T-HEAD CPU 调试技巧

2022 年 11 月 11 日

平头哥半导体有限公司 T-HEAD Semiconductor Co.,LTD

地址: 杭州市余杭区向往街 1122 号欧美金融城 (EFC) 英国中心西楼 T6

邮编: 311121

网址: www.t-head.cn

版本历史

版本	描述	日期
1.0	初始版本	2020.09.01
2.0	修改 T-HEAD 命名	2022.11.10

T-HEAD CPU 调试技巧

第一章　使用介绍

一般来说 T-HEAD 调试器主要调试的是 C/C++ 程序和汇编程序。要进行程序的调试，首先在编译时，我们必须要把调试信息加到可执行文件中。使用编译器的 -g 参数可以做到这一点。如：

玄铁 900 系列的命令如下所示：

riscv64-unknown-elf-gcc -g hello.c -o hello

riscv64-unknown-elf-gcc -g hello.cpp -o hello

玄铁 800 系列的命令如下所示：

csky-abiv2-elf-gcc -g hello.c -o hello

csky-abiv2-elf-g++ -g hello.cpp -o hello

如果没有-g，你将看不见程序的变量名，所代替的全是运行时的内存地址。当你用-g 把调试信息加入之后，并成功编译目标代码以后，让我们来看看如何用调试器来调试它。

1. 启动 T-HEAD 调试器的方法：

玄铁 900 系列：

riscv64-unknown-elf-gdb <program>

玄铁 800 系列：

csky-abiv2-elf-gdb <program>

program 也就是你的被调试的执行文件，一般在当前目录下。

2. 查看 T-HEAD 调试器命令的帮助：

T-HEAD 调试器的命令很多，T-HEAD 调试器把它们分成许多个种类。help 命令只是列出 T-HEAD 调试器的命令种类，如果要看种类中的命令，可以使用 help <class> 命令，如：help breakpoints，查看设置断点的所有命令。也可以直接 help <command> 来查看命令的帮助。

T-HEAD 调试器中，输入命令时，可以不用打全命令，只用打命令的前几个字符就可以了，当然，命令的前几个字符应该要标志着一个唯一的命令，在 Linux 下，你可以敲击两次 TAB 键来补齐命令的全称，如果有重复的，那么 T-HEAD 调试器会把其列出来。

3. 退出 T-HEAD 调试器：

要退出 T-HEAD 调试器时，只用发 quit 或命令简称 q 就行了。

4.T-HEAD 调试器中运行 UNIX 的 shell 程序

在 gdb 环境中，你可以执行 UNIX 的 shell 的命令，使用 T-HEAD 调试器的 shell 命令来完成：

shell <command string>

调用 UNIX 的 shell 来执行 <command string>，环境变量 SHELL 中定义的 UNIX 的 shell 将会被用来执行 <command string>，如果 SHELL 没有定义，那就使用 UNIX 的标准 shell：/bin/sh。（在 Windows 中使用 Command.com 或 cmd.exe）

还有一个 T-HEAD 调试器命令是 make：

make <make-args>

可以在 T-HEAD 调试器中执行 make 命令来重新 build 自己的程序。这个命令等价于 “shell make <make-args>”。

注意： 由于 T-HEAD 调试器是用于调试 T-HEAD 体系结构的应用程序，那么如果用户需要在硬件目标板调试程序，则还需要先启动调试代理服务程序 T-Head DebugServer，具体请参考 T-HEAD DebugServer 的用户手册。

第二章　GDB 标准调试命令

T-HEAD 调试器是一个强大的命令行调试工具。大家知道命令行的强大就是在于，其可以形成执行序列，形成脚本。Unix 或 Linux 下的软件全是命令行的，这给程序开发提代供了极大的便利，命令行软件的优势在于，它们可以非常容易的集成在一起，使用几个简单的已有工具的命令，就可以做出一个非常强大的功能。

当以 T-HEAD 调试器 <program> 方式启动 T-HEAD 调试器后，T-HEAD 调试器会在 PATH 路径和当前目录中搜索 <program> 的源文件。

2.1　File 命令

用于打开被调试的程序，并根据文件基本信息，更新调试器内部状态，如大小端等。不管启动 T-HEAD 调试器是否已经打开被调试的程序文件，您都可以使用该命令来指定或重新指定被调试程序。

file 被调试文件名

2.2　Target 命令

连接目标调试系统，目前有以下集中目标调试系统：

target jtag jtag://127.0.0.1:1025

连接调试代理服务程序（和实际目标板连接调试），其中 127.0.0.1 是调试代理服务程序的所在计算机的 ip 地址，1025 则是调试代理服务程序启动时的 socket 端口号，缺省值为 1025

target remote 192.168.0.102：1025

连接 T-HEAD 实现的 RISC-V/CSKY 体系结构的软件仿真器 xuantie-qemu，其中 192.168.0.102 是 qemu 所在 host 机上的 ip 地址，1025 则是 socket 端口号。请参考仿真器的用户手册。

2.3　Directory 命令

用于设置被调是用户程序的源代码目录，以便在调试时方便结合源代码进行信息调试分析。

directory

将源代码搜索路径初始化为空

directory 源代码搜索路径

添加新的路径到搜索 PATH

2.4 Run 命令

在 T-HEAD 调试器中，运行程序使用 r 或是 run 命令。

暂停或恢复程序运行

调试程序中，暂停程序运行是必须的，T-HEAD 调试器可以方便地暂停程序的运行。你可以设置程序的在哪行停住，在什么条件下停住。以便于你查看运行时的变量，以及运行时的流程。

当程序被 T-HEAD 调试器停住时，你可以使用 info program 来查看程序被暂停的原因。

在 T-HEAD 调试器中，我们可以有以下几种暂停方式：

断点（BreakPoint）、观察点（WatchPoint）停止程序运行：

按 CTRL+C 停止运行的程序

如果要恢复程序运行，可以使用 c 或是 continue 命令。

2.5 设置断点（BreakPoint）

我们用 break 或 b 命令来设置断点，例如：

*b * 程序代码断地址*

b 函数名

b 文件名：行号

b 当前程序 pc 指针所在文件行号

break +offset 或 b +offset

break -offset 或 b –offset

在当前行号的前面或后面的 offset 行停住。offiset 为自然数。

注解： offset 的范围为：只要存在（当前行数 +offset）这一行就可以。

我们用 hb 来设置硬件断点，注意，硬件断点由于受到硬件限制，具体来说，在 E906 和 E907 系列的 CPU 中，硬断点最多设置两个；在 E802 和 E803 两款 CPU 中，最多只能设置一个硬断点。

*hb * 代码段数据断等任何程序地址*

hb 函数名

hb 文件名：行号

hb 当前程序 pc 指针所在文件行号

hb 变量名。

hbreak +offset 或 *hb +offset*

hbreak -offset 或 *hb –offset*

在当前行号的前面或后面的 offset 行停住。offiset 为自然数。

查看断点时，可使用 info 命令，如下所示：（注：n 表示断点号）

info breakpoints [n]

info break [n]

注解： offset 的范围为：只要存在（当前行数 +offset）这一行就可以。

2.6 设置观察点（WatchPoint）

用 watch 命令来设置观察点。

观察点一般来观察某个表达式（变量也是一种表达式）的值是否有变化了，如果有变化，马上停住程序。我们有下面的几种方法来设置观察点：

watch 表达式

查看观察点时，可使用 info 命令

info watchpoints

列出当前所设置了的所有观察点。

2.7 维护停止点（断点或观察点）

在 T-HEAD 调试器中，如果你觉得已定义好的停止点没有用了，你可以使用 delete、clear、disable、enable 这几个命令来进行维护。

clear

1. 如果没有参数，删除当前行所有的断点。

2. 如果后面加行数，则删除该行所有的断点。

3. 如果后面加函数名，则删除在函数首定义的所有断点。

delete [breakpoints] [range···]

删除指定的断点，breakpoints 为断点号。如果不指定断点号，则表示删除所有的断点。range 表示断点号的范围（如：3-7 或者 3 4 5 6 7 用空格分隔）。其简写命令为 d。

比删除更好的一种方法是 disable 停止点，disable 了的停止点，T-HEAD 调试器不会删除，当你还需要时，enable 即可，就好像回收站一样。

disable [breakpoints] [range···]

disable 所指定的停止点，breakpoints 为停止点号。如果什么都不指定，表示 disable 所有的停止点。简写命令是 dis.

enable [breakpoints] [range···]

enable 所指定的停止点，breakpoints 为停止点号。简写为 en。

2.8 恢复程序运行和单步调试

当程序被停住了，可以使用 continue 继续运行，step，next 等命令单步跟踪

continue 或 c

命令恢复程序的运行直到程序结束，或下一个断点到来。

step <count>

单步跟踪，如果有函数调用，他会进入该函数。进入函数的前提是，此函数被编译有 debug 信息。

next <count> 或 n <count>

同样单步跟踪，如果有函数调用，他不会进入该函数。

stepi 或 si

nexti 或 ni

单步跟踪一条机器指令。一条程序代码有可能由数条机器指令完成，stepi 和 nexti 可以单步执行机器指令.

2.9 查看栈信息

当程序被停住了，你需要做的第一件事就是查看程序是在哪里停住的。当你的程序调用了一个函数，函数的地址，函数参数，函数内的局部变量都会被压入“栈”（Stack）中。你可以用 T-HEAD 调试器命令来查看当前的栈中的信息。

下面是一些查看函数调用栈信息的 T-HEAD 调试器命令：

backtrace 或 bt

打印当前的函数调用栈的所有信息。如：

(gdb) bt

#0 func (n=250) at tst.c:6

#1 0x08048524 in main (argc=1, argv=0xbffff674) at tst.c:30

#2 0x400409ed in ___libc_start_main () from /lib/libc.so.6

根据以上输出可以看出函数的调用栈信息：___libc_start_main –> main() –> func()

GDB 跳转到相应的堆栈 Frame 命令：

frame Id

(gdb) frame 2 跳到 frame 2，即 ___libc_start_main 的 frame

用户可以查看当前堆栈 frame 的一些局部变量等信息

2.10 查看源程序

显示源代码 list 或 l

T-HEAD 调试器可以打印出所调试程序的源代码，当然，在程序编译时一定要加上-g 的参数，把源程序信息编译到执行文件中。不然就看不到源程序了。当程序停下来以后，T-HEAD 调试器会报告程序停在了那个文件的第几行上。你可以用 list 命令来打印程序的源代码。

list 或 l

显示当前行后面的源程序。一般是打印当前行的上 5 行和下 5 行。

1.disassemble

查看源程序的当前执行时的机器码，这个命令会把目前内存中的指令 dump 出来。如下面的示例表示查看函数 func 的汇编代码。可以如下执行命令：

disassemble

disassemble 地址值, 地址值 *2*

disassemble $pc,$pc+offset

disassemble func

2.11 查看运行时数据

在你调试程序时，当程序被停住时，你可以使用 print 命令（简写命令为 p），或是同义命令 inspect 来查看当前程序的运行数据。print 命令的格式是：

print <expr>

print /<f> <expr>

<expr> 是表达式，是你所调试的程序的语言的表达式，<f> 是输出的格式，比如，如果要把表达式按 16 进制的格式输出，那么就是/x。

一般来说，T-HEAD 调试器会根据变量的类型输出变量的值。但你也可以自定义 T-HEAD 调试器的输出的格式。例如，你想输出一个整数的十六进制，或是二进制来查看这个整型变量的中的位的情况。要做到这样，你可以使用 T-HEAD 调试器的数据显示格式：

x 按十六进制格式显示变量。

d 按十进制格式显示变量。

u 按十六进制格式显示无符号整型。

o 按八进制格式显示变量。

t 按二进制格式显示变量。

a 按十六进制格式显示变量。

c 按字符格式显示变量。

f 按浮点数格式显示变量。

2.12 查看内存

你可以使用 examine 命令（简写是 x）来查看内存地址中的值。x 命令的语法如下所示：

x/<n/f/u> <addr>

n、f、u 是可选的参数。

n 是一个正整数，表示显示内存的长度，也就是说从当前地址向后显示几个地址的内容。

f 表示显示的格式，参见上面。如果地址所指的是字符串，那么格式可以是 s，如果地址是指令地址，那么格式可以是 i。

u 表示从当前地址往后请求的字节数，如果不指定的话，T-HEAD 调试器默认是 4 个 bytes。u 参数可以用下面的字符来代替，b 表示单字节，h 表示双字节，w 表示四字节，g 表示八字节。当我们指定了字节长度后，T-HEAD 调试器会从指内存定的内存地址开始，读写指定字节，并把其当作一个值取出来。

<addr> 表示一个内存地址。

n/f/u 三个参数可以一起使用。例如：

命令：x/3uh 0x54320 表示，从内存地址 0x54320 读取内容，h 表示以双字节为一个单位，3 表示三个单位，u 表示按十六进制显示。

2.13 自动显示

你可以设置一些自动显示的变量，当程序停住时，或是在你单步跟踪时，这些变量会自动显示。相关的 T-HEAD 调试器命令是 display。

display <expr>

display/<fmt> <expr>

display/<fmt> <addr>

expr 是一个表达式，fmt 表示显示的格式，addr 表示内存地址，当你用 display 设定好了一个或多个表达式后，只要你的程序被停下来，T-HEAD 调试器会自动显示你所设置的这些表达式的值。

格式 i 和 s 同样被 display 支持，一个非常有用的命令是：

display/i $pc

$pc 是 T-HEAD 调试器的环境变量，表示着指令的地址，/i 则表示输出格式为机器指令码，也就是汇编。于是当程序停下后，就会出现源代码和机器指令码相对应的情形，这是一个很有意思的功能。

2.14 维护自动显示

上面说了如何设置程序的自动显示。在 T-HEAD 调试器中，如果你觉得已定义好的自动显示点没有用了，你可以使用 delete、disable、enable 这几个命令来进行维护。

delete display [display points]

删除指定的断点，*display points* 为自动显示号。如果不指定断点号，则表示删除所有的断点。range 表示断点号的范围（如 3 4 5 6 7 用空格分隔）。其简写命令为 d display。

另外一种方法是 disable 自动显示点，disable 了的自动显示点，T-HEAD 调试器不会删除，当你还需要时，enable 即可。

disable display [display points]

disable 所指定的停止点，breakpoints 为停止点号。如果什么都不指定，表示 disable 所有的停止点。简写命令是 dis display.

enable display [display points]

enable 所指定的停止点，breakpoints 为停止点号。简写为 en display。

注意： 与维护停止点有所不同。比如 d 3 是删除 3 号断点。d display 3 是删除 3 号自动显示点。可以写 d 3-5 但是不能写 d display 3-5 只能是 d display 3 4 5。

2.15 查看寄存器

要查看寄存器，很简单，可以使用如下命令：

info registers

查看 CPU 通用寄存器和 psr，epsr，pc，epc。

info registers <regname···>

查看具体的某个寄存器，regname 表示具体寄存器名字，可以是多个。

寄存器中放置了运行时数据，比如程序当前运行的指令地址（pc），程序的当前堆栈指针（sp）等等。你同样可以使用 print 命令来访问寄存器的情况，只需要在寄存器名字前加一个 $ 符号就可以了。如：p $pc。

由于 CPU 内部寄存器较多，gdb 内部对所有 CPU 对应的寄存器进行了分组，T-head gdb 支持寄存器的分组查看功能，只需要在普通查看下加上寄存器组名即可。

info registers general

查看 CPU 通用寄存器。

info registers cr

查看 CPU 全部控制寄存器。

info registers fr

查看 CPU 全部浮点寄存器。

info registers vr

查看 CPU 全部向量寄存器，仅存在与 CK810 处理器中。

info registers mmu

查看 CPU 全部内存管理相关的控制寄存器。

info registers profiling

查看 CPU 全部 profiling 相关寄存器，仅存在于 C810 处理器中。

2.16 修改变量值

修改被调试程序运行时的变量值，在 T-HEAD 调试器中很容易实现，使用 T-HEAD 调试器的 set 命令即可完成。如：

(gdb) set x=4（比如 set endcommand=7）

2.17 修改寄存器值

修改寄存器值在 T-HEAD 调试器中很容易实现，使用 T-HEAD 调试器的 set 命令即可完成。如：

(gdb) set $r0=4

2.18 restore 命令

调试器可以在动态调试中，随时将目标文件中的程序加载到目标板的相应的内存中。

restore <filename> <offset> [<startaddr> <endaddr>]

其中 filename 是需要加载的文件名，可以使 elf 文件，也可以是 bin 文件或者 hex 文件，offset 是程序在文件中指定的地址与即将加载地址之间的偏移；可选参数 startaddr、endaddr 如果给出，则表示仅向目标板加载这个地址区间的程序，超过这个地址范围的程序则会忽略。

2.19 dump/append 命令

调试器不仅支持动态的加载文件到内存，还支持实时的将目标板内存中的信息抽取出来，并以文件的形式保存下来，这个功能由 dump/append 命令来实现。

dump/append binary

将目标板上的指令和数据以二进制流保存到文件。

dump ihex

将目标板上的指令和数据以十六进制数据流保存到文件。

dump/append memory

将目标板上的内存的数据以二进制流保存到文件。

dump/append value

将目标板上的表达式的值以二进制流保存到文件。

2.20 call function 命令

在调试程序的时候，调试器允许用户自由的调用程序中的任意函数，用户可以通过自己输入函数的参数，然后观察函数返回值。

call <function(arg1,arg2···)>

其中，function 是函数名，arg 是参数列表，当命令完成后，调试器会输出函数执行的返回结果。

2.21 source 命令

调试器在开启后的任何时候，都可以通过 source 命令来执行脚本。

source <scriptfilepath>

其中 <scriptfilepath> 是脚本名称。

第三章　T-HEAD 拓展调试命令

3.1　pctrace 命令

T-HEAD CPU 硬件上提供了一个记录 PC 跳转轨迹的单元，可以记录最近 8 次跳转指令发生的地址的值，可以通过 T-HEAD 调试器的 pctrace 命令来查看程序运行时的 PC 跳转轨迹。直接在命令栏输入 pctrace，或者其简写命令 pc 即可。

pctrace

对于结果，标号靠前的地址是最近的跳转轨迹。值得一提的是，pc 跳转轨迹需要 T-HEAD CPU 相关硬件支持，如 E802 和 E803 不支持该功能，此外，模拟器 qemu 也没有 pctrace 功能。

3.2　设置显示 T-HEAD 调试器自身的一些属性

修改 T-HEAD 调试器自身的一些参数属性可以使用 T-HEAD 调试器的 set 命令, 例如：

(*gdb*) *set download-write-size 1024*

此命令用来设置每次下载到目标板数据的最大长度。

同样也可以查看 T-HEAD 调试器自身的一些参数属性设置，此时则用 show 命令，例如：

(*gdb*) *show download-write-size*

由于 T-HEAD 调试器自身的参数属性特别的，这里不一一列举，您可以使用如下命令获取相关帮助：

(gdb)help show

(gdb)help set

3.3　reset 命令

输入 reset 则可以调用 NReset，该功能依赖于开发板连接具有 Nreset 信号线，且该信号连接在 CKLINK 对应的管脚上。

执行完 reset 后，调试器会发起进调试操作，再次将 CPU 拉进调试模式。

3.4 sreset 命令

命令格式：sreset -c 0xABCD1234，其中 0xABCD1234 针对不同的 CPU 设定而不同。

命令执行后 CPU 将执行软件复位，复位后调试器会发起进调试操作，保证 CPU 再次进入调试模式。

第四章　调试示例

源程序：tst.c

```
1 #include  “stdio.h”
2
3 int func(int n)
4 {
5         int sum=0,i;
6         for(i=0; i<n; i++)
7         {
8                 sum+=i;
9         }
10         return sum;
11 }
12
13
14 main()
15 {
16         int i;
17         long result = 0;
18         for(i=1; i<=100; i++)
19         {
20                 result += i;
21         }
22
23        printf("result[1-100] = %d \\n", result );
24        printf("result[1-250] = %d \\n", func(250) );
25 }
```

编译生成执行文件：（Linux 下）

```
> csky-linux-gcc -g tst.c -o tst
```

使用 T-HEAD 调试器调试：

1. **启动 GDB**

```
csky-linux-gdb tst
```

```
warning: A handler for the OS ABI "GNU/Linux" is not built into this
configuration of GDB. Attempting to continue with the default csky
settings.

GNU gdb (GDB) 7.2 (built on Mar 3 2012)

Copyright (C) 2010 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or late
`<http://gnu.org/licenses/gpl.html>`_

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Type "show copying"and "show warranty" for details.

This GDB was configured as "--host=i686-pc-cygwin --target=csky-linux".
For bug reporting instructions, please see:
`<http://www.gnu.org/software/gdb/bugs/>`_
```

2. **链接目标**

```
(gdb) target remote 192.168.0.201:1025
Remote CSKY debugging using jtag://192.168.0.201:1025
0x2800028e in ?? ()
CSKY Target version 0.2
```

3. **列出源码 (l 命令相当于 list)**

```
(gdb) l

1       #include <stdio.h>
2
3       int func(int n)
4       {
5               int sum=0,i;
6               for(i=0; i<n; i++)
7               {
8                       sum+=i;
9               }
10              return sum;
```

4. **直接回车表示，重复上一次命令**

```
(gdb)

11      }
12
13
14      main()
15      {
16              int i;
17              long result = 0;
18              for(i=1; i<=100; i++)
19              {
20                      result += i;
```

5. **设置断点 (在源程序第 16 行处)**

```
(gdb) break 16

Breakpoint 1 at 0x8048496: file tst.c, line 16.
(gdb) break func                              /* 设置断点，在函数 func() 入口处 */
Breakpoint 2 at 0x8048456: file tst.c, line 5.
```

6. **查看断点信息**

```
(gdb) info break

Num Type           Disp Enb Address    What
1   breakpoint     keep y   0x08048496 in main at tst.c:16
2   breakpoint     keep y   0x08048456 in func at tst.c:5
```

7. **运行程序 (run 命令简写)**

```
(gdb) r           <---------------------

Starting program: /home/hchen/test/tst

Breakpoint 1, main () at tst.c:17    <---------- 在断点处停住。
17              long result = 0;
```

8. **单条语句执行，next 命令简写**

```
(gdb) n

```

(下页继续)

(续上页)

```
18              for(i=1; i<=100; i++)
(gdb) n
20                      result += i;
(gdb) n
18              for(i=1; i<=100; i++)
(gdb) n
20                      result += i;
```

9. **继续运行程序，continue 命令简写**

```
(gdb) c

Continuing.
result[1-100] = 5050       <----------程序输出。
```

```
Breakpoint 2, func (n=250) at tst.c:5
```

```
5               int sum=0,i;
(gdb) n
6               for(i=1; i<=n; i++)
```

10. **打印变量 i 的值，print 命令简写**

```
(gdb) p i

$1 = 134513808
(gdb) n
8                       sum+=i;
(gdb) n
6               for(i=1; i<=n; i++)
(gdb) p sum
$2 = 1
(gdb) n
8                       sum+=i;
(gdb) p i
$3 = 2
(gdb) n
6               for(i=1; i<=n; i++)
(gdb) p sum
$4 = 3
```

11. **查看函数堆栈**

```
(gdb) bt

#0  func (n=250) at tst.c:5
#1  0x080484e4 in main () at tst.c:24
#2  0x400409ed in \__libc_start_main () from /lib/libc.so.6
(gdb) c     <---------------------- 继续运行。
Continuing.
result[1-250] = 31375    <----------程序输出。

Program exited with code 027. <--------程序退出，调试结束。
```

12. **退出 T-HEAD 调试器**

```
(gdb) q
>
```

第五章　T-HEAD CPU 调试技巧

5.1　中断和异常

【E804】异常处理机制：

第一步，处理器保存 PSR 和 PC 到影子寄存器（EPSR 和 EPC）中。

第二步，将 PSR 中的超级用户模式设置位 S 位置 1（不管发生异常时处理器处于哪种运行模式），使处理器进入超级用户模式。

第三步，将 PSR 中的异常向量号 VEC 域更新为当前发生的异常向量号，标识异常类别以及支持共享异常服务的情况。

第四步，将 PSR 中的异常使能位 EE 位清零，禁止异常响应。在 EE 为零时发生的任何异常（除了普通中断），处理器都将其作为不可恢复错误异常处理。不可恢复的错误异常发生时，EPSR 和 EPC 也会被更新。

第五步，将 PSR 中的中断使能位 IE 位清零，禁止响应中断。以上第二步、第三步和第四步，同时发生。

第六步，处理器首先根据 PSR 中的异常向量号计算得到异常入口地址，然后用该地址获得异常服务程序的第一条指令的地址。将异常向量乘以 4 后加上异常向量基准地址（存在向量基准地址寄存器 VBR 中，当 VBR 不存在时该值恒为零）即得到异常入口地址，以该异常入口地址从存储器中读取一个字，并将该字的 [31:1] 转载到程序计数器中作为异常服务程序的第一条指令的地址（PC 的最低位始终是 0，与异常向量表中取得的异常入口地址值的最低位无关）。对于向量中断，异常向量由外部的中断控制器提供；对于其它的异常，处理器根据内部逻辑决定异常向量。

最后，处理器从异常服务程序的第一条指令处开始执行并将 CPU 的控制权转交给异常服务程序，开始异常的处理。

5.1.1　中断不来问题

1）先查外设模块的中断状态位和使能位

2）再查中断控制器的中断状态位和使能位

3）再查 CPU 的控制器寄存器，例如 CSKY CPU 的 PSR 的 EE/IE 位、RISCV 的 MIE 等

4）如果步骤 1-3 都检查过了，中断还是不来，那么有可能是优先级高的中断没有退出（仅限于有嵌套中断控制器的情况）。

5.1.2 出异常调试

例如：方法：设置断点到 Default_Handler 处（异常向量入口），出现一场后，停在了 Default_Handler gdb 输入如下命令，查看全部寄存器信息：

(gdb) i r a

2）步骤二，查看异常原因

【CSKY 架构】

1）看 epc，出异常的 pc 地址，找到对应 C 函数或者汇编出错点

2）看 psr[16:23]，查看异常类型 1、断点断在 Default_Handler 2、停下来后，输入如下命令查看函数调用栈，定位出错点：

(gdb) set $pc=$epc

(gdb) bt

【RISCV 架构】

1）看 mepc，出异常的 pc，找到对应 C 函数或者汇编出错点

2）看 mcause，查看异常类型 1、断点断在 Default_Handler 2、停下来后，输入如下命令查看函数调用栈，定位出错点：

(gdb) set $pc=$mepc

(gdb) bt

5.1.3 程序跑飞

程序中有回调函数未初始化，或者堆栈溢出，会导致程序跑飞，即 pc 指针乱指。此时可以使用 gdb 的 pctrace 功能追踪，(gdb bt 功能由于栈被冲掉失效)。方法如下：

例如程序跑飞到 0x00000000 地址：

1）先设置硬断点到 0x0 地址

*(gdb) hb *0x0*

2）运行程序，当 pc 指针为 0x0 时，cpu 会进入调试模式

(gdb) c

3）输入 pctrace 命令，得到 8 条 PC 跳转的轨迹

(gdb) pctrace

5.1.4 非法指令异常

例如 C 程序在编译的时候由于编译选项不对，导致编译出与架构不符合的指令，导致非法指令异常。方法如下：

例如 E804 编译出 DSP 指令：

1）先设置硬断点到异常入口地址，如：Default_Handler

(gdb) hb Default_Handler

2）运行程序，当 pc 运行到非法指令时，cpu 会进入调试模式，gdb 查看 epc

(gdb) i r mepc

3）反汇编 epc 地址处汇编，查看非法指令

(gdb) set $pc=$mepc

(gdb) disassemble $pc , $pc+offset

4. 若 epc 处反汇编为 DPS 指令，则修改编译选项，确保不再编译出 DSP 指令

5.1.5 访问错误异常

例如 C 程序在编译的时候由于地址空间不对或者程序堆栈溢出等问题，导致访问了非法地址的指令，导致访问错误异常。方法如下：

例如 E804 访问非法地址：

1）先设置硬断点到异常入口地址，如：Default_Handler

(gdb) hb Default_Handler

2）运行程序，当 pc 运行到非法地址访问时，cpu 会进入调试模式，gdb 查看 epc

(gdb) i r mepc

3）反汇编 epc 地址处汇编，查看非法地址

(gdb) set $pc=$mepc

(gdb) disassemble $pc , $pc+offset

4）epc 一般为 load/store 指令，查看具体访问地址非法指令，找出对应 C 代码位置。

(gdb) i r

第六章　IDE 调试介绍

1. CDS 调试介绍，具体见 CDS help 文档。
2. CDK 调试介绍，具体见 CDK help 文档。