PA1第一阶段实验报告

实现正确的寄存器结构体：

对gpr,一个gpr就是一个32位寄存器，其中每个32位寄存器与一个16位寄存器共享后16位内存，所以对uint32\_t \_32和uint32\_t \_16使用union，而16位寄存器又与2个8位寄存器共享内存，所以对uint16\_t \_16和uint8\_t \_8[2]使用union，这构成了一个32位寄存器。

一个gpr与eax等有具体名称的一个32位寄存器相同，所以对gpr和8个有具体名称的32位寄存器使用union，由于eax等寄存器内存连续，所以把eax等放入一个无名struct中，使其内存连续。

代码如下：

typedef struct {

18

19 union{

20 union{

21

22 uint32\_t \_32;

23 union{

24 uint16\_t \_16;

25 uint8\_t \_8[2];

26 };

27

28 } gpr[8];

29 struct{

30

31 uint32\_t eax, ecx, edx, ebx, esp, ebp, esi, edi;

32 };

33

34 };

35 /\* Do NOT change the order of the GPRs' definitions. \*/

36

37 //uint32\_t eax, ecx, edx, ebx, esp, ebp, esi, edi;

38

39 swaddr\_t eip;

40

41 } CPU\_state;

单步执行

形式：si [N] 说明：让程序单步执行 N 条指令后暂停执行,

当 N 没有给出时, 缺省为 1

单步执行的功能十分简单, 而且框架代码中已经给出了模拟CPU执行方式的函数, 为cpu\_exec()，只要使用相应的参数去调用它就可以了。

另外，用

for(i=0;i<len;i++)

sum=sum\*10+\*(arg+i)-'0'

来解析出需要执行的步数。

代码如下：

83 static int cmd\_si(char \*args) {

84 char \*arg = strtok(NULL, " ");

85 int i,sum=0,len;

86 if(arg == NULL) {

87 cpu\_exec(1);

88 }

89 else{

90 len=strlen(arg);

91 for(i=0;i<len;i++)

92 sum=sum\*10+\*(arg+i)-'0';

93 cpu\_exec(sum);

94 }

95 return 0;

96 }

97

打印寄存器

形式：info SUBCMD 说明：打印寄存器状态

打印寄存器也很简单, 执行 info r 之后, 直接用 printf() 输出所有寄存器的值即可。这里，我分别输出了两次，一次以16进制格式输出，一次以无符号int形式输出。

代码如下：

98 static int cmd\_info(char \*args){

99 char \*arg = strtok(NULL, " ");

100 if((\*arg)=='r'){

101 printf("reg meaning\n");

102 printf("eax %%x:0x%x %%u:%u\n",cpu.eax,cpu.eax);

103 printf("edx %%x:0x%x %%u:%u\n",cpu.edx,cpu.edx);

104 printf("ecx %%x:0x%x %%u:%u\n",cpu.ecx,cpu.ecx);

105 printf("ebx %%x:0x%x %%u:%u\n",cpu.ebx,cpu.ebx);

106 printf("ebp %%x:0x%x %%u:%u\n",cpu.ebp,cpu.ebp);

107 printf("esi %%x:0x%x %%u:%u\n",cpu.esi,cpu.esi);

108 printf("edi %%x:0x%x %%u:%u\n",cpu.edi,cpu.edi);

109 printf("esp %%x:0x%x %%u:%u\n",cpu.esp,cpu.esp);

110 }

111 return 0;

112 }

扫描内存

形式：x N EXPR 说明：求出表达式 EXPR 的值, 将结果作为起始内存

地址, 以十六进制形式输出连续的 N 个4字节

这里规定表达式 EXPR 中只能是一个十六进制数，对命令进行解析之后, 输出连续N个连续的四字节内存的值，这里没有按内存序号递增输出，而是对每个四字节以内存序号从高到低输出，使其类似一个int的形式。

其中，以arg指向个数N的首地址，用sum计算出N，以a指向16进制内存的首地址，用unsigned int s计算出起始内存地址，并在swaddr\_read(uint32\_t,size\_t)函数中传递给uint32\_t。

代码如下：

113 static int cmd\_x(char \*args){

114 char \*arg = strtok(NULL, " ");

115 int i=0,len,lena,sum=0;

116 unsigned int s=0;

117 char \*a=strtok(NULL," ");

118 len=strlen(arg);

119 lena=strlen(a);

120 for(i=0;i<len;i++)

121 sum=sum\*10+\*(arg+i)-'0';

122

123 for(i=0;i<lena-2;i++)

124 {

125 if(((\*(a+i+2)-'0')>=0)&&((\*(a+i+2)-'0')<=9))

126 s=s\*16+\*(a+i+2)-'0';

127 else{

128 s=s\*16+\*(a+i+2)-'a'+10;

129 }

130 }

131

132 uint32\_t x;

133 for(i=0;i<sum;i++){

134 x= swaddr\_read(s+3,1);

135 printf("0x %x ",x);

136 x= swaddr\_read(s+2,1);

137 printf("%x ",x);

138 x= swaddr\_read(s+1,1);

139 printf("%x ",x);

140 x= swaddr\_read(s,1);

141 printf("%x",x);

142 printf("\n");

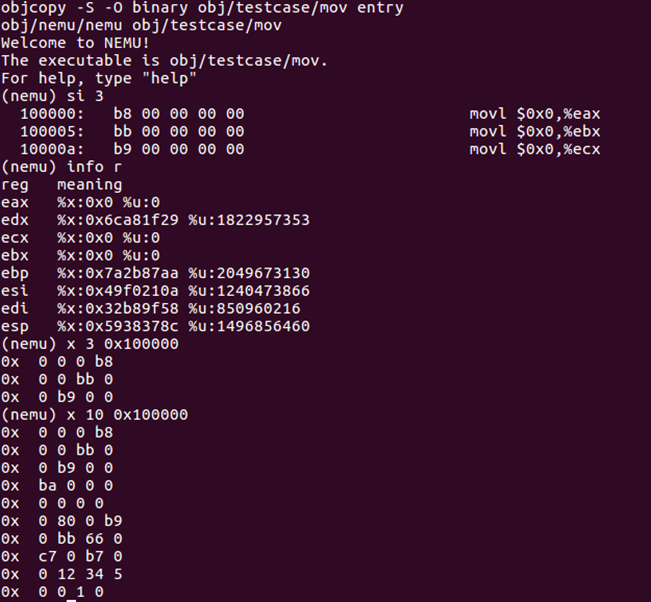
143 s=s+4;

144 }

145 return 0;

146 }

最终运行结果如下所示：



必答题：



CF 50-55

标志位CF是进位标志位（Carry Flag）。

当两个数相加时，若最高位向上形成进位，则CF=1；

当两个数相减时，若最高位向上形成借位，则CF=1；

当两个无符号数相乘时，若乘积的高一半为0，则CF=0；

当两个带符号数相乘时，若乘积的高一半是低一半的符号扩展，则CF=0。

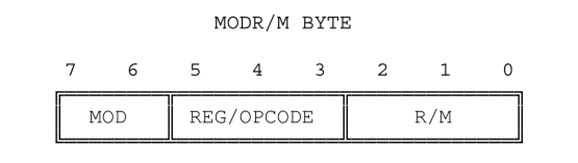
ModR/M 240-248 412-416

The ModR/M byte contains three fields of information：

The mod field, which occupies the two most significant bits of the byte, combines with the r/m field to form 32 possible values: eight registers and 24 indexing modes

The reg field, which occupies the next three bits following the mod field, specifies either a register number or three more bits of opcode information. The meaning of the reg field is determined by the first (opcode) byte of the instruction.

The r/m field, which occupies the three least significant bits of the byte, can specify a register as the location of an operand, or can form part of the addressing-mode encoding in combination with the field as described above



mov 345-351

mov 通用寄存器,立即数 (如:mov ax,1000h)

mov 通用寄存器,通用寄存器 (如:mov ax,bx)

mov 段寄存器,通用寄存器 (如:mov ds,ax )

mov 通用寄存器,内存单元 (如:mov ax,ds:[0])

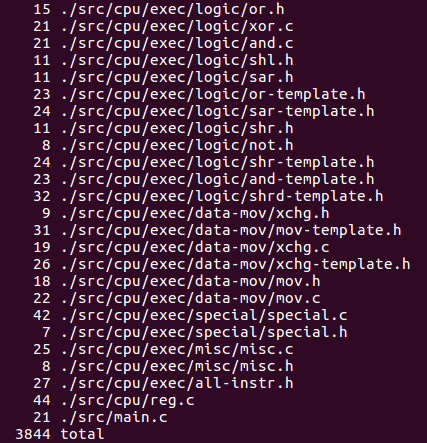
mov 内存单元,通用寄存器 (如:mov ds:[0],ax)

mov 内存单元,立即数 (如:mov ds:[0],1000h)

1. 用命令：wc -l `find -name '\*.\*'`

具体例子如下：





共3833 lines

3.gcc -Wall和-Werror

-Wall 打开gcc的所有警告

-Werror，它要求gcc将所有的警告当成错误进行处理

遇到的问题：

刚开始实现寄存器结构的时候，我想的是把gpr和eax等8个寄存器单独实现，对eax等和gpr之中的uint32\_t使用union

如

union{uint32\_t...}gpr[8];

union{gpr[0].\_32;uint32\_t eax};union{};...

后来由于gpr已经定义了，在后面就不能重新定义，后面就总是报错，后面在室友的帮助下认识到其实gpr[0]和gpr[0].\_32所占内存其实是一样的，gpr[0]只是也是union的名字而已，后面就做出来了，汉。。。

之后就没什么大问题了，可以不断推进，主要是耐心读代码，找接口和查资料。

一些想法：

我现在觉得，对一个稍大的工程来说，你在很多文件的某个文件的某些行加点注释，读的人估计会很难受，几乎要把所有文件的所有代码全部读完才知道到底是讲的什么。一个很有效的办法是，工程的作者在工程的源目录下写一个readme，介绍工程的主要思路和各个文件的主要内容，读者只要花个几十分钟把readme读完就对整个工程有详细具体的了解了。

另外，对于复杂的工程来说，抽象是很有必要的，对工程进行合理地抽象对理解整个工程有很大的帮助。就比如NEMU分成 monitor, CPU, 存储管理, 设备四个部分，在了解了这四个部分的交互之后就可以详细了解每个部分的具体实现了，比如monitor又分成单步执行等各项功能。

最后，我认为看问题有不同的角度是很好的，如果只抱住一个角度不放，看到的东西是有限的，而且很可能是被扭曲的，多个角度能让我们更客观准确地分析问题。