CS 344: OPERATING SYSTEMS LABS

GROUP 12

ASSIGNMENT_0A-1

220123029 KAVURI VEDA VARSHA

220123030 KODIBOYINA LEELA VARSHINI

220123031 KOJJA VAMSI KRISHNA

220123052 RAMINENI HARDHIKA

EXERCISE 1:

Modified code for ex1.c is as follows:

```c
#include <stdio.h>
int main(int argc, char **argv)
{
        int x = 1;
        printf("Hello x = %d\n", x);
        asm ("movl %1, %0;":"=r"(x):"r"(x+1):);
        printf("Hello x = %d after increment\n", x);
        if(x == 2) printf("OK\n");
        else printf("ERROR\n");
}
```

Output:

```
guest@iitg-Vostro-3910:~/Desktop/hardy$ gcc ex1.c
guest@iitg-Vostro-3910:~/Desktop/hardy$ ./a.out
Hello x = 1
Hello x = 2 after increment
OK
```

Explanation:

The format of GNU assembler assembly code in C is:

asm [volatile](AssemblerTemplate

: Output Operands

[: Input Operands

[: Clobbers]])

In asm: %0 -> first variable(here x), %1 -> second variable(here x+1)

movl src, dst moves the contents of source to destination.

The “r” in the operand constraint string indicates that the operand must be located in a register. Each output operand must have “=”

in its constraint.

EXERCISE 2:

When executing the ‘si’ command multiple times, we observed that:

Upon a processor reset, the processor switches to real mode, setting the Code Segment (CS) to 0xf000 and the IP to 0xfff0. This configuration causes execution to start at the address specified by these segment registers (CS:IP). Then the BIOS jumps backward to an earlier location in the BIOS.

'lidtl' command sets up an interrupt descriptor table and 'lgdtl' sets up a global descriptor table. Along with this, it initializes various devices such as the VGA display.

```
(gdb) source .gdbinit
+ target remote localhost:26000
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
warning:  A  handler  for  the  OS  ABI  "GNU/Linux"  is  not  built  into  this
configuration
of GDB.  Attempting to continue with the default i8086 settings.

(gdb) si
[f000:e05b]    0xfe05b: cmpw   $0xffc8,%cs:(%esi)
0x0000e05b in ?? ()
(gdb) si
[f000:e062]    0xfe062: jne    0xd241d416
0x0000e062 in ?? ()
(gdb) si
[f000:e066]    0xfe066: xor    %edx,%edx
0x0000e066 in ?? ()
(gdb) si
[f000:e068]    0xfe068: mov    %edx,%ss
0x0000e068 in ?? ()
(gdb) si
[f000:e06a]    0xfe06a: mov    $0x7000,%sp
0x0000e06a in ?? ()
(gdb) si
[f000:e070]    0xfe070: mov    $0x2d4e,%dx
0x0000e070 in ?? ()
(gdb) si
[f000:e076]    0xfe076: jmp    0x5575ff02
0x0000e076 in ?? ()
```

```
(gdb) si
[f000:ff00]    0xfff00: cli
0x0000ff00 in ?? ()
(gdb) si
[f000:ff01]    0xfff01: cld
0x0000ff01 in ?? ()

(gdb) si
[f000:ff02]    0xfff02: mov    %ax,%cx
0x0000ff02 in ?? ()
(gdb) si
[f000:ff05]    0xfff05: mov    $0x8f,%ax
0x0000ff05 in ?? ()
(gdb) si
[f000:ff0b]    0xfff0b: out    %al,$0x70
0x0000ff0b in ?? ()
(gdb) si
[f000:ff0d]    0xfff0d: in     $0x71,%al
0x0000ff0d in ?? ()
(gdb) si
[f000:ff0f]    0xfff0f: in     $0x92,%al
0x0000ff0f in ?? ()
(gdb) si
[f000:ff11]    0xfff11: or     $0x2,%al
0x0000ff11 in ?? ()
(gdb) si
[f000:ff13]    0xfff13: out    %al,$0x92
0x0000ff13 in ?? ()
(gdb) si
[f000:ff15]    0xfff15: mov    %cx,%ax
0x0000ff15 in ?? ()
(gdb) si
[f000:ff18]    0xfff18: lidtl  %cs:(%esi)
0x0000ff18 in ?? ()
```

```
(gdb) si
[f000:ff1e]    0xfff1e: lgdtl  lgdtl  %cs:(%esi)
0x0000ff1e in ?? ()
(gdb) si
[f000:ff24]    0xfff24: mov    %cr0,%ecx
0x0000ff24 in ?? ()
(gdb) si
[f000:ff27]    0xfff27: and    $0xffff,%cx
0x0000ff27 in ?? ()
(gdb) si
[f000:ff2e]    0xfff2e: or     $0x1,%cx
0x0000ff2e in ?? ()
(gdb) si
[f000:ff32]    0xfff32: mov    %ecx,%cr0
0x0000ff32 in ?? ()
(gdb) si
[f000:ff35]    0xfff35: ljmpw  $0xf,$0xff3d
0x0000ff35 in ?? ()
(gdb) si
The target architecture is assumed to be i386
=> 0xfff3d: mov    $0x10,%ecx
0x000fff3d in ?? ()
(gdb) si
=> 0xfff42: mov    %ecx,%ds
```

EXERCISE 3:

Observations:

In tracing the readsect( )and bootmain( ) functions in bootmain.c and the bootblock.asm assembly code:

1. The loop starts at address 0x7d8f. The entry condition is checked at 0x7d83; if true, the jmp at 0x7d8d directs execution to the loop at 0x7d96. Subsequent iterations of the loop run from 0x7d8f to 0x7db4.
2. The loop ends at address 0x7db4. Just before exiting, the jbe instruction at 0x7d94 is executed if the loop continuation condition at 0x7d92 is not met.
3. After the loop, execution moves to address 0x7d87, where a call to the entry( ) function is made. Setting a breakpoint at 0x7d87, and continuing execution, shows that the last instruction executed iscall *0x10018 . Using si after this instruction leads to entering the kernel at address 0x0010000c, as shown in the output image.

```
for(; ph < eph; ph++){
    7d83:   39 f3                   cmp    %esi,%ebx
    7d85:   72 0f                   jb     7d96 <bootmain+0x5b>
  entry();
    7d87:   ff 15 18 00 01 00       call   *0x10018
    7d8d:   eb d5                   jmp    7d64 <bootmain+0x29>
  for(; ph < eph; ph++){
    7d8f:   83 c3 20                add    $0x20,%ebx
    7d92:   39 de                   cmp    %ebx,%esi
    7d94:   76 f1                   jbe    7d87 <bootmain+0x4c>
    pa = (uchar*)ph->paddr;
    7d96:   8b 7b 0c                mov    0xc(%ebx),%edi
    readseg(pa, ph->filesz, ph->off);
    7d99:   ff 73 04                pushl  0x4(%ebx)
    7d9c:   ff 73 10                pushl  0x10(%ebx)
    7d9f:   57                      push   %edi
    7da0:   e8 53 ff ff ff          call   7cf8 <readseg>
    if(ph->memsz > ph->filesz)
    7da5:   8b 4b 14                mov    0x14(%ebx),%ecx
    7da8:   8b 43 10                mov    0x10(%ebx),%eax
    7dab:   83 c4 0c                add    $0xc,%esp
    7dae:   39 c1                   cmp    %eax,%ecx
    7db0:   76 dd                   jbe    7d8f <bootmain+0x54>
      stosb(pa + ph->filesz, 0, ph->memsz - ph->filesz);
    7db2:   01 c7                   add    %eax,%edi
    7db4:   29 c1                   sub    %eax,%ecx
```

1. The processor begins executing 32-bit code at address 0x7c31, where it encounters the instruction mov $0x10, %ax. The CR0 register, a 32-bit control register in the 386 processor, contains the PE flag, which determines whether the processor is in Protected mode (PE = 1) or Real mode (PE = 0). To transition from 16-bit to 32-bit mode, several steps are performed: first, a global descriptor table (GDT) is set up and loaded using the lgdt command. Next, the PE flag in the CR0 register is set to 1 to enable Protected mode. Finally, a long jump (ljmp) is executed to address start32, ensuring that the processor correctly reloads the code segment (%cs) and instruction pointer (%eip) in 32-bit mode. This setup uses identity mapping for the segment descriptors, which means no address translation is applied.

```
  # Switch from real to protected mode.  Use a bootstrap GDT that makes
  # virtual addresses map directly to physical addresses so that the
  # an effective memory map doesn't change during the transition.
  lgdt    gdtdesc
  movl    %cr0, %eax
  orl     $CR0_PE, %eax
  movl    %eax, %cr0

//PAGEBREAK!
  # Complete the transition to 32-bit protected mode by using a long jmp
  # to reload %cs and %eip.  The segment descriptors are set up with no
  # translation, so that the mapping is still the identity mapping.
  ljmp    $(SEG_KCODE<<3), $start32

.code32  # Tell assembler to generate 32-bit code now.
start32:
  # Set up the protected-mode data segment registers
  movw    $(SEG_KDATA<<3), %ax    # Our data segment selector
```

2. The last instruction of the bootloader that was executed is call *0x10018 which calls the entry function and hence the bootloader transfers control to the kernel which is at address 0x0010000c. The first instruction of the kernel it just loaded is mov %cr4, %eax at address 0x0010000c. The below image shows the outcomes.

```
(gdb) source .gdbinit
+ target remote localhost:26000
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
warning: A handler for the OS ABI "GNU/Linux" is not built into this configuration
of GDB.  Attempting to continue with the default i8086 settings.

(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.
[   0:7c00] => 0x7c00:  cli

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) si 18
[   0:7c29] => 0x7c29:  mov    %eax,%cr0
0x00007c29 in ?? ()
(gdb) si
[   0:7c2c] => 0x7c2c:  ljmp   $0xb866,$0x87c31
0x00007c2c in ?? ()
(gdb) si
The target architecture is assumed to be i386
=> 0x7c31:      mov    $0x10,%ax
0x00007c31 in ?? ()
(gdb) b *0x7d87
Breakpoint 2 at 0x7d87
(gdb) c
Continuing.
=> 0x7d87:      call   *0x10018

Thread 1 hit Breakpoint 2, 0x00007d87 in ?? ()
(gdb) si
=> 0x10000c:    mov    %cr4,%eax
0x0010000c in ?? ()
(gdb) x/1x 0x10018
0x10018:        0x0010000c
(gdb)
```

3. The disk sector size is 512 bytes, as specified in bootmain.c on line 13. When the bootloader loads the kernel, it first reads the 4096-byte ELF header, which spans 8 sectors (implemented in bootmain.c on line 28). After loading the ELF header, the

bootloader proceeds to load each kernel segment. Running readelf -a kernel revealed 16 sections, each aligned to 512 bytes. To determine the total number of sectors read by the bootloader, we calculate the sum of the ceiling values of each section's size divided by the sector size (512 bytes). This calculation showed that 436 sectors are needed to fetch the entire kernel. The terminal screenshot below demonstrates the command output used to compute the number of sectors read.

```
guest@iitg-Vostro-3910:~/Downloads/xv6/xv6-public-master$ readelf -a -e kernel
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x10000c
  Start of program headers:          52 (bytes into file)
  Start of section headers:          201244 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         3
  Size of section headers:           40 (bytes)
  Number of section headers:         17
  Section header string table index: 16

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        80100000 001000 0071e8 00  AX  0   0 16
  [ 2] .rodata           PROGBITS        80107200 008200 0009cf 00   A  0   0 32
  [ 3] .data             PROGBITS        80108000 009000 002576 00  WA  0   0 4096
  [ 4] .bss              NOBITS          8010a580 00b576 00afb0 00  WA  0   0 32
  [ 5] .debug_line       PROGBITS        00000000 00b576 006aee 00      0   0  1
  [ 6] .debug_info       PROGBITS        00000000 012064 010ee3 00      0   0  1
  [ 7] .debug_abbrev     PROGBITS        00000000 022f47 0044aa 00      0   0  1
  [ 8] .debug_aranges    PROGBITS        00000000 0273f8 0003b0 00      0   0  8
  [ 9] .debug_str        PROGBITS        00000000 0277a8 000e16 01  MS  0   0  1
  [10] .debug_loclists   PROGBITS        00000000 0285be 0050b1 00      0   0  1
  [11] .debug_rnglists   PROGBITS        00000000 02d66f 000845 00      0   0  1
  [12] .debug_line_str   PROGBITS        00000000 02deb4 000146 01  MS  0   0  1
  [13] .comment          PROGBITS        00000000 02dffa 00002b 01  MS  0   0  1
  [14] .symtab           SYMTAB          00000000 02e028 001fa0 10     15  66  4
  [15] .strtab           STRTAB          00000000 02ffc8 0011a4 00      0   0  1
  [16] .shstrtab         STRTAB          00000000 03116c 0000ad 00      0   0  1
```

EXERCISE 4:

Output of pointers.c:

```
guest@iitg-Vostro-3910:~/Downloads$ gcc pointers.c
guest@iitg-Vostro-3910:~/Downloads$ ./a.out
1: a = 0x7ffd6a2564a0, b = 0x55bd593412a0, c = 0x7ffd6a2568d9
2: a[0] = 200, a[1] = 101, a[2] = 102, a[3] = 103
3: a[0] = 200, a[1] = 300, a[2] = 301, a[3] = 302
4: a[0] = 200, a[1] = 400, a[2] = 301, a[3] = 302
5: a[0] = 200, a[1] = 128144, a[2] = 256, a[3] = 302
6: a = 0x7ffd6a2564a0, b = 0x7ffd6a2564a4, c = 0x7ffd6a2564a1
```

First an array a is declared as int a[4] so a would store the address of the first element of the array a. int *b = malloc(16); would allocate a 16 byte block of memory and store the address of this newly allocated memory. int *c; declares the pointer variable c is assigned garbage value as it is not assigned a value.

Before line 2, c is set to a and it will store the address of the first element of array a. Values of array a are set to 100, 101, 102,103.

Before line 3, a[1] = 301 and set to c[1]. *(c+2) is equivalent to c[2] = 302 changes the value of a[3] = 302 then the value of array is 200, 300, 301, 302.

Before line 4, c = c+1 changes c to the second element of a. So *c = 400 sets a[1] = 400. Array a is 200, 400, 301, 302.

Just after printing line 4, c points to the memory location 0x7ffd6a2564a4, the integer a[1] is stored from 0x7ffd6a2564a4 - 0x7ffd6a2564a7. The statement, c = (int *) ((char *) c + 1); changes c to 0x7ffd6a2564a5 as sizeof(char) = 1, and typecasting c into a char* and incrementing it would increase the memory location by 1 instead of 4 locations. Now *c = 500 would change contents of the memory locations from 0x7ffd6a2564a5 - 0x7ffd6a2564a8, which would change the contents of a[1] and a[2] both, as the memory locations 0x7ffd6a2564a5 - 0x7ffd6a2564a7 is a part of a[1] and the location 0x7ffd6a2564a8 is a part of a[2]. So the values stored in a[1] and a[2] are not garbage although they seem to be corrupted.

As b is of type int* , and a points to the location 0x7ffd6a2564a0, b would point to 0x7ffd6a2564a4 as sizeof(int) = 4. (char*)a points to the memory location 0x7ffd6a2564a0 and (char*)a + 1 would be 0x7ffd6a2564a1, so finally b points to 0x7ffd6a2564a1.

objdump commands:

```
guest@iitg-Vostro-3910:~/Downloads/xv6/xv6-public-master$ objdump -h kernel

kernel:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000071e8  80100000  00100000  00001000  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .rodata       000009cf  80107200  00107200  00008200  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .data         00002576  80108000  00108000  00009000  2**12
                  CONTENTS, ALLOC, LOAD, DATA
  3 .bss          0000afb0  8010a580  0010a580  0000b576  2**5
                  ALLOC
  4 .debug_line   00006aee  00000000  00000000  0000b576  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  5 .debug_info   00010ee3  00000000  00000000  00012064  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  6 .debug_abbrev 000044aa  00000000  00000000  00022f47  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  7 .debug_aranges 000003b0  00000000  00000000  000273f8  2**3
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  8 .debug_str    00000e16  00000000  00000000  000277a8  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  9 .debug_loclists 000050b1  00000000  00000000  000285be  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 10 .debug_rnglists 00000845  00000000  00000000  0002d66f  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 11 .debug_line_str 00000146  00000000  00000000  0002deb4  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 12 .comment      0000002b  00000000  00000000  0002dffa  2**0
                  CONTENTS, READONLY
```

objdump -h kernel: As we can see in the above screenshot, VMA and LMA of .text section is different indicating that it loads and executes from different addresses.

```
guest@iitg-Vostro-3910:~/Downloads/xv6/xv6-public-master$ objdump -h bootblock.o

bootblock.o:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000001c3  00007c00  00007c00  00000074  2**2
                  CONTENTS, ALLOC, LOAD, CODE
  1 .eh_frame     000000b0  00007dc4  00007dc4  00000238  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .comment      0000002b  00000000  00000000  000002e8  2**0
                  CONTENTS, READONLY
  3 .debug_aranges 00000040  00000000  00000000  00000318  2**3
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  4 .debug_info   00000585  00000000  00000000  00000358  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  5 .debug_abbrev 0000023c  00000000  00000000  000008dd  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  6 .debug_line   00000283  00000000  00000000  00000b19  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  7 .debug_str    0000021a  00000000  00000000  00000d9c  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  8 .debug_line_str 00000055  00000000  00000000  00000fb6  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  9 .debug_loclists 0000018d  00000000  00000000  0000100b  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 10 .debug_rnglists 00000033  00000000  00000000  00001198  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
```

objdump -h bootblock.o: As we can see in the above screenshot, VMA and LMA of .text section is same indicating that it loads and executes from the same address.

EXERCISE 5:

In the Makefile, we updated the link address to 0x7c04 from the original 0x7c00. This change caused an error: the bootloader, which initially switched from 16-bit real mode to 32-bit protected mode, started executing from the BIOS memory again instead of entering 32-bit mode. This issue arises because the 'ljmp' instruction requires a correct absolute address, and if the link and load addresses are mismatched, the jump target address becomes incorrect.

```
(gdb) source .gdbinit
+ target remote localhost:26000
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
warning: A handler for the OS ABI "GNU/Linux" is not built into this configuration
of GDB.  Attempting to continue with the default i8086 settings.

(gdb) si
[f000:e05b]    0xfe05b: cmpw   $0xffc8,%cs:(%esi)
0x0000e05b in ?? ()
(gdb) si
[f000:e062]    0xfe062: jne    0xd241d416
0x0000e062 in ?? ()
(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.
[   0:7c00] => 0x7c00:  cli

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) si 18
[   0:7c29] => 0x7c29:  mov    %eax,%cr0
0x00007c29 in ?? ()
(gdb) si
[   0:7c2c] => 0x7c2c:  ljmp   $0xb866,$0x87c35
0x00007c2c in ?? ()
(gdb) si
[f000:e05b]    0xfe05b: cmpw   $0xffc8,%cs:(%esi)
0x0000e05b in ?? ()
(gdb) si
[f000:e062]    0xfe062: jne    0xd241d416
0x0000e062 in ?? ()
(gdb)
```

Output of objdump -f kernel:

The start address is the starting point (0x0010000c) at which the bootloader enters the kernel.

```
guest@iitg-Vostro-3910:~/Downloads/xv6/xv6-public-master$ objdump -f kernel

kernel:     file format elf32-i386
architecture: i386, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x0010000c
```

EXERCISE 6:

After restarting the machine and examining 8 words of memory at address 0x00100000:

1. When the BIOS enters the bootloader, the memory at this location is all zeros.
2. When the bootloader enters the kernel, the memory at this location contains non-zero values.

This is because, before the kernel is loaded, the bootloader has not yet populated this memory area, resulting in zero values. Once the bootloader has loaded the kernel, this memory area contains data from the kernel, so the values become non-zero.

```
(gdb) source .gdbinit
+ target remote localhost:26000
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
warning: A handler for the OS ABI "GNU/Linux" is not built into this configuration
of GDB.  Attempting to continue with the default i8086 settings.

(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.
[   0:7c00] => 0x7c00:  cli

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) x/8x 0x00100000
0x100000:       0x00000000      0x00000000      0x00000000      0x00000000
0x100010:       0x00000000      0x00000000      0x00000000      0x00000000
(gdb) b *0x7d87
Breakpoint 2 at 0x7d87
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0x7d87:      call   *0x10018

Thread 1 hit Breakpoint 2, 0x00007d87 in ?? ()
(gdb) si
=> 0x10000c:    mov    %cr4,%eax
0x0010000c in ?? ()
(gdb) x/8x 0x00100000
0x100000:       0x1badb002      0x00000000      0xe4524ffe      0x83e0200f
0x100010:       0x220f10c8      0x9000b8e0      0x220f0010      0xc0200fd8
(gdb)
```