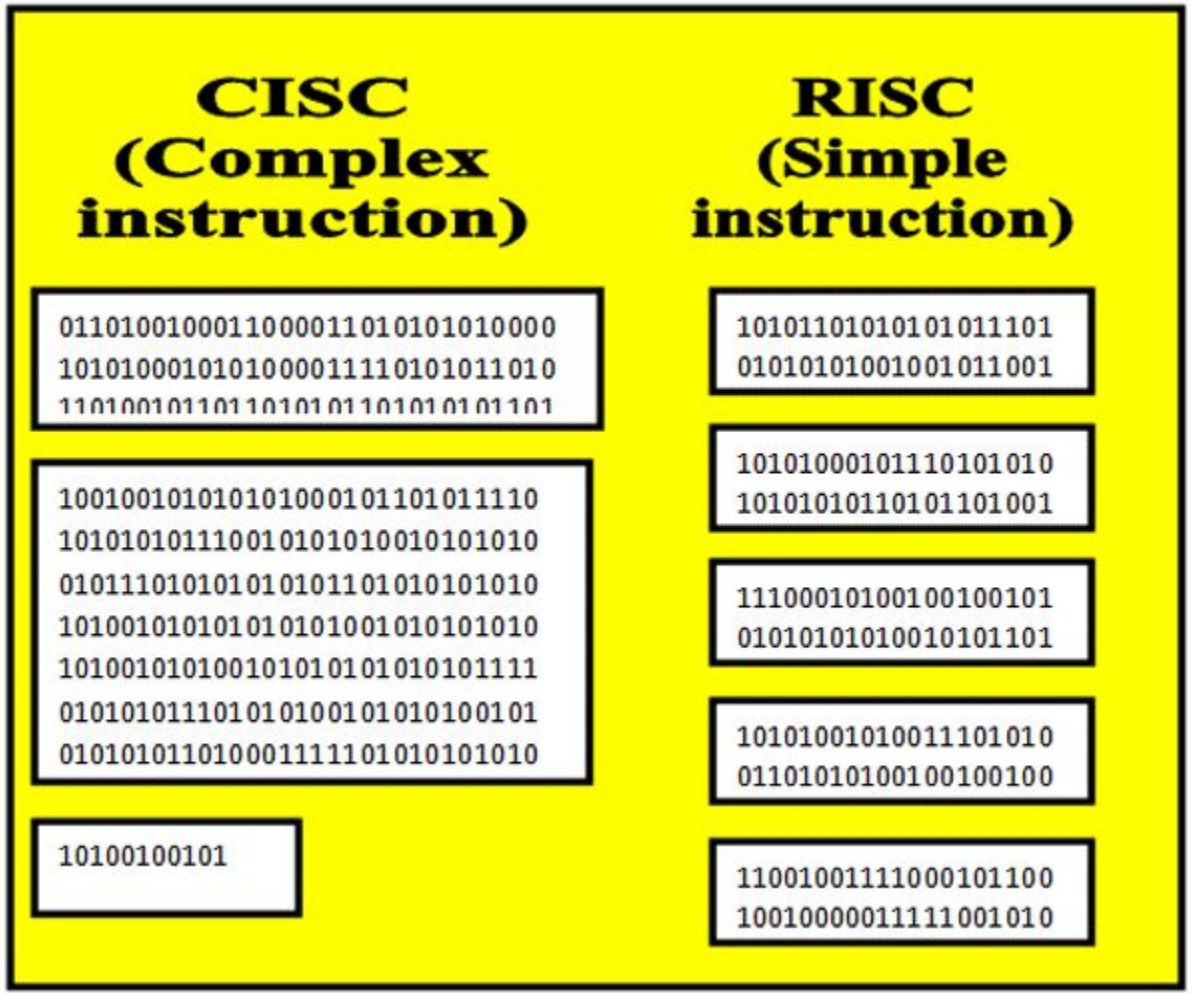
XV6 Introduction

Kowshic Roy

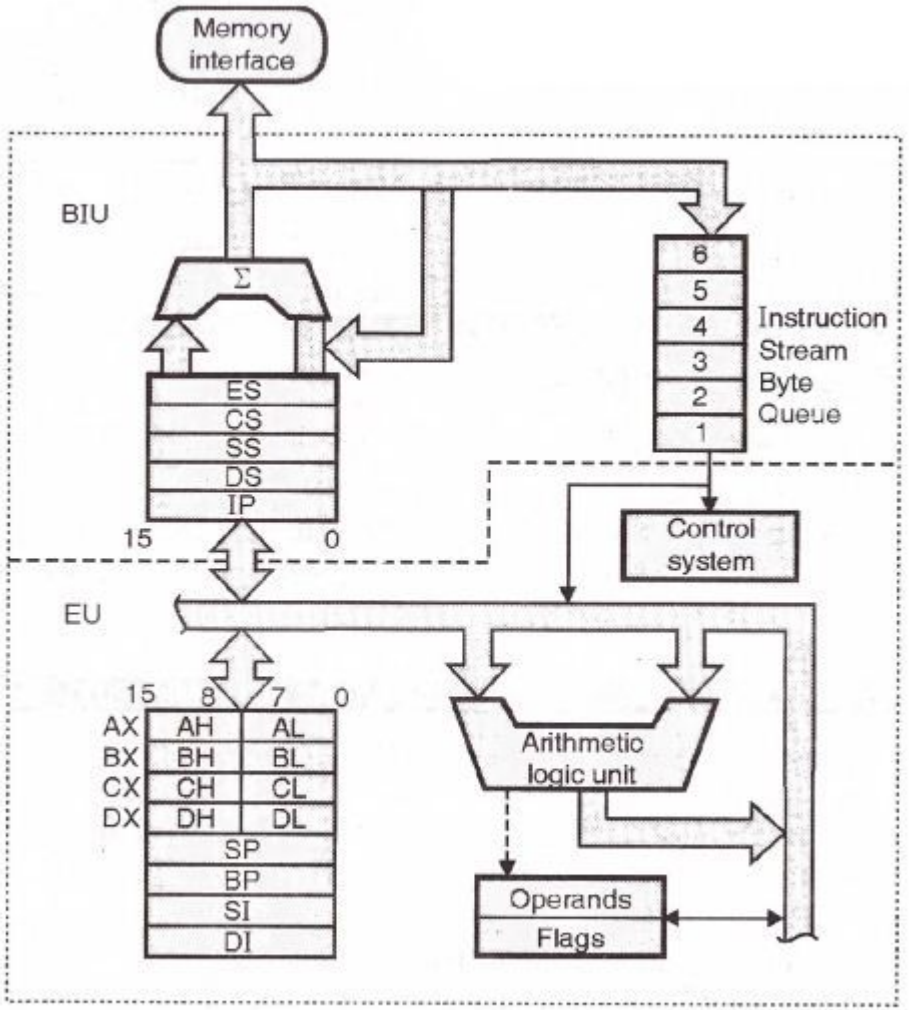
xv6 origin

- A teaching OS developed in MIT.
- Unix v6
- RISC - V



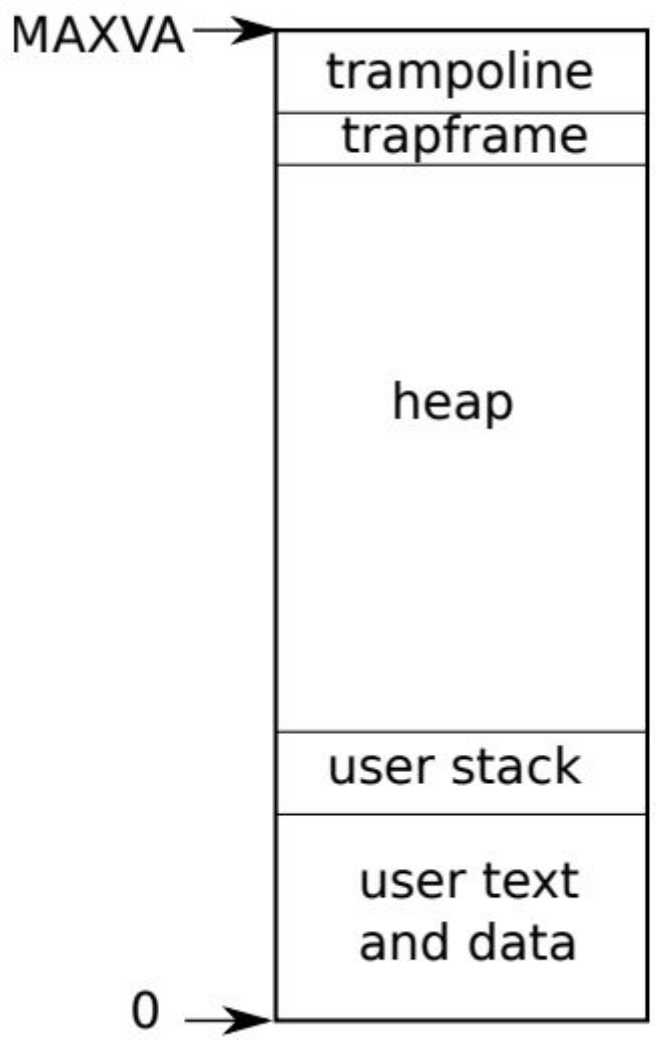
A machine

- How many bit?



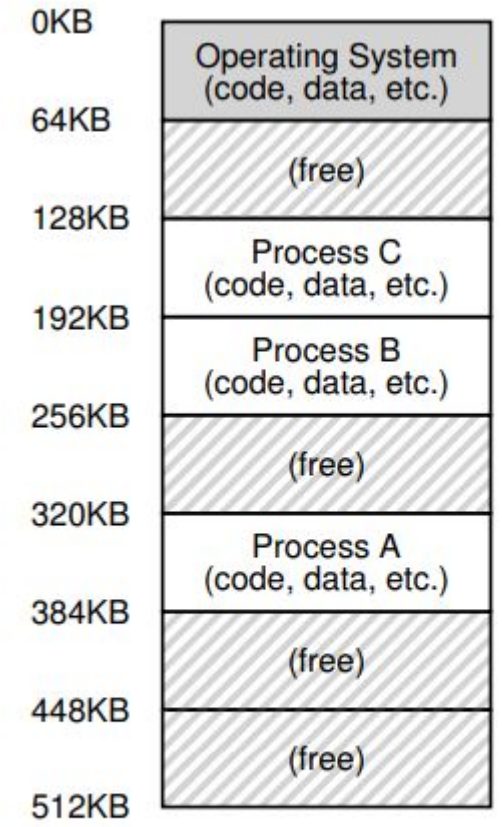
8086 internal architecture

A process


MAXVA
trampoline
trapframe
heap
user stack
user text
and data
0

Why we need OS?

- A machine that is accessed by all of you
- Picture oa ram with 64 KB
- Easy snooping by providing physical address.
- Extra malloc to hijack other's memory address.



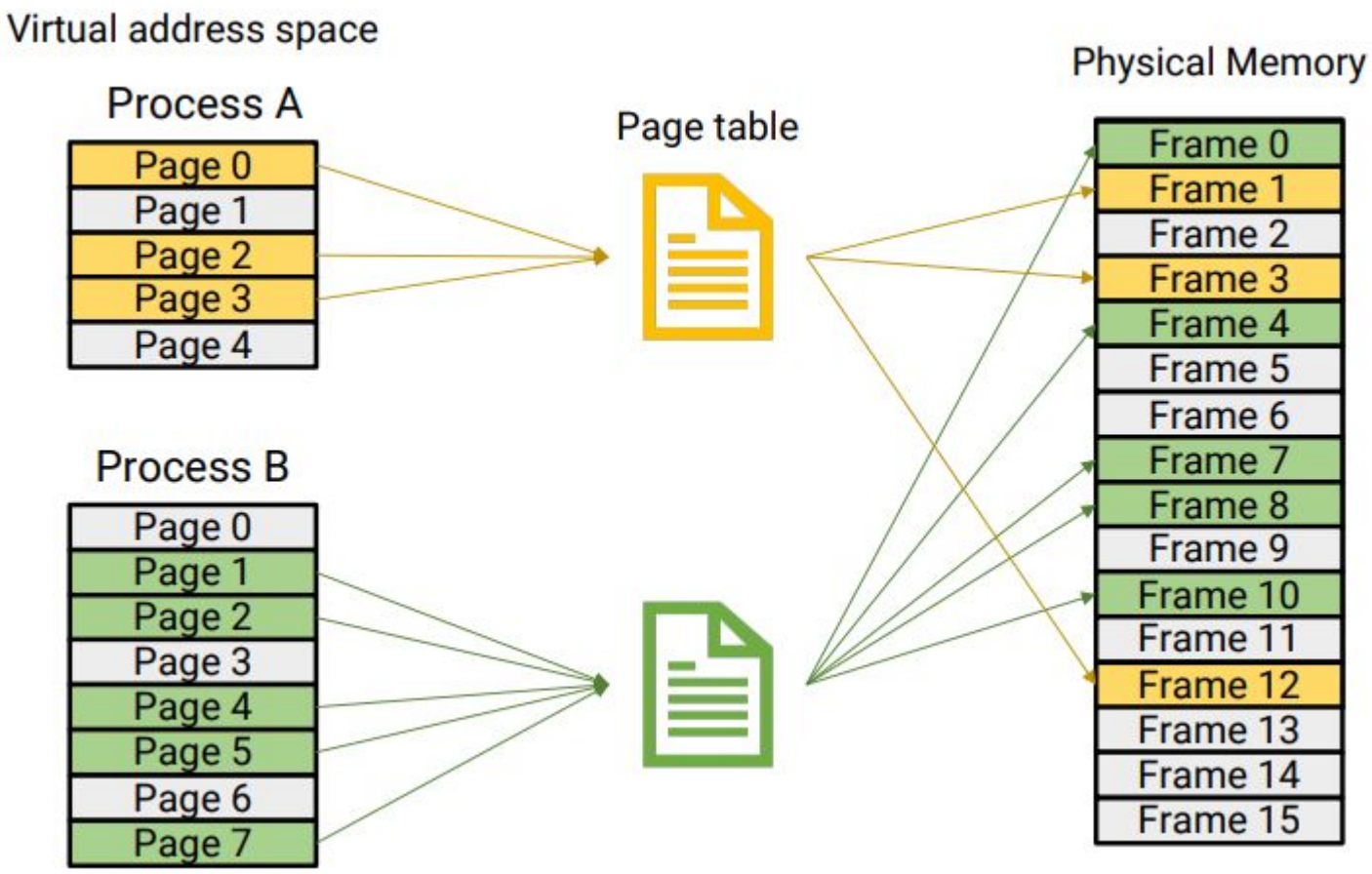
13.2: **Three Processes: Sharing Memory**

Assembly Code

`/* Type your code here, or load an example. */`
`int square(int num) {`
`    return num * num;`
`}`
	`square:`
	`        push    rbp`
	`        mov     rbp, rsp`
	`        mov     DWORD PTR [rbp-4], edi`
	`        mov     eax, DWORD PTR [rbp-4]`
	`        imul    eax, eax`
	`        pop     rbp`
	`        ret`

`fptr = fopen("filename.txt", "w");`	`        lui     a5,%hi(.LC0)`
	`        addi    a1,a5,%lo(.LC0)`
	`        lui     a5,%hi(.LC1)`
	`        addi    a0,a5,%lo(.LC1)`
	`        call    fopen`
	`        mv      a5,a0`
	`        sw      a5,-20(s0)`

Virtual Memory

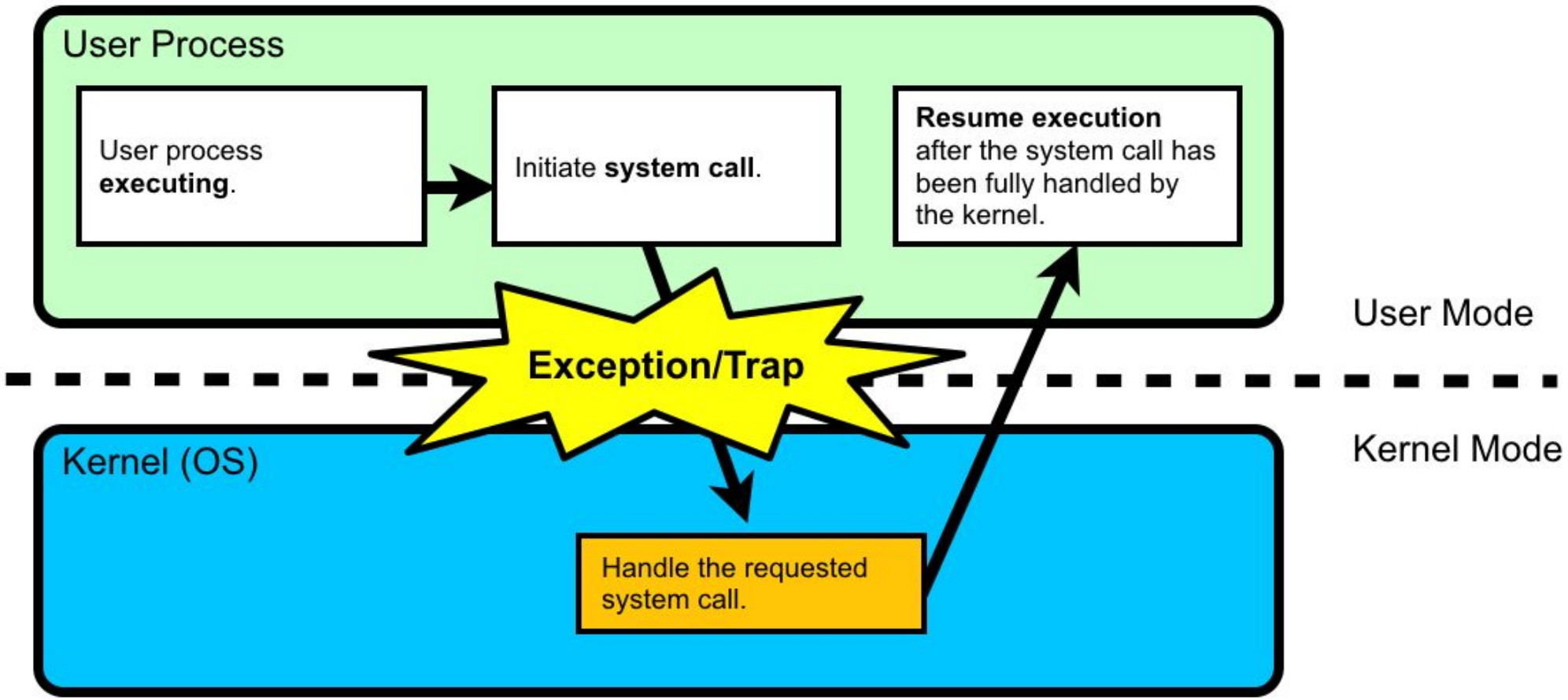


Traps

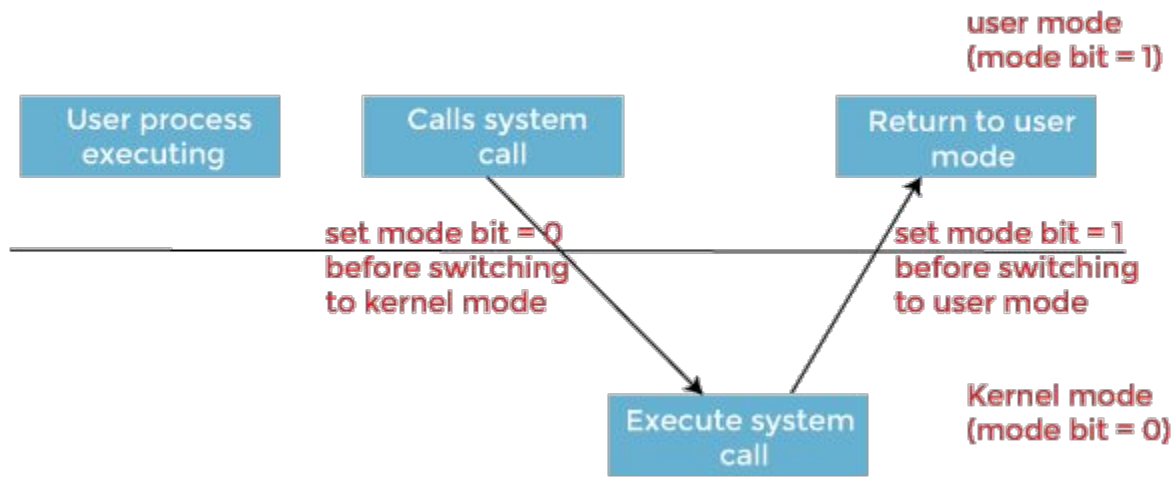
- System Calls
- Exceptions
- Interrupts

User Space	Kernel Space
push rbp mov rbp, rsp mov DWORD PTR [rbp-4], edi mov eax, DWORD PTR [rbp-4] imul eax, eax pop rbp ret	. . . sbrk: push rbp mov rbp, rsp . . .

System call mechanism



System call mechanism

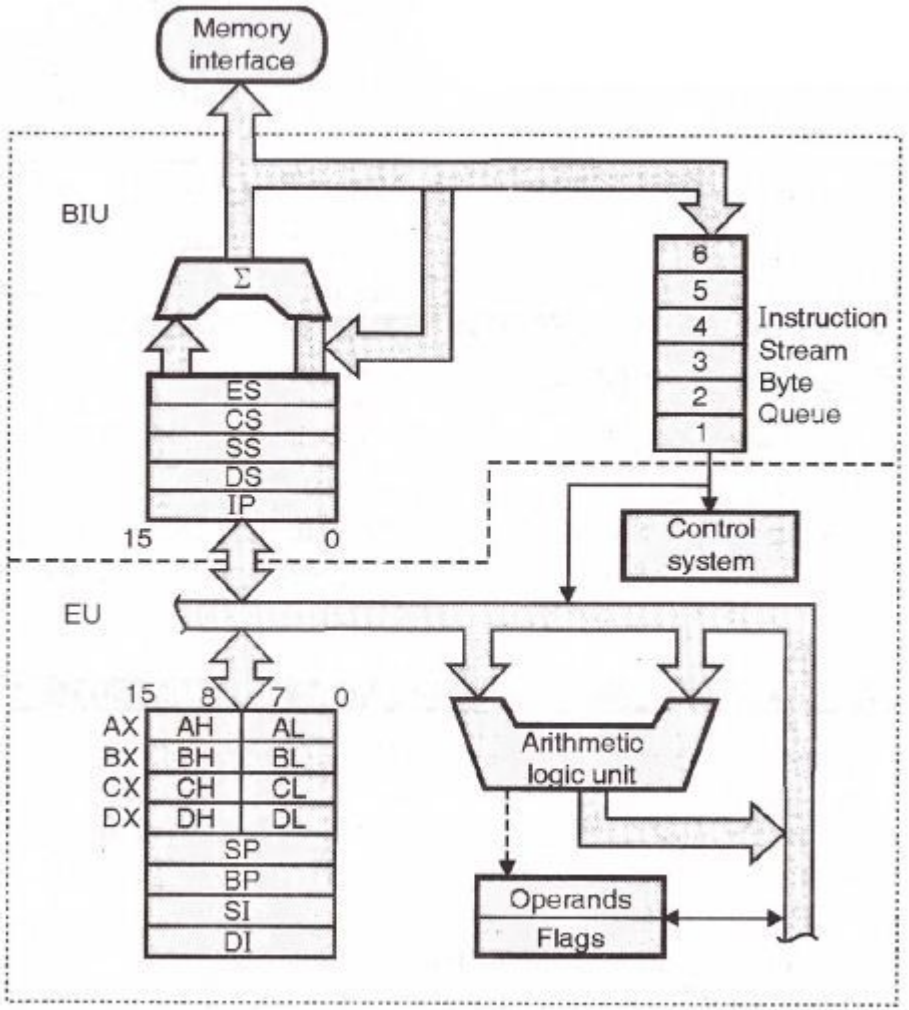


Trap table

System call Name	ID	Memory Address	
Fork	0	0xFFFFF…	
Open	5	0x03FFF….	

A machine

- How many bit?



8086 internal architecture

OS @ run (kernel mode)	Hardware	Program (user mode)
Create entry for process list		
Allocate memory for program		
Load program into memory		
Setup user stack with argv		
Fill kernel stack with reg/PC		
return-from-trap		
	restore regs	
	(from kernel stack)	
	move to user mode	
	jump to main	
		Run main()
		...
		Call system call
		trap into OS
	save regs	
	(to kernel stack)	
	move to kernel mode	
	jump to trap handler	
Handle trap		
Do work of syscall		
return-from-trap		
	restore regs	
	(from kernel stack)	
	move to user mode	
	jump to PC after trap	
		...
		return from main
		trap (via `exit()`)
Free memory of process		
Remove from process list		

Figure 6.2: **Limited Direct Execution Protocol**

Sy

System call	Description
int fork()	Create a process, return child's PID.
int exit(int status)	Terminate the current process; status reported to wait(). No return.
int wait(int *status)	Wait for a child to exit; exit status in *status; returns child PID.
int kill(int pid)	Terminate process PID. Returns 0, or -1 for error.
int getpid()	Return the current process's PID.
int sleep(int n)	Pause for n clock ticks.
int exec(char *file, char *argv[])	Load a file and execute it with arguments; only returns if error.
char *sbrk(int n)	Grow process's memory by n bytes. Returns start of new memory.
int open(char *file, int flags)	Open a file; flags indicate read/write; returns an fd (file descriptor).
int write(int fd, char *buf, int n)	Write n bytes from buf to file descriptor fd; returns n.
int read(int fd, char *buf, int n)	Read n bytes into buf; returns number read; or 0 if end of file.
int close(int fd)	Release open file fd.
int dup(int fd)	Return a new file descriptor referring to the same file as fd.
int pipe(int p[])	Create a pipe, put read/write file descriptors in p[0] and p[1].
int chdir(char *dir)	Change the current directory.
int mkdir(char *dir)	Create a new directory.
int mknod(char *file, int, int)	Create a device file.
int fstat(int fd, struct stat *st)	Place info about an open file into *st.
int stat(char *file, struct stat *st)	Place info about a named file into *st.
int link(char *file1, char *file2)	Create another name (file2) for the file file1.
int unlink(char *file)	Remove a file.

Figure 1.2: Xv6 system calls. If not otherwise stated, these calls return 0 for no error, and -1 if there's an error.

And many mores we will add …