【嵌入式】堆栈8字节对齐

转载

AsCoolCucmber 于 2019-12-14 20:41:01 发布 893 收藏 6

文章标签： 嵌入式 stm32 堆栈

原文链接：https://www.cnblogs.com/sky1991/archive/2012/10/13/2722482.html

一、为什么要保证堆栈8字节对齐

AAPCS规则要求堆栈保持8字节对齐。

如果不对齐，调用一般的函数也是没问题的。但是当调用需要严格遵守AAPCS规则的函数时可能会出错。

例如调用sprintf输出一个**浮点数**时，栈必须是8字节对齐的，否则结果可能会出错。

实验验证：

```
#include "stdio.h"
#include "string.h"
float fff=1.234;
char buf[128];
int main(void)
{
 sprintf(buf,"%.3f\n\r",fff); //A
 while(1);
}
```

1. 在A处设置断点，让程序全速运行至A
2. 在MDK中修改MSP的值使MSP满足8字节对齐
3. 全速运行程序，观察buf中的字符为 1.234 结果正确
4. 回到第2步，修改MSP使之只满足4字节对齐而不满足8字节对齐
5. 全速运行程序，观察buf中的字符为 -2.000 结果错误

该实验证明了调用sprintf输出一个浮点数必须要保证栈8字节对齐。

二、编译器（底层汇编）是怎么做的

先看一个实验

```
#include "stdio.h"
#include "string.h"
float fff=1.234;
char buf[128];

void fun(int a,int b,int c,int d){ int v; v=v;}
void test(void){}

int main(void)
{
 fun(1,2,3,4);
 test();//A
 while(1);
}
```

0. 保证初始的时候堆栈是8字节对齐的
1. 在A处设置断点
2. 全速运行至A，观察MSP=0x2000025c，没有8字节对齐
   (为什么说没有对齐呢？因为MSP的结尾是c，说明是一个奇数，如果是有对齐的话必然是偶数，这是微机原理的知识——规则字)
3. 略微修改一下main函数代码如下，其他部分代码不变

```
int main(void)
{
 fun(1,2,3,4);
 //test();
 sprintf(buf,"%.3f\n\r",fff);//A
 while(1);
}
```

4. 同样在A处设置断点
5. 全速运行至A，观察MSP=0x200002d8，这次8字节对齐了

这个实验说明了如果编译器发现了某个函数需要调用浮点库时会自动调整编译生成的汇编代码，从而保证调用这些浮点库函数时堆栈是8字节对齐的。换句话说如果我们保证了栈初始的时候是8字节对齐的，那么编译器可以保证以后调用浮点库时堆栈仍是8字节对齐的。

三、os应该怎样设置任务堆栈

由上面的讨论可知给任务分配栈时需要保证栈是8字节对齐的，不然在该任务中凡是调用sprintf的函数均会出错，因为栈一开始就是不对齐的。

四、中断中的栈对齐问题

是否保证了栈初始是8字节对齐了就万事大吉了呢?
——NO!
大家请看一种特殊的情况：

```
#include "stdio.h"
#include "string.h"
float fff=1.234;
char buf[128];
void fun(int a,int b,int c,int d){ int v; v=v;}
int main(void)
{
 fun(1,2,3,4);
 while(1);
}
void SVC_Handler(void){ sprintf(buf,"%.3f\n\r",fff); //B}
```

main函数的反汇编如下：

```
0x080001DC B500 PUSH {lr}
0x080001DE 2304 MOVS r3,#0x04 ;A
0x080001E0 2203 MOVS r2,#0x03
0x080001E2 2102 MOVS r1,#0x02
0x080001E4 2001 MOVS r0,#0x01
0x080001E6 F7FFFFF5 BL.W fun (0x080001D4)
0x080001EA BF00 NOP
0x080001EC E7FE B 0x080001EC
```

0. 保证初始的时候堆栈是8字节对齐的
1. 在A处设置断点
2. 全速运行至A，观察此时MSP=0x200002e4 未对齐
3. 在MDK中将SVC的挂起位置1
4. 在B处设置断点
5. 全速运行至B，观察此时MSP=0x200002b4 未对齐
6. 继续全速执行，观察buf中的字符为:-2.000 出错了

这个实验说明了即使保证栈初始是8字节对齐的，编译器也只能保证在调用sprintf那个时刻栈是8字节对齐的
但不能保证任意时刻栈都是8字节对齐的，如果恰巧在MSP没有8字节对齐的时刻发生了中断，而中断中又调用了sprintf，这种情况下仍会出错

五、Cortex内核为我们做了什么

Cortex内核提供了一种硬件机制来解决上述这种中断中栈不对齐问题。
CM3中可以把NVIC配置控制寄存器的STKALIGN置位，来保证中断中的栈8字节对齐，
具体实现过程如下：
当发生中断时由硬件自动检测MSP是否8字节对齐，如果对齐了，则不进行任何操作，
如果没有对齐，则自动将MSP减4这样便对齐了，同时将xPSR的第9位置位来记录这个MSP的非正常的变化，在中断返回若发现xPSR的第9位是置位的则自动将MSP加4调整回原来的值。

实验验证：

```
#include "stdio.h"
#include "string.h"
float fff=1.234;
char buf[128];
void fun(int a,int b,int c,int d){ int v; v=v;}
int main(void)
{
 fun(1,2,3,4);
 while(1);
}
void SVC_Handler(void){ sprintf(buf,"%.3f\n\r",fff); //B}
```

mian函数的反汇编如下：

```
0x080001DC B500 PUSH {lr}
0x080001DE 2304 MOVS r3,#0x04 ;A
0x080001E0 2203 MOVS r2,#0x03
0x080001E2 2102 MOVS r1,#0x02
0x080001E4 2001 MOVS r0,#0x01
0x080001E6 F7FFFFF5 BL.W fun (0x080001D4)
0x080001EA BF00 NOP
0x080001EC E7FE B 0x080001EC
```

1. 在A处设置断点
2. 全速运行至A，观察此时MSP=0x200002e4 未对齐
3. 在MDK中将SVC的挂起位置1，同时将0xE000ED14处的值由0x00000000改为0x00000200 (即将NVIC配置控制寄存器的STKALIGN置位)
4. 在B处设置断点
5. 全速运行至B，观察此时MSP=0x200002b0 对齐了
6. 观察中断返回时的MSP=0x200002e4 调整回来了
7. 继续全速执行，观察buf中的字符为:1.234 正确

这个实验说明了将NVIC配置控制寄存器的STKALIGN置位可以保护中断时栈仍是8字节对齐

六、总结

综上所述，为了能够安全的使用严格遵守AAPCS规则的函数(比如sprintf)需要做到以下几点：

1. 保证MSP在初始的时候是8字节对齐的
2. 如果用到OS的话需要保证给每个任务分配的栈是保持8字节对齐的
3. 如果用的是基于CM3内核的处理器需将NVIC配置控制寄存器的STKALIGN置位