Booting

Backward compatibility: x86 segments, descriptor tables

BIOS, Bootloader, Read/write disk sectors, ELF format

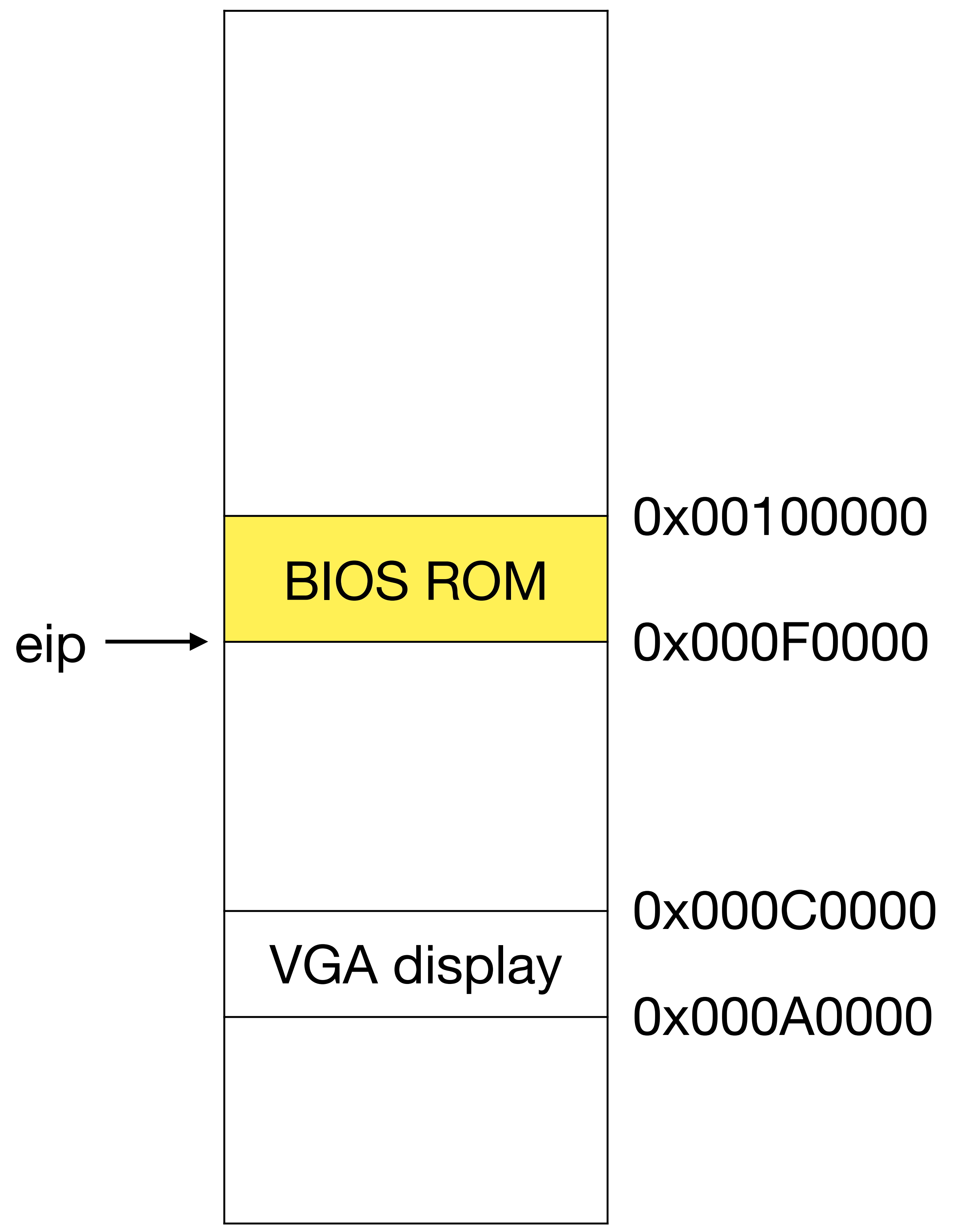
Abhilash Jindal

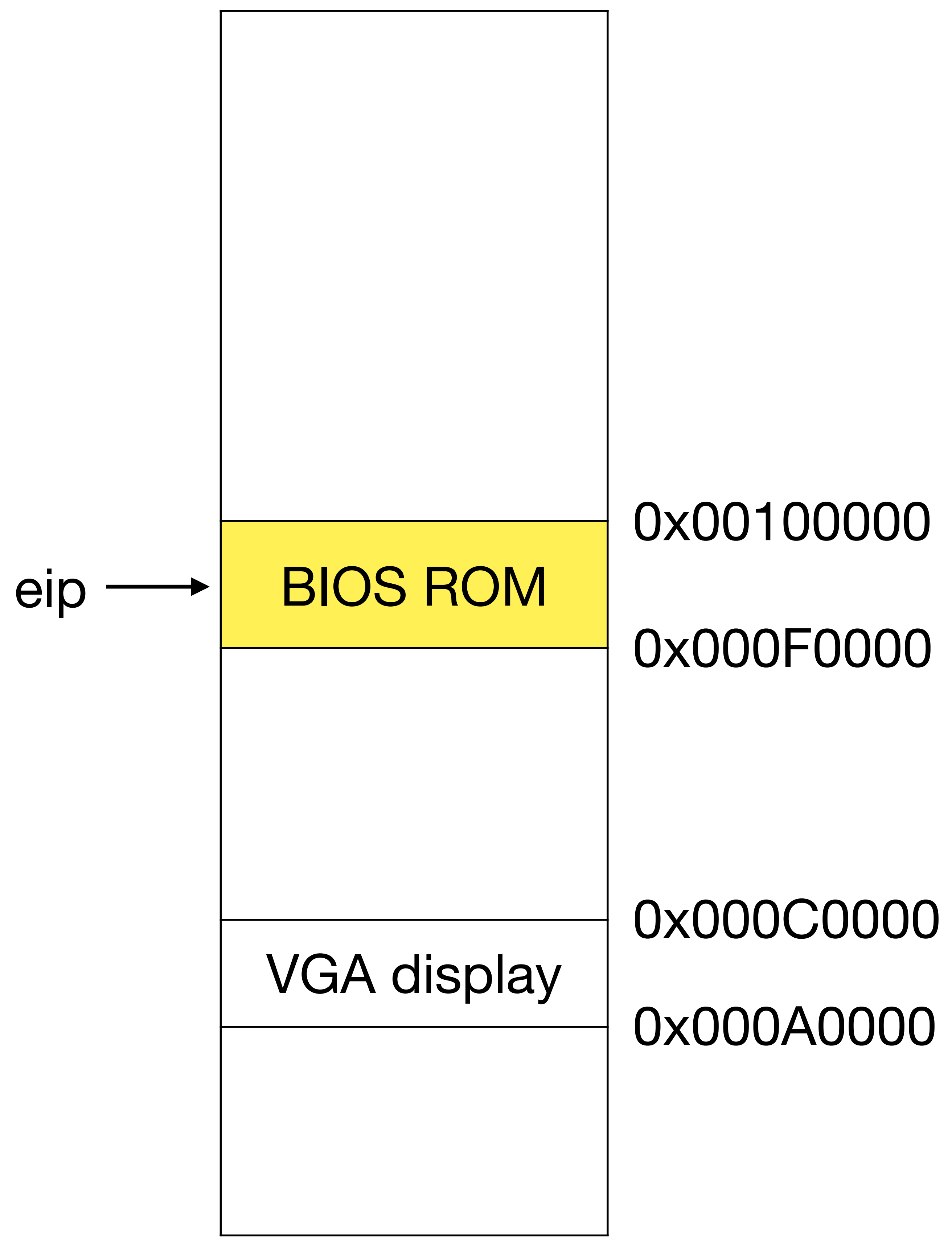
Reference. xv6 book: Appendix B

Agenda

- Boot a minimal OS!
- Understand code
    - Learn what happens when we power on a computer
    - Learn few more x86 details required for booting

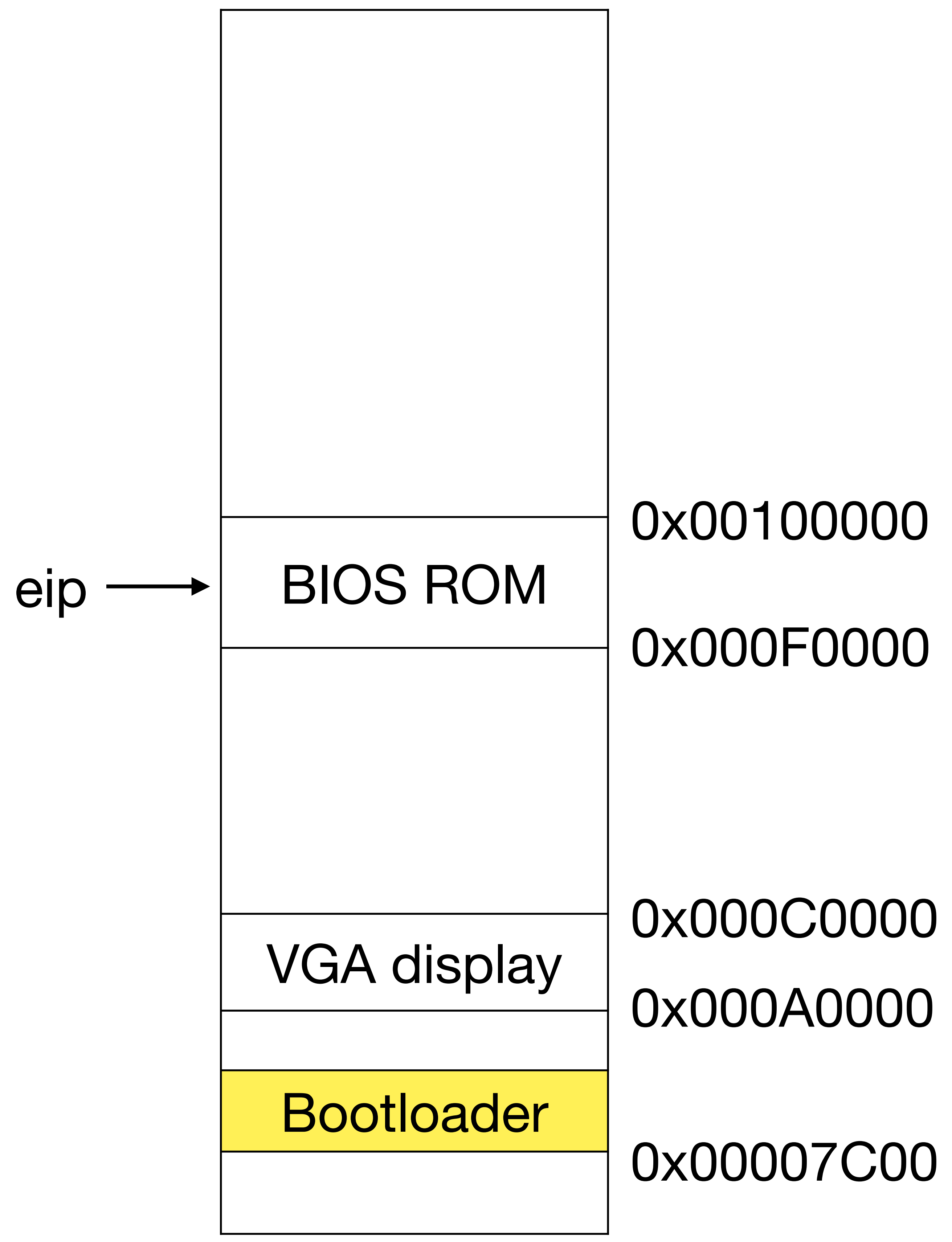
Boot up sequence (1): BIOS

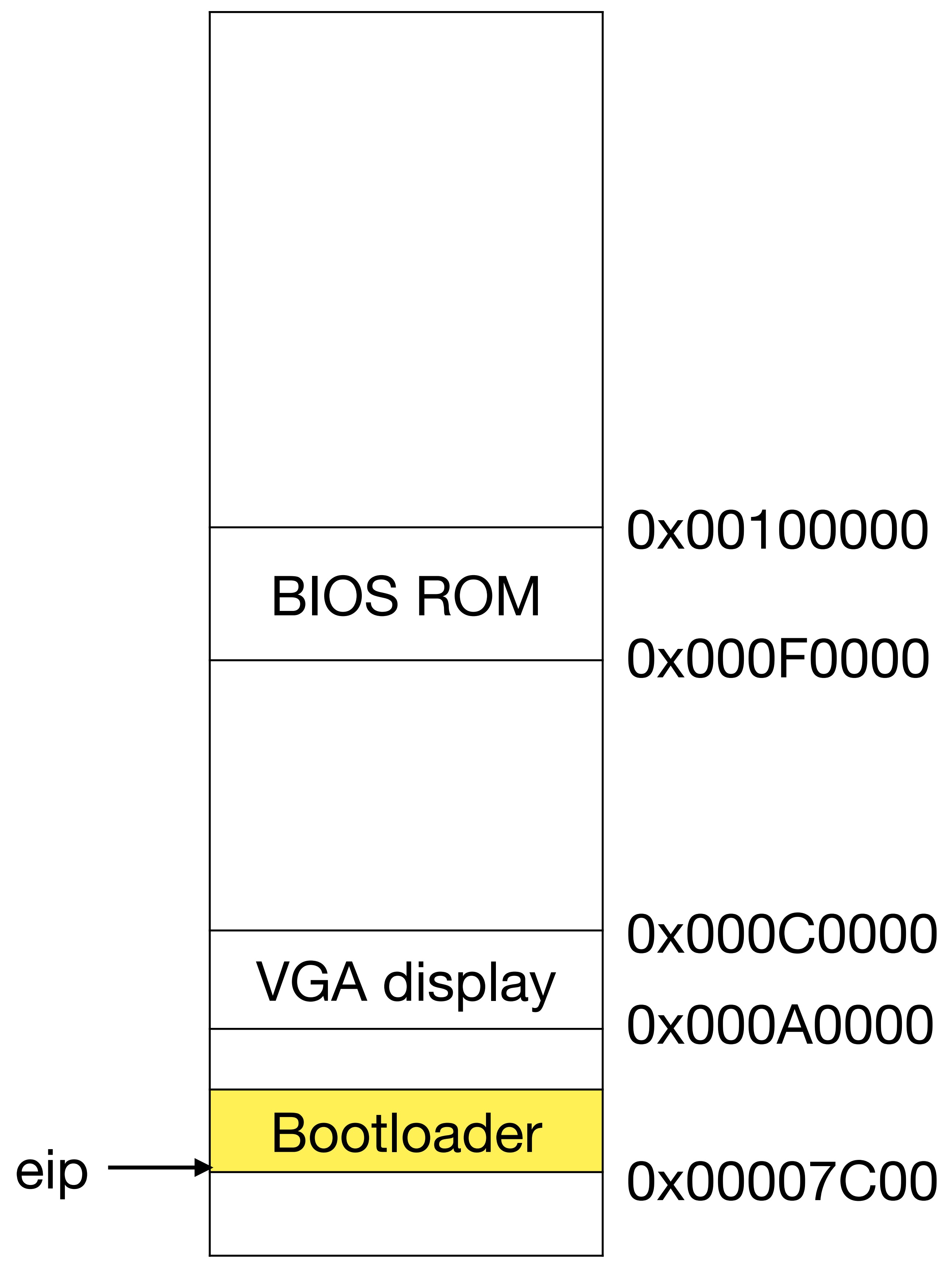




- For backward compatibility, PC boots in 16-bit mode
- BIOS does initial hardware check: are CPUs, memory, disk functional?

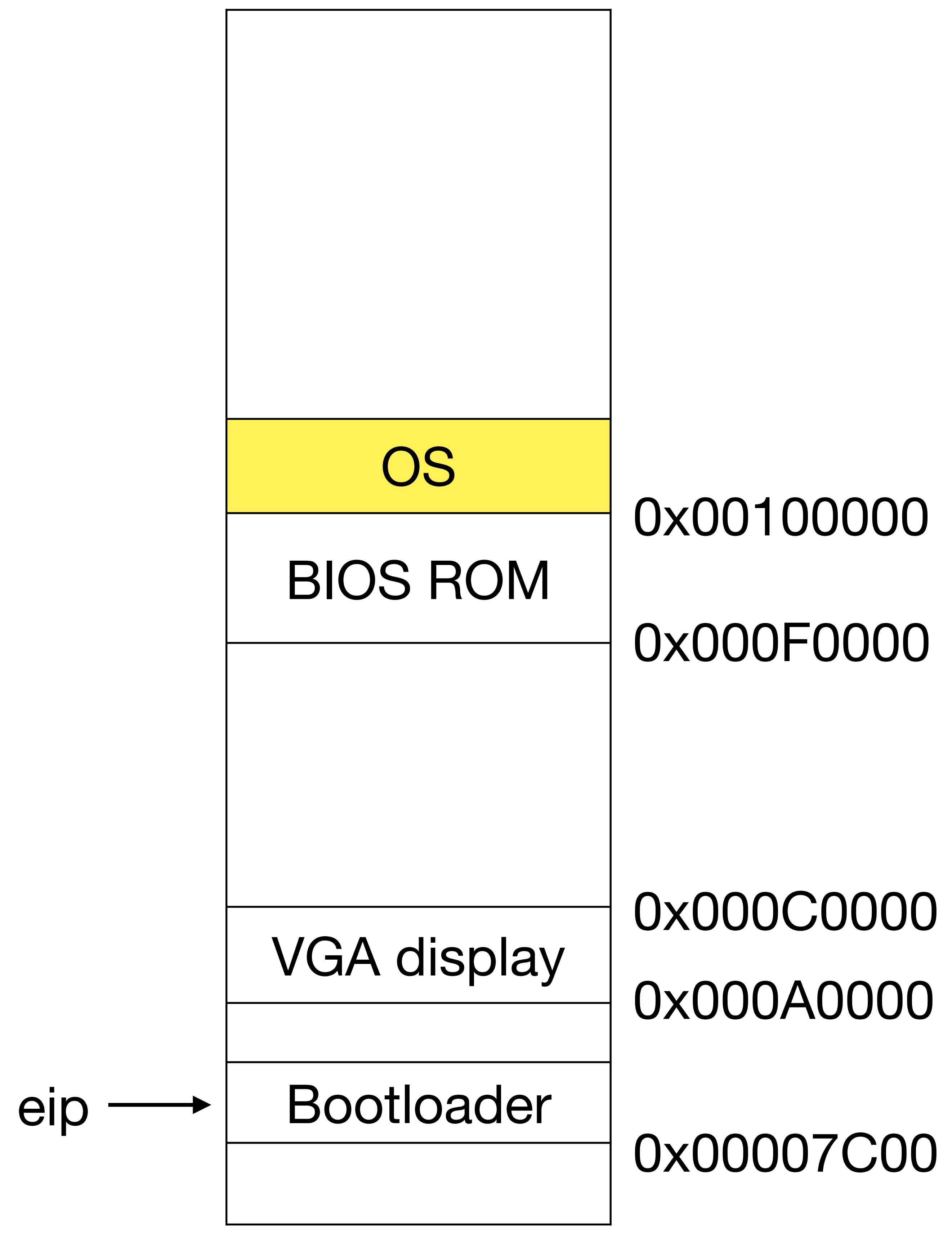
Boot up sequence (2): Bootloader

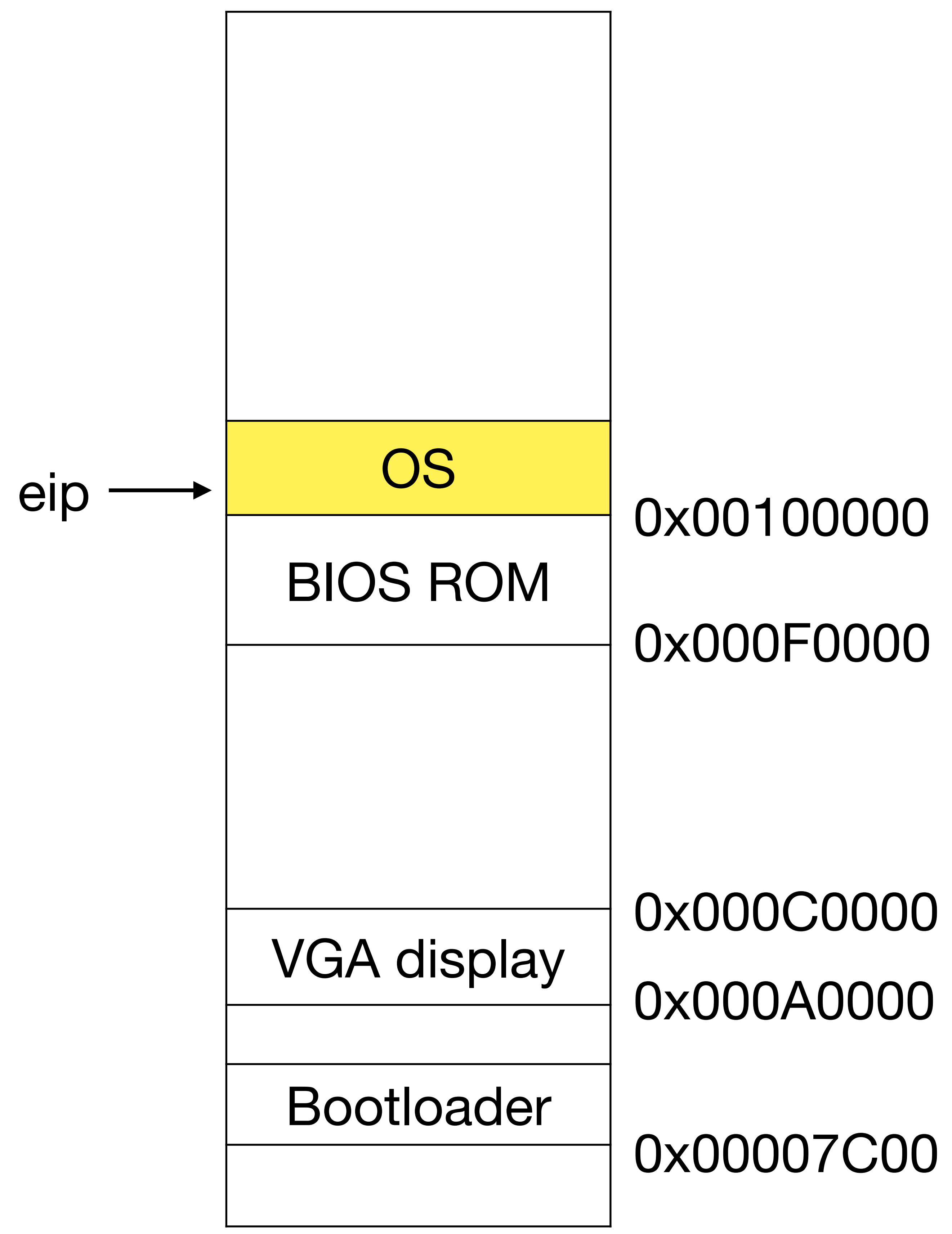




- BIOS loads first disk sector (512 bytes) at 0x7c00 and gives control
    - We need to write a boot loader that fits in the first sector of disk
- Boot loader changes to 32 bit mode

Boot up sequence (3): OS





- gcc prepares OS image in Executable and Linkable Format (ELF)
- Bootloader copies OS image starting from disk sector 1 to 0x100000 and transfers control to it
- We need to tell gcc that image will be loaded at 0x100000

Backward compatibility

- When boot loader gets control, the CPU is in *16-bit mode*
    - This is for backward compatibility. OS written for 16-bit mode should just work for 32-bit and 64-bit machines
- Bootloader explicitly switches from 16-bit mode to 32-bit mode
- Understand how hardware provides backward compatibility and some historical details of 16-bit architecture

16-bit registers

- All registers were 16-bit on 16-bit CPU
- movw %ax, %bx : move low 16 bits of %eax into 16 bits of %ebx
- movb %al, %bl : move low 8 bits of %eax into 8 bits of %ebx
- When CPU is in 16-bit mode, 32-bit and 64-bit machines continue to support same opcode for these instructions

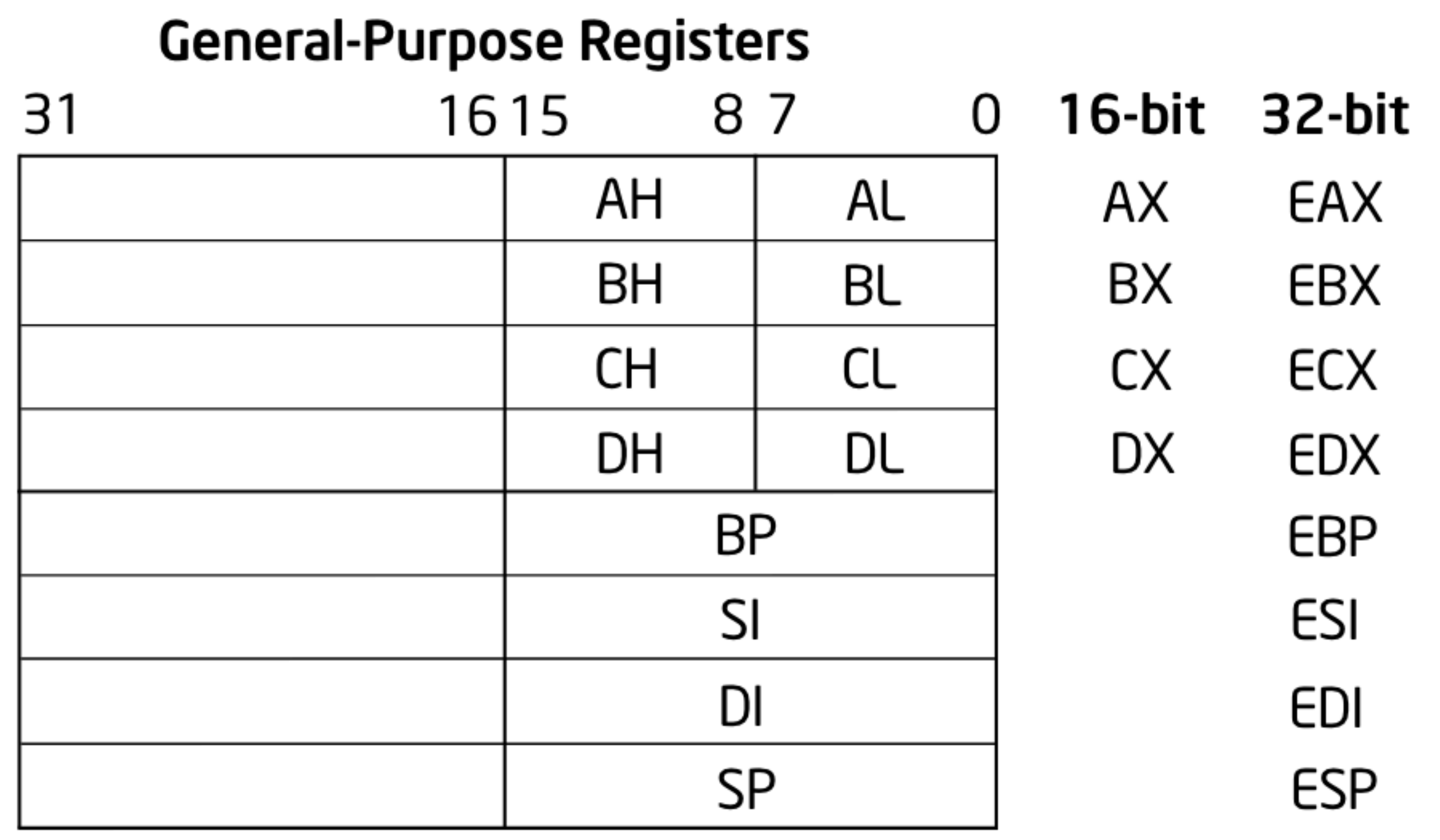
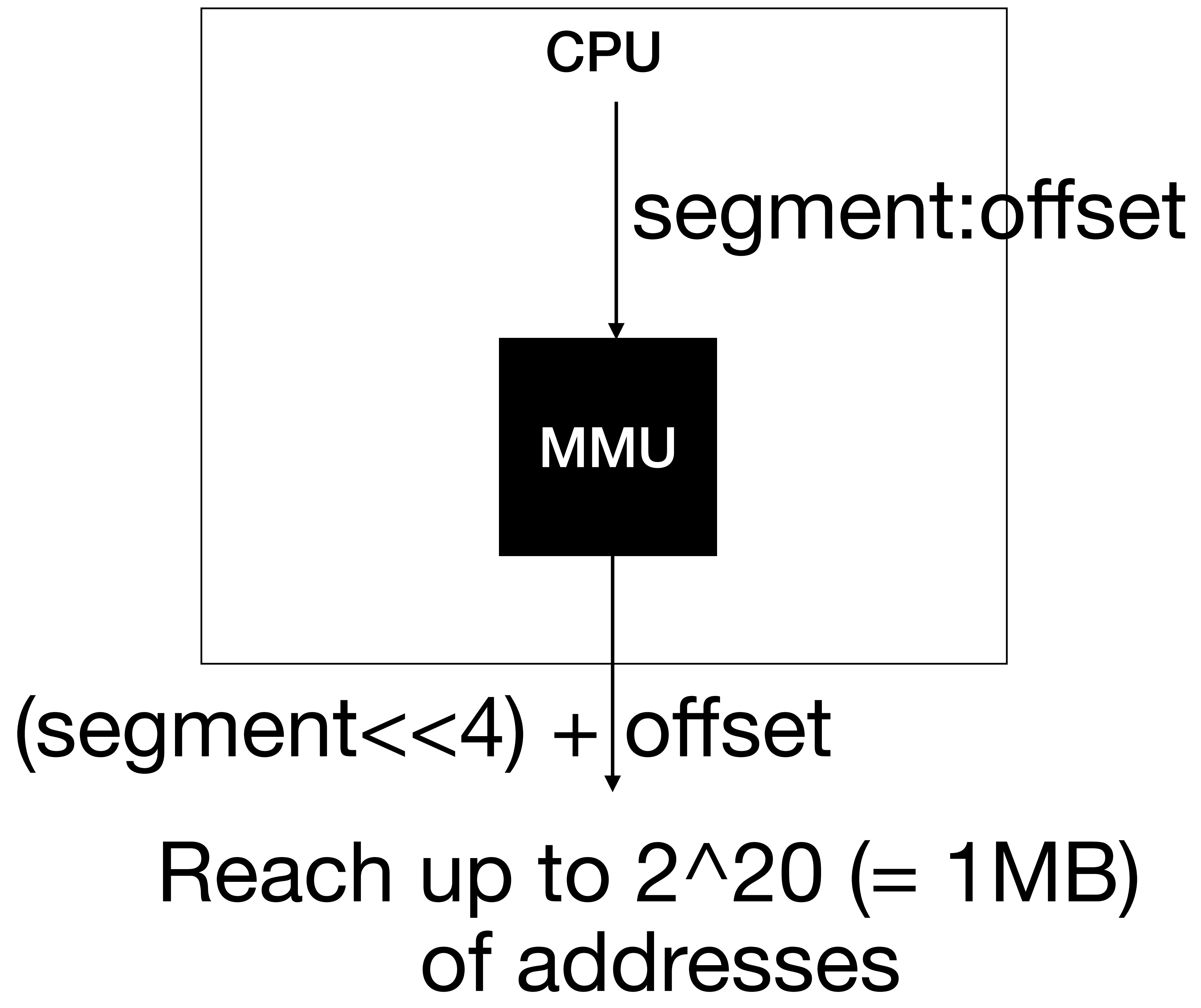


Figure 3-5. Alternate General-Purpose Register Names

Segment registers

- 16-bit registers can only point up to 2^16 (=64KB) addresses in DRAM
- Full address = (segment registers : offset)
  - code segment (cs): ip
  - stack segment (ss): sp / bp.
    push, pop
  - data segment (ds): ax, bx, cx, dx.
    mov (%bx) %ax
  - extra segment (es): si, di
    movsb



Far pointers in 16-bit x86

```c
#include <stdio.h>
int foo() {
   char far *p =(char far *)0x55550005;
   char far *q =(char far *)0x53332225;
   *p = 80;
   (*p)++;
   printf("%d",*q);
   return 0;
}

Outputs 81
```

- p points to (0x5555 << 4) + 0x0005 = 0x55555
- q points to (0x5333 << 4) + 0x2225 = 0x55555
- Multiple ways of referencing same address making them awkward to control

Segment registers in 32-bit

- 32-bit registers can point to 2^32 (=4GB) memory.
- “Protected mode”: extend segment registers for protection

GDTR: Global descriptor table register

cs: top 13 bits
0b…10xxx

Global Descriptor Table

S.No.*	Permission	Base	Limit
0x01	WRITE		
0x02	EXECUTE	0x1F	0xFF
…	…	…	…

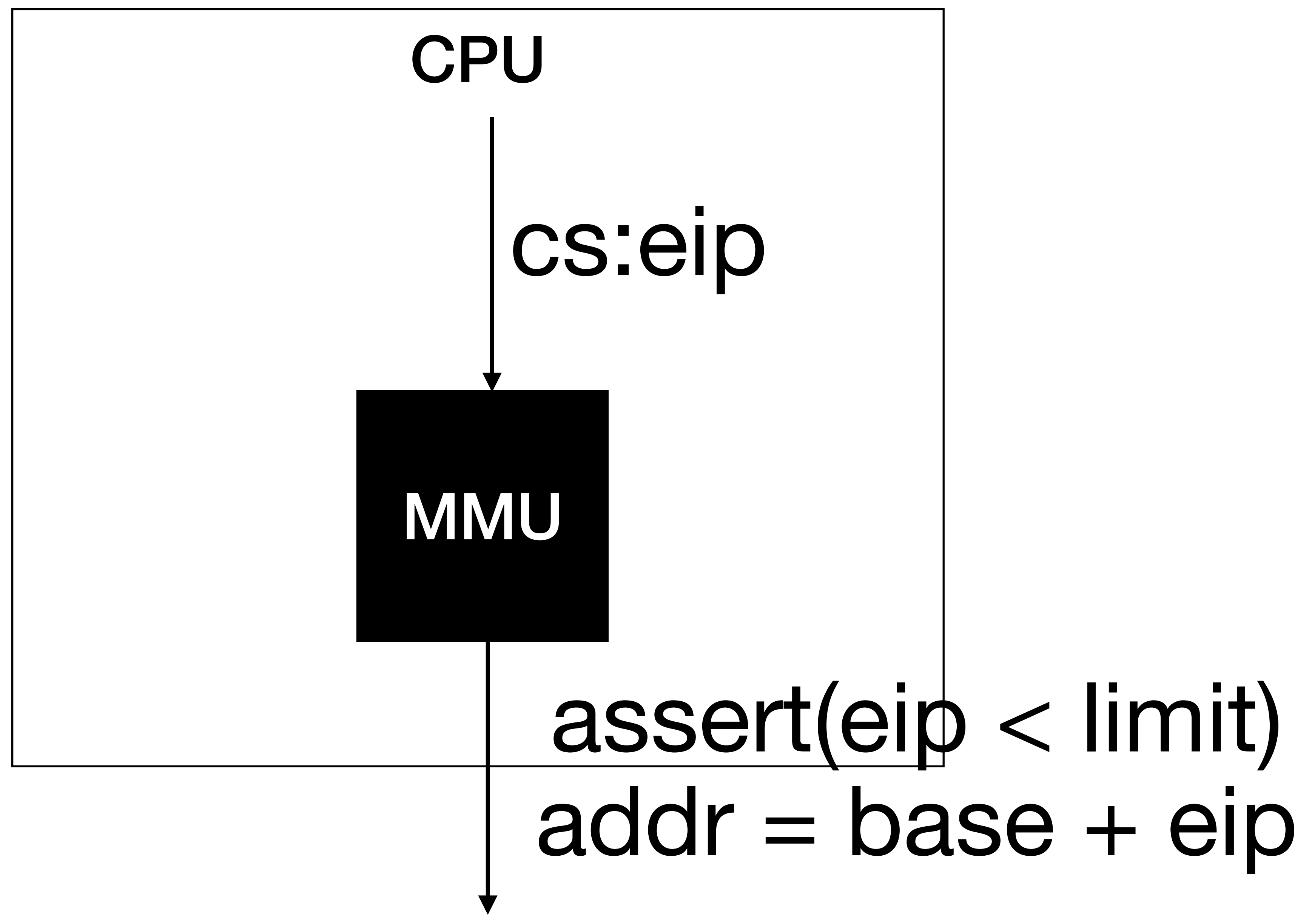
*: S.No. added only for illustration

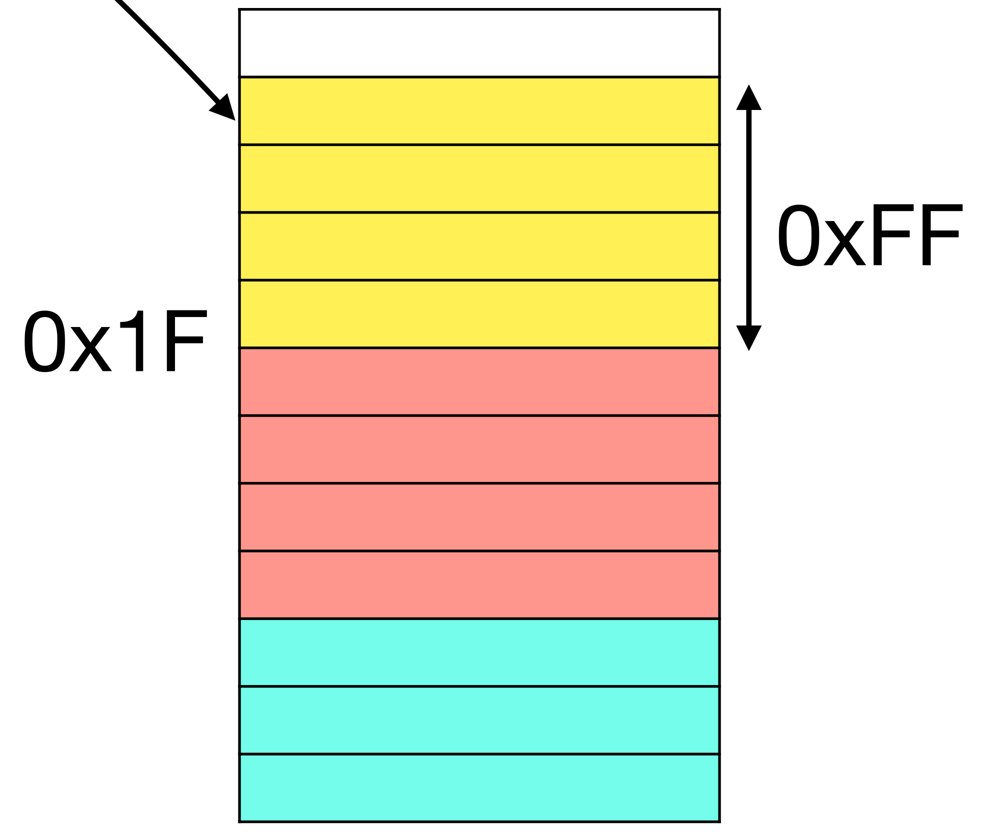
Address translation

cs: top 13 bits
0b…10xxx

Global Descriptor Table

S.No.*	Permission	Base	Limit
0x01	WRITE		
0x02	EXECUTE	0x1F	0xFF
…	…	…	…

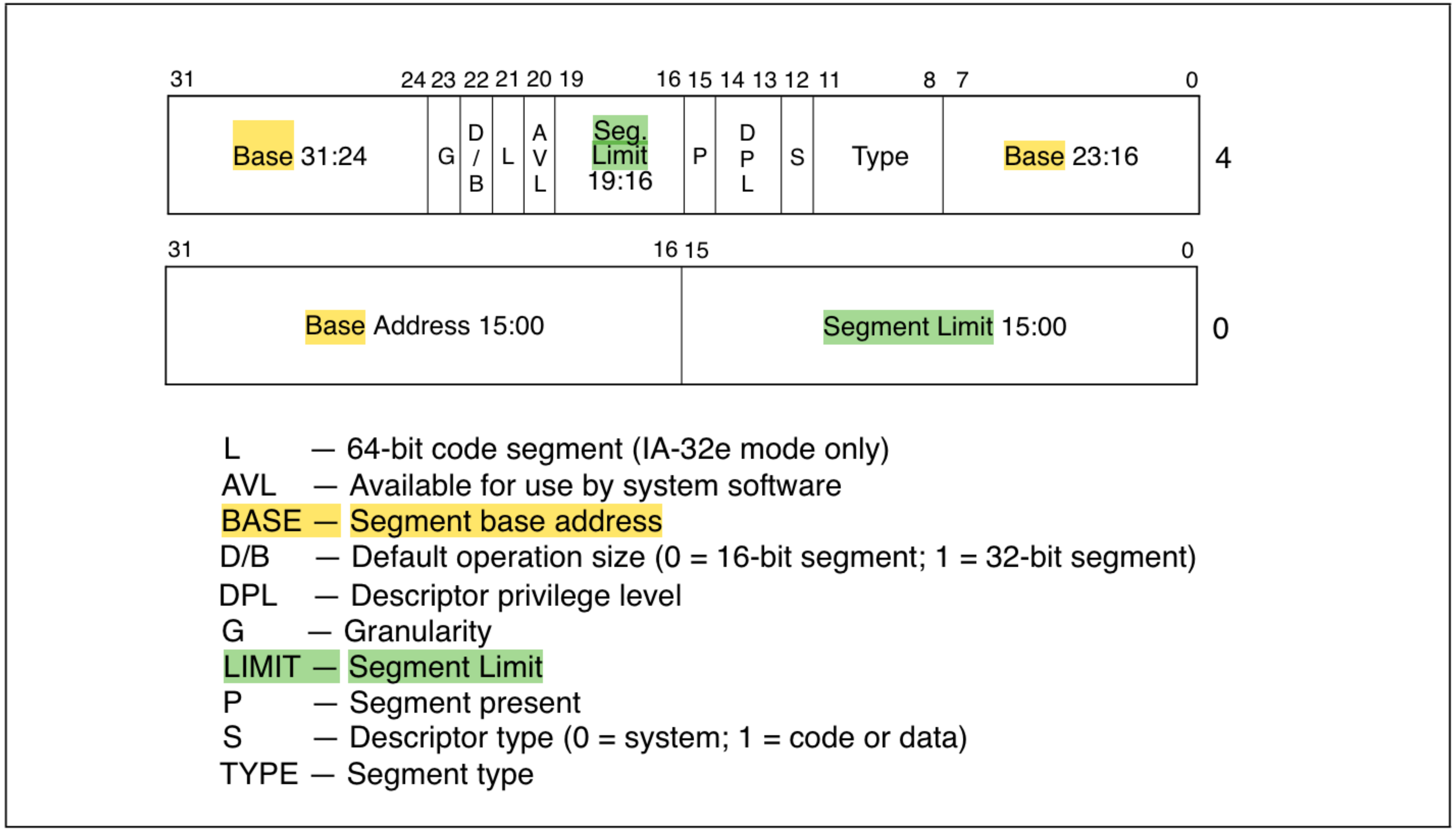




- Can “protect” different segments from each other

Global descriptor table

- Upto 2^13 (=8192) segment descriptors
- Segment descriptor:
    - 32 bit base
    - 20 bit limit
    - If G=1, granularity=4KB.
    - Max memory within 1 segment = 2^20*2^12 = 4GB



L	— 64-bit code segment (IA-32e mode only)
AVL	— Available for use by system software
BASE	— Segment base address
D/B	— Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL	— Descriptor privilege level
G	— Granularity
LIMIT	— Segment Limit
P	— Segment present
S	— Descriptor type (0 = system; 1 = code or data)
TYPE	— Segment type

Figure 3-8. Segment Descriptor

Segmented memory model

- Stack cannot grow into code section

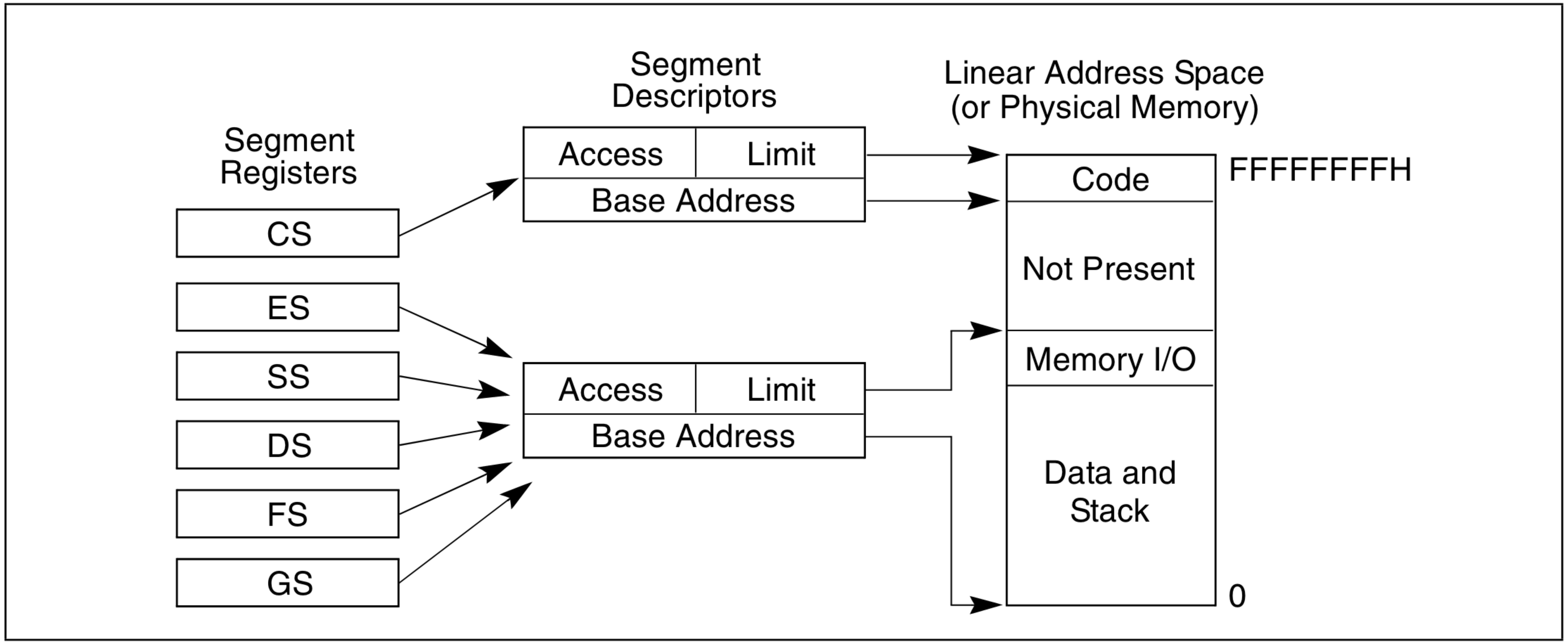


Figure 3-3. Protected Flat Model

Multi-segment model

- Best protection
- Difficult to program

movl %esp %ecx

addl $1 (%ecx)

Does not add 1 to the value at top of the stack!

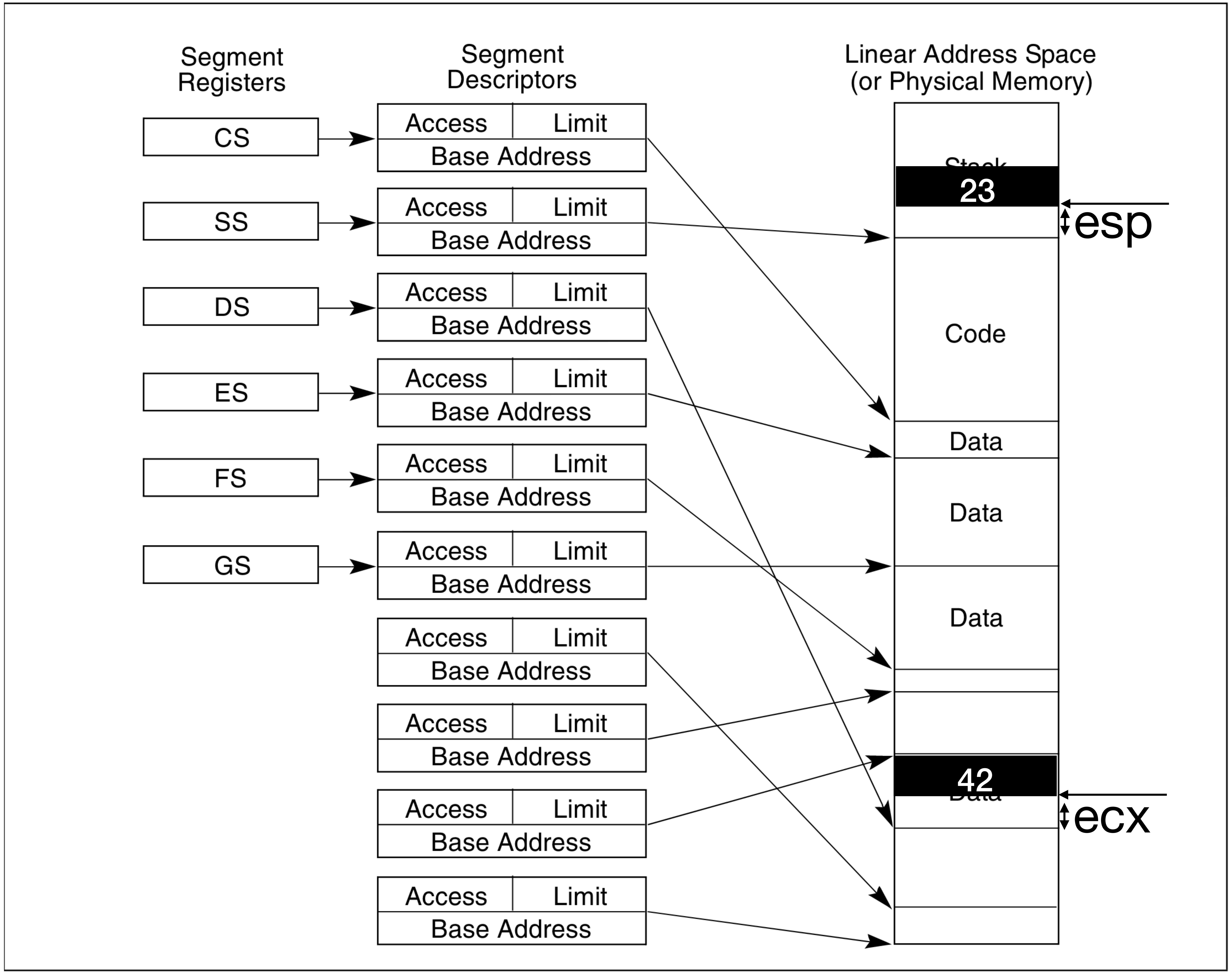


Figure 3-4. Multi-Segment Model

Flat memory model

- Easier to program
- Used by xv6

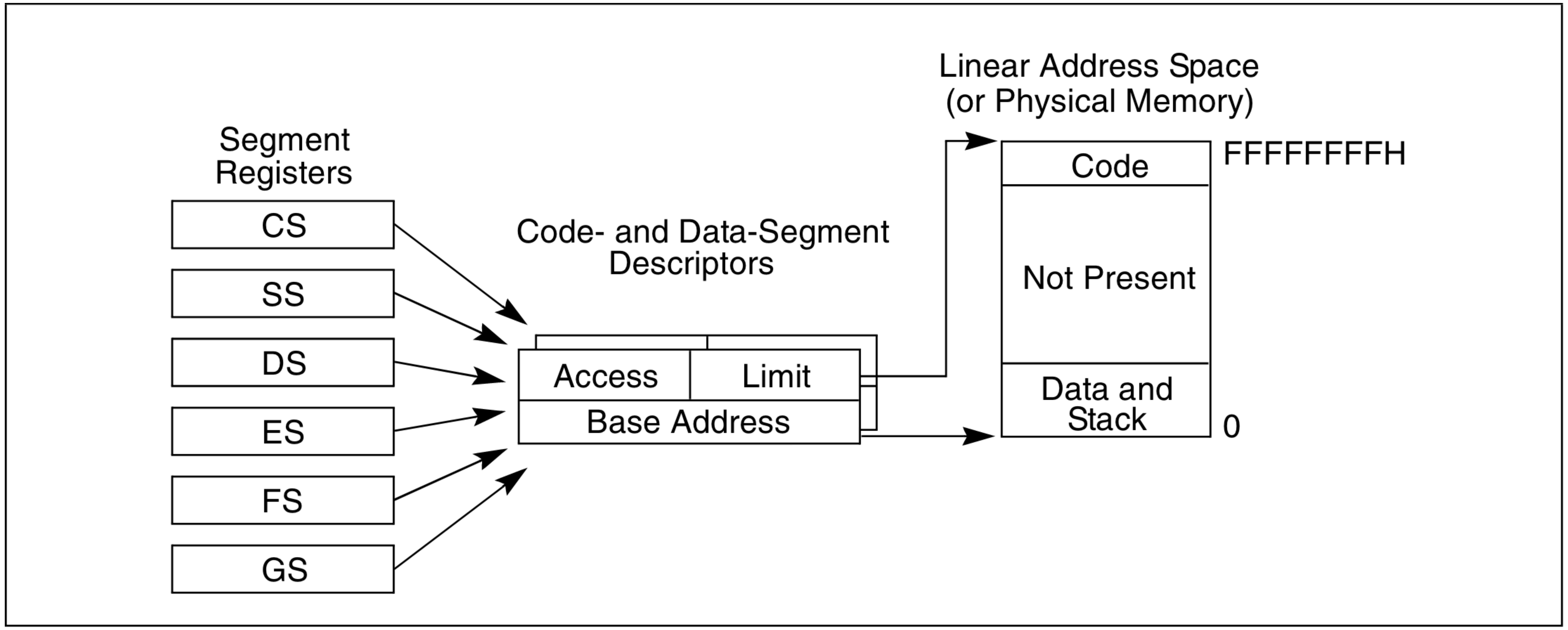


Figure 3-2. Flat Model