一、代码重定位及内存控制

1.代码重定位相关概念

运行地址：程序运行中当前指令代码所在的存储单元地址；

链接地址：编译程序所指指定的程序运行时应该处在的运行地址；

位置无关代码：代码中所使用的都是相对地址(PC数值加上偏移量),代码不在链接时指定的运行空间地址，也可以执行；

位置有关代码：地址与代码所处位置相关，使用了绝对地址，位置有关代码要求链接地址和运行地址必须一致；

重定位：当程序的加载地址与链接地址不一致时，如果使用位置相关代码则需要重定位。重定位的操作是把链接脚本中需要重定位的段的内容复制到链接地址处，重定位只需重定位代码段(text)和数据段(data)。

```
SECTION  // 段声明
{
  .=0xD0024000; //定义段首链接地址
.text:
     {
      start.o //指明首先链接start.o文件
      *(.text)
     }

.bss:
     {
      *(.bss) //定义bss段并且跟在text段之后
     }
.data:
     {
      *(.data) //定义data段
     }

}
```

在第一行中，"."一个特殊符号，它是定位器，一个位置指针，指向程序地址空
随后，".text:"后面的花括号内列出了应该放入这个输出段中的输入段的名字，

2.代码重定位汇编源码

adr伪指令:

adr伪指令是小范围地址寻址伪指令，将基于PC的地址或基于寄存器的地址读取到寄存器中。adr都到的地址是位置无关的地址，如果adr指令中的地址是基于PC的，则改地址与adr伪指令通过PC加偏移的方式获取标号地址，是位置无关的。

```
start:
      mov r0,#0x10
      adr r4,start
      在上面的程序中，start的地址加载给寄存器r4，但是此地址是基于PC的，
```

相关代码分析:

Makefile文件

```
link.bin: start.o main.o
          arm-linux-ld -Tlink.lds -o link.elf $^        //指定链接文
          arm-linux-objcopy -O binary link.elf link.bin//将elf文件
          arm-linux-objdump -D link.elf > link_elf.dis //进行反汇编
          gcc mkv210_image.c -o mkmini210               //加校验头
          ./mkmini210 link.bibn 210.bin                 //将校验头加
%.o : %.S
          arm-linux-gcc -o $@ $< -c
%.o : %.c
          arm-linux-gcc -o $@ $< -c
clean:
          arm *.o *.elf *.bin *.dis mkmini210 -f
```

Link.lds文件

```
SECTION
{
  .=0xD0024000;       //链接地址的定位
.text:
     {
      start.o
      *(.text)
     }

.data:
     {
      *(.data)
     }
bss_start = .;       //定义符号,bss的起始地址
.bss:
     {
      *(.bss)
     }
bss_end = .;         //定义符号,bss的起始地址
}
bss_end = .;
```

start.S文件

```
.global _start
_start:
       //关闭看门狗
       ldr r0, =0xE2700000
       mov r1, #0
       str r1, [r0]
       // 指定堆栈指针
       ldr sp, =0xD0037D80

       //adr指令是相对地址
       adr r0, _start
```

```
        //ldr指令是绝对地址即链接地址送给r1
        ldr r1, =_start
        ldr r2, =bss_start
        //比较r0和r1的数值是否相等
        cmp r0, r1
        //如果r0和r1的数值相等则进行跳转否则顺序执行
        beq clean_bss

copy_loop:
        //将r0地址中的内容放进r3寄存器然后r0的地址自增
        ldr r3, [r0], #4
        //将r3中的内容放进r1所指向的内存中然后r1地址自增
        str r3, [r1], #4
        //比较r1和r2是否相等,若相等则意味copy完成否怎继续执行copy_loop
        cmp r1, r2
        bne copy_loop
clean_bss:
        //判断bss段是否有数据,如果有则请零没有则直接跳转执行
        ldr r0, =bss_start
        ldr r1, =bss_end
        cmp r0, r1
        beq run_one_dram
        mov r2, #0
clean_loop:
        //将bss段数据进行清零
        str r2, [r0], #4
        cmp r0, r1
        bne clean_loop
run_on_dram:
        //跳进c文件的main函数进行执行程序
        ldr pc, =main
halt:
        b halt
```

main.c文件

```
#define GPJ2CON (*(volatile unsigned long *) 0xE0200280)
#define GPJ2DAT (*(volatile unsigned long *) 0xE0200284)
void delay(unsigned long count)
{
   volatile unsigned long i = count;
   while(i--);
}
void main()
{

   GPJ2CON = 0x00001111;
   while(1)
   {
      GPJ2DAT = 0;
      delay(0x100000);
      GPJ2DAT = 0xF;
      delay(0x100000);
   }
}
```

结尾:

初学S5PV210将其分段整理成笔记供自己参考也供与大家学习，如有错误请大佬们直言指出，如果感觉有用那就点个赞留个言，谢谢观众老爷们的赏脸。

若想获得上述内容的PDF版本移步到GitHub下载。

地址：https://github.com/QianquanChina/Study-Notes

-----缱绻