最近跟着韦老师视频学习嵌入式linux，毕业多年，在学校学的汇编知识早已还给老师，也不可能从头看书<<汇编语言>>，那么今天就来复习一下常用的ARM指令。养成良好习惯，也方便以后查找,下面很多资料也参考了网上博客的，感谢前辈们的付出。

**常用ARM指令的功能介绍**

**1．汇编伪指令**

（1）  
ENTRY

ENTRY 伪指令用于指定汇编程序的入口点。在一个完整的汇编程序中至少要有一个ENTRY，但在一个源文件里最多只能有一个ENTRY（也可以没有）。

（2）  
伪指令AREA

语法格式：

AREA 段名      属性1,属性2

AREA用于定义一个代码段或数据段。其中，段名若以数字开头，则该段名

需要”|”括起来，如|1\_test|。

属性字段表示代码段（或数据段）的相关属性，多个属性用逗号分隔。常用的属性如下：

CODE属性，用于定义代码段，默认为READONLY。

DATA属性，用于定义数据段，默认为READWRITE。

READONLY属性，指定本段只读，代码段默认为READONLY。

除了上面的属性，还有ALIGN属性和COMMON属性。

（3）  
伪指令EQU

EQU用于为程序中的常量、标号等定义一个等效的字符名称，类似于C语言中的#define。其中EQU可用“\*”代替。

语法格式：

名称 EQU 表达式{,类型}

（4）  
伪指令END

END用于通知编译器已经到了源程序的结尾。

**2．加载/存储指令**

ARM微处理器支持加载/存储指令，用于在寄存器和存储器之间传送数据，加载指令用于将存储器中的数据传送到寄存器，存储指令则完成相反的操作。常用的加载存储器指令有字数据加载指令LDR、字节数据加载指令LDRB、半字数据加载指令LDRH、字数据存储指令STR、字节数据存储指令STRB及半字数据存储指令STRH。

实验中主要用到了LDR和STR指令，下面将介绍两条指令的用法（其他指令的用法可度娘看下）

（1）  
LDR指令

LDR指令的格式为：

LDR{条件} 目的寄存器,&lt;存储器地址&gt;

LDR指令用于从存储器中将一个32位的字数据传送到目的寄存器中。该指令通常用于从存储器中读取32位的字数据到通用寄存器，然后对数据进行处理。当程序计数器PC作为目的寄存器时，指令从存储器中读取的字数据被当作目的地址，从而可以实现程序流程的跳转。该指令在程序中比较常用，且殉职方式灵活多样。例如、

1. LDR
2. r0,=0x56000054 //将地址0x56000054存放到寄存器r0中
3. LDR r2, =SDRAM\_BASE //参看sdram试验代码，结合
4. .equ SDRAM\_BASE, 0x30000000 //语句，是将0x30000000这个地址（也是个32位的字数据）放到r2寄存器中
5. LDR PC, =on\_sdram //跳到SDRAM中继续执行
6. LDR r0,0x56000054 //将内存0x56000054地址中存储的数据搬到r0寄存器中

复制代码

（2）  
STR指令

STR指令的格式为：

STR{条件} 源寄存器,&lt;存储器地址&gt;

STR指令用于从源寄存器中将一个32位的字数据传送到存储器中。该指令在程序设计中比较常用，且寻址方式灵活多样，使用方式可以参考LDR。例如：

LED实验中的一段代码

1. LDR R0,=0x56000054         //R0设为GPBDAT寄存器。此寄存器用于读/写端口B各引脚的数据
2. MOV R1,#0x00000000       //此值改为0x00000010,可让LED1熄灭
3. STR  R1,[R0]                       //GPF4输出0，LED1点亮，将R1中的数//据0x0000000存放到寄存器R0中

复制代码

**3．数据处理指令**

数据处理指令可分为数据传送指令、算术逻辑运算指令和比较指令等。数据传送指令用于在寄存器和存储器之间进行数据的双向传输。算术逻辑运算指令完成常用的算术与逻辑运算，该类指令不但将运算结果保存在目的寄存器中，同时更新CPSR中的相应条件标志位。比较指令不保存运算结果，只更新CPSR中对于得条件标志位。

（1）  
MOV指令

MOV指令的格式为：

MOV{条件}{S} 目的寄存器，源操作数

MOV指令可完成从另外一个寄存器、被移位的寄存器或将一个立即数加载到目的寄存器。其中S选项决定指令的操作是否影响CPSR中条件标志位的值，当没有S时，指令不更新CPSR中条件标志位的值。例如：

MOV PC,LR //将寄存器LR的数据传送到PC（R15）寄存器中

（2）  
CMP指令

CMP{条件} 操作数1，操作数2

CMP指令用于吧一个寄存器的内容和另外一个寄存器的内容或立即数进行比较，同时更新CPSR中条件标志位的值。该指令进行一次减法运算，但不存储结果，只更改条件标志位。标志位表示的数是操作数1与操作数2的关系（大、小、相等）。例如

SDRAM实验中的39行代码

1. cmp r1, r3                       //将R1寄存器的值与R3寄存器的值相比较，
2. //结合上下代码在实验中的意思判断是否完成：源地址等于Steppingstone的末地址？

（3）  
跳转指令

跳转指令用于实现持续流程的跳转，在ARM程序中有两种方法可以实现程序流程的跳转：

使用专门的跳转指令。

直接向程序计数器PC写入跳转地址值

通过向程序计数器PC写入跳转地址值，可以实现在4GB的地址空间任意跳转，在跳转之前结合使用。“MOV LR，PC” 等类似指令可以保存将来的返回地址，从而实现在4GB连续的线性地址空间的子字程序调用。ARM指令集中的跳转指令可以完成从当前指令向前或向后的32MB地址空间的跳转，包括以下4条指令：

B 跳转指令；

BL 带返回的跳转指令；

BLX 带返回和状态切换的跳转指令；

BX 带状态切换的跳转指令。

1）  
B指令

B指令是最简单的跳转指令。一旦遇到一个B指令，ARM处理器将立即跳转到给定的目标地址，从那里继续执行。注意，存储在跳转指令中的实际值是相对当前PC值得一个偏移量，而不是一个绝对地址，它的指由汇编器来计算（参考寻址方式中的相对寻址）。它是24位有符号数，左移两位后又符号扩展为32位，表示的有效偏移为26位（前32MB的地址空间）。例如：

1. halt\_loop:                                                 //这两句意思是一致跳转下去。。死胡同
2. b  halt\_loop                              //跳转到halt\_loop程序段

2）  
BL指令

BL指令的格式为：

BL{条件} 目标地址

BL是另外一个跳转指令，但跳转之前，会在寄存器R14种保存PC的当前内容，因此，可以通过R14的内容重新加载到PC中来返回到跳转指令之后的那个指令处执行。该指令的实现程子程序条用的一个基本但常用的手段。例如：

1. bl  main  //调用主程序

复制代码

具体实例（在JZ2440开发板SDRAM实验中的head.S代码）

1. @\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. @ File：head.S
3. @ 功能：设置SDRAM，将程序复制到SDRAM，然后跳到SDRAM继续执行
4. @\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
5. .equ MEM\_CTL\_BASE, 0x48000000
6. .equ SDRAM\_BASE, 0x30000000
7. .text
8. .global \_start
9. \_start:
10. bl disable\_watch\_dog @ 关闭WATCHDOG，否则CPU会不断重启
11. bl memsetup @ 设置存储控制器
12. bl copy\_steppingstone\_to\_sdram @ 复制代码到SDRAM中
13. ldr pc, =on\_sdram @ 跳到SDRAM中继续执行
14. on\_sdram:
15. ldr sp, =0x34000000 @ 设置堆栈
16. bl main
17. halt\_loop:
18. b halt\_loop
19. disable\_watch\_dog:
20. @ 往WATCHDOG寄存器写0即可
21. mov r1, #0x53000000
22. mov r2, #0x0
23. str r2, [r1]
24. mov pc, lr @ 返回
25. copy\_steppingstone\_to\_sdram:
26. @ 将Steppingstone的4K数据全部复制到SDRAM中去
27. @ Steppingstone起始地址为0x00000000，SDRAM中起始地址为0x30000000
28. mov r1, #0
29. ldr r2, =SDRAM\_BASE
30. mov r3, #4\*1024
31. 1:
32. ldr r4, [r1],#4 @ 从Steppingstone读取4字节的数据，并让源地址加4
33. str r4, [r2],#4 @ 将此4字节的数据复制到SDRAM中，并让目地地址加4
34. cmp r1, r3 @ 判断是否完成：源地址等于Steppingstone的未地址？
35. bne 1b @ 若没有复制完，继续
36. mov pc, lr @ 返回
37. memsetup:
38. @ 设置存储控制器以便使用SDRAM等外设
39. mov r1, #MEM\_CTL\_BASE @ 存储控制器的13个寄存器的开始地址
40. adrl r2, mem\_cfg\_val @ 这13个值的起始存储地址
41. add r3, r1, #52 @ 13\*4 = 52 //每个寄存器都是四个字节，13个寄存器共13\*4=52个字节
42. 1:
43. ldr r4, [r2], #4 @ 读取设置值，并让r2加4
44. str r4, [r1], #4 @ 将此值写入寄存器，并让r1加4
45. cmp r1, r3 @ 判断是否设置完所有13个寄存器
46. bne 1b @ 若没有写成，继续
47. mov pc, lr @ 返回
48. .align 4
49. mem\_cfg\_val:
50. @ 存储控制器13个寄存器的设置值
51. .long 0x22011110 @ BWSCON
52. .long 0x00000700 @ BANKCON0
53. .long 0x00000700 @ BANKCON1
54. .long 0x00000700 @ BANKCON2
55. .long 0x00000700 @ BANKCON3
56. .long 0x00000700 @ BANKCON4
57. .long 0x00000700 @ BANKCON5
58. .long 0x00018005 @ BANKCON6
59. .long 0x00018005 @ BANKCON7
60. .long 0x008C07A3 @ REFRESH
61. .long 0x000000B1 @ BANKSIZE
62. .long 0x00000030 @ MRSRB6
63. .long 0x00000030 @ MRSRB7

复制代码

后续再做个汇编与C语言的混合编程笔记