作业1.1　x86寄存器详解

一、通用寄存器[[[1]](#footnote-1)](General-Purpose Registers)

32位处理器在16位处理器的基础上，拓展(Extend)了8个通用寄存器(AX、BX、CX、DX、SI、DI、BP、SP)的长度，并将它们命名为EAX、EBX、ECX、EDX、ESI、EDI、ESP和EBP，同时支持8位和16位操作，用法和8086相同。其示意图如下：



虽然这些寄存器都叫通用寄存器，但它们每一个都有自己的特别之处[[[2]](#footnote-2)]：

1. 数据寄存器

数据寄存器主要用来保存操作数和运算结果等信息，从而节省读取操作数所需占用总线和访问存储器的时间。

* EAX是累加器(Accumulator Register)，它是很多加法乘法指令的缺省寄存器。
* EBX是基址寄存器(Base Register)，在内存寻址时用于存放基地址。
* ECX是计数器(Count Register)，用于记录循环次数以及移位时用CL指明移位的位数。
* EDX是数据寄存器(Data Register)，它可在计算乘除法时用于存放默认操作数，也可用于存放I/O的端口地址。

1. 变址寄存器

寄存器ESI、EDI、SI和DI称为变址寄存器(Index Register)，其中ESI和SI称为源地址指针寄存器，EDI和DI称为目的地址指针寄存器，它们主要用于存放存储单元在段内的偏移量，它们可作一般的存储器指针使用。在很多字符串操作指令中，DS:ESI指向源串，而ES:EDI指向目标串。

1. 地址指针寄存器

32位CPU有2个32位通用寄存器EBP和ESP。它们主要用于访问堆栈内的存储单元，并且规定：

* EBP是基址指针寄存器(Base Pointer Register)，一般作为当前堆栈的最后单元，用它可直接存取堆栈中的数据；该寄存器可以被作为BP或者EBP寻址，其缺省段寄存器为SS。
* ESP是堆栈指针寄存器(Stack Pointer Register)，专门用作堆栈指针，称之为栈顶指针，在32位平台上，ESP每次减少4字节。

二、段寄存器[[[3]](#footnote-3)](Segment Registers)

除了8086中的四个段(CS、DS、ES、SS)外，32位处理器增加了两个段(FS、GS)，这些段寄存器都是16位的，它们的含义如下：

* + CS：代码段(Code Segment)
  + DS：数据段(Data Segment)
  + ES：附加段(Extra Segment)
  + SS：堆栈段(Stack Segment)
  + FS：标志段(Flag Segment)
  + GS：全局段(Global Segment)

在32位模式下，传统的段寄存器，如CS、SS、DS、ES，保存的不再是16位段基地址，而是段的选择子，即，用于选择所要访问的段，因此，严格地说，它的新名字叫做段选择器。除了段选择器之外，每个段寄存器还包括一个64位的不可见部分，称为描述符高速缓存器，里面有段的基地址和各种访问属性（界限、权限）。这部分内容程序不可访问，由处理器自动使用。示意图如下：



三、指令指针寄存器(Instruction Pointer Register)

指令指针寄存器EIP的低16位就是8086的IP寄存器，为了生成32位物理地址，处理器需要使用32位的指令指针寄存器。示意图如下：



四、标志寄存器[[[4]](#footnote-4)][[[5]](#footnote-5)][[[6]](#footnote-6)](Program Status and Control Register)

EFLAGS和8086的16位标志寄存器相比，增加了4个控制位(IOPL、NT、RF、VM)，但这4个标志位在实模式下不起作用，它们的位置如下图：



相关的控制/标志位含义如下：

* CF(Carry Flag)：进位标志位
* PF(Parity Flag)：奇偶标志位
* AF(Auxiliary Carry Flag)：辅助进位标志位
* ZF(Zero Flag)：零标志位
* SF(Sign Flag)：符号标志位
* IF(Interrupt Enable Flag)：中断允许标志位，由CLI、STI两条指令来控制；设置IF使CPU可识别外部(可屏蔽)中断请求。复位IF则禁止中断。IF对不可屏蔽外部中断和故障中断的识别没有任何作用。
* DF(Direction Flag)：向量标志位，由CLD、STD两条指令来控制。
* OF(Overflow Flag)：溢出标志位。
* IOPL(I/O Privilege Level)：I/O特权级字段，它的宽度为2位，它指定了I/O指令的特权级。如果当前的特权级别在数值上小于或等于IOPL，那么I/O指令可以执行。否则，将发生一个保护性故障中断。
* NT(Nested Task Flag)：控制中断返回指令IRET，它宽度为1位。若NT=0，则用堆栈中保存的值恢复EFlags、CS和EIP从而实现中断返回；若NT=1，则通过任务切换实现中断返回。
* TF(Trap Flag)：自陷(追踪)标志位，当TF被置为1时，CPU进入单步执行方式，即每执行一条指令，产生一个单步中断请求。这种方式主要用于程序的调试。
* RF(Restart Flag)：重启动标志位，RF用来控制是否接受调试故障。规定：RF=0时，表示“接受”调试故障，否则拒绝之。在成功执行完一条指令后，处理机把RF置为0，当接受到一个非调试故障时，处理机就把它置为1。
* VM(Virtual 8086 Mode)：虚拟8086方式标志位，如果VM的值为1，则表示处理机处于虚拟的8086方式下的工作状态，否则处理机处于一般保护方式下的工作状态。
* AC(Alignment Check / Access Control)、VIF(Virtual Interrupt Flag)、VIP(Virtual Interrupt Pending)、ID(ID Flag)是Pentium处理器增加的标志位。

五、系统地址寄存器[[[7]](#footnote-7)](Memory Management Registers)

80386有4个系统地址寄存器，它们是全局描述符表寄存器GDTR(Global Descriptor

Table Register)、中断描述符表寄存器IDTR(Interrupt Descriptor Table Register)、局部描述表

寄存器LDTR(Local Descriptor Table Register)和任务寄存器TR(Task Register)。它们主要用

来在保护模式下，管理用于生成线性地址和物理地址的4个系统管理描述符表，这些寄存器不直接被程序访问。[[[8]](#footnote-8)]



GDTR、LDTR、IDTR和TR都是段基址寄存器，这些段中含有分段机制的重要信息表。GDTR、IDTR和LDTR用于寻址存放描述符表的段。TR用于寻址一个特殊的任务状态段(Task State Segment，TSS)。TSS中包含着当前执行任务的重要信息。

* 全局描述符表寄存器GDTR(Global Descriptor Table Register)：48位寄存器，GDTR寄存器中用于存放全局描述符表GDT的32位的线性基地址和16位的表限长值。基地址指定GDT表中字节0在线性地址空间中的地址，表长度指明GDT表的字节长度值。指令LGDT和SGDT分别用于加载和保存GDTR寄存器的内容。在机器刚加电或处理器复位后，基地址被默认地设置为0，而长度值被设置成0xFFFF。在保护模式初始化过程中必须给GDTR加载一个新值。
* 中断描述符表寄存器IDTR(Interrupt Descriptor Table Register)：48位寄存器，与GDTR的作用类似，IDTR寄存器用于存放中断描述符表IDT的32位线性基地址和16位表长度值。指令LIDT和SIDT分别用于加载和保存IDTR寄存器的内容。在机器刚加电或处理器复位后，基地址被默认地设置为0，而长度值被设置成0xFFFF。
* 局部描述符表寄存器LDTR(Local Descriptor Table Register)：16位寄存器，LDTR寄存器中用于存放局部描述符表LDT的32位线性基地址、16位段限长和描述符属性值。指令LLDT和SLDT分别用于加载和保存LDTR寄存器的段描述符部分。包含LDT表的段必须在GDT表中有一个段描述符项。当使用LLDT指令把含有LDT表段的选择符加载进LDTR时，LDT段描述符的段基地址、段限长度以及描述符属性会被自动地加载到LDTR中。当进行任务切换时，处理器会把新任务LDT的段选择符和段描述符自动地加载进LDTR中。在机器加电或处理器复位后，段选择符和基地址被默认地设置为0，而段长度被设置成0xFFFF。
* 任务寄存器TR(Task Register)：16位寄存器，TR寄存器用于存放当前任务TSS段的16位段选择符、32位基地址、16位段长度和描述符属性值。它引用GDT表中的一个TSS类型的描述符。指令LTR和STR分别用于加载和保存TR寄存器的段选择符部分。当使用LTR指令把选择符加载进任务寄存器时，TSS描述符中的段基地址、段限长度以及描述符属性会被自动加载到任务寄存器中。当执行任务切换时，处理器会把新任务的TSS的段选择符和段描述符自动加载进任务寄存器TR中。

六、控制寄存器[[[9]](#footnote-9)](Control Registers)

32位的控制寄存器(CR0~CR3) 保存全局性、和任务无关的机器状态，用于控制和确定处理器的操作模式以及当前执行任务的特性，如下图所示：



* CR0中包含了6个预定义标志，0位是保护允许位PE(Protection Enable)，用于启动保护模式，如果PE位置1，则保护模式启动，如果PE=0，则在实模式下运行。1位是监控协处理位MP(Monitor Coprocessor)，它与第3位一起决定：当TS=1时操作码WAIT是否产生一个“协处理器不能使用”的出错信号。第3位是任务转换位(Task Switch)，当一个任务转换完成之后，自动将它置1。随着TS=1，就不能使用协处理器。CR0的第2位是模拟协处理器位 EM (Emulate Coprocessor)，如果EM=1，则不能使用协处理器，如果EM=0，则允许使用协处理器。第4位是微处理器的扩展类型位ET(Processor Extension Type)，其内保存着处理器扩展类型的信息，如果ET=0，则标识系统使用的是287协处理器，如果 ET=1，则表示系统使用的是387浮点协处理器。CR0的第31位是分页允许位(Paging Enable)，它表示芯片上的分页部件是否允许工作。由PG位和PE位定义的操作方式如下表所示：

|  |  |  |
| --- | --- | --- |
| **PG** | **PE** | 方式 |
| 0 | 0 | 实模式， |
| 0 | 1 | 未开启分页机制的保护模式 |
| 1 | 0 | 不存在 |
| 1 | 1 | 开启了分页机制的保护模式 |

* CR1是未定义的控制寄存器，供将来的处理器使用。
* CR2是页故障线性地址寄存器，保存最后一次出现页故障的全32位线性地址。
* CR3是页目录基址寄存器，保存页目录表的物理地址，页目录表总是放在以4K字节为单位的存储器边界上，因此，它的地址的低12位总为0，不起作用，即使写上内容，也不会被理会。CR3中含有页目录表物理内存基地址，因此该寄存器也被称为页目录基地址寄存器PDBR(Page-Directory Base address Register)。

七、调试寄存器[[[10]](#footnote-10)][[[11]](#footnote-11)](Debug Registers)

32位处理器有8个32位的调试寄存器DR0~DR7，调试寄存器具有如下功能：

* 设置发生断点的地址（线性地址）。
* 设置断点的长度（可以为1、2、4个字节，但是执行断点只能是1个字节）。
* 设置在调试异常产生的地址执行的操作。
* 设置断点是否可用。
* 在调试异常产生时，调试条件是否是可用。

其中DR0~DR3这四个寄存器用于存储线性断点地址，DR4和DR5是Intel保留寄存器，DR6主要是在调试异常产生后，报告产生调试异常的相关信息，DR7包含一些控制位，用于控制断点的方式。示意图如下：



下面详细介绍部分调试寄存器的具体作用：

1. 调试地址寄存器DR0~DR3：

每个调试地址寄存器保存一个32位的断点线性地址。断点比较发生在物理地址转换之前。调试地址寄存器无论有没有开启分页机制，都是有效的，塔里木存储的是线性地址。如果开启了分页机制，线性地址会通过处理器的分页处理单元被转化为物理地址。如果没有开启分页机制，线性地址就等同于物理地址。

但是需要注意的是当分页开启的时候，不同的任务有不同的线性地址到物理地址的映射。在这种情况下，调试寄存器中的地址会被关联到其中一个任务上。由于这个原因，80386在DR7寄存器中有全局和局部开启位。这些位表明了调试地址是全局关联的（所有的任务）还是局部关联的（当前任务）。

1. 调试寄存器DR4~DR5：

当调试扩展被开启的时候（控制寄存器CR4中的DE标志置位），保留调试寄存器DR4和DR5。任何试图引用这两个寄存器的行为都将会触发一个无效机器码异常。DE不置位时，DR4和DR5是DR6和DR7的别名。

1. 调试状态寄存器DR6：

这个寄存器主要是在调试异常产生后，报告产生调试异常的相关信息，处理器从不清空该寄存器的内容。该寄存器标志位显示下列信息：

* B0~B3指明检测到断点条件。如果其中某个位被置位，则表示是相应的DR0~DR3断点引发了调试陷阱。此外还存在着一种情况，不管DR7寄存器中的Gi、Li(i=0,1,2,3)如何设置，Bi都会被置位。因此，只要是遇到DRi指定的断点，总会设置Bi，如果看到多个Bi置位，则可以通过Gi、Li的情况判断究竟是哪个调试寄存器引发的调试陷阱。
* BD位（调试寄存器访问检测）指明指令流中的下一个指令将会访问某个调试寄存器，当调试控制寄存器DR7中的GD(General Detect)标志置位时才会生效。
* BS位（单步）指明调试异常是由单步执行模式触发的。
* BT位（任务切换）指明调试异常是任务切换导致的，目标任务的TSS中调试陷阱标志置位。

1. 调试控制寄存器DR7：

调试控制寄存器可以开启或关闭断点，并设置断点条件。它的标志和域控制以下信息：

* L0~L3置位时（局部断点有效），启动断点模式，即激活当前任务的相关断点。当检测到断点条件，对应的Li位置位，产生调试异常。处理器在任务切换时自动清空这些标志，以避免一个不必要的断点出现在新任务中。
* G0~G3置位时（全局断点有效），启动断点模式，即激活所有任务的相关断点。当检测到断点条件，对应的Gi位置位，产生调试异常。处理器在任务切换时不清空标志，这样断点可以出现在所有的任务中。
* LE和GE（局部和全局精确断点）开启后处理器能够精确定位引发数据中断条件的那条指令。
* GD用于保护DRi，如果GD位为1，则对DRi的任何访问都会产生调试异常，并导致进入1号调试陷阱，即IDT的对应入口，这样可以保证调试器在必要的时候完全控制DRi。当检测到对DRi寄存器的访问时，调试状态寄存器DR6中的BD就会被置位，并产生一个异常。
* R/W0~R/W3指明对应断点的断点条件，是读、写还是执行断点或是I/O端口断点。
* LEN0~LEN3表示长度。

八、测试寄存器[[[12]](#footnote-12)](Test Registers)

80386设置了8个32位的测试寄存器TR0到TR7，其中TR0~ TR5由Intel公司保留，用户只能访问TR6、TR7，这两个寄存器用于在转换旁视缓冲器(Translation Lookaside Buffer)中测试随机存储器(RAM)和相联存储器(CAM)。TR6是测试控制寄存器，其内存放测试控制命令，TR7是测试状态寄存器，其内保存转换旁路缓冲器测试的数据。

1. [] 李忠，王晓波，余洁．x86汇编语言：从实模式到保护模式[M]．北京：电子工业出版社，2012：183-184． [↑](#footnote-ref-1)
2. [] 飘零过客．32位处理器的寄存器介绍[DB/OL]．https://blog.csdn.net/xuehuafeiwu123/article/details/76019828，2017-07-24． [↑](#footnote-ref-2)
3. [] 李忠，王晓波，余洁．x86汇编语言：从实模式到保护模式[M]．北京：电子工业出版社，2012：185-186． [↑](#footnote-ref-3)
4. [] 陈渝，向勇．操作系统实验指导[M]．北京，清华大学出版社，2013：33-34． [↑](#footnote-ref-4)
5. [] qintangtao．指令指针寄存器和标志寄存器[DB/OL]．http://www.cnblogs.com/qintangtao/p/4161912.html，2014-12-13． [↑](#footnote-ref-5)
6. [] feng.qi．指令指针寄存器和标志寄存器[DB/OL]．https://www.cnblogs.com/feng-qi/articles/3055729.html，2013-05-03． [↑](#footnote-ref-6)
7. [] yyt7529．x86中内存管理寄存器[DB/OL]．https://blog.csdn.net/yyt7529/article/details/4325980，2009-07-06． [↑](#footnote-ref-7)
8. [] 陈光军，傅越千．微机原理与接口技术[M]．北京，北京大学出版社，2007：31． [↑](#footnote-ref-8)
9. [] feng.qi．指令指针寄存器和标志寄存器[DB/OL]．https://www.cnblogs.com/feng-qi/articles/3055729.html，2013-05-03． [↑](#footnote-ref-9)
10. [] Tweek．[总结]调试寄存器 原理与使用：DR0-DR7[DB/OL]．https://bbs.pediy.com/thread-107515.htm，2010-02-21． [↑](#footnote-ref-10)
11. [] halfdead．Mistifying the debugger, ultimate stealthness[J/OL]．https://www.docin.com/p-36299796.html，2009-12-06． [↑](#footnote-ref-11)
12. [] zmcomputer．调试寄存器和测试寄存器[DB/OL]．https://blog.csdn.net/zmcomputer/article/details/5908227，2010-09-26． [↑](#footnote-ref-12)