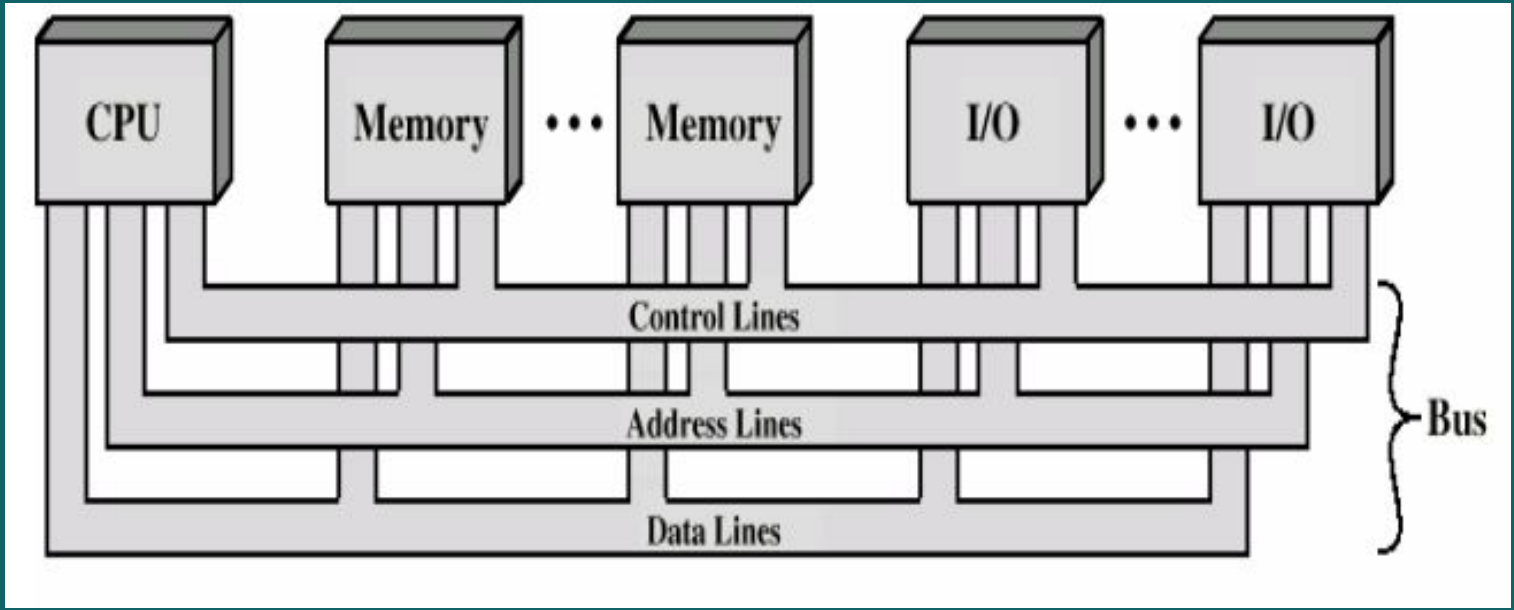
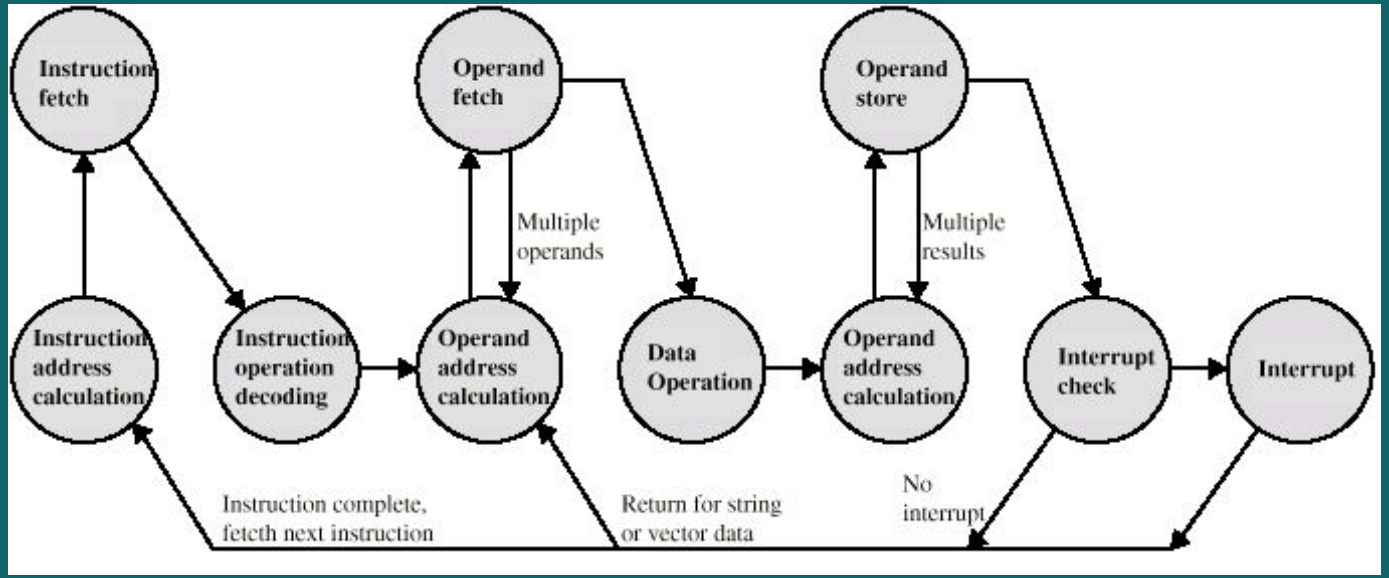
Interrupciones y E/S en 8086

Arquitectura de Computadoras - Práctico 10

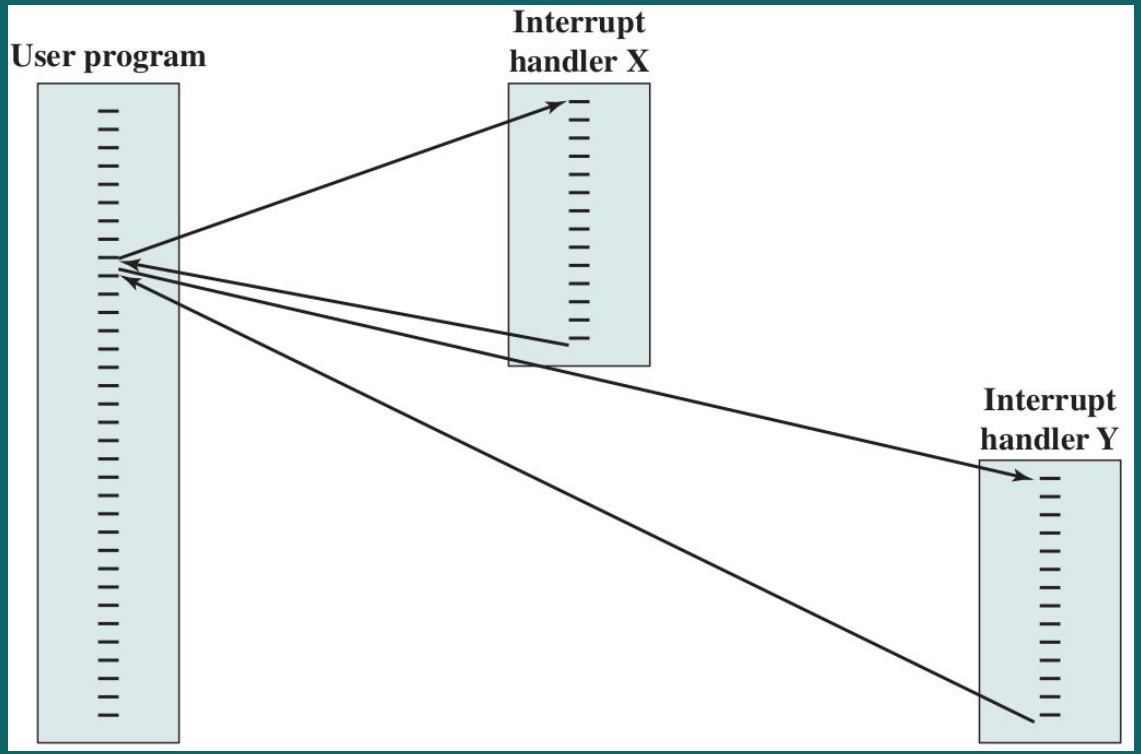
Contextualizando



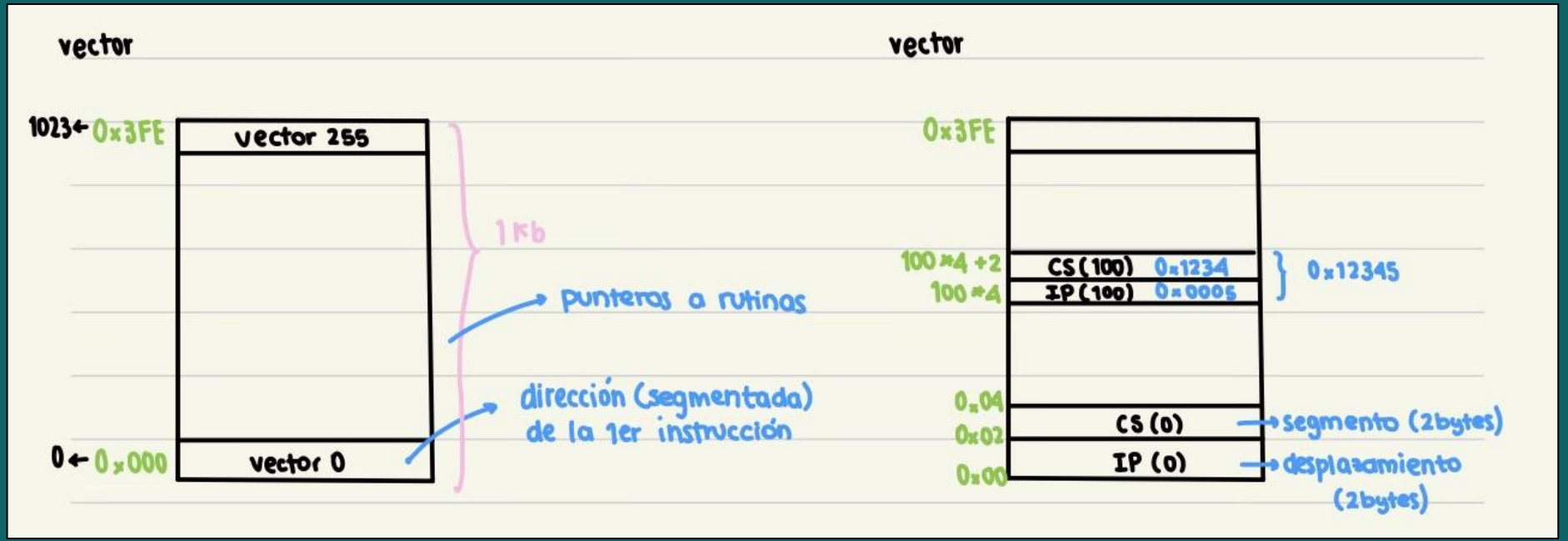
Contextualizando



Contextualizando



Vector interrupciones 8086



Aspectos de bajo nivel

- Compilar el alto nivel para ejecutar en un procesador.
- Detalles de 8086
  - in, out, enable y disable
  - iret
  - vector de interrupciones
  - instalación de manejadores de interrupción
  - otras instrucciones pushf, popf

Aspectos de bajo nivel

- Guardar contexto
- Requerimiento de transparencia

Ejercicio

Se desea controlar una bomba de agua mediante un microprocesador. La bomba se enciende colocando el valor 1 en el bit 2 del puerto de E/S BOMBA, y se apaga colocando el valor 0 en el mismo bit. Dado que el encendido no es inmediato, existe un sensor externo que interrumpe cuando se activa efectivamente la bomba.

Cuando la bomba está activa se debe encender un LED conectado al bit 4 del puerto BOMBA, en otro caso se debe desactivar el LED.

Escribir la rutina de atención a la interrupción y las subrutinas ActivarBomba() y DesactivarBomba().

Alto nivel

¿Quién decide cuándo se encender o apagar la máquina?

¿La máquina es dedicada?

¿La rutina ActivarBomba() debe garantizar que la bomba está encendida luego de su ejecución?

Alto nivel

```
void init{
        disable()
        /Instalar interrupciones
        short bom_efectivo=0
        short led=0
        enable()
}
void ActivarBomba(){
        out(BOMBA, 2+led)
        while bom_efectivo!=1{}
}
void DesactivarBomba(){
        out(BOMBA, 0)
        bom_efectvio=0
        led=0
}
void InterrupcionBomba{
        bom_efectivo=1
        led=8
        out(BOMBA, 2+led)
}
```

Asumamos que garantiza encendido

¿Qué variaría del código si no fuese necesario dar esta garantía?

¿Cuál es la diferencia para el programador del programa principal?

Instalación de interrupciones

```
void init{
disable()
/Instalar interrupciones
short Bom_Efectivo=0
short Led=0
enable ()
```

```
BOMBA EQU puerto
ID_INT_BOMBA EQU valor

Bom_Efectivo dw 0
Led dw 0

Init:
XOR AX, AX
MOV ES, AX
cli
MOV ES:[4*ID_INT_BOMBA+2], SEGMENT InterrupcionBomba
MOV ES:[4*ID_INT_BOMBA], OFFSET InterrupcionBomba
sti
RET
```

Ensamblado

```
void ActivarBomba(){
        out(BOMBA, 2+led)
        While Bom_Efectivo!=1{}

}
```

```
ActivarBomba:
    PUSH AX
    MOV AX, [Led]
    ADD AX, 2
    OUT BOMBA, AX \\Prendo la bomba
    XOR AX, AX
Comp:                \\Espero interrupción
    CMP AX, word ptr[Bom_Efectivo]
    JE Comp
    POP AX
    RET
```

Ensamblado

```
void DesactivarBomba(){
        out(BOMBA, 0)
        Bom_Efectivo=0
        Led=0
}

InterrupcionBomba{
        Bom_Efectivo = 1
        Led = 8
        out(BOMBA, 2+led)
}
```

```
DesactivarBomba:
    PUSH AX
    XOR AX, AX
    OUT BOMBA, AX
    MOV word ptr[Bom_Efectivo], 0
    MOV word ptr[Led], 0
    POP AX
    RET

InterrupcionBomba:
    PUSH AX
    MOV word ptr[Bom_Efectivo], 1
    MOV word ptr[Led], 8

    MOV AX, 10
    OUT BOMBA, AX \\Prendo el led
    POP AX
    IRET
```