x86寻址方式及使用capstone获得指令信息总结

## 1. 寄存器

32位CPU所含有的寄存器有：

4个数据寄存器(EAX、EBX、ECX和EDX)

2个变址和指针寄存器(ESI和EDI) 2个指针寄存器(ESP和EBP)

6个段寄存器(ES、CS、SS、DS、FS和GS)

1个指令指针寄存器(EIP) 1个标志寄存器(EFlags)

段寄存器中存储段号（segment selector，也译作“段选择符”），再由段号映射到存在内存中的GDT（global (segment) descriptor table，全局段号记录表），读取段的信息。

## 2. 寻址方式

X86 寻址方式有以下几种:

### 1）立即数寻址

数据在指令中， 如： MOV ax, 0

### 2）寄存器寻址

数据在寄存器：如： MOV ax, bx

**以下是数据在内存中的寻址方式。**

### 3）直接寻址

如： MOV ax, [0]

### 4）寄存器间接寻址

如： MOV ax, [bx]

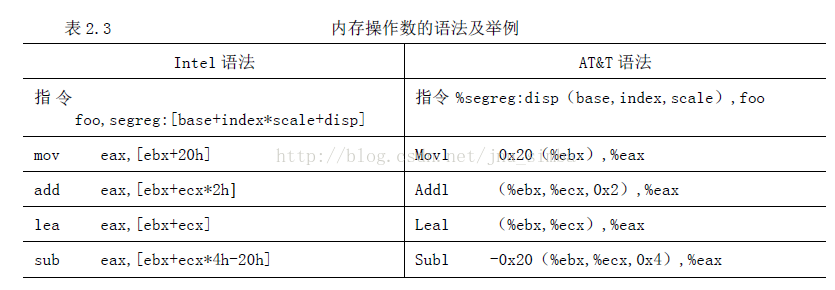
### 5）寄存器相对寻址

段寄存器:[基址寄存器 + 变址寄存器 +位移量]

如：MOV EAX,GS:[EBP+EDI+100H]

比例变址寻址，如：MOV EAX, [EBX+4\*EDI+100H]

## 3. AT&T 与 Intel 语法的比较



## 4.实验

准备汇编代码如下（从汇编文件汇编得到二进制，利用objdump查看汇编结果）

4ac: a1 44 33 22 11 mov 0x11223344,%eax

4b1: 8b 03 mov (%ebx),%eax

4b3: 8b 45 00 mov 0x0(%ebp),%eax

4b6: 8b 02 mov (%edx),%eax

4b8: 8b 43 02 mov 0x2(%ebx),%eax

4bb: 8b 45 04 mov 0x4(%ebp),%eax

4be: 8b 42 06 mov 0x6(%edx),%eax

4c1: 8b 44 3b 02 mov 0x2(%ebx,%edi,1),%eax

4c5: 8b 44 bd 04 mov 0x4(%ebp,%edi,4),%eax

4c9: 8b 44 32 06 mov 0x6(%edx,%esi,1),%eax

4cd: 36 8b 44 3b 02 mov %ss:0x2(%ebx,%edi,1),%eax

4d2: 3e 8b 44 bd 04 mov %ds:0x4(%ebp,%edi,4),%eax

4d7: 65 8b 44 32 06 mov %gs:0x6(%edx,%esi,1),%eax

以下是用于测试的代码(testCapstoneMem.c):

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <capstone/capstone.h>

**#include** <inttypes.h>

//To deep-dive x86 addresing and Capstone x86\_op\_mem structure.

//Compiled with:

// clang -g -m32 testCapstoneMem.c -lcapstone

**int** main(){

csh handle;

cs\_arch arch = CS\_ARCH\_X86;

cs\_mode mode = CS\_MODE\_32;

cs\_insn \*insn;

**char** code[] =

"\xa1\x44\x33\x22\x11"

"\x8b\x03"

"\x8b\x45\x00"

"\x8b\x02"

"\x8b\x43\x02"

"\x8b\x45\x04"

"\x8b\x42\x06"

"\x8b\x44\x3b\x02"

"\x8b\x44\xbd\x04"

"\x8b\x44\x32\x06"

"\x36\x8b\x44\x3b\x02"

"\x3e\x8b\x44\x3b\x02"

"\x65\x8b\x44\x32\x06";

size\_t size = **sizeof**(code);

uint64\_t address = 0x1000; //(ASSUMED) address of the first instruction

**int** err = cs\_open(arch, mode, &handle);

cs\_option(handle, CS\_OPT\_SYNTAX, CS\_OPT\_SYNTAX\_ATT);

cs\_option(handle, CS\_OPT\_DETAIL, CS\_OPT\_ON); //detail is needed

**int** v = (**int**) cs\_disasm(handle, (uint8\_t \*) code, size, address, 0, &insn); //memory of insn is allocated by cs\_disasm()

**int** j;

**if** (v) {

printf("Disasm:\n");

**for** (j = 0; j < v; j++) {

printf("0x%"PRIx64":\t%s\t\t%s\n",

insn[j].address,

insn[j].mnemonic,

insn[j].op\_str);

x86\_op\_mem mem = insn[j].detail->x86.operands[0].mem;

}

} **else** {

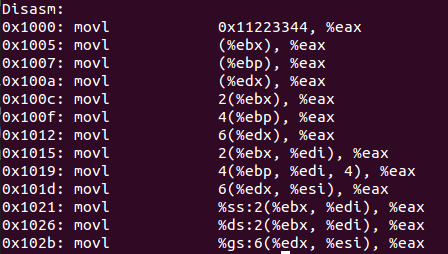
printf("Disasm failed.\n");

}

**return** 0;

}

运行结果：



capstone得到的指令detail信息：

0x1000: movl 0x11223344, %eax

{segment = 19(X86\_REG\_EAX?), base = 0, index = 0, scale = 1, disp = 0x11223344}

0x1005: movl (%ebx), %eax

{segment = 0, base = 21(X86\_REG\_EBX), index = 0, scale = 1, disp = 0}

0x1007: movl (%ebp), %eax

{segment = 0, base = 20(X86\_REG\_EBP), index = 0, scale = 1, disp = 0}

0x100a: movl (%edx), %eax

{segment = 0, base = 24(X86\_REG\_EDX), index = 0, scale = 1, disp = 0}

0x100c: movl 2(%ebx), %eax

{segment = 0, base = 21, index = 0, scale = 1, disp = 2}

0x100f: movl 4(%ebp), %eax

{segment = 0, base = 20, index = 0, scale = 1, disp = 4}

0x1012: movl 6(%edx), %eax

{segment = 0, base = 24, index = 0, scale = 1, disp = 6}

0x1015: movl 2(%ebx, %edi), %eax

{segment = 0, base = 21, index = 23, scale = 1, disp = 2}

0x1019: movl 4(%ebp, %edi, 4), %eax

{segment = 0, base = 20, index = 23, scale = 4, disp = 4}

0x101d: movl 6(%edx, %esi), %eax

{segment = 0, base = 24, index = 29, scale = 1, disp = 6}

0x1021: movl %ss:2(%ebx, %edi), %eax

{segment = 49(X86\_REG\_SS), base = 21, index = 23, scale = 1, disp = 2}

0x1026: movl %ds:2(%ebx, %edi), %eax

{segment = 17(X86\_REG\_DS), base = 21, index = 23, scale = 1, disp = 2}

0x102b: movl %gs:6(%edx, %esi), %eax

{segment = 33, base = 24, index = 29, scale = 1, disp = 6}

### 附：capstone部分寄存器编码

|  |  |  |
| --- | --- | --- |
| **寄存器** | **编码（16进制）** | **编码（10进制）** |
| X86\_REG\_ES | 0x1c | 28 |
| X86\_REG\_DS | 0x11 | 17 |
| X86\_REG\_SS | 0x31 | 49 |
| X86\_REG\_CS | 0xb | 11 |
| X86\_REG\_GS | 0x21 | 33 |
| X86\_REG\_FS | 0x20 | 32 |
| X86\_REG\_EBP | 0x14 | 20 |
| X86\_REG\_EAX | 0x13 | 19 |
| X86\_REG\_EBX | 0x15 | 21 |
| X86\_REG\_EDI | 0x17 | 23 |
| X86\_REG\_ESI | 0x1d | 29 |