SDB简易调试器：实现监视点

😁 监视点的功能

通过监视点我们能够更加清楚地得知程序运行背后你想知道的某些值的变化，从而在进行debug的时候清晰地定位到问题所在。

😁 监视点实现思路

1.监视点的结构体实现：

从PA的文档中得知，监视点的核心功能检测就是cpu每执行一条指令之前，寄存器的值是否发生改变，所以在结构体中需要添加两个成员变量一个是原寄存器的值old_value,另外一个是运行程序后寄存器新的值new_value。由于我们是通过监视点池wp_pool[NR_WP]来管理监视点，所以我们还需要通过设置标志位use_flag来标记哪些监视点使用过。

2.监视点的创建函数和删除函数实现：

WP* new_wp() 实现思路：遍历监视点池，如果当前监视点的use_flag还没被使用，就使用当前监视点池下标创建监视点。

void free_wp(WP* wp) 实现思路：链表的删除操作。

3.监视点功能函数的实现：

void check_watchpoint（）实现思路：通过每次比较监视点old_value和new_value，如果old_value和new_value的值是相等的，就继续执行。否则暂停，进入到sdb_mainloop()中等待用户命令。

😁 监视点实现的难点

1.在实现监视点的创建和监视点的删除时，需要使用工作指针，如果处理思路不清晰，很容易出现指针乱指对象，从而引发程序的一些bug。

2.如何进行设计监视点功能函数，每次执行程序之前扫描的寄存器的值，判断寄存器的old_value和new_value是否相等，如果不相等程序就暂停，否则就继续运行。

😁 在实现监视点的时候遇到的问题，以及解决方案

遇到的问题1：

在cpu/cpu-exec.c下使用wp_pool这个监视点池。但是不管如何进行链接都是失败的，一直显示无法识别这个变量。

解决方案：

原因是这个线程池是使用static进行修饰，所以只能在watchpoint.c这个文件夹下使用。

遇到的问题2：

既然无法在cpu-exe.c识别出wp_pool这个监视点池，那么如何对监视点进行一个扫描。

解决方案：

但是后来阅读文档和捋清思路后发现实现监视点功能函数的思路出了点问题，我需要扫描的不是监视点池，而是组织监视点的链表。通过扫描链表中每个监视点的new_value(执行指令后的新值)是否等old_value(原来的值)，来判断每次执行程序前监视点值有没有变化，如果发生变化，就停止运行。所以后面在程序watchpoint.c中实现check_watchpoint()这个函数来扫描监视点。

遇到的问题3：

没有正确实现监视点功能函数，以至于nemu进入暂停时机错误，触发bug。

解决方案：

经过思考是由于在第一次比较old-value和new-value不同之后，退出程序之前没有将old-value的值更新成new-value的值。所以后面每次扫描都会触发暂停。修改代码更新old_value的值之后，这样在执行后面的指令的时候才不会因为每次执行前扫描监视点设置的值前后不一致而不断进入暂停状态。