第1周 研究hurlex 小内核

- x86寄存器体系
- **Intel** & **AT&T** 汇编
- ld 链接器文件
- 计算机启动过程
- 操作系统
    - 学习路线

基础

x86寄存器体系

下图描述了一个IA-32提供给我们的一个基本环境中包含的硬件单元。

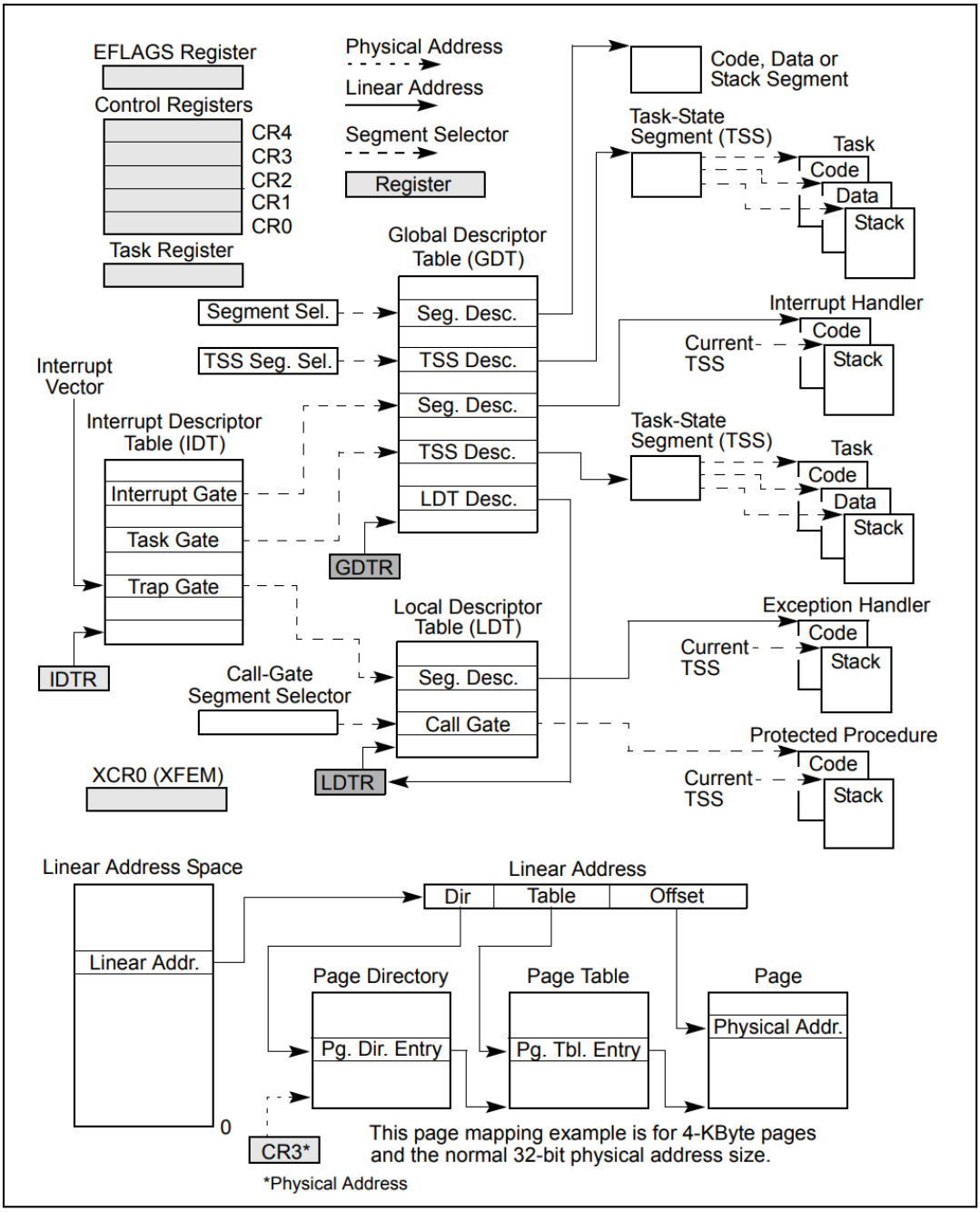


Figure 2-1. IA-32 System-Level Registers and Data Structures

该架构主要提供以下功能

- 内存管理
- 软件模块的保护
- 多任务处理
- 异常和中断处理
- 多核
- 缓存管理
- 硬件资源和电源管理
- 调试和性能监视

几个地址

- 逻辑地址（logical address）：段基址 加上 段内偏移
- 线性地址（linear address）：段基址与段内偏移合成后的地址

- 物理地址（physical address）：如果没有分页，线性地址就是物理地址；有分页，则需要经过分页机制的转换才能得到物理地址。

IA-32处理器运行模式

- 实模式：处理器开机/重启后一开始处于的模式，最大可访问内存为1Mb
- 保护模式：现代处理器支持的原生模式，上电后，处理器开机进入实模式后，从实模式跳转至保护模式。
- 系统管理模式（SMM）：主要供os进行电源管理和OEM差异功能的实现
- 虚拟8086模式：可以在保护模式下运行8086程序

  Intel64架构支持IA-32的所有运行模式和IA-32e模式（简单来说，就是64位对32位的兼容）

Intel & AT&T 汇编语法

CSAPP 中使用的是 AT&T 语法，但Intel 相对简洁，所以一起了解下。

两种语法区别

项目	AT&T	Intel	说明
寄存器命名	%eax	eax	Intel的不带%
操作数顺序	movl %eax, %edx	mov edx, eax	将eax的值赋值给edx
常数\立即数	movl $3, %eax movl $0x10, %eax	mov eax, 3 mov eax, 0x10	将3赋值给eax, 将0x10赋值给eax AT&T的常数加 $ 前缀
jmp指令	jmp *%edx jmp *0x4001002 jmp *(%eax)	jmp edx jmp 0x4001002 jmp [eax]	在AT&T的jmp地址前面要加星号*

操作数长度	movl %eax, %edx
	movb $0x10, %al
	leaw 0x10(%dx), %ax
		mov edx, eax
		mov al, 0x10
		lea ax, [dx + 0x10]
			b = byte (8-bit)
			s = short (16-bit integer or 32-bit floating point)
			w = word (16-bit)
			l = long (32-bit integer or 64-bit floating point)
			q = quad (64 bit)
			t = ten bytes (80-bit floating point)

访问内存高度	后缀b、w、l表示字节、字、长型	前缀byte ptr, word ptr, dword ptr	
引用全局或静态变量var的值	_var	[_var]	
引用全局或静态变量var的地址	$_var	_var	
引用局部变量	基于栈指针（ESP）	基于栈指针（ESP）	
内存直接寻址	imm(base, index, indexscale)	[base + index * indexscale + imm]	两种结果的实际寻址都是 imm + base + index * indexscale
立即数变址寻址	-4(%ebp)	[ebp - 4]	
整数数组寻址	0x40014(, %eax, 3)	[0x40014 + eax * 3]	
寄存器变址寻址	0x40014(%ebx, %eax, 2)	[ebx + eax * 2 + 0x40014]	
寄存器间接寻址	movw $6, %ds:(%eax)	mov word ptr ds:[eax], 6	

AT&T语法

AT&T Style Syntax

Intel语法

Intel Style Syntax

计算机启动过程

实验

编写代码实验

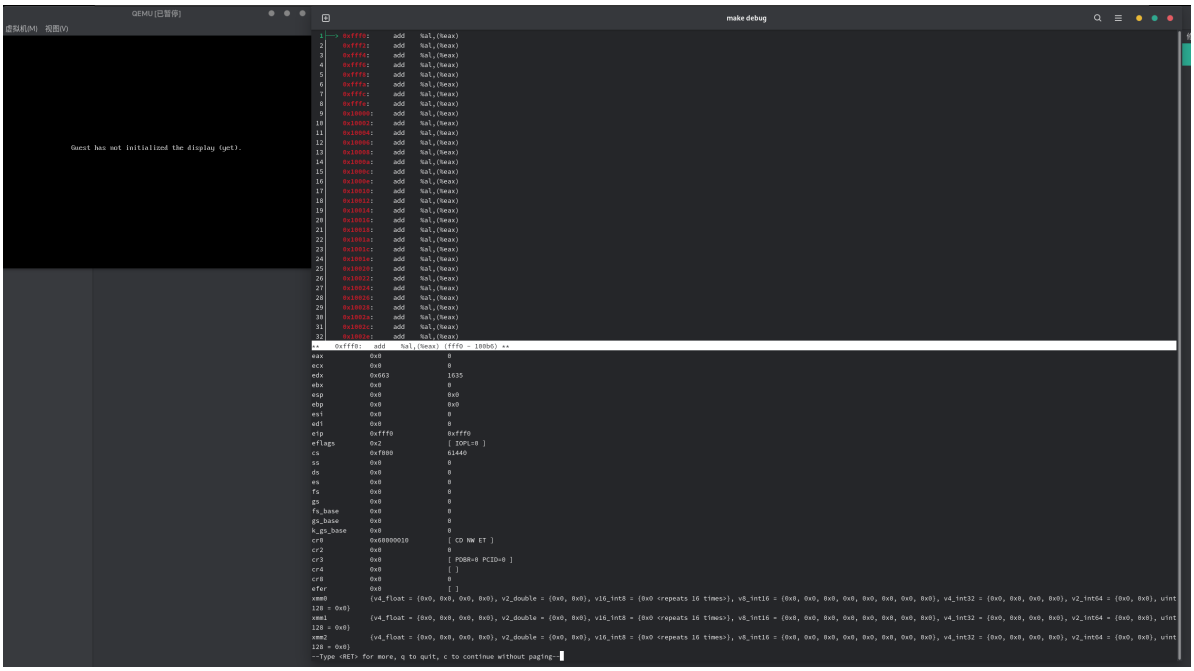
`make debug` 进入调试模式，我们通过阅读以上知识了解到，CPU 地址：$cs * 16 + eip$。

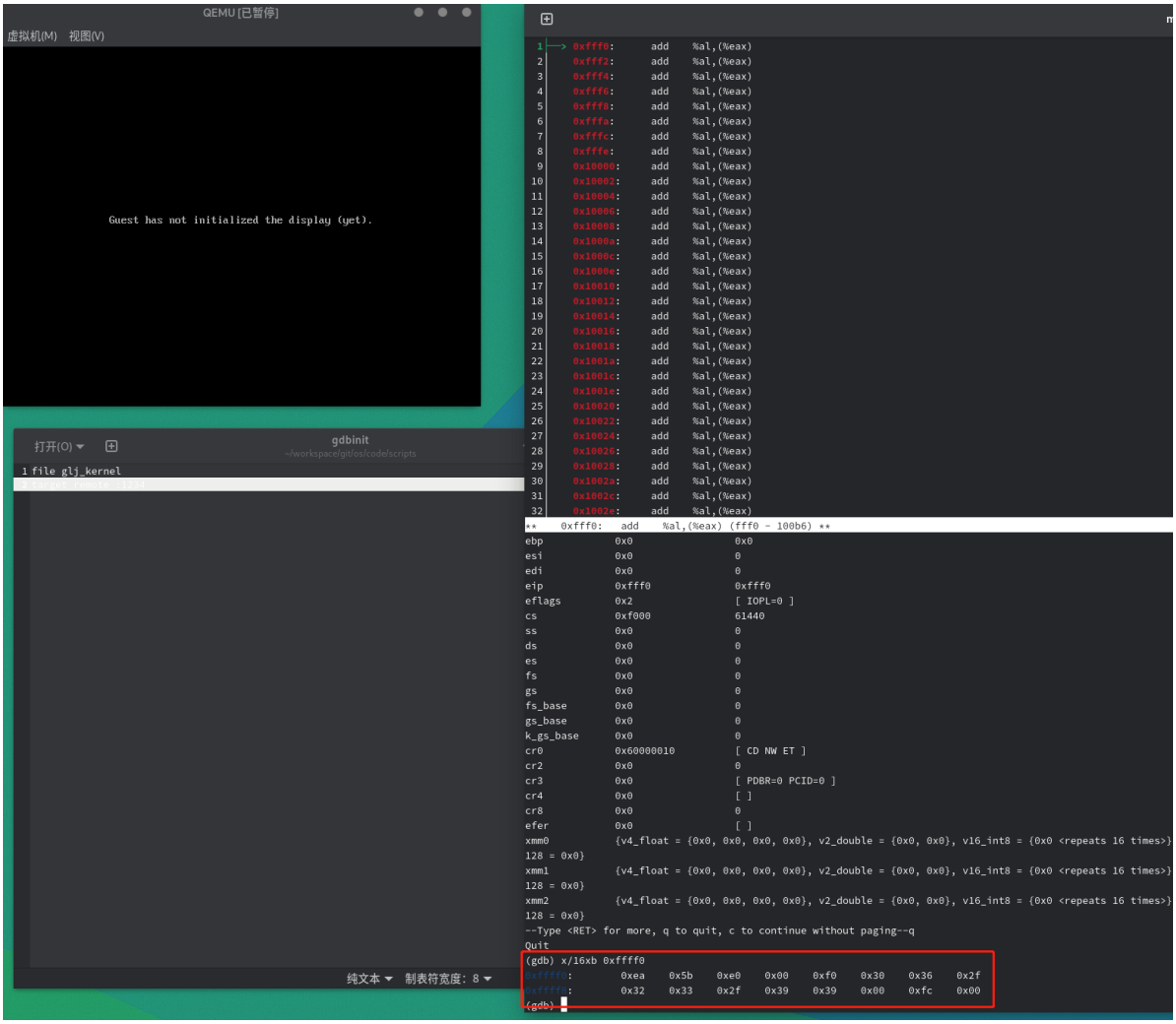
在CPU上电后，通过命令`info registers`查看寄存器数据，其中 cs寄存器和eip寄存器分别被初始化为`0xf000`、0xfff0（见图一），实际访问地址为`0xffff0`，但在20位地址（实模式）下，最大访问空间也就是`1Mb`（`0xfffff`）,此处仅有16字节，空间太小以至于无法存储bios的代码，所以一般此处为一个跳转指令（使用命令`x/16xb 0xffff0`，结果见图二），跳转bios真正的代码处，然后去执行bios中代码。

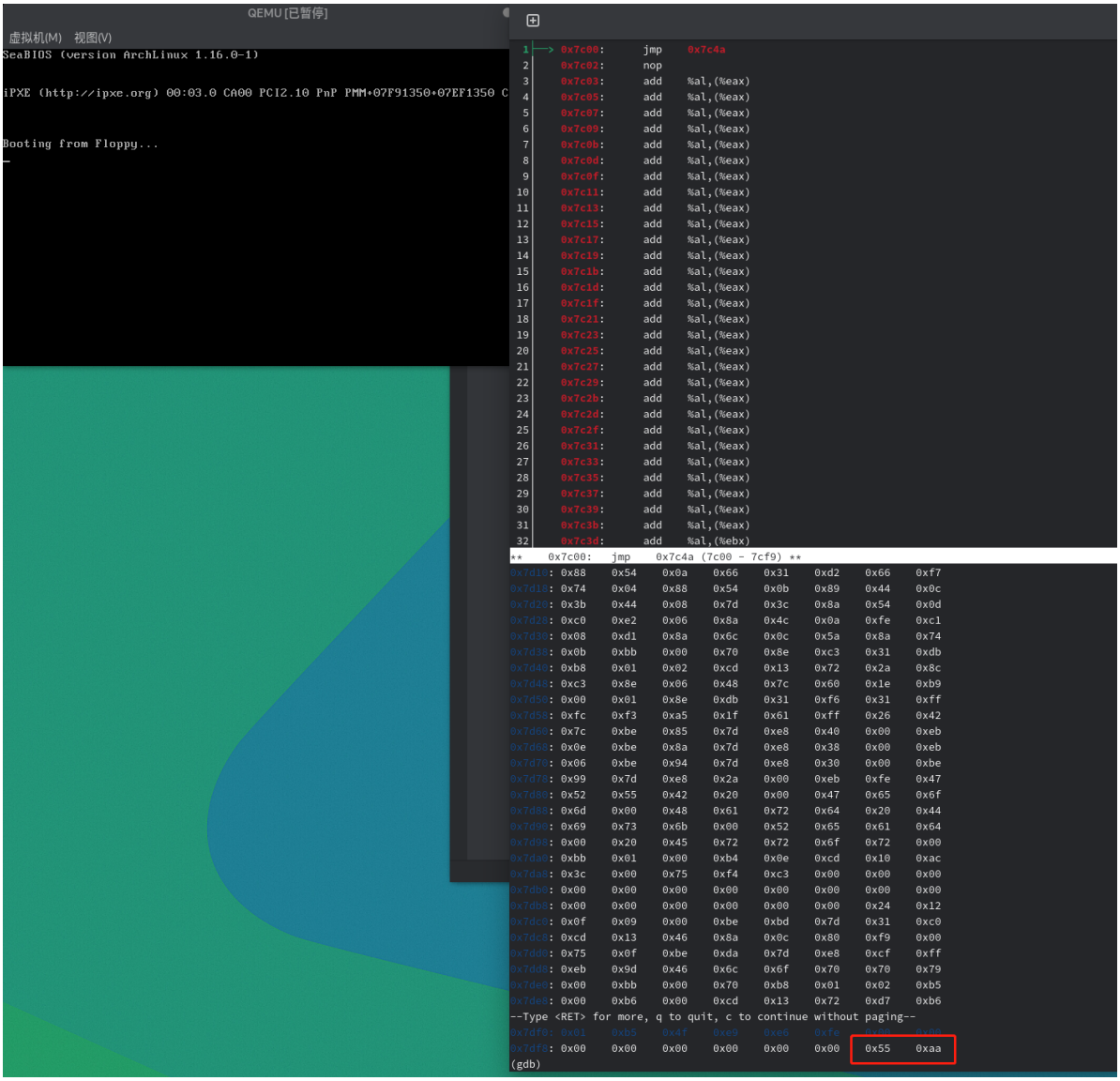
在bios执行结束后，会将第一个可启动存储设备（第一个扇区最后两字节为`0x55aa`）的第一个扇区（512字节）加载至`0x7c00`，然后跳转到`0x7c00`处，去执行相应的代码。所以我们在`0x7c00`处打个断点（`b *0x7c00`），跳转后，查看`0x7c00`处的最后两字节是否为`0x55 0xaa`（使用命令`x/512xb 0x7c00`，结果如图3）。

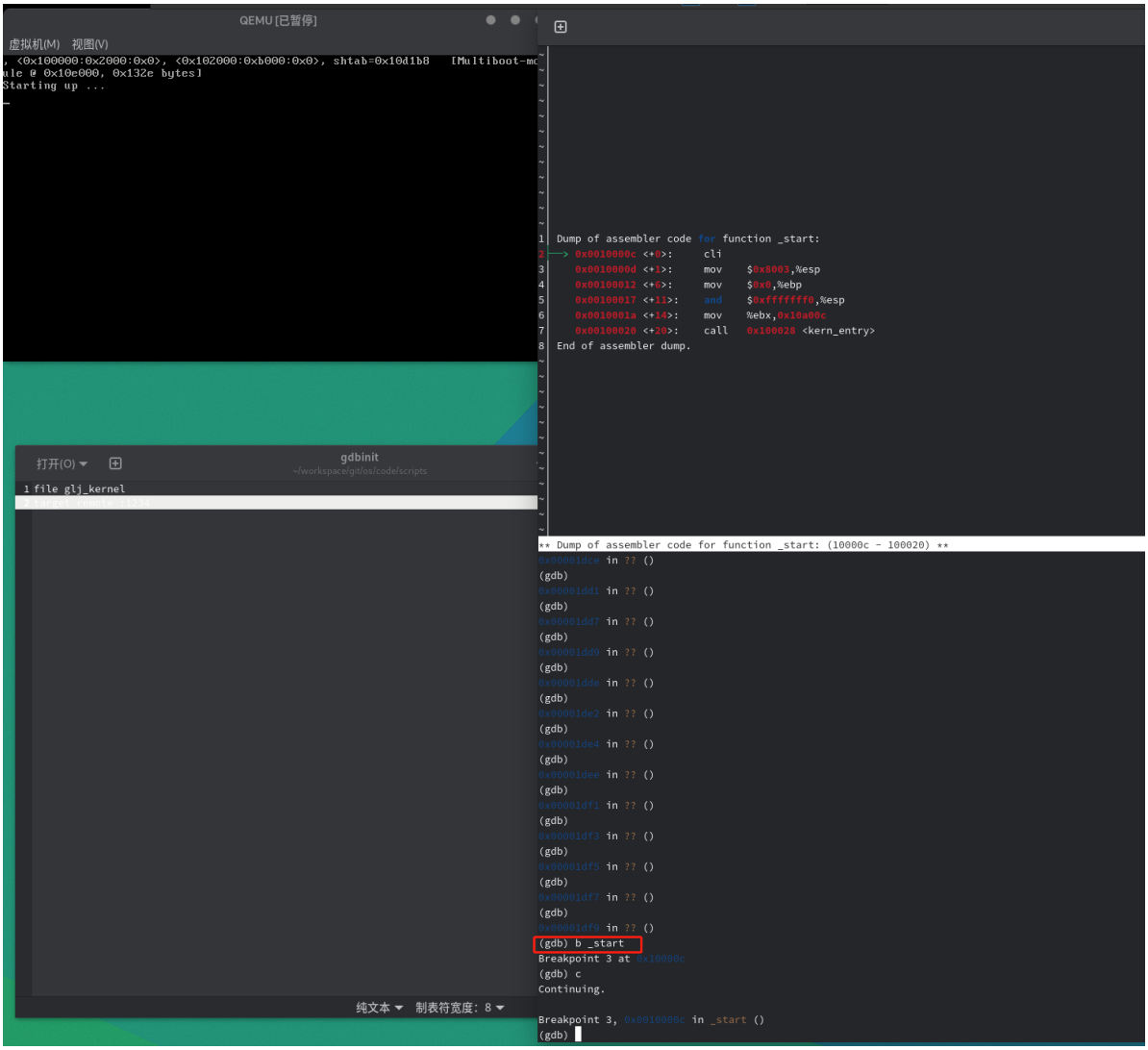
此处执行结束后跳转至`_start`汇编代码处，此时才真正进入到我们的代码中，在`_kern_entry`处打断点，进入到我们c语言的世界。

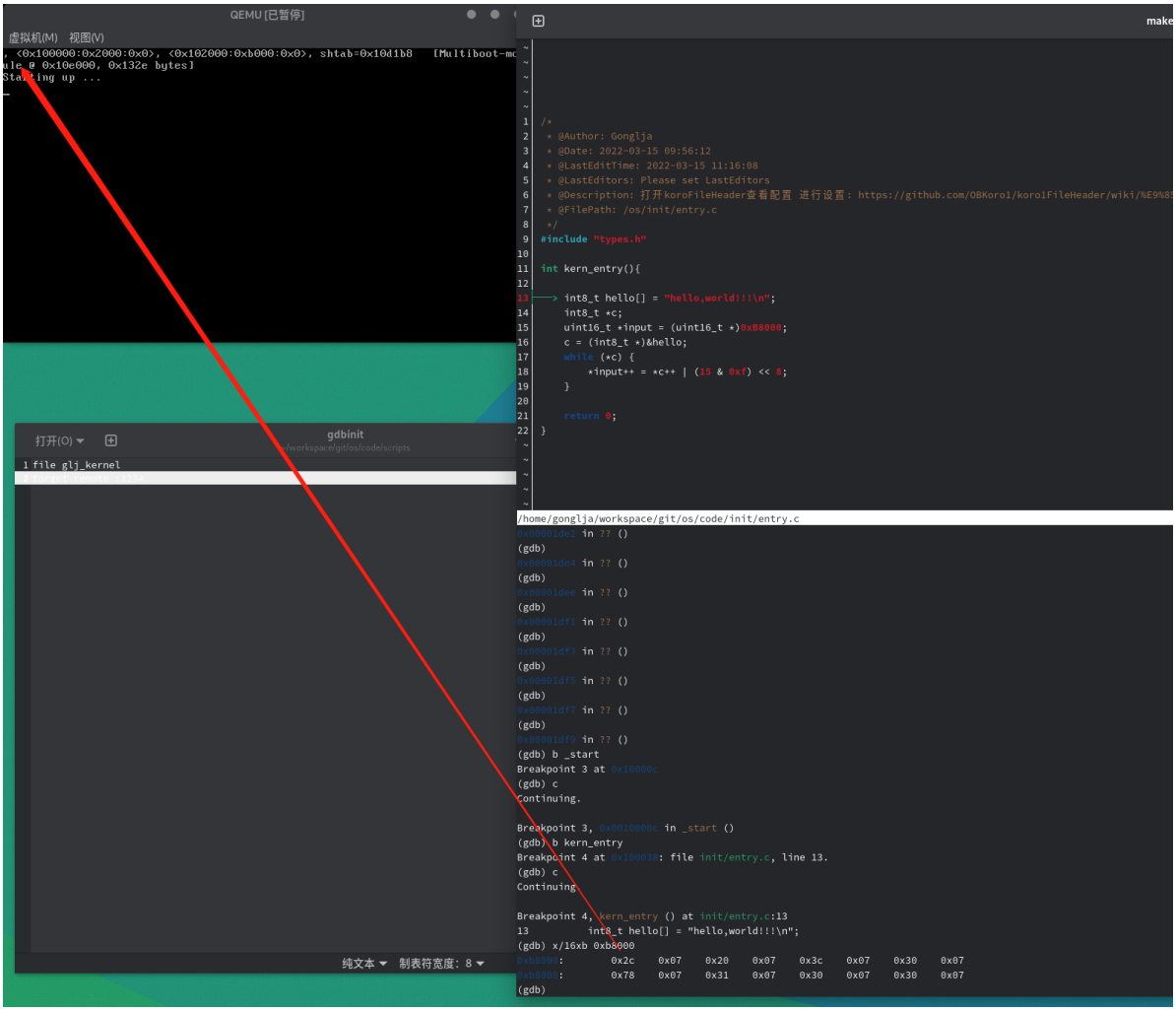
c语言的代码为`printf("hello,world!!!");`，实际上就是把数据写入到`0xb8000`处。图4和图5为显示不同的内容。

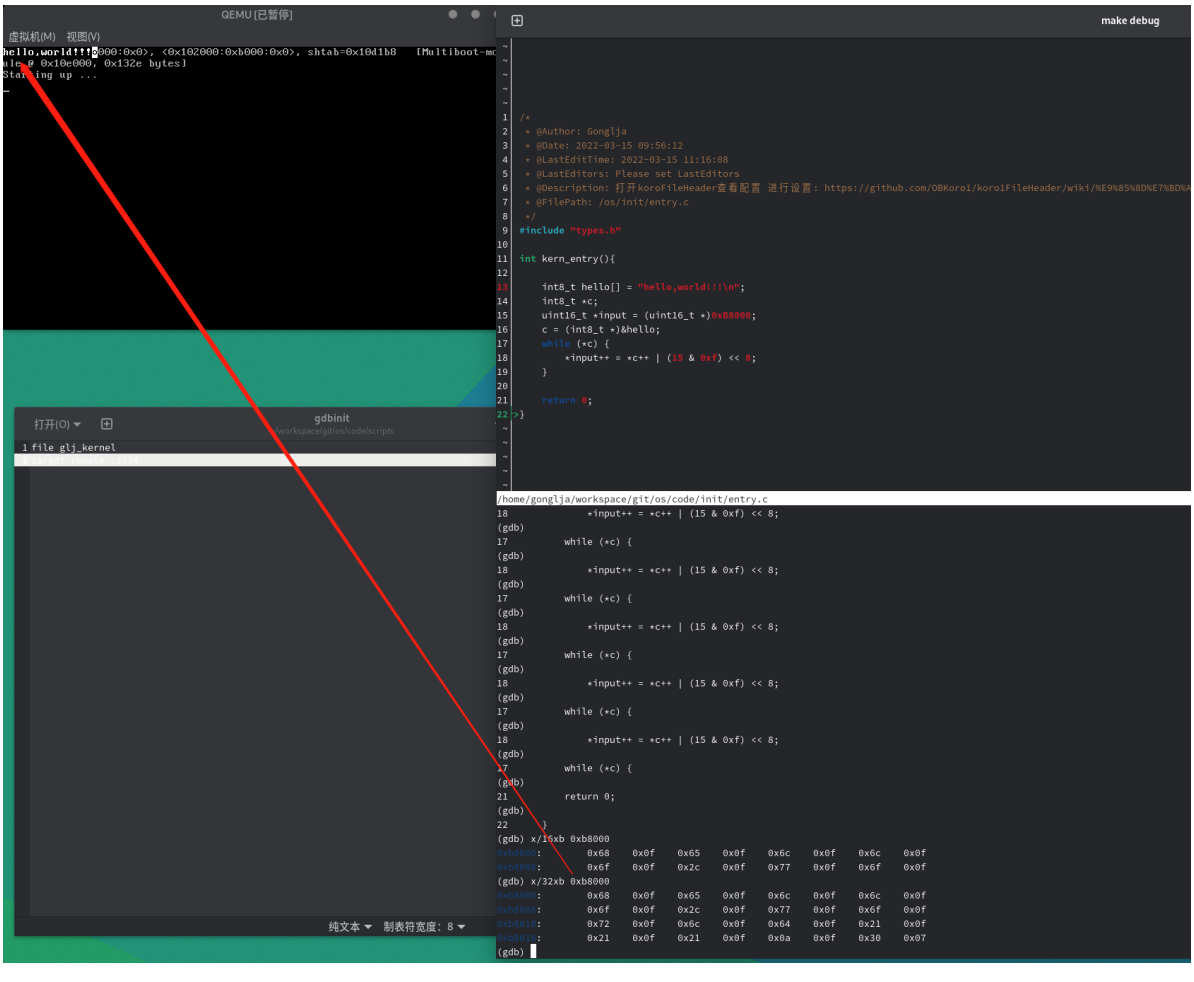



QEMU [已暂停]
虚拟机(M) 视图(V)
Guest has not initialized the display (yet).
打开(O)
gdbinit
~/workspace/git/os/code/scripts
1 file glj_kernel
纯文本
制表符宽度: 8
1 -> 0xfff0: add %al,(%eax)
2 0xfff2: add %al,(%eax)
3 0xfff4: add %al,(%eax)
4 0xfff6: add %al,(%eax)
5 0xfff8: add %al,(%eax)
6 0xfffa: add %al,(%eax)
7 0xfffc: add %al,(%eax)
8 0xfffe: add %al,(%eax)
9 0x10000: add %al,(%eax)
10 0x10002: add %al,(%eax)
11 0x10004: add %al,(%eax)
12 0x10006: add %al,(%eax)
13 0x10008: add %al,(%eax)
14 0x1000a: add %al,(%eax)
15 0x1000c: add %al,(%eax)
16 0x1000e: add %al,(%eax)
17 0x10010: add %al,(%eax)
18 0x10012: add %al,(%eax)
19 0x10014: add %al,(%eax)
20 0x10016: add %al,(%eax)
21 0x10018: add %al,(%eax)
22 0x1001a: add %al,(%eax)
23 0x1001c: add %al,(%eax)
24 0x1001e: add %al,(%eax)
25 0x10020: add %al,(%eax)
26 0x10022: add %al,(%eax)
27 0x10024: add %al,(%eax)
28 0x10026: add %al,(%eax)
29 0x10028: add %al,(%eax)
30 0x1002a: add %al,(%eax)
31 0x1002c: add %al,(%eax)
32 0x1002e: add %al,(%eax)
** 0xfff0: add %al,(%eax) (fff0 - 100b6) **
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0xfff0 0xfff0
eflags 0x2 [IOPL=0]
cs 0xf000 61440
ss 0x0 0
ds 0x0 0
es 0x0 0
fs 0x0 0
gs 0x0 0
fs_base 0x0 0
gs_base 0x0 0
k_gs_base 0x0 0
cr0 0x60000010 [CD NW ET]
cr2 0x0 0
cr3 0x0 [PDBR=0 PCID=0]
cr4 0x0 [ ]
cr8 0x0 0
efer 0x0 [ ]
xmm0 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}}
128 = 0x0}
xmm1 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}}
128 = 0x0}
xmm2 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}}
128 = 0x0}
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) x/16xb 0xffff0
0xffff0: 0xea 0x5b 0xe0 0x00 0xf0 0x30 0x36 0x2f
0xffff8: 0x32 0x33 0x2f 0x39 0x39 0x00 0xfc 0x00
(gdb)


QEMU [已暂停]
虚拟机(M) 视图(V)
SeaBIOS (version ArchLinux 1.16.0-1)
iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F91350+07EF1350 C
Booting from Floppy...
1 --> 0x7c00: jmp 0x7c4a
2 0x7c02: nop
3 0x7c03: add %al,(%eax)
4 0x7c05: add %al,(%eax)
5 0x7c07: add %al,(%eax)
6 0x7c09: add %al,(%eax)
7 0x7c0b: add %al,(%eax)
8 0x7c0d: add %al,(%eax)
9 0x7c0f: add %al,(%eax)
10 0x7c11: add %al,(%eax)
11 0x7c13: add %al,(%eax)
12 0x7c15: add %al,(%eax)
13 0x7c17: add %al,(%eax)
14 0x7c19: add %al,(%eax)
15 0x7c1b: add %al,(%eax)
16 0x7c1d: add %al,(%eax)
17 0x7c1f: add %al,(%eax)
18 0x7c21: add %al,(%eax)
19 0x7c23: add %al,(%eax)
20 0x7c25: add %al,(%eax)
21 0x7c27: add %al,(%eax)
22 0x7c29: add %al,(%eax)
23 0x7c2b: add %al,(%eax)
24 0x7c2d: add %al,(%eax)
25 0x7c2f: add %al,(%eax)
26 0x7c31: add %al,(%eax)
27 0x7c33: add %al,(%eax)
28 0x7c35: add %al,(%eax)
29 0x7c37: add %al,(%eax)
30 0x7c39: add %al,(%eax)
31 0x7c3b: add %al,(%eax)
32 0x7c3d: add %al,(%ebx)
** 0x7c00: jmp 0x7c4a (7c00 - 7cf9) **
0x7d10: 0x88 0x54 0x0a 0x66 0x31 0xd2 0x66 0xf7
0x7d18: 0x74 0x04 0x88 0x54 0x0b 0x89 0x44 0x0c
0x7d20: 0x3b 0x44 0x08 0x7d 0x3c 0x8a 0x54 0x0d
0x7d28: 0xc0 0xe2 0x06 0x8a 0x4c 0x0a 0xfe 0xc1
0x7d30: 0x08 0xd1 0x8a 0x6c 0x0c 0x5a 0x8a 0x74
0x7d38: 0x0b 0xbb 0x00 0x70 0x8e 0xc3 0x31 0xdb
0x7d40: 0xb8 0x01 0x02 0xcd 0x13 0x72 0x2a 0x8c
0x7d48: 0xc3 0x8e 0x06 0x48 0x7c 0x60 0x1e 0xb9
0x7d50: 0x00 0x01 0x8e 0xdb 0x31 0xf6 0x31 0xff
0x7d58: 0xfc 0xf3 0xa5 0x1f 0x61 0xff 0x26 0x42
0x7d60: 0x7c 0xbe 0x85 0x7d 0xe8 0x40 0x00 0xeb
0x7d68: 0x0e 0xbe 0x8a 0x7d 0xe8 0x38 0x00 0xeb
0x7d70: 0x06 0xbe 0x94 0x7d 0xe8 0x30 0x00 0xbe
0x7d78: 0x99 0x7d 0xe8 0x2a 0x00 0xeb 0xfe 0x47
0x7d80: 0x52 0x55 0x42 0x20 0x00 0x47 0x65 0x6f
0x7d88: 0x6d 0x00 0x48 0x61 0x72 0x64 0x20 0x44
0x7d90: 0x69 0x73 0x6b 0x00 0x52 0x65 0x61 0x64
0x7d98: 0x00 0x20 0x45 0x72 0x72 0x6f 0x72 0x00
0x7da0: 0xbb 0x01 0x00 0xb4 0x0e 0xcd 0x10 0xac
0x7da8: 0x3c 0x00 0x75 0xf4 0xc3 0x00 0x00 0x00
0x7db0: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x7db8: 0x00 0x00 0x00 0x00 0x00 0x00 0x24 0x12
0x7dc0: 0x0f 0x09 0x00 0xbe 0xbd 0x7d 0x31 0xc0
0x7dc8: 0xcd 0x13 0x46 0x8a 0x0c 0x80 0xf9 0x00
0x7dd0: 0x75 0x0f 0xbe 0xda 0x7d 0xe8 0xcf 0xff
0x7dd8: 0xeb 0x9d 0x46 0x6c 0x6f 0x70 0x70 0x79
0x7de0: 0x00 0xbb 0x00 0x70 0xb8 0x01 0x02 0xb5
0x7de8: 0x00 0xb6 0x00 0xcd 0x13 0x72 0xd7 0xb6
--Type <RET> for more, q to quit, c to continue without paging--
0x7df0: 0x01 0xb5 0x4f 0xe9 0xe6 0xfe 0x00 0x00
0x7df8: 0x00 0x00 0x00 0x00 0x00 0x00 0x55 0xaa
(gdb)


QEMU [已暂停]
虚拟机(M) 视图(V)
, <0x100000:0x2000:0x0>, <0x102000:0xb000:0x0>, shtab=0x10d1b8 [Multiboot-mc
ule @ 0x10e000, 0x132e bytes]
Starting up ...
打开(O)
gdbinit
~/workspace/git/os/code/scripts
1 file glj_kernel
2 target remote :1234
纯文本
制表符宽度: 8
1 Dump of assembler code for function _start:
2 -> 0x0010000c <+0>: cli
3 0x0010000d <+1>: mov $0x8003,%esp
4 0x00100012 <+6>: mov $0x0,%ebp
5 0x00100017 <+11>: and $0xfffffff0,%esp
6 0x0010001a <+14>: mov %ebx,0x10a00c
7 0x00100020 <+20>: call 0x100028 <kern_entry>
8 End of assembler dump.
** Dump of assembler code for function _start: (10000c - 100020) **
0x0001dce in ?? ()
(gdb)
0x0001dd1 in ?? ()
(gdb)
0x0001dd7 in ?? ()
(gdb)
0x0001dd9 in ?? ()
(gdb)
0x0001dde in ?? ()
(gdb)
0x0001de2 in ?? ()
(gdb)
0x0001de4 in ?? ()
(gdb)
0x0001dee in ?? ()
(gdb)
0x0001df1 in ?? ()
(gdb)
0x0001df3 in ?? ()
(gdb)
0x0001df5 in ?? ()
(gdb)
0x0001df7 in ?? ()
(gdb)
0x0001df9 in ?? ()
(gdb) b _start
Breakpoint 3 at 0x10000c
(gdb) c
Continuing.
Breakpoint 3, 0x0010000c in _start ()
(gdb)


QEMU [已暂停]
虚拟机(M) 视图(V)
Starting up ...
make
/*
* @Author: Gonglja
* @Date: 2022-03-15 09:56:12
* @LastEditTime: 2022-03-15 11:16:08
* @LastEditors: Please set LastEditors
* @Description: 打开koroFileHeader查看配置 进行设置: https://github.com/OBKoro1/koro1FileHeader/wiki/%E9%8
* @FilePath: /os/init/entry.c
*/
#include "types.h"
int kern_entry(){
int8_t hello[] = "hello,world!!!\n";
int8_t *c;
uint16_t *input = (uint16_t *)0xB8000;
c = (int8_t *)&hello;
while (*c) {
*input++ = *c++ | (15 & 0xf) << 8;
}
return 0;
}
打开(O)
gdbinit
~/workspace/git/os/code/scripts
1 file glj_kernel
纯文本 制表符宽度: 8
/home/gonglja/workspace/git/os/code/init/entry.c
(gdb)
(gdb) b _start
Breakpoint 3 at 0x10000c
(gdb) c
Continuing.
Breakpoint 3, 0x0010000c in _start ()
(gdb) b kern_entry
Breakpoint 4 at 0x100038: file init/entry.c, line 13.
(gdb) c
Continuing.
Breakpoint 4, kern_entry () at init/entry.c:13
13 int8_t hello[] = "hello,world!!!\n";
(gdb) x/16xb 0xb8000
0xb8000: 0x2c 0x07 0x20 0x07 0x3c 0x07 0x30 0x07
0xb8008: 0x78 0x07 0x31 0x07 0x30 0x07 0x30 0x07
(gdb)


QEMU [已暂停]
虚拟机(M) 视图(V)
hello,world!!!
Starting up ...
make debug
/*
* @Author: Gonglja
* @Date: 2022-03-15 09:56:12
* @LastEditTime: 2022-03-15 11:16:08
* @LastEditors: Please set LastEditors
* @Description: 打开koroFileHeader查看配置 进行设置: https://github.com/OBKoro1/koro1FileHeader/wiki/%E9%85%8D%E7%BD%A
* @FilePath: /os/init/entry.c
*/
#include "types.h"
int kern_entry(){
int8_t hello[] = "hello,world!!!\n";
int8_t *c;
uint16_t *input = (uint16_t *)0xB8000;
c = (int8_t *)&hello;
while (*c) {
*input++ = *c++ | (15 & 0xf) << 8;
}
return 0;
}
打开(O)
gdbinit
~/workspace/git/os/code/scripts
1 file glj_kernel
纯文本 制表符宽度: 8
/home/gonglja/workspace/git/os/code/init/entry.c
(gdb) x/16xb 0xb8000
0xb8000: 0x68 0x0f 0x65 0x0f 0x6c 0x0f 0x6c 0x0f
0xb8008: 0x6f 0x0f 0x2c 0x0f 0x77 0x0f 0x6f 0x0f
(gdb) x/32xb 0xb8000
0xb8000: 0x68 0x0f 0x65 0x0f 0x6c 0x0f 0x6c 0x0f
0xb8008: 0x6f 0x0f 0x2c 0x0f 0x77 0x0f 0x6f 0x0f
0xb8010: 0x72 0x0f 0x6c 0x0f 0x64 0x0f 0x21 0x0f
0xb8018: 0x21 0x0f 0x21 0x0f 0x0a 0x0f 0x30 0x07
(gdb)

Q&A

参考

1. intel 汇编 https://www.cs.virginia.edu/~evans/cs216/guides/x86.html
2. AT&T 汇编