

2021/01/29_调试器_第5课_调试器硬件断点的编写

笔记本： 调试器

创建时间： 2021/1/29 星期五 10:50

作者： ileemi

断点等异常其实就是指令本身产生的异常，CPU执行完指令时产生异常（"int 3" 产生异常可以理解为 CPU 执行这条指令时，出现错误）。单步步入其实就是利用硬件功能实现，指令本身没有问题，通过修改 "TF" 标志位，让 CPU 支持这条指令后，停下来。自动单步（循环执行 "t" 功能）。

CPU硬件的功能


CPU提供一些硬件断点，只需要将要下断点的地址告诉CPU即可，不需要修改程序的代码，CPU指定到该地址后，就会停下来。这样以来，程序检测是否被调试就不便于检测，内存访问分页问题页就几乎不存在。

CPU为了方便设置断点，提供了一些调试寄存器（debug register）。只需要将设置断点的地址告诉CPU即可。

DR0~DR7，DR4、DR5保留：

3.7.2 Register Operands

Source and destination operands can be any of the following registers, depending on the instruction being executed:

- 32-bit general-purpose registers (EAX, EBX, ECX, EDX, ESI, EDI, ESP, or EBP)
- 16-bit general-purpose registers (AX, BX, CX, DX, SI, DI, SP, or BP)
- 8-bit general-purpose registers (AH, BH, CH, DH, AL, BL, CL, or DL)
- segment registers (CS, DS, SS, ES, FS, and GS)
- EFLAGS register
- x87 FPU registers (ST0 through ST7, status word, control word, tag word, data operand pointer, and instruction pointer)
- MMX registers (MM0 through MM7)
- XMM registers (XMM0 through XMM7) and the MXCSR register
- control registers (CR0, CR2, CR3, and CR4) and system table pointer registers (GDTR, LDTR, IDTR, and task register)
- debug registers (DR0, DR1, DR2, DR3, DR6, and DR7) 
- MSR registers

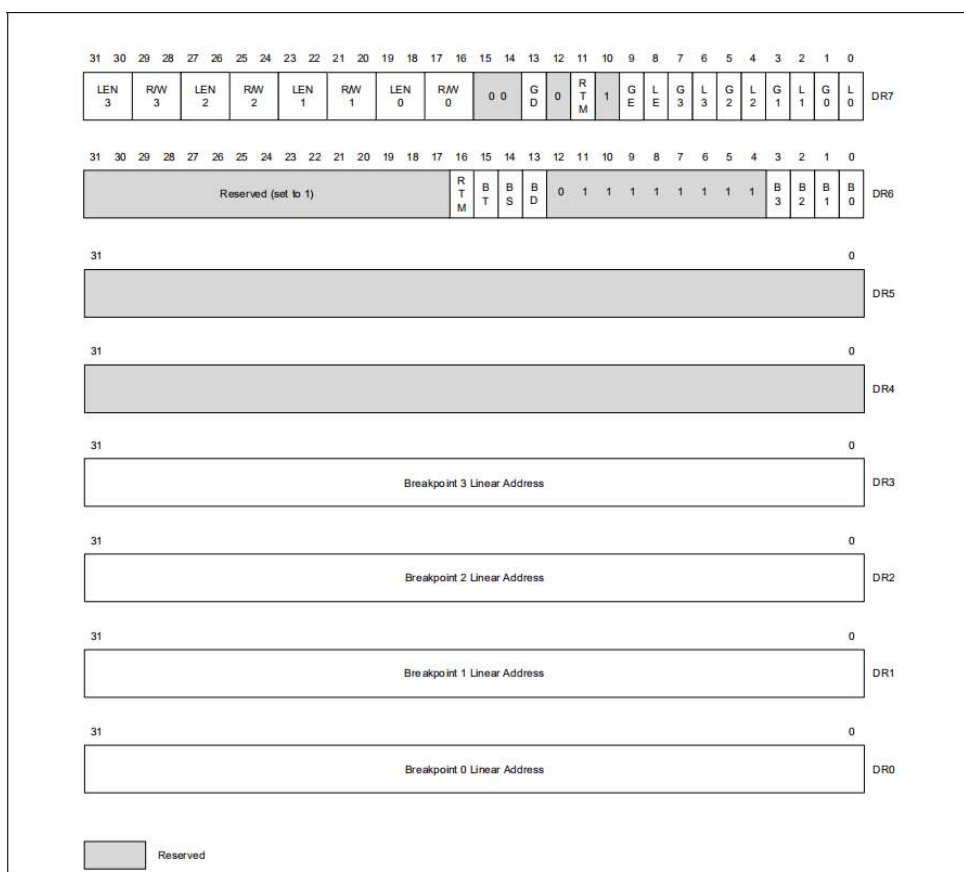


Figure 17-1. Debug Registers

调试寄存器以线程为单位，一个线程只能下四个硬件断点。别的进程需要使用时，可将寄存器环境进行保存。硬件断点是一次性的，使用后会自动复位。使用硬件断点产生的异常时在指令指令完后产生的，之前手动设置的内存断点等都是指令没有指令完，产生的异常。

L0~L3：局部断点（以线程为单位）

G0~G3：全局断点（以进程为单位）

DR7

调试控制寄存器(DR7)启用或禁用断点并设置断点条件（参见上图所示）。在设置硬件断点的时候，还需要指明断点的条件以及位置大小（长度）。设置执行断点时，其长度应设置为：0字节，此寄存器中的标志和字段控制以下内容：

- **L0 through L3 (local breakpoint enable) flags (bits 0, 2, 4, and 6)：** 启用（设置时）当前任务（**局部**）相关断点的断点条件。当检测到断点条件并设置其关联的Ln标志时，将生成调试异常。处理器自动清除每个任务开关上的这些标志，以避免新任务中不必要的断点条件。
- **G0 through G3 (global breakpoint enable) flags (bits 1, 3, 5, and 7)：** 启用（设置时）所有任务（**全局**）相关断点的断点条件。当检测到断点条件并设置其关联的Gn标志时，将生成调试异常。处理器不清除任务开关上的这些标志，允许为所有任务启用断点。

- R/W0 through R/W3 (read/write) fields (bits 16, 17, 20, 21, 24, 25, 28, and 29)** — Specifies the breakpoint condition for the corresponding breakpoint. The DE (debug extensions) flag in control register CR4 determines how the bits in the R/Wn fields are interpreted. When the DE flag is set, the processor interprets bits as follows:
 - 00 — Break on instruction execution only.
 - 01 — Break on data writes only.
 - 10 — Break on I/O reads or writes.
 - 11 — Break on data reads or writes but not instruction fetches.

When the DE flag is clear, the processor interprets the R/Wn bits the same as for the Intel386™ and Intel486™ processors, which is as follows:

 - 00 — Break on instruction execution only.
 - 01 — Break on data writes only.
 - 10 — Undefined.
 - 11 — Break on data reads or writes but not instruction fetches.
- LEN0 through LEN3 (Length) fields (bits 18, 19, 22, 23, 26, 27, 30, and 31)** — Specify the size of the memory location at the address specified in the corresponding breakpoint address register (DR0 through DR3). These fields are interpreted as follows:
 - 00 — 1-byte length.
 - 01 — 2-byte length.
 - 10 — Undefined (or 8 byte length, see note below).
 - 11 — 4-byte length.

指定对应断点的断点条件

在相应的断点地址寄存器(DR0到DR3)中指定的地址指定内存位置的大小

17-4 Vol. 3B

硬件内存断点

使用硬件内存断点在目标进程中的入口地址下断点。"CONTEXT_FULL" 宏不包含调试寄存器。使用硬件断点产生固定的异常为：单步异常（80000004H），所以就需要通过硬件断点寄存器Dr6判断目标进程中产生的 "单步异常" 是否是硬件断点异常。

在目标进程中使用硬件内存断点，目标进程的 "EIP" 会指向硬件内存断点地址的下一条指令上。

硬件执行断点，必须清空标志位。执行断点和内存断点的处理手动有些差异。

附加

被附加的程序不能被其它调试器调试。

附加目标程序时，目标程序处于被调试状态，不能进行操作，取消附加后，被附加的程序就可以正常运行。

暂停：让运行的目标程序产生一条int异常。DebugBreakPoint--让目标进程产生一个int。

DebugBreak：让自己的程序产生一个int3异常。

FatalExit：目标进程退出时产生一条int3。

利用附加功能可以实现双进程反调试。