ C语言的语法回顾

内存管理和存储架构：

C语言允许程序变量在定义时就确定内存地址，通过作用域，以及关键字extern，static，实现了精细的处理机制，按照在硬件的区域不同，内存分配有三种方式：

1). **从静态存储区域分配**。内存在程序**编译**的时候就已经分配好，这块内存在程序的**整个运行期间**都存在。例如**全局变量**，**static 变量**。

2). **在栈上创建**。在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元**自动被释放**。栈内存分配运算内置于处理器的指令集中，**效率很高**，但是分配的内存容量有限。

3). **从堆上分配**，亦称动态内存分配。程序在运行的时候用 malloc 或 new 申请任意多少的内存，程序员自己负责在何时用 free 或 delete 释放内存。动态内存的生存期由程序员决定，使用非常**灵活**，但同时遇到**问题也最多**。

➢ C语言的语法回顾

内存管理和存储架构：

```c
#include <stdio.h>
#include <stdlib.h>

static int st_val;                      //静态全局变量 -- 静态存储区
int ex_val;                             //全局变量 -- 静态存储区
int main(void)
{
   int a = 0;                           //局部变量 -- 栈上申请
   int *ptr = NULL;                     //指针变量
   static int local_st_val = 0;         //静态变量
   local_st_val += 1;
   a = local_st_val;
   ptr = (int *)malloc(sizeof(int)); //从堆上申请空间
   if(ptr != NULL)
   {
      printf("*p value:%d", *ptr);
      free(ptr);
      ptr = NULL;
      //free后需要将ptr置空，否则会导致后续ptr的校验失效，出现野指针
   }
}
```

➢ C语言的语法回顾

内存管理和存储架构：

- **数组**是由相同类型元素构成，当它被声明时，编译器就根据内部元素的特性在内存中分配一段空间
- C语言也提供多维数组，
- 指针则是提供使用地址的符号方法，C语言的指针具有最大的灵活性，在被访问前，可以指向任何地址，这大大方便了对硬件的操作，但同时也对开发者有了更高的要求。

不同类型指针加法的运算结果

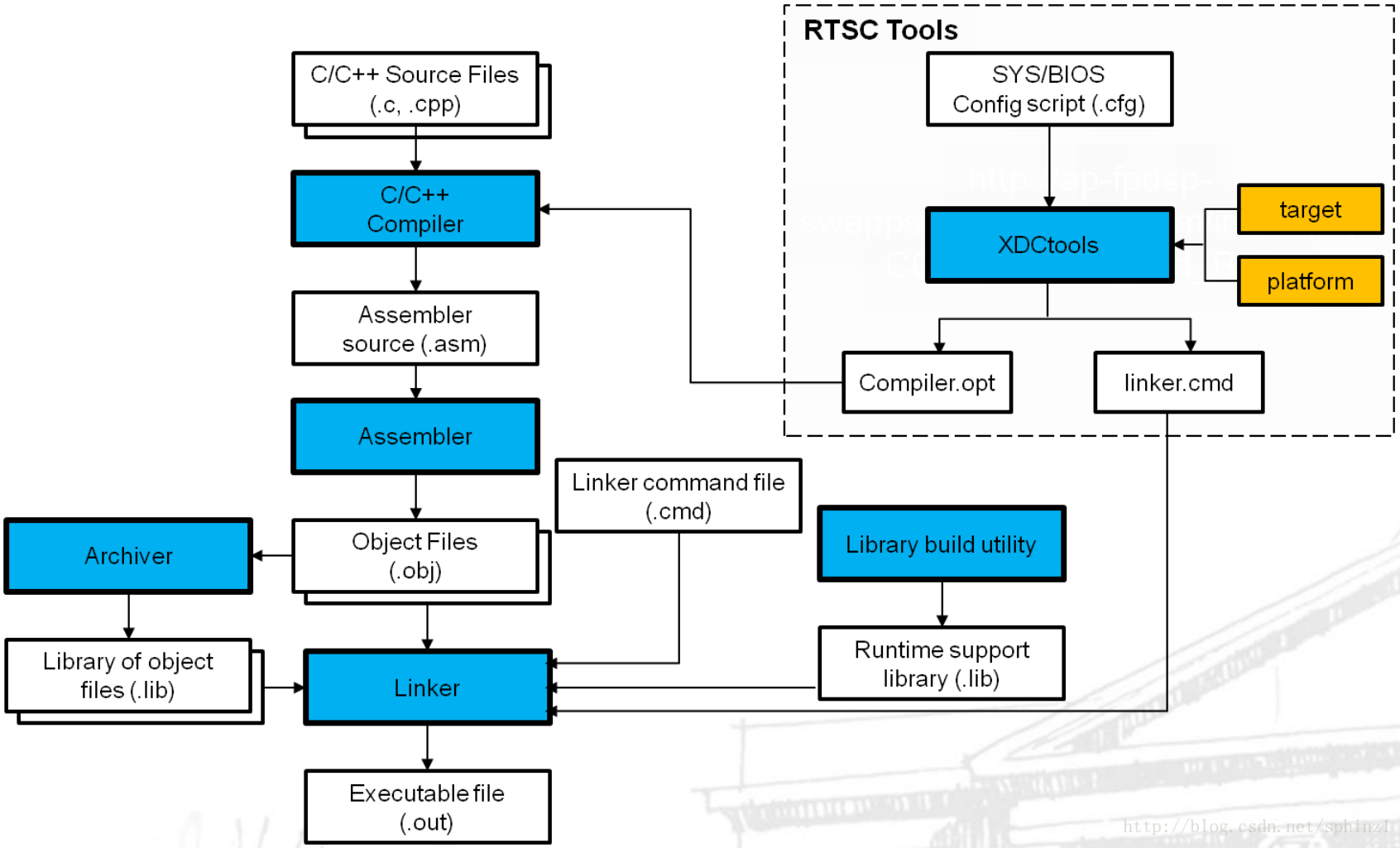
```
int main(void)
{
  char cval[] = "hello";
  int i;
  int ival[] = {1, 2, 3, 4};
  int arr_val[][2] = {{1, 2}, {3, 4}};
  const char *pconst = "hello";
  char *p;
  int *pi;
  int *pa;
  int **par;

  p = cval;
  p++;              //addr增加1
  pi = ival;
  pi+=1;            //addr增加4
  pa = arr_val[0];
  pa+=1;            //addr增加4
  par = arr_val;
  par++;           //addr增加8
  for(i=0; i<sizeof(cval); i++)
  {
```

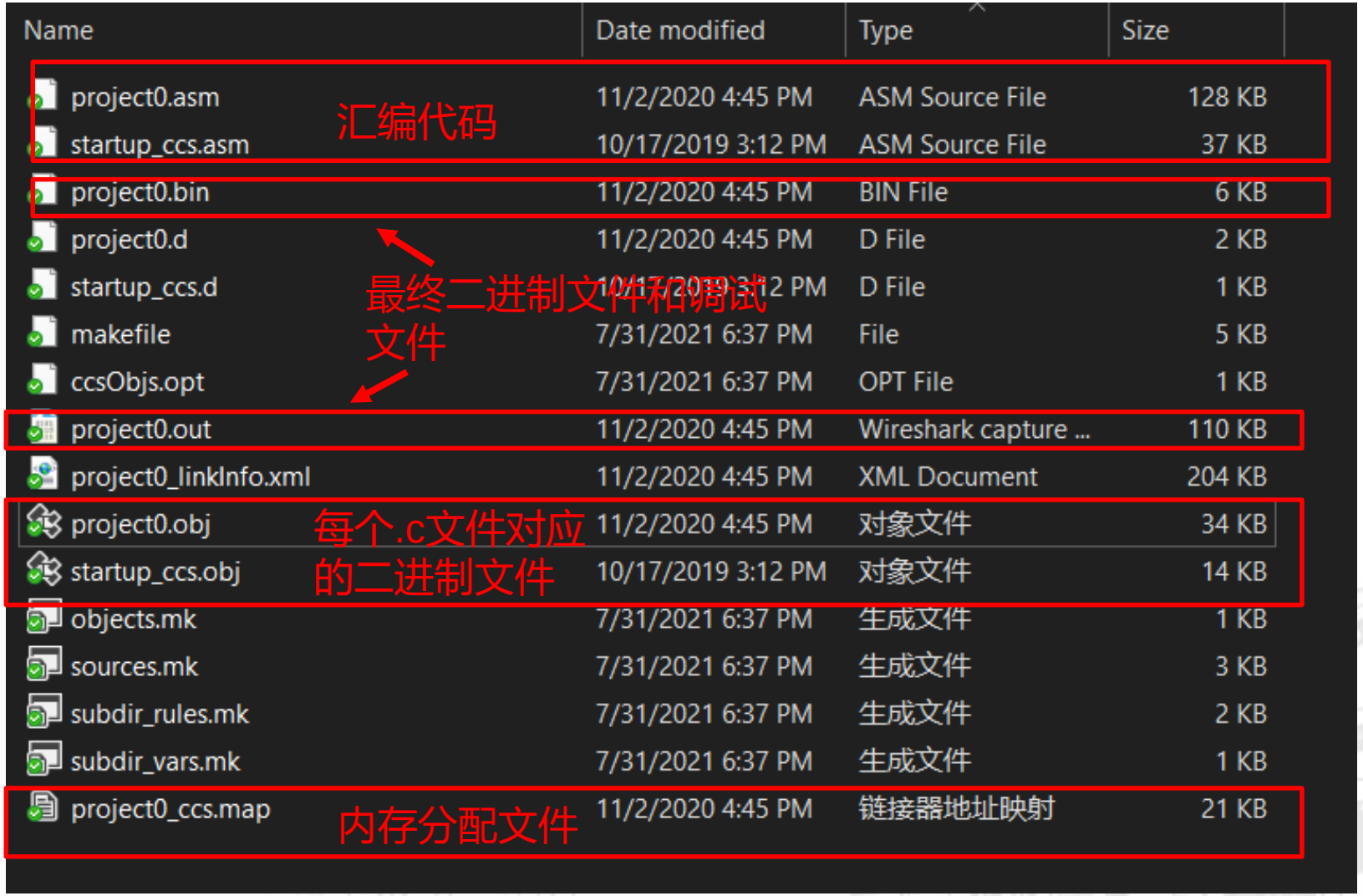
C语言编译过程

- **预处理(Preprocessing)**，即完成宏定义和include 文件展开等工作；
- **编译(Compilation)**，根据**编译**参数进行不同程序的优化，**编译成汇编代码**；
- **汇编(Assemble)**，用**汇编器**把上一阶段生成的汇编代码进一步生成**目标代码**；
- **链接(Linking)**，用**链接器**把上一阶段生成的目标代码、其他一些相关的系统提供的目标代码（如crtx.o）和系统或用户提供的**库**链接起来，生成最终的执行代码。生成**可执行文件**。

➢ CCS的编译过程

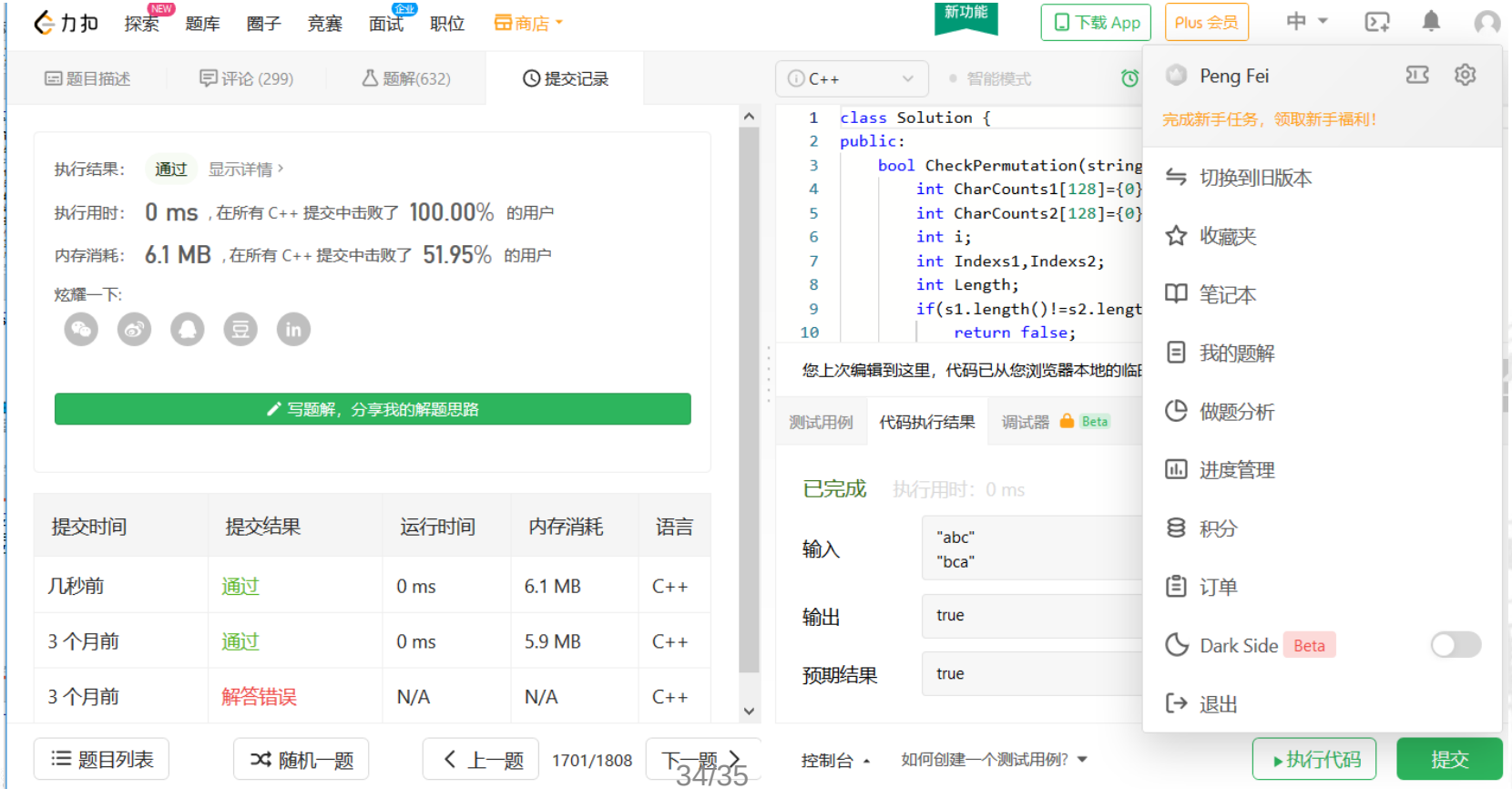

RTSC Tools
SYS/BIOS
Config script (.cfg)
target
XDCtools
platform
Compiler.opt
linker.cmd
C/C++ Source Files
(.c, .cpp)
C/C++
Compiler
Assembler
source (.asm)
Assembler
Linker command file
(.cmd)
Archiver
Object Files
(.obj)
Library build utility
Library of object
files (.lib)
Linker
Runtime support
library (.lib)
Executable file
(.out)

编译过程生成的文件


Name
Date modified
Type
Size
project0.asm
11/2/2020 4:45 PM
ASM Source File
128 KB
startup_ccs.asm
10/17/2019 3:12 PM
ASM Source File
37 KB
汇编代码
project0.bin
11/2/2020 4:45 PM
BIN File
6 KB
project0.d
11/2/2020 4:45 PM
D File
2 KB
startup_ccs.d
D File
1 KB
最终二进制文件和调试文件
makefile
7/31/2021 6:37 PM
File
5 KB
ccsObjs.opt
7/31/2021 6:37 PM
OPT File
1 KB
project0.out
11/2/2020 4:45 PM
Wireshark capture ...
110 KB
project0_linkInfo.xml
11/2/2020 4:45 PM
XML Document
204 KB
project0.obj
11/2/2020 4:45 PM
对象文件
34 KB
每个.c文件对应的二进制文件
startup_ccs.obj
10/17/2019 3:12 PM
对象文件
14 KB
objects.mk
7/31/2021 6:37 PM
生成文件
1 KB
sources.mk
7/31/2021 6:37 PM
生成文件
3 KB
subdir_rules.mk
7/31/2021 6:37 PM
生成文件
2 KB
subdir_vars.mk
7/31/2021 6:37 PM
生成文件
1 KB
project0_ccs.map
11/2/2020 4:45 PM
链接器地址映射
21 KB
内存分配文件

作业

- 1. 在力扣或者牛客网做三道简单C语言算法题
- 2. 要求在自己电脑IDE上做，调试完成后再提交到力扣上检验，不可直接用力扣的在线IDE做。
- 3. 提交作业需包括本地IDE代码、结果截图，力扣运行截图。



谢谢！