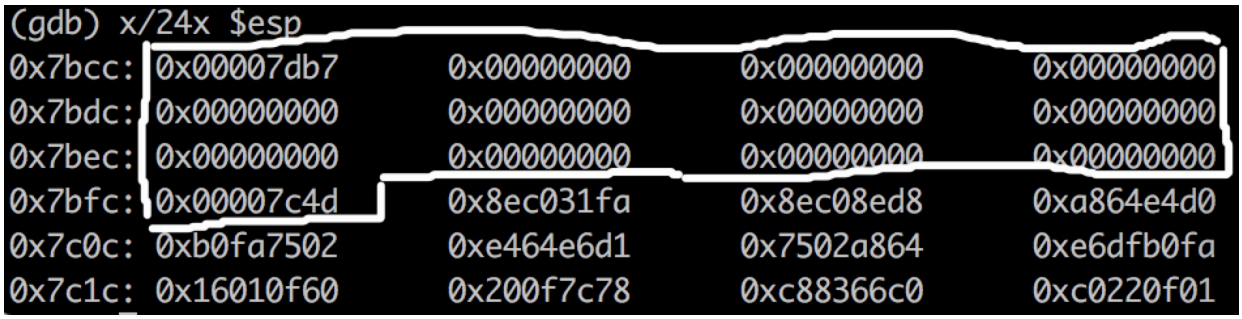
Homework 2

Exercise 1: What is on the stack

The address of _start is 0x100000. After probing the registers, I get the registers and their corresponding values.

register	value	explanation
eax	0x0	result of last function
ecx	0x0	result of last function
edx	0x1f0	last used value
ebx	0x10074	last used value for source address pointer
esp	0x7bcc	the stack pointer
ebp	0x7bf8	the bottom of current stack frame
esi	0x10074	last used value for source address pointer
edi	0x0	return result of last function
eip	0x10000c	the current instruction address, the entry of kernel
eflags	0x46	flags during calculation
cs	0x8	code segment base address, set with ljmp to SEG_KCODE<<3
ss	0x10	stack segment base address, set to SEG_KDATA<<3
ds	0x10	data segment base address, set to SEG_KDATA<<3
es	0x10	extra segment base address, set to SEG_KDATA<<3
fs	0x0	flag segment, set to zero
gs	0x0	global segment, set to zero

The part of stack ranges from address 0x7bcc to address 0x7bfc.



Some explanation is shown below.

value	explanation
0x00007db7	the address of code after bootmain function
0x00000000	sub $0x1c,%esp
0x00000000	sub $0x1c,%esp
0x00000000	sub $0x1c,%esp
0x00000000	sub $0x1c,%esp
0x00000000	sub $0x1c,%esp
0x00000000	sub $0x1c,%esp
0x00000000	sub $0x1c,%esp
0x00000000	push %ebx
0x00000000	push %esi
0x00000000	push %edi
0x00000000	push %ebp
0x00007c4d	the address that bootmain() will return to

Some questions and their answers:

- Q: Start by setting a break-point at 0x7c00, the start of the boot block (bootasm.S). Single step through the instructions. Where in bootasm.S the stack pointer is initialized?

  A: on the address 0x7c43, with the code `mov $0x7c00,%esp`, and the esp is initialized to value 0x7c00.

- Q: Single step through the call to `bootmain`; what is on the stack now?

  A: Since the esp has just been initialized in last step, there will be nothing on the stack.

- Q: What do the first assembly instructions of bootmain do to the stack? Look for bootmain in bootblock.asm.

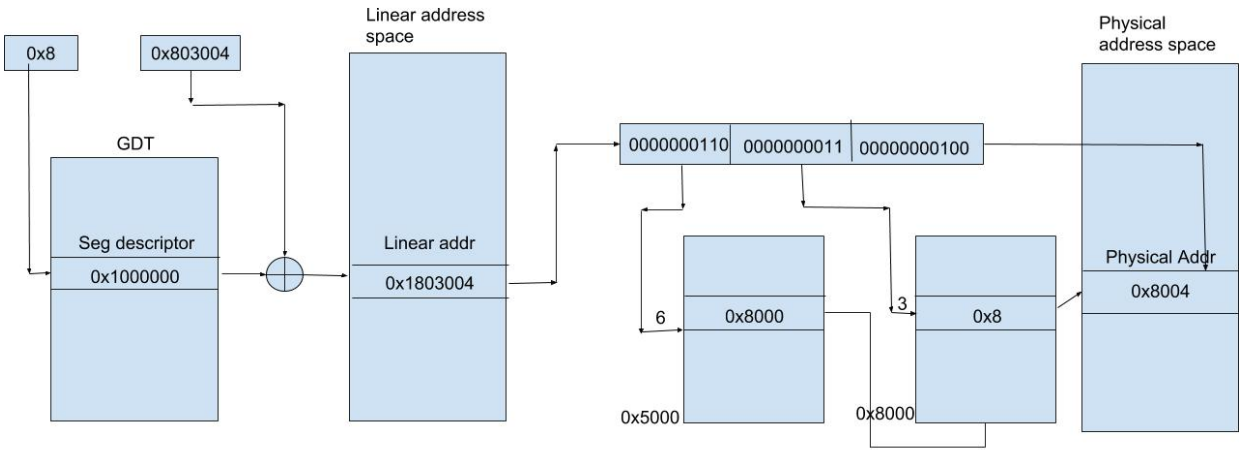
  A: push ebp to the stack, in order to keep the stack frame.

- Q: Look in bootmain in bootblock.asm for the call that changes `eip` to 0x10000c. What does that call do to the stack?

  A: It will push the return address 0x7db7 to the stack.

Exercise 2: Understanding address translation

Question 1: Explain how logical to physical address translation works



Extra credit (5%): Can you explain the nature of the memory access in the question above?

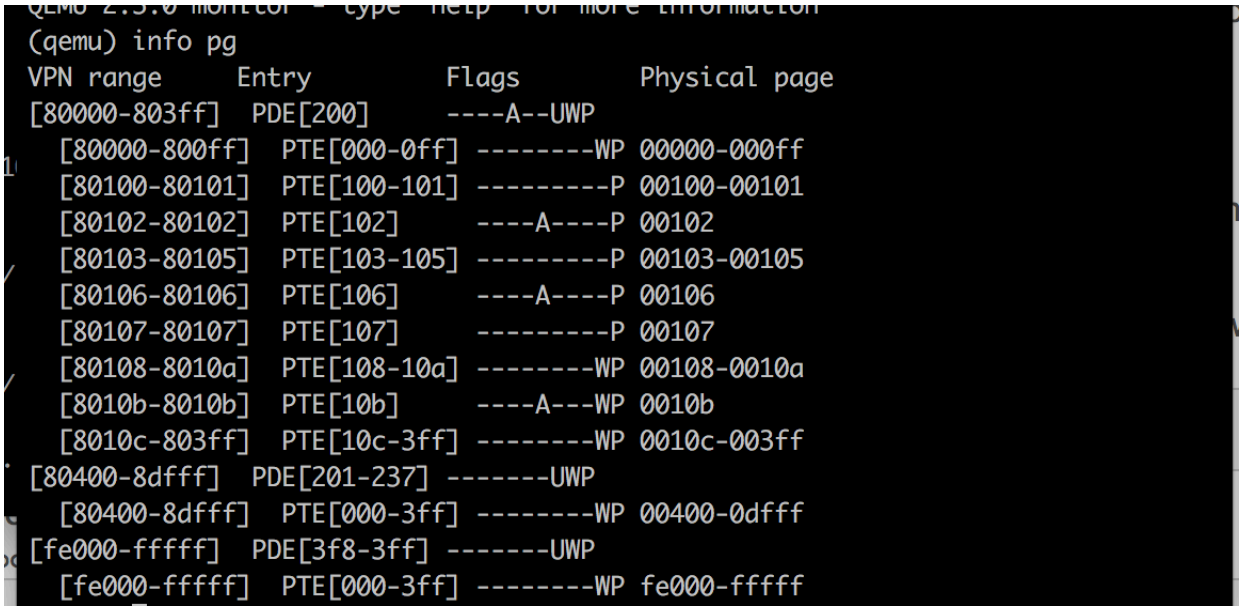
- nature of segmentation:

  Base address plus offset to isolate all the processes from each other and simplify the archetechure of the process memory by using base registers cs, ds, ss and so on.

- nature of paging:

  Devide all the physical memory into multiple pages size of 4kb. A process can make better use of memory: it just need to load the pages it needs into the memory instead the whole segmentation and also can avoid fragments in memory.

Question 2: What is the state of page tables after xv6 is done initializing the first 4K page table?



- 0x80000000-0x803fffff PDE, accessible and user access and writable and present
  - 0x80000000-0x800fffff, 0x80108000-0x8010afff, 0x8010c000-0x803fffff

    PTE, writable and present
  - 0x80100000-0x80101fff, 0x80103000-0x80105fff, 0x80107000-0x80107fff

    PTE, present
  - 0x80102000-0x80102fff, 0x80106000-0x80106fff

    PTE, accessible and present
  - 0x8010b000-0x8010bfff

    PTE, accessible and writable and present
- 0x80400000-0x8dffffff

  PDE, user access and writable and present
  - 0x80400000-0x8dffffff

    PTE, writable and present
- 0xfe000000-0xffffffff

  PDE, user access and writable and present
  - 0xfe000000-0xffffffff

    PTE, writable and present