汇编语言与逆向技术

第2章 IA32处理器结构

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院

2024-2025学年

本章知识点

- 计算机体系结构
  - 重点：寄存器
- IA32位处理器体系结构
  - 重点：保护模式、EFLAGS寄存器
- IA32的内存管理
  - 难点：段模式、页模式
- 实验：“Hello World”

主观题 10分

设置

大家都知道有哪些品牌或者类型的处理器?

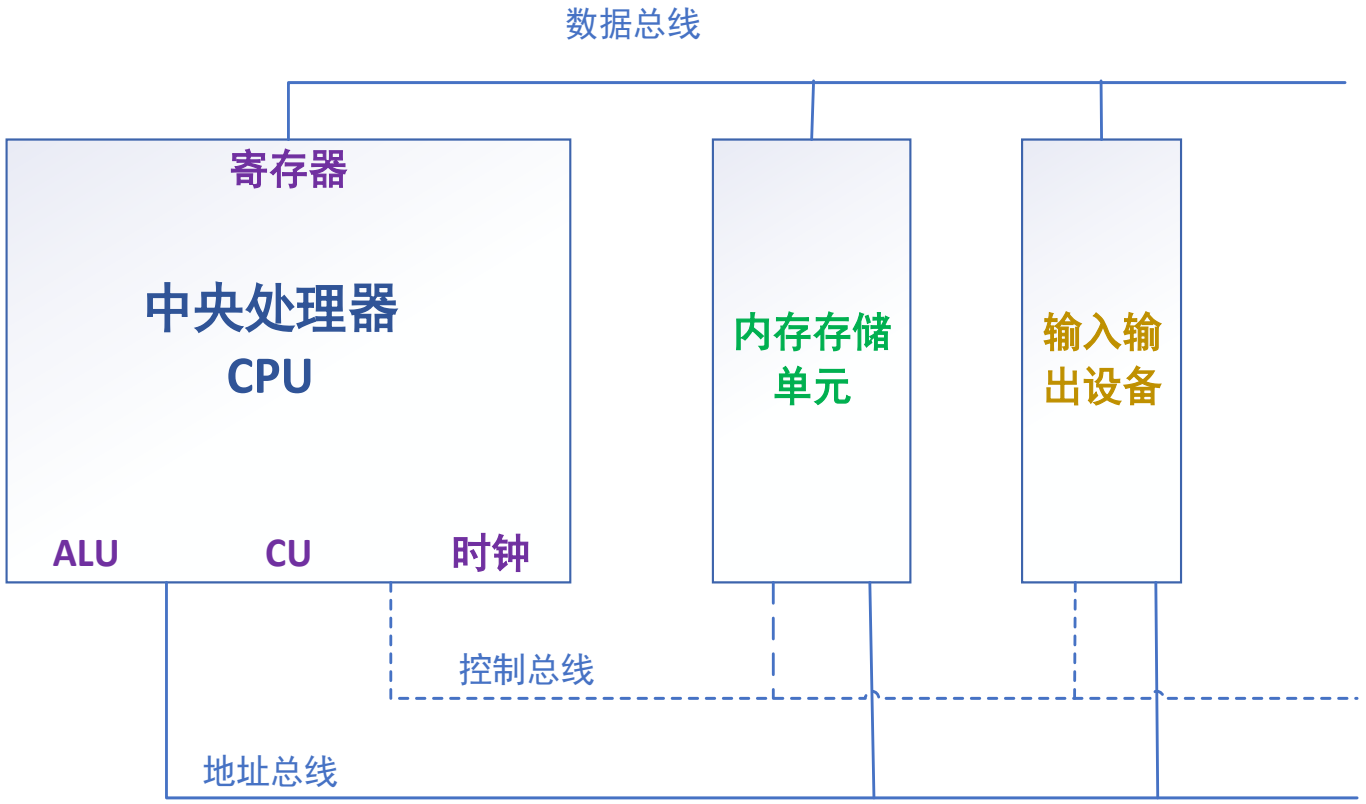
作答

计算机体系结构

计算机基本概念

- 计算机基本结构
- 指令执行周期
- 内存的读取
- 程序是如何运行的

计算机基本结构



计算机基本结构

- 中央处理器（CPU，Central Processor Unit)进行计算和逻辑操作的地方
  - 寄存器（Register）
  - 时钟（clock）
  - 控制单元（CU，Control Unit）
  - 算数逻辑单元（ALU，Arithmetic Logic Unit）

CPU

- 寄存器：数据存储，数量有限
- 时钟：同步CPU的内部操作
- 控制单元：控制机器指令的执行步骤
- 算数逻辑单元：算术运算、逻辑运算

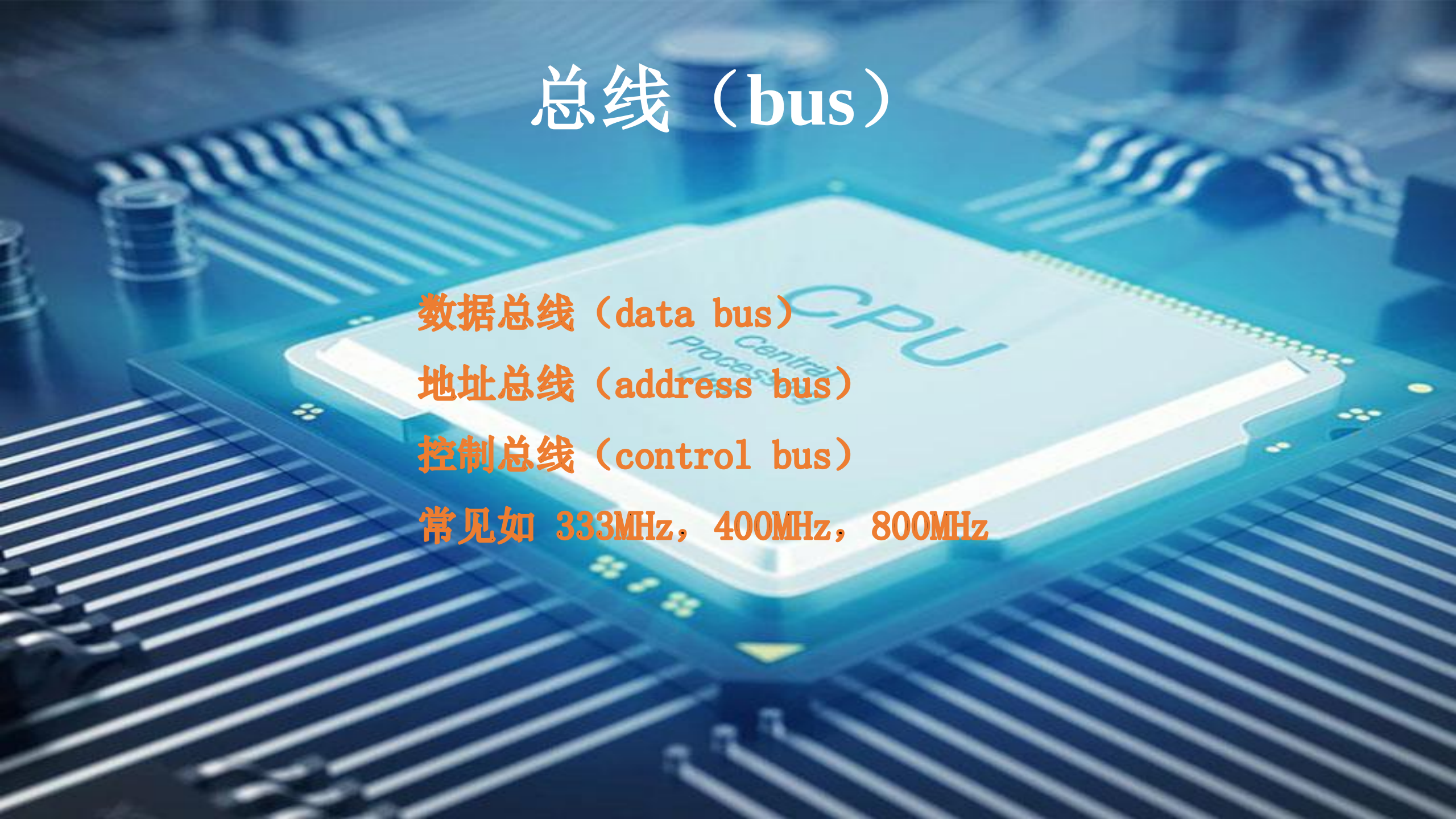
CPU时钟

- 每个时钟周期CPU完成一步操作
- 时钟频率=1/时钟周期
- 时钟频率反映了CPU速度的快慢
- 当前流行的CPU主频3GHz
- 可通过软件优化提升主频利用率，实现超频

内存存储单元

- Memory storage unit存放指令和数据的地方
- 核心频率133MHz~200MHz （存储单元频率、刷新频率）
- DDR2-800、DDR3-1600、DDR4-2400等，并不是内存的真正频率，而是业界约定俗成的等效频率（数据传送频率）
- 是硬盘的20-30倍

总线（bus）



数据总线（data bus）

地址总线（address bus）

控制总线（control bus）

常见如 333MHz，400MHz，800MHz

总线

- 地址总线的宽度决定了CPU的寻址能力；
- 数据总线的宽度决定了CPU与其它器件进行数据传送时的一次数据传送量；
- 控制总线宽度决定了CPU对系统中其它器件的控制能力。

指令执行周期

- 单条机器指令的执行包括一系列操作
  - 取指令：指令指针IP
  - 解码：控制单元CU确定执行什么操作
  - 取操作数：从内存读操作数
  - 执行：算数逻辑单元ALU
  - 存储输出操作数：向内存写入

多选题 10分

设置

CPU内部包括哪些基本部件?

A ALU

B CU

C Register

D Clock

提交

南開大學 Nankai University

单选题 10分

设置

下面哪个设备的时钟频率最高?

A CPU

B 总线

C 内存

提交

单选题 10分

设置

数据在CPU中存储在什么位置?

A 寄存器Register

B 时钟Clock

C 控制单元（CU）

D 算数逻辑单元（ALU）

提交

主观题 10分

设置

一条CPU指令的执行周期包括哪些操作？ 其中占用时间最多的操作有哪些?

作答

IA-32处理器体系结构

IA-32处理器体系结构

- IA-32（Intel Architecture 32-bit）英特尔32位体系结构
  - 1985年 80386 CPU首先使用
  - 32位内存地址、32位数据操作数

工作模式

- 实地址模式 （Real-Address Mode）
  - 16位，8086程序设计环境
- 保护模式（Protected Mode）
  - 32位，IA-32程序设计环境
  - 提供了虚拟8086模式：执行和兼容旧的8086程序

CPU模式	区分方式	所处时间	备注
实模式	从段寄存器中直接拿取段起始地址	系统刚启动时	此时是兼容16位的
保护模式	间接地先从段寄存器找到表格中的一项，再从表格中的一项中拿到段起始地址	需要更多内存时时	遵循一定的规则，进行一系列的操作地切换

实地址模式

- 16位的8086程序设计环境
  - 20条地址线、可存储空间1MB（2的20次方）
  - 16位寄存器如何兼容？
    - 使用一个段选择器（通常是一个段寄存器，如CS、DS、ES等）和一个偏移地址相结合来产生一个20位的物理地址
    - **物理地址（20位） = 16位段地址 << 4（即*16）+ 16位偏移地址**

保护模式

- IA-32 CPU的存储管理和保护机制
  - 多任务操作系统
  - 程序有独立的4GB内存存储空间（2的32次方）

地址空间

- IA-32 CPU 4GB 地址空间
  - 32位的寻址上限
- 8086只有1MB地址空间

单选题 10分

设置

哪种操作模式，程序可以拥有4GB的地址空间?

A 保护模式

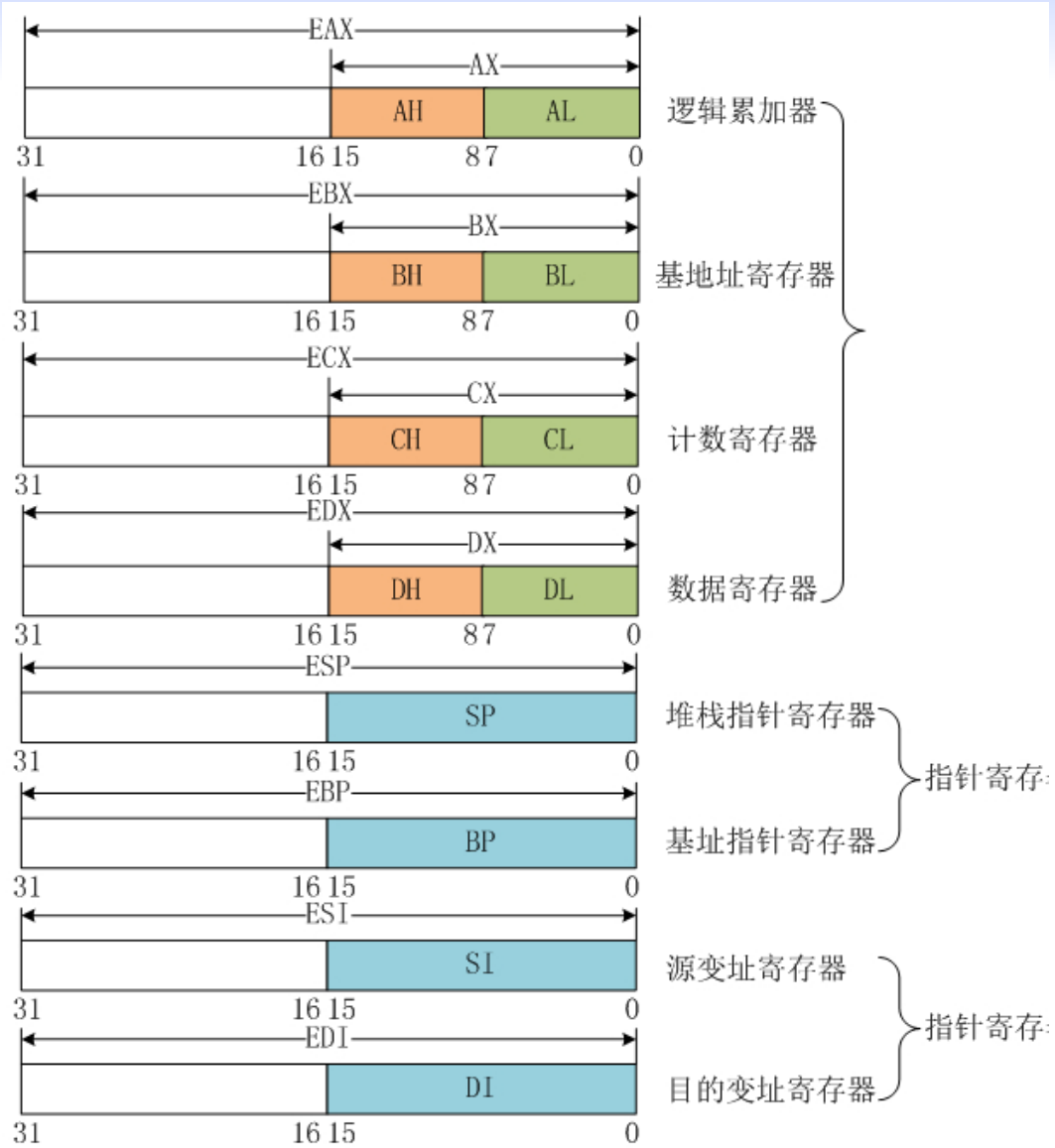
B 实地址模式

C 虚拟8086模式

提交

寄存器（Register）

- 寄存器是CPU内部的高速存储单元
  - 比内存的访问速度快很多
  - 优化循环结构执行速度，把循环计数变量放到寄存器中。



通用寄存器

8个32位通用寄存器

EAX

EBX

ECX

EDX

ESP

EBP

ESI

EDI

段寄存器

- CS：Code Segement，代码段寄存器
- SS：Stack Segment，栈段寄存器
- DS：Data Segment，数据段寄存器
- ES：Extra(Data) Segment，数据段寄存器
- FS：Data Segment，数据段寄存器
- GS：Data Segment，数据段寄存器

```
00261DD7   8B55 E0          mov edx,dword ptr ss:[ebp-0x20]
00261DDA   8B02             mov eax,dword ptr ds:[edx]
00261DDC   8945 D8          mov dword ptr ss:[ebp-0x28],eax
```

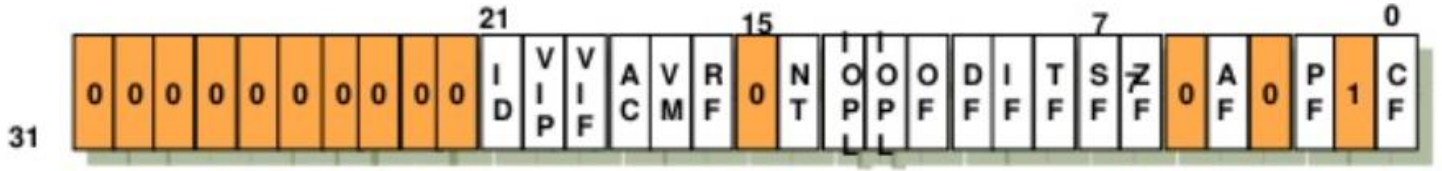
主观题 10分

设置

高级语言的if、for、while等条件判断过程在CPU中是如何实现的?

作答

EFLAGS寄存器



ID	X	ID Flag (CPUID support)	DF	C	Direction Flag
VIP	X	Virtual Interrupt Pending	IF	X	Interrupt Enable Flag
VIF	X	Virtual Interrupt Flag	TF	X	Trap Flag
AC	X	Alignment Check	SF	S	Sign Flag
VM	X	Virtual 8086 Mode	ZF	S	Zero Flag
RF	X	Resume Flag	AF	S	Auxiliary Carry Flag
NT	X	Nested Task	PF	S	Parity Flag
IOPL	X	I/O Privilege Level	CF	S	Carry Flag
OF	S	Overflow Flag			

Bit Positions shown as "0" or "1" are Intel reserved.

S = Status Flag
C = Control Flag
X = System Flag

零标志

- 零标志（ZF）：若算数或者逻辑运算结果为0则将其置1，反之清零

```
xor eax, eax
jz    //当z为1时则跳转（与je功能上相同）
```

进位标志

- 进位标志（CF）：在无符号算术运算的结果最高有效位(most-significant bit)发生进位或借位则将其置1，反之清零。

  add eax，0xffffffff

  jc

溢出标志

- 溢出标志（OF）：在有符号算术运算的结果是较大的正数或较小的负数，并且目的操作数无法容纳时，将该位置1，反之清零。
- 常用于为带符号整型运算指示溢出状态。

```
MOV AL, 100      ; AL = 100
ADD AL, 50       ; AL = AL + 50
```

符号标志

- 符号标志（SF）：该标志被设置为有符号整型的最高有效位。
- 0表示算术或者逻辑运算结果为正
- 1表示算数或者逻辑运算结果为负

奇偶标志

- 奇偶标志（PF）：如果结果的最低有效字节(least-significant byte)包含偶数个1位则该位置1，否则清零。
- 常用于数据校验

辅助进位标志

- 辅助进位标志（AC）：如果算术操作在结果的第3位发生进位或借位则将该标志置1，否则清零。
- 这个标志在BCD(binary-code decimal)算术运算中被使用。

控制标志

- 方向标志（DF）
  - 控制串指令(MOVS, CMPS, SCAS, LODS以及STOS)
  - 设置DF标志使得串指令自动递减（从高地址向低地址方向处理字符串），清除该标志则使得串指令自动递增
  - STD以及CLD指令分别用于设置以及清除DF标志。

系统标志

- TF：将该位设置为1以允许单步调试模式，清零则禁用该模式。
- 调试器的单步调试功能

单选题 10分

设置

两个无符号整数A、B，A减B之后的CPU状态寄存器的CF和ZF标志位结果如下，哪种状态表示A大于B？

A CF=0，ZF=0

B CF=1，ZF=0

C CF=0，ZF=1

D CF=1，ZF=1

提交

南開大學 Nankai University

指令指针

- 指令指针寄存器（EIP）存放下一条机器指令的内存地址。
- 跳转指令可以修改指令指针寄存器

单选题 10分

设置

哪个寄存器存储了下一条要执行机器指令的内存地址？

A EAX

B EFLAGS

C EIP

D ESP

提交

南開大學 Nankai University

多选题 10分

设置

下列哪些是通用寄存器？

A EAX

B EFLAGS

C EIP

D ESP

E FS

提交

南開大學 Nankai University

IA32内存管理

主观题 10分

设置

如果一台电脑装有32GB的内存，是否最多只可以运行8个程序?

作答

IA-32内存管理

- IA32保护模式的内存管理比实地址模式要复杂
  - 多任务
  - 多用户
  - 段模式、页模式
  - 页模式也是基于段模式的，通常称为段页式

平坦模式（FLAT）

- 每个程序有独立的4GB虚拟地址空间
  - 数据
  - 指令
  - 数据 ←→ 指令
- 虚拟地址到物理地址的转换是透明的

段管理

- 一般保护模式的程序有3个段
  - 代码段，CS
  - 数据段，DS
  - 堆栈段，SS
- 段是一块连续的内存空间
- 段基址加偏移的寻址方式 Segment：Offset

段管理

- GDT（Global Descriptor Table）全局描述符表
  - 整个系统只有一个GDT（64bit）
  - 寄存器GDTR存放GDT的入口地址
- LDT（Local Descriptor Table）局部描述符表
  - 每个任务可以有自己的LDT
  - 寄存器LDTR 存储LDT的入口地址
  - 因为在任何时刻只能有一个任务在运行，所以LDTR也只需要有一个

Protected Mode Registers

GDTR

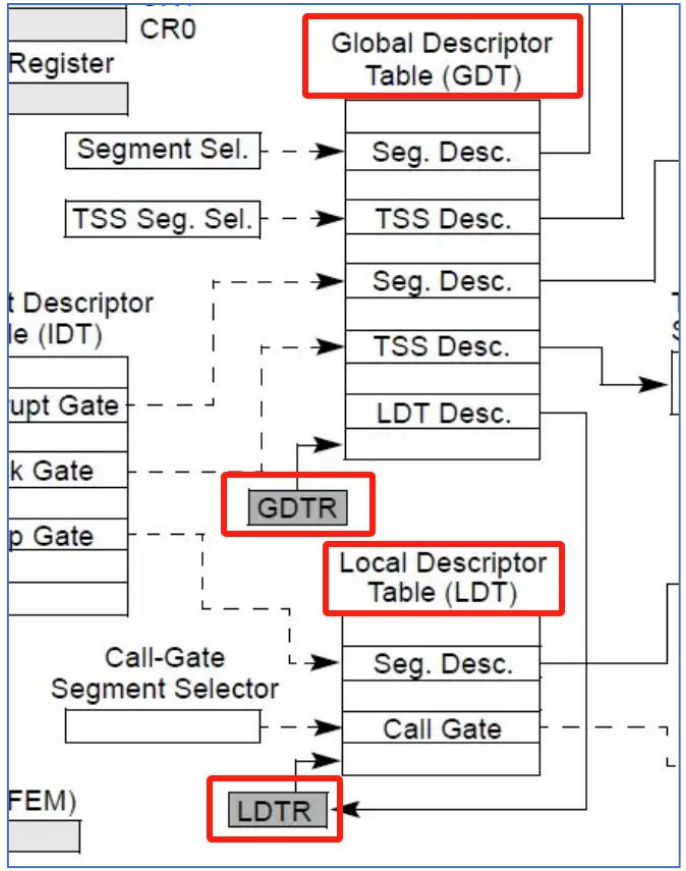
Bits	Label	Description
0-15	Limit	(Size of GDT) - 1
16-47	Base	Starting address of GDT

Stores the segment selector of the GDT.

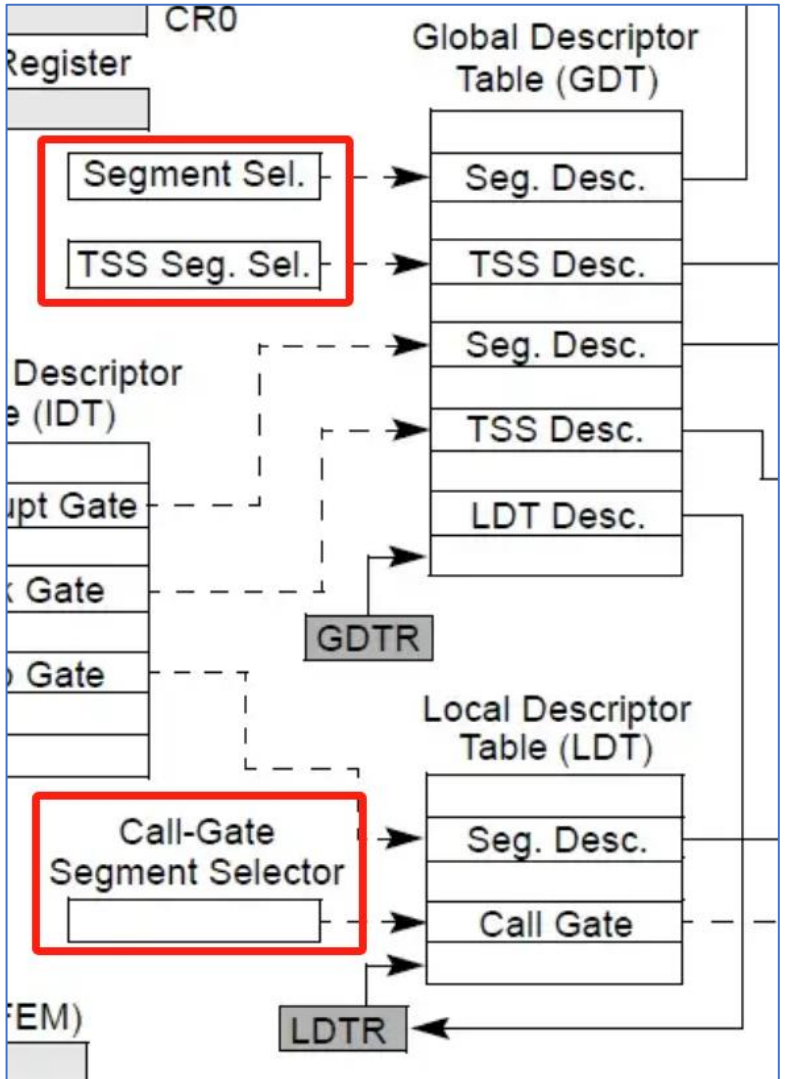
LDTR

Bits	Label	Description
0-15	Limit	(Size of LDT) - 1
16-47	Base	Starting address of LDT

Stores the segment selector of the LDT.



段选择子（Segment Selector）



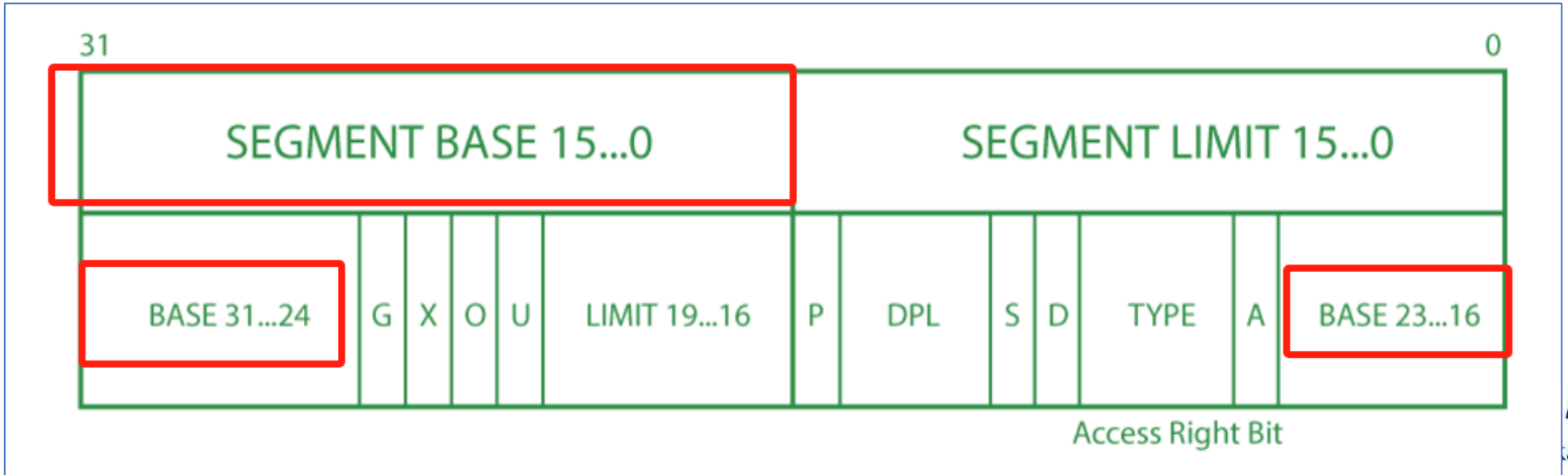
- 段选择子用于索引GDT和LDT中的表项
  - index（13个比特位），表索引，最大8192
  - TI（1个比特位），0是GDT，1是LDT
  - RPL（2个比特位），Request Privilege Level
    - 段的访问权限，包括0和3两个取值
    - Ring 0, Kernel Mode，Ring 3, User Mode

```
15                                                  3    2        0
+---------------------------------------------------+----+--------+
|          Index                                    | TI |   RPL  |
+---------------------------------------------------+----+--------+

TI = Table Indicator: 0 = GDT, 1 = LDT
```

段描述符

- 段描述符（64个比特位）
  - Segment Base：段的基地址，32个比特位
  - Segment Limit：段的长度，20个比特位
  - Segment Attribute：段的属性描述符，12个比特位



段模式的寻址过程

- Segment：Offset模式
  - 段寄存器（16比特位）里面存储的是一个段选择子（Segment Selector）
  - 段选择子从GDT/LDT中找到段描述符（Segment Descriptor）
  - 从段描述符中找到段基址（Base Address）
  - 段基址加上Offset得到物理内存地址

主观题 10分

设置

思考题：基于段的内存管理方式有哪些缺点?

作答

分页机制（虚拟内存）

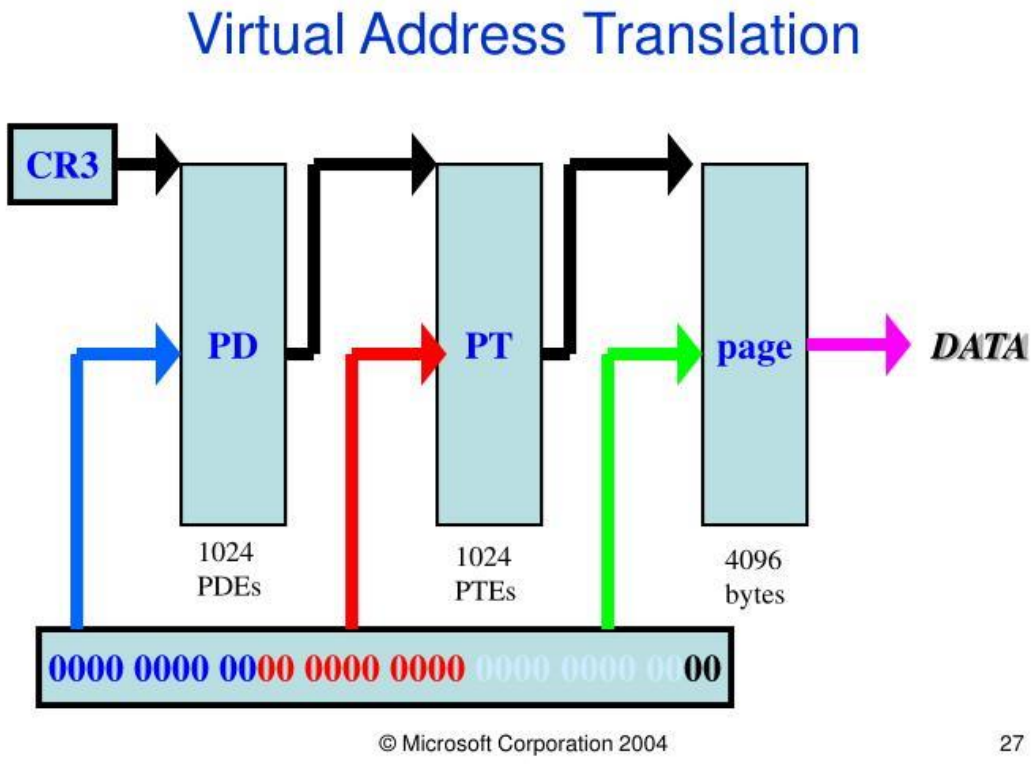
- 段又被分割成内存页（page）（ 4096字节）
- 实现了运行程序的总内存远大于计算机的物理内存
- 提高内存的利用率，减少内存碎片
- 页交换，不使用的内存页被交换到硬盘上
  - 虚拟内存空间大于实际的物理内存空间
  - 页交换降低程序执行速度

分页机制

- 分页机制实现虚拟地址到物理地址的转换
- 每个进程都有其独立的虚拟地址空间（最大为4GB，即2^32）
- 读取虚拟内存地址的数据需要访问内存3次
  - 页目录（Page Directory，PD）
  - 页表（Page Table，PT）
  - 页（Page）

分页机制

- 读取CPU的CR3寄存器，读取页目录的物理内存地址
- 以虚拟地址的前10个比特位作为索引，在页目录PD中找到页表的物理内存地址（PDE）。
- 以虚拟地址的中间10个比特位作为索引，在页表找到页的物理内存地址。
- 页的物理内存地址与虚拟地址最后12个比特位相加，得到物理地址。



单选题 10分

设置

通过虚拟地址读取一个内存数据，至少要访问多少次内存?

A 1

B 2

C 3

D 4

提交

南開大學 Nankai University

填空题 10分

设置

内存中可以有 [填空1] 个GDT， [填空2] 个LDT。

选项：A：1个，B：多个

作答

实验：“Hello World”

汇编、链接和运行程序

- 源文件：用文本编辑器编写的asm文本文件
- 汇编：汇编器把汇编源文件翻译成机器语言，生成目标文件(.obj)
- 链接：链接器从库中复制所需的过程，并将其同目标文件合并在一起生成可执行文件(.exe)

hello.asm

```
.386
.model flat, stdcall
option casemap :none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\masm32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\masm32.lib
```

hello.asm

```
.data
HelloWorld db "Hello World!", 0
.code
start:
invoke StdOut, addr HelloWorld
invoke ExitProcess, 0
end start
```

hello.asm

- .386
  - 允许汇编80386处理器的非特权指令，禁用其后处理器引入的汇编指令
- .model 初始化程序的内存模式
  - flat：平坦模式，4GB内存空间
  - stdcall：调用约定， stdcall是Win32 API函数的调用约定

hello.asm

- option casemap: none
  - 大小写敏感
- include …inc 函数的常量和声明
- includelib …lib 链接库

hello.asm

- .DATA
  - 定义已初始化数据段的开始
- .CODE
  - 定义代码段的开始
- start：，指令标号，标记指令地址

hello.asm

- StdOut, **masm32.inc**中定义的函数，将内存数据输出到命令行窗口上
- ExitProcess， **Kernel32.inc**中定义的函数，退出程序执行

hello.asm

- END start
  - 标记模块的结束
  - 指定程序的入口点

编译

- \masm\bin\ml /c /Zd /coff hello.asm
- **ml** 程序可以用来汇编并链接一个或多个汇编语言源文件
- ml的命令行选项是大小写敏感的

编译

- **/c** Assemble without linking
  - 只编译、不链接
- **/Zd** Add line number debug info
  - 在目标文件中生成行号信息
- **/coff** generate COFF format object file
  - 生成Microsoft公共目标文件格式（common object file format）的文件

链接

- \masm32\bin\link /SUBSYSTEM:CONSOLE hello.obj
- link.exe 链接器，将obj文件合并，生成可执行文件
- /SUBSYSTEM:CONSOLE，生成命令行程序

汇编语言与逆向技术

第2章 IA32处理器结构

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院

2024-2025学年