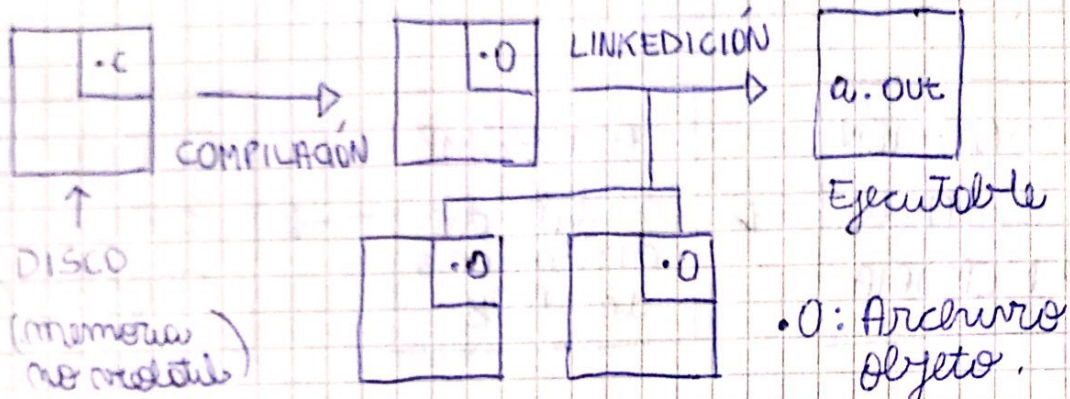
Arquitectura de computadoras

Programas y Binarios =



compilación y linkedición en C =

Programa .C

⇩

Preprocesador (CPP) .C → realiza la macroexpansión de directivas (include, define, ...)

⇩

compilador de C a ASM (gcc) .S → convierte el codigo C en ASM puede ser generado con sintaxis intel o AT&T.

⇩

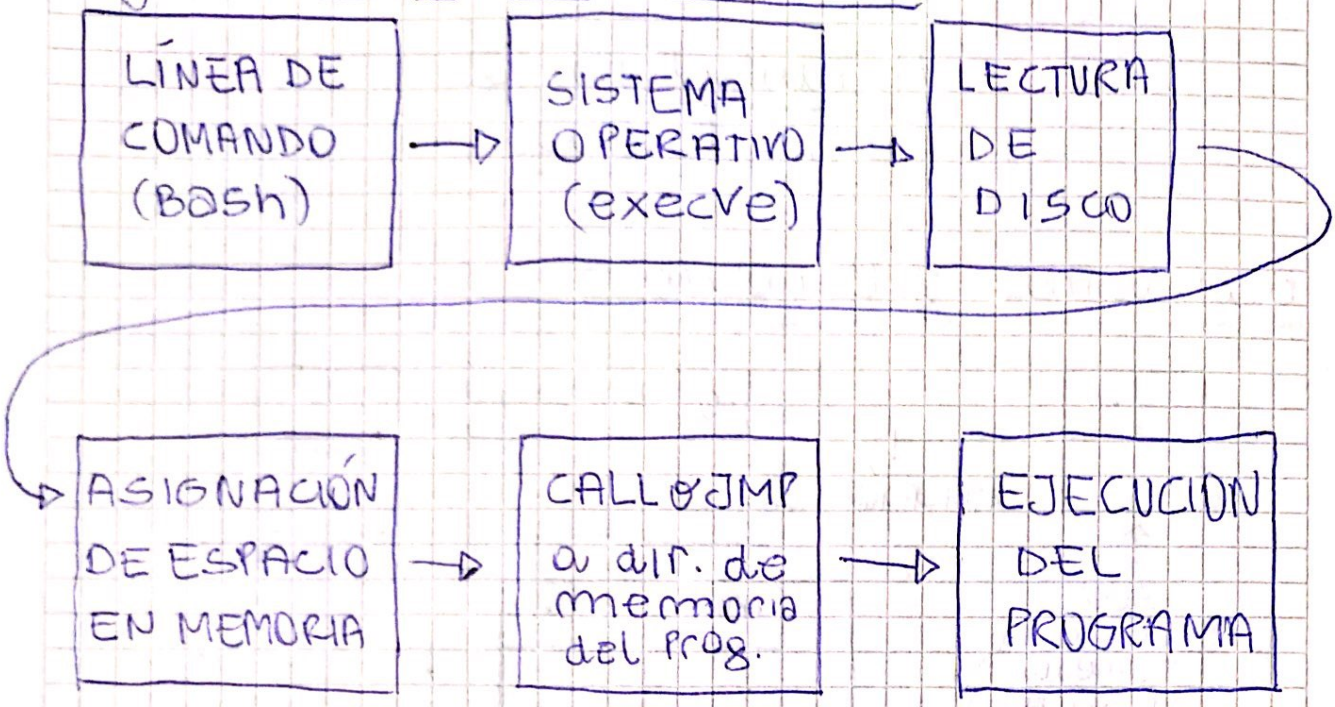
compilador de ASM (gas) .O → genera el codigo objeto.

⇩

Linkeditor (ld) .out → El ejecutable puede tener varios formatos ELF, A.OUT y también puede generar modelo FLAT o Binario.

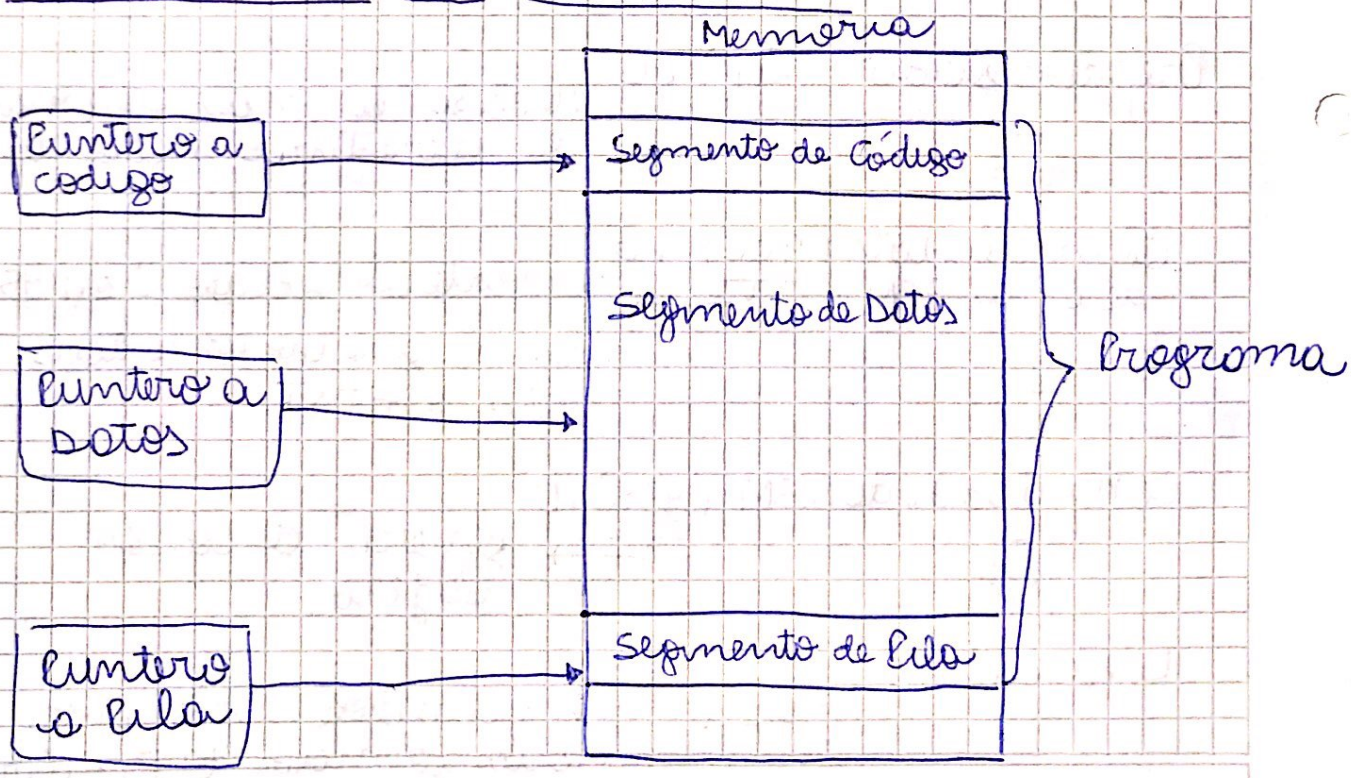
Scanned by CamScanner

Ejecución de un Programa =



Para ejecutar un archivo ejecutable el mismo debe ser llevado a la memoria RAM

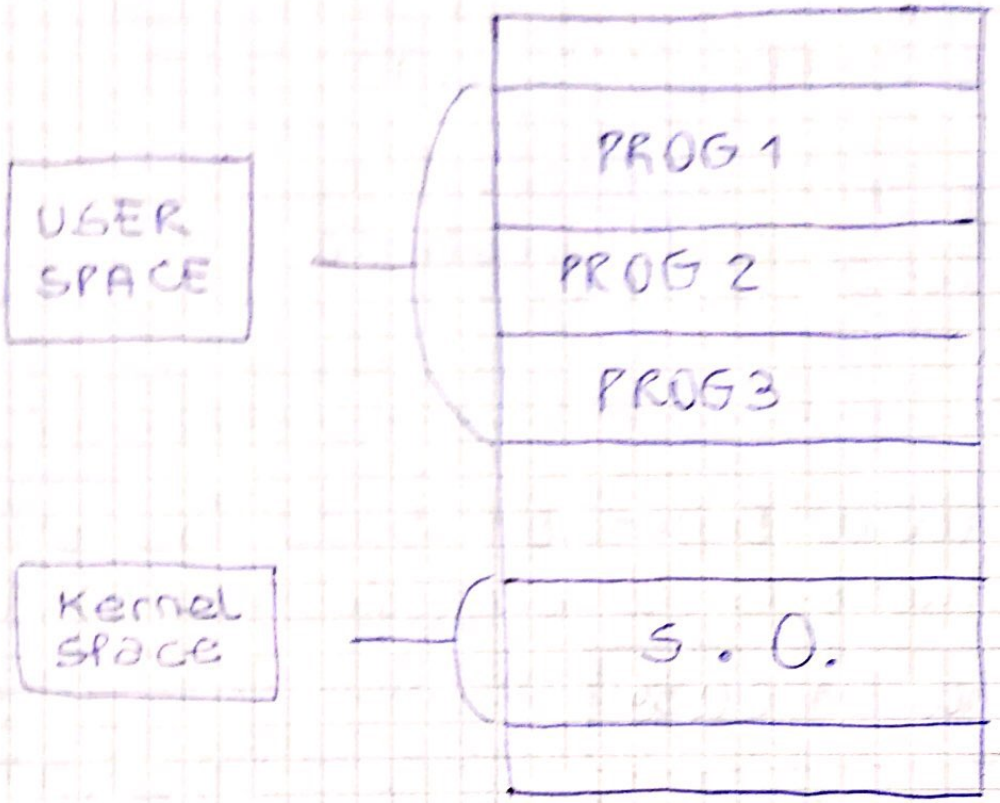
Programa en Memoria =



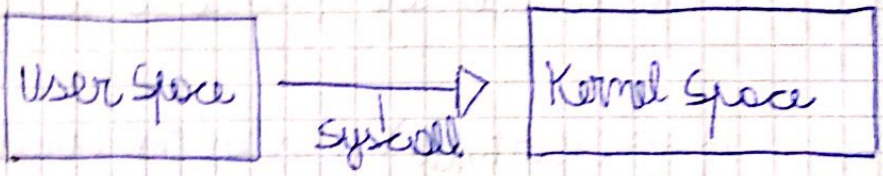
NOTA

Scanned by CamScanner

Introducción a S.O. =



System calls =

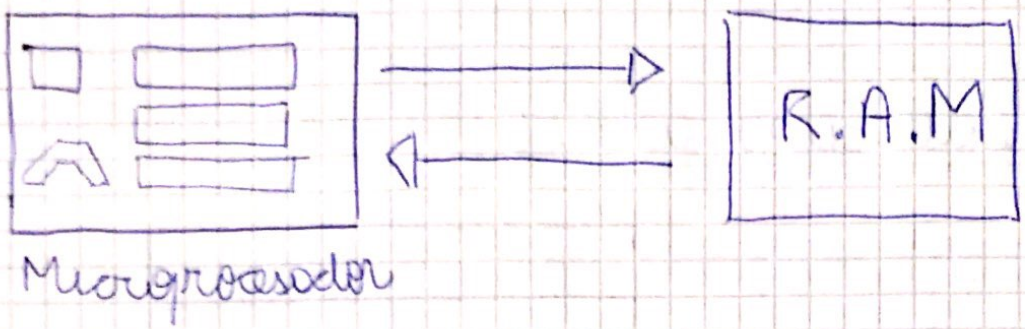


Formas de ejecutar system call =

- INT 80h Interrupción nro 80 (consume más tiempo)
- Instrucciones SYSCALL/SYSRET (INTEL)
  SYSENTER/SYSEXIT (AMD)
- VSYSCALL y VDSO: Syscall virtuales y Virtual dinamic shared Object

LINUX crea páginas de memoria en U.E. para acelerar tiempos

Scanned by CamScanner



Decodificar Instrucción → Ejecutarla → Ir a Buscar la próxima

Existe una Biyección entre Assembler e instrucciones de código de maquina

Programa en Disco

- → Datos inicializados (zona de datos) { .rodata (read only), .data }
- → Instrucciones (Zona de Código) .text
- → BSS (Virtual) (Datos no inicializados o en 0) .bss
  - ↓ "Better save space"

NOTA

Programa en Memoria

- → Código (apuntado por IP) (*)
- → Datos (también se incluye .bss y deja de ser virtual (expansión de bss)).
- → Stack

(*) <u>Punto de Entrada</u> = es necesario conocer el punto de entrada al programa para saber cual es la primera instrucción, esto se hace con la etiqueta "_start" que da el pie a la primera instrucción que se va a ejecutar.

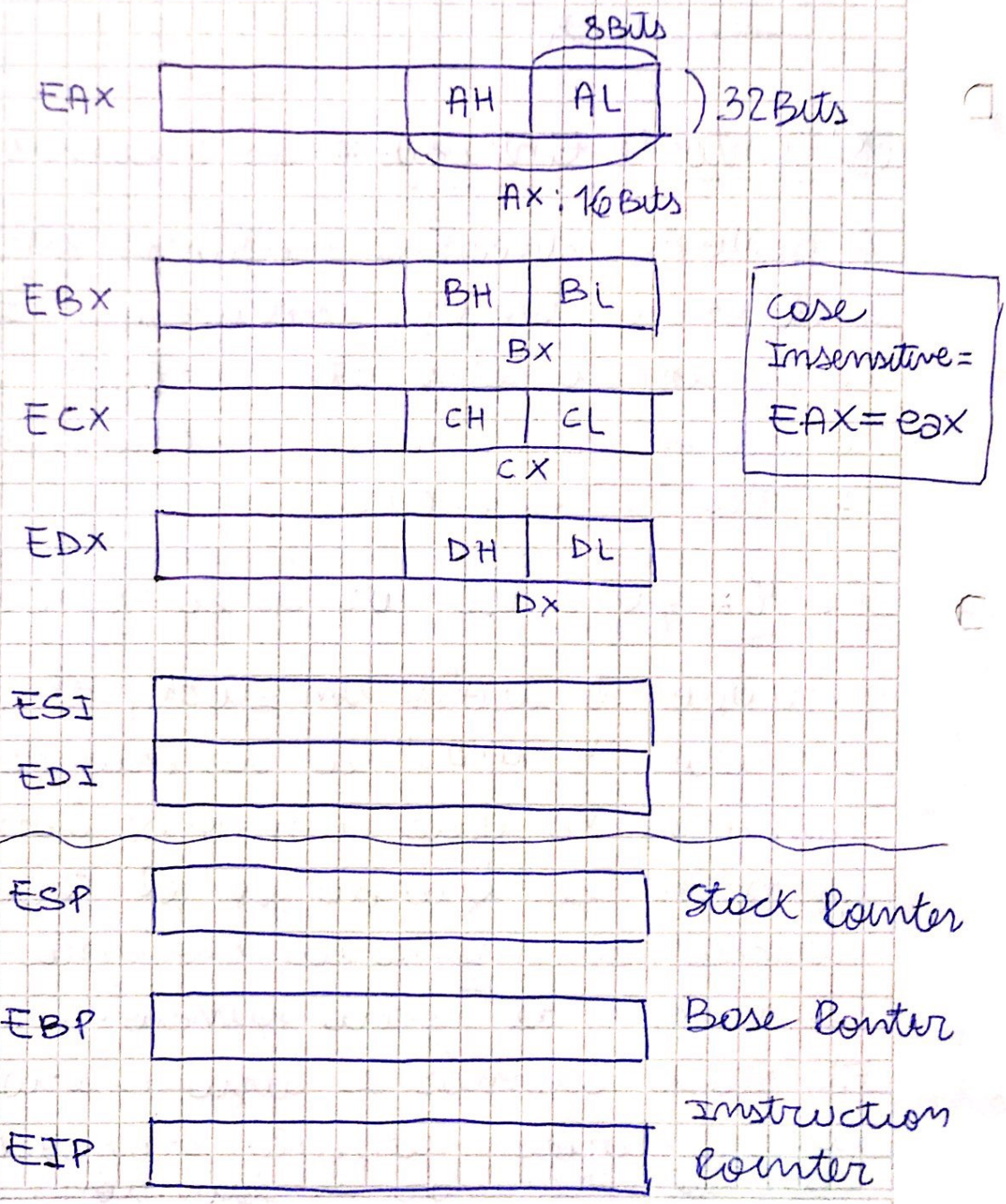
<u>Stack</u> = para llamar a las funciones, es necesario el uso de un stack, de esta forma se puede "conocer" la historia de dichas funciones. Ya que cuando se llama a una función, se guarda en el stack la dirección de memoria próxima a ejecutar al volver de dicha función.
El stack además es una buena zona para declarar variables privadas (automáticas) pues al terminar la función, el stack

Scanned by CamScanner

debería volver al estado en el que fue entregado para no tener ningún problema. Es decir, al momento de ejecutar la instrucción ret, el stack debe estar como se recibió

Lenguaje Assembler (en Intel) =

Registros de Proposito general = (32 Bits)



NOTA

Scanned by CamScanner

Declaración de Variables estáticas =

Em .data

db : Un Byte
dw : 2 Bytes (word)
dd : 4 Bytes (Double)

Se pueden nombrar antecediendo un nombre.

Ej =
```
.data
var db 64 ; Se guarda 64 en un Byte
            en la dir referenciada por "var".

var db ? ; Byte sin inicializar
var dw ? ; 2 Bytes sin inicializar
var dw 20000; 2 Bytes con 20000 en "var"
```

Arreglos → Elementos contiguos en memoria

Directiva DUP = Duplica la expresión
Es una MACRO.

4DUP(2) ≡ 2, 2, 2, 2

Ej =
```
z dd 1, 2, 3 ; Tres valores de 4 Bytes
               contiguos 1, 2, 3
                         ↓  ↓  ↓
                         z  z+4 z+8

bytes db 10DUP(?) ; 10 Bytes sin inicializar
arr   db 100 DUP (0) ; 100 Bytes en 0
str   db 'hola', 0 ; 5 Bytes 'h', 'o', 'l', 'a', 0
```

NOTA

Scanned by CamScanner

Referenciado de Memoria =

- Puede referenciar 2^{32} Bytes con 32 Bits
- Se pueden sumar dos registros (no más!)
  ↳ uno de ellos se puede premultiplicar por 2, 4, 8
- O sumar o restar una constante de 32 Bits

Ej =

```
                    DEST ← ORIG
mov eax, [ebx]
mov [var], ebx
      ↓
   de 32 Bits

mov eax, [esi-4]
mov edx, [esi+4*ebx]
```

NO PUEDO RESTAR REGISTROS
NO PUEDO MOVER DE MEM A MEM

Directivas de Tamaño =

En general assembler infiere el tamaño de la dirección de memoria que se referencia a partir de los operandos.
Si levanto algo de memoria a un registro de 32 Bits, se infiere que se levantan 4 Bytes

Debe especificarse para evitar ambiguedades =

BYTE PTR	→	en 1 Byte
WORD PTR	→	2 Bytes
DWORD PTR	→	4 Bytes

NOTA

Scanned by CamScanner

Instrucciones =

<reg>: registros
<mem>: memoria
<const>: constante

1) mov dest, orig

<reg>, <reg>
<reg>, <mem>
<mem>, <reg>
<reg>, <const>
<mem>, <const>

2) PUSH <reg32> ; 4Bytes!!!
<mem> ; Toma 4 Bytes
<const32>

Pushea al stack donde debe ser según ESP
- decrementa ESP en 4 ("El stack crece hacia abajo") (de FFFFFFFF a 00000000)
- luego copia la información a [ESP]
- SIEMPRE 4 BYTES!!!

3) POP <reg32> ; mete [ESP] en <reg32>
<mem> ; pop a memoria.

Saca lo que está en [ESP] e incrementa ESP en 4.

NOTA

Scanned by CamScanner

4) lea <reg32>, <mem>

Load Effective Address = carga el puntero a la dirección de memoria en el registro indicado.

Ej:
```
struct Point {
    int xcord;
    int Ycord;
};
```

```
int y = Points[i].ycord;
```

donde Points[] es un arreglo de Point
Asumiendo que la dirección de Points[] está en EBX y la variable i en EAX,

xcord y Ycord ambos campos son de 32 Bits entonces ycord es un Offset de 4 Bytes dentro del struct. =>

$$\text{MOV EDX, } \underbrace{[EBX + 8 * EAX + 4]}_{\text{"Effective Address"}}$$

de esta forma "y" termina en EDX.
Ahora si utilizamos el Operador & (Dir de memo)

```
int *P = & Points[i].ycoord;
```

=> LEA ESI, [EBX + 8 * EAX + 4]
carga la dirección en ESI

NOTA

Scanned by CamScanner

Operaciones Aritméticas =

1) add <reg>, <reg>
<mem>, <reg>
<reg>, <mem>
<reg>, <const>
<mem>, <const>

Se adiciona el segundo operando al primero y se guarda en el primero.

2) sub <reg>, <reg>
<reg>, <mem>
<mem>, <reg>
<reg>, <const>
<mem>, <const>

se resta el segundo operando en el primero

3) inc <reg> Incremento en 1
<mem>

4) dec <reg> Decremento en 1
<mem>

Scanned by CamScanner

5) IMUL <reg32>, <reg32>

<reg32>, <mem>

<reg32>, <reg32>, <const>

<reg32>, <mem>, <const>

Multiplicación entera (32 Bits) =

- con 2 operandos: Multiplico ambos y guardo en el 1er.
- con 3 operandos: Multiplico OP_2 y OP_3 y guardo en OP1

OP1: debe ser un registro

OP3: (si lo hay) debe ser una constante.

6) IDIV <reg32>

<mem>; se toman 4 Bytes

División entera (32 Bits) =

Divide el registro de 64 Bits EDX:EAX (Parte alta: EDX Bajo: EAX) por el operando especificado.

El resultado se guarda en EAX y el resto en EDX.

NOTA

Scanned by CamScanner

Operaciones lógicas "Bit a Bit" =

1) and <reg>, <reg>
<reg>, <mem>
<mem>, <reg>
<reg>, <const>
<mem>, <const>

2) or <reg>, <reg>
<reg>, <mem>
<mem>, <reg>
<reg>, <const>
<mem>, <const>

3) xor <reg>, <reg>
<reg>, <mem>
<mem>, <reg>
<reg>, <const>
<mem>, <const>

4) not <reg> cambia los Bits
<mem> Ej: not BYTE PTR [var]

5) neg <reg>
<mem>

NOTA
Negación o complemento a la Base 2 (Invertir Bits y sumar 1)

Scanned by CamScanner

6) SHL <reg>, <const8>

<mem>, <const8>

<reg>, CL

<mem>, CL

Shifteo hacia la izquierda rellenando con ceros (Hasta 32 lugares)

7) SHR <reg>, <const8>

<mem>, <const8>

<reg>, CL

<mem>, CL

Shifteo hacia la derecha rellenando con ceros.

Saltos y Control de Flujo =

El procesador tiene un Instruction Pointer que apunta a la instruccion a ejecutar. En general se incrementa en 1 cuando termina una instrucción.

No se puede manipular directamente pero si implicitamente para control de flujo.

LABELS O ETIQUETAS ⟶ label:

JMP <label> salto incondicional

NOTA

Scanned by CamScanner

Saltos condicionales =

USANDO LOS FLAGS DE L REGISTRO EFLAGS

MACHINE STATUS WORD → info de la ultima operación Aritmetica / operación cmp

1) je <label> ; salta cuando son iguales.

2) jne <label> ; " " no son iguales

3) JZ <label> ; " " fue cero.

4) Jg <label> ; " " fue mayor

5) Jge <label> ; " " " " igual

6) JL <label> ; " " fue menor

7) JLe <label> ; " " " " igual.

operación de comparación =

cmp <reg>, <reg>
<reg>, <mem>
<mem>, <reg>
<reg>, <const>

comparo mediante una resta para setear los Flags. Equivalente a SUB pero no guardo el resultado.

Scanned by CamScanner

Control de subrrutinas =

1) call <label>

Pushea la ubicación a la proxima instrucción despues del call (la posición indicada por el IP). Luego salta a lugar indicado en la etiqueta. (cambia IP).

2) ret

Popea un puntero a codigo del stack y salta ahí (para volver).
En un principio, si no hubo problemas con el manejo del stack en la subrrutina dicha dirección de retorno debe ser la misma que pusheo el llamador.

Argumentos de línea de comando =

En un sistema tipo Unix, el sistema operativo antes de darle control al programa configura el stack que le va a otorgar con los argumentos del programa

ESP : Cantidad de Argumentos
ESP+4 : Path al Programa
ESP+8 : Dirección del 1er Argumento
ESP+(n+2)*4 : Dirección del n-argumento
: NULL (Ø) (4 Bytes)
VARIABLES DE ENTORNO
: NULL

NOTA

Scanned by CamScanner

$\boxed{\text{syscall}}$ → Función escrito en Kernel Space

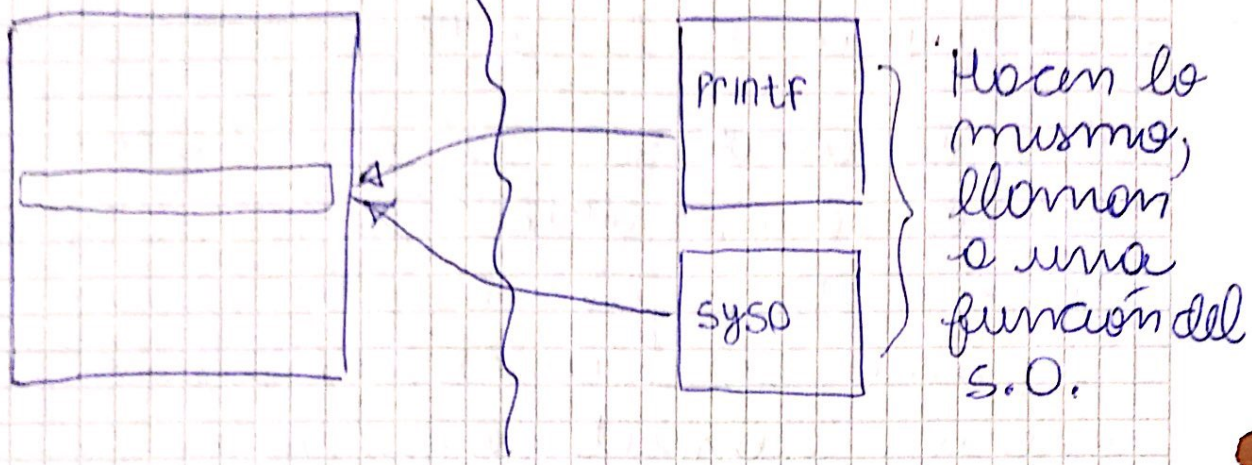
Interrupciones de software =

Los programas se corren en espacios distintos de memoria que en el que está el S.O., que también es un programa. Si consideramos que el OS posee un gran set de funciones que pueden ser llamadas por los programas, resultaría muy poco práctico acceder a dichas funciones mediante la instrucción call de ensembler. Para evitar el problema de conocer para cada versión del S.O., la posición exacta de cada función se utilizan las interrupciones de software que el S.O administra haciendo uso de la interrupt descriptor Table (IDT). Al iniciar la computadora la BIOS (Basic Input / Output System) tiene control del IDT y luego al arrancar el S.O. toma control del mismo para "indexar" las funciones utilitarias.
En el caso de Linux, se hace uso de la entrada 80h para acceder al S.O. y colocando en EAX el ID de la syscall a utilizar.

Scanned by CamScanner

System calls =

S.O. Kernel Space | Programas - User Space

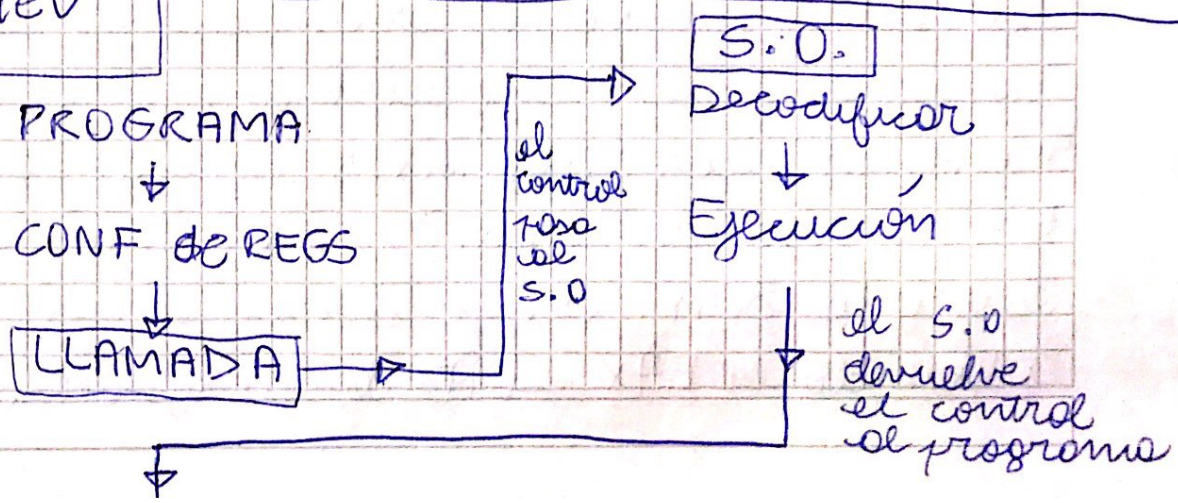


(*) no conocemos la dirección de memoria de las funciones del S.O (poco práctico)
=> USAMOS INTERRUPCIONES DE SOFTWARE

(*) la syscall es nuestra comunicación con el sistema operativo

(*) Queremos que nuestro programa sirva para varios S.O.

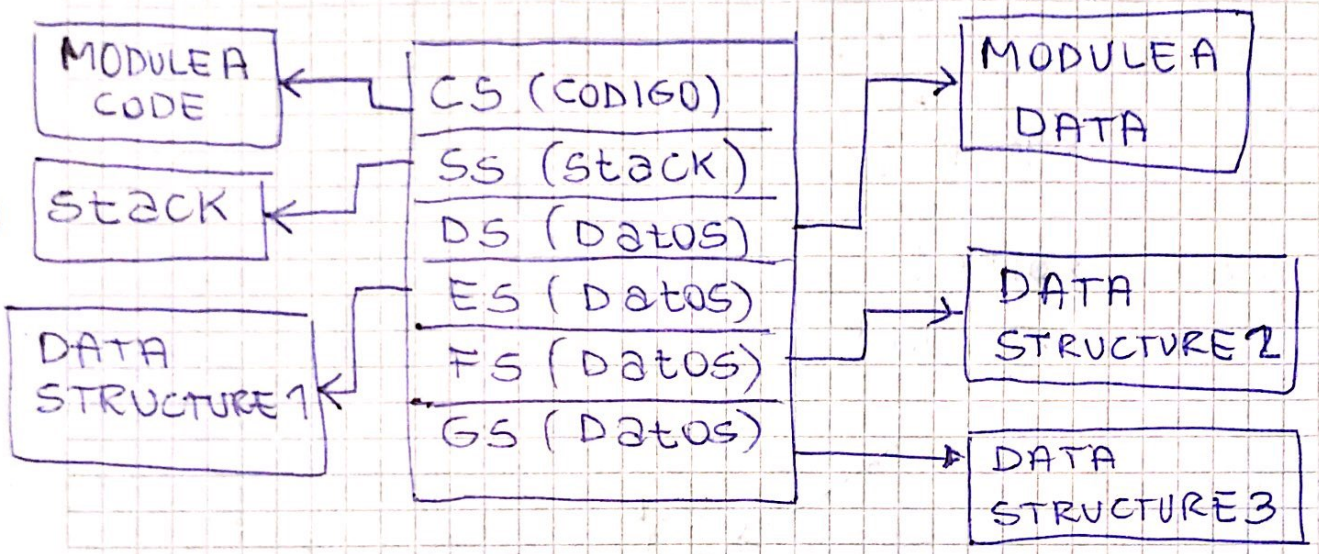
(#) En los sistemas tipo unix se tratarán los periféricos como un sistema de archivos (fopen, fclose, cd, etc)
=> /dev



NOTA

Segmentación de Memoria en 80386

(*) Registros de segmentación para apuntar a las distintas zonas o segmentos CODIGO/DATOS/PILA



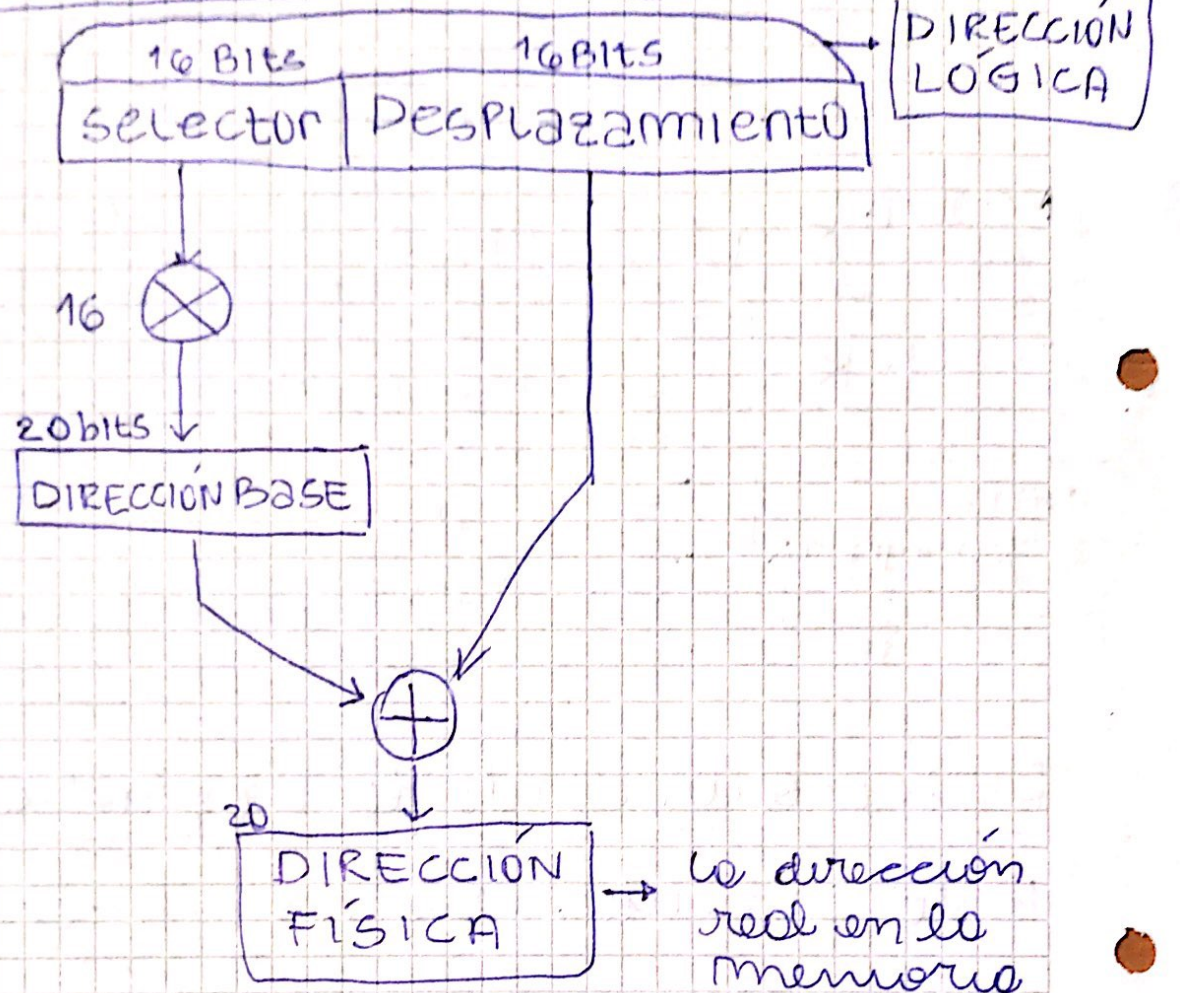
Direccionamiento de Memoria =

Historicamente desde los microprocesadores 8086 y 8088 se utilizo direccionamiento en MODO REAL (se accede directamente a la memoria) pero no tenia las características necesarias para los sistemas operativos multitarea y multiusuario. En el 80386 se consolido el MODO PROTEGIDO (se pasa por una tabla para ver si es posible el acceso.) Pero se

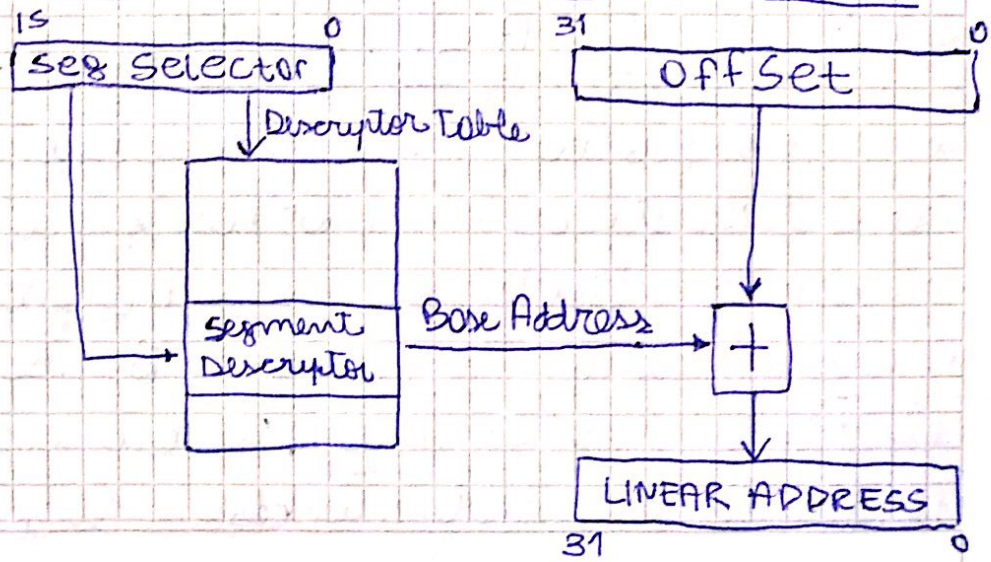
Scanned by CamScanner

mantuvo como modo de inicio del microprocesador, el modo real para mantener la compatibilidad

Direccionamiento en Modo Real =



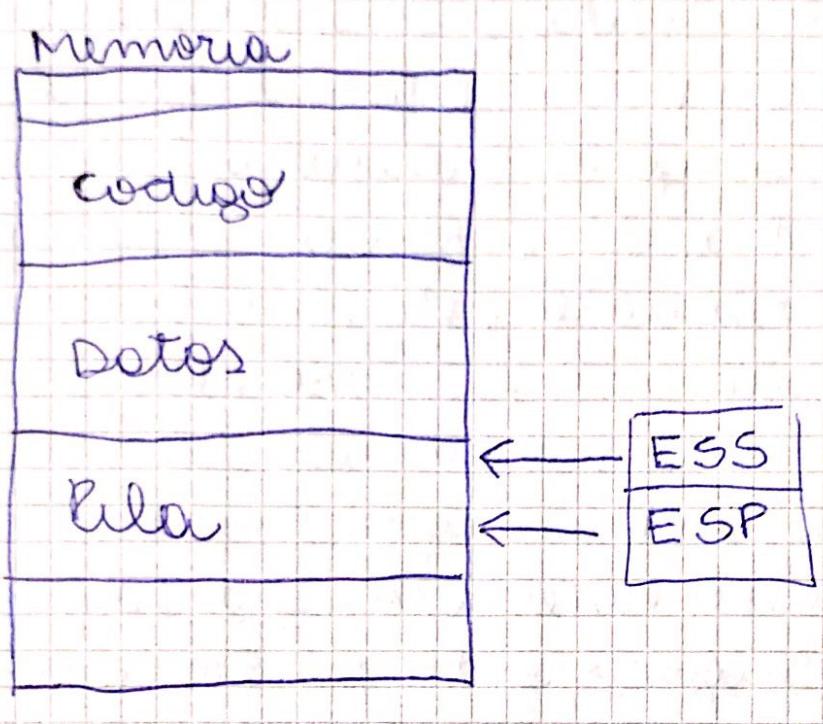
Direccionamiento en modo protegido =



NOTA

Scanned by CamScanner

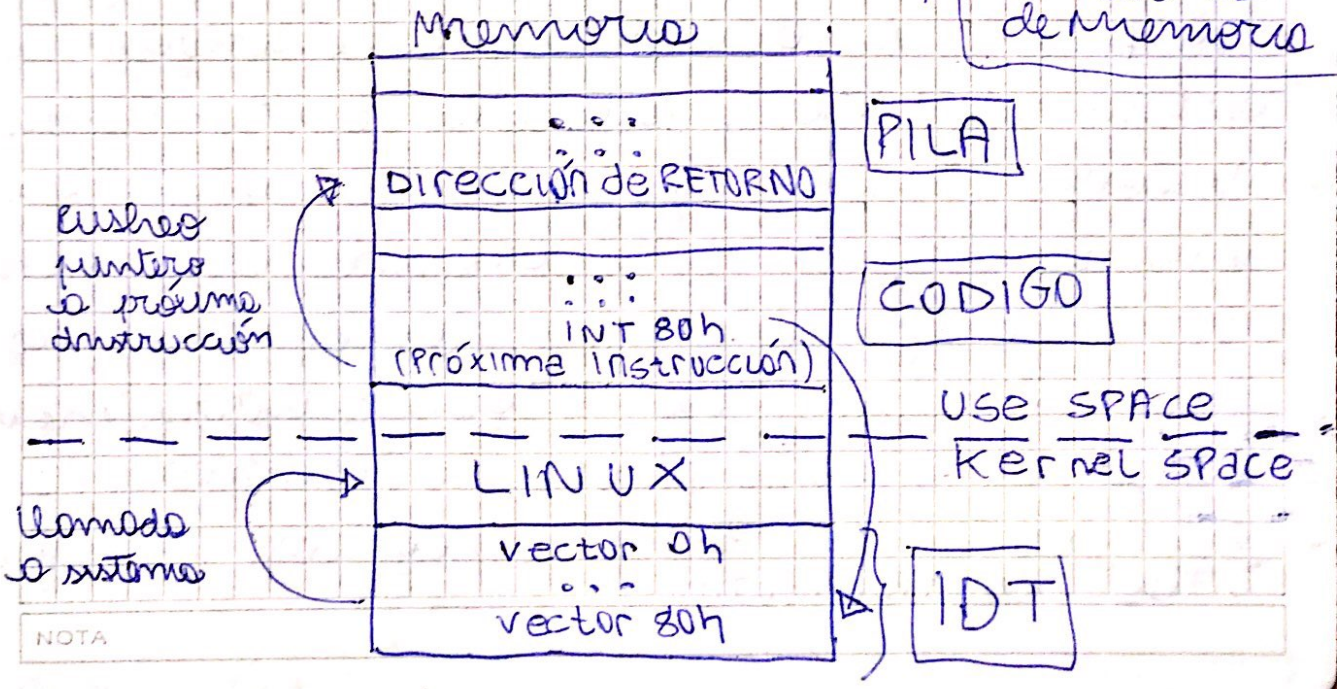
Repaso de Pila =



ESS: Apunta al segmento
ESP: se mueve en la Pila.

Pila →

El Stack "crece" decrementando posiciones de memoria



Scanned by CamScanner

Assembler y C =

Todo programa que corra en una computadora, tiene que estar en un lenguaje que ésta entienda, en codigo de máquina. El codigo de maquina hace uso de los registros del procesador, del stack, de las interrupciones, etc. Así que un programa escrito en C, en Assembler, COBOL, Pascal, Ada, etc. a fin de cuentas termina en codigo maquina, lo que implica es que siempre se puede hacer una Interface para conectar los lenguajes entre sí. En particular, cómo entre codigo de máquina y Assembler hay una relación Biyectiva, es fácil conectar este lenguaje con cualquier otro, ya que con Assembler hay que respetar el funcionamiento interno del otro lenguaje para poder interactuar. Es interesante poder hacer una interface con Assembler ya que éste tiene control directo sobre las operaciones del procesador. No como los lenguajes de propósito general que se abstraen de la arquitectura. Con este lenguaje se pueden deshabilitar las interrupciones, habilitar las interrupciones, habilitar modo protegido, cambiar la dirección del directorio de paginas, modificar stack pointer, etc.

NOTA

Scanned by CamScanner

El lenguaje C es por excelencia el lenguaje de elección para desarrollo básico de un sistema operativo por su bajo nivel, versatilidad, manejo de memoria y su independencia de servicios que necesitan otros lenguajes. Otros lenguajes de programación de más alto nivel, por ejemplo C++, necesitan configuraciones especiales para funcionar.

La convención de como realizar las llamadas entre distintos lenguajes, cómo se reserva espacio para variables, cómo se alinean los datos, etc. es parte de lo:

APPLICATION BINARY INTERFACE (ABI)

Llamadas de Assembler a C =

Si recordamos que a final de cuentas, una función en C es una dirección de memoria, no debería ser difícil hacer una llamada desde ASM a C.

Convenciones en C en x86 =

(*) Parámetros de una función =

```
FUNCION_EN_C(Param_a, Param_b, Param_c)
```

Los parametros se pushean en el stack de DERECHA a IZQUIERDA

Scanned by CamScanner

En este caso primero param_c, luego param_b y finalmente param_a.

(*) Llamado a la función =

Se ejecuta la instrucción CALL que guarda en el stack el EIP para el retorno y efectuo JMP al primer Byte de la función

(*) Armado del Stack Frame =

Resguardo y actualización de EBP, una vez en la función se resguarda el valor anterior de EBP. Y luego se le asigna el valor actual del registro ESP.

```
PUSH ebp
mov ebp, esp
```

De esta manera se puede utilizar el registro EBP para acceder a los parametros que quedaron en la pila.

(*) Valores a retornar =

- Si el valor es menor a 32 Bits, se retorna en EAX
- Si es mayor, se retorna la parte alta en EDX y la parte Baja en EAX.
- Si es un dato más complejo (ej: Estructura de datos) retorna un puntero formado por EDX:EAX.

NOTA

Scanned by CamScanner

La función llamador debe =

1. Guardar, si los necesita los registros EAX, ECX y EDX que son los que pueden modificar las subrutinas (se pushean al stack)

2. Pasar los parametros pusheando de Derecha a izquierda De forma que se puedan popear en orden en la función llamada.

3. Llamar a la subrutina con call lo cual pushea la dirección de retorno al stack

Después del retorno =

I. Sacar los parametros del stack (sumando al stack pointer "crece para abajo")

II. Restauro los registros EAX, ECX y EDX vía POP.

Scanned by CamScanner

La función llamada, o la subrutina en sí debe =

1. Armar el Stack Frame: Pushear el EBP al stack (el Base Pointer de la función llamador debe preservarse para ser recuperado previo al retorno).
Copiar en EBP el ESP. mi Base Pointer es la posición actual del stack, es el punto de referencia para encontrar los parametros en el stack.
2. Alocar espacio para variables locales o automaticas:
Para ello hay que decrementar el stack pointer
3. Guardar los registros que deben ser preservados: EBX, EDI, ESI, EBP y ESP

Luego del cuerpo de la subrutina =

1. dejar el valor de retorno según corresponda.
2. Recuperar los registros resguardados ESI, EDI, EBP, ESP, EBX.
3. Destrucción del Stack Frame:
Desalocar variables locales mov ESP, EBP
recuperar el EBP del llamador POP EBP
4. Ejecutar ret (saca del stack el valor de retorno)

Scanned by CamScanner

SPP (Stack Smashing Protector) =

- lo implementa gcc
- utiliza la función _stack_chk_fail
- Ubica un valor entre EBP y las variables locales que se lo denomina CANARY
- Antes de retornar verifica que el CANARY no haya sido modificado
- Si lo fue termina la ejecución.

Alineamiento a palabra =

El procesador cada vez que accede a memoria, lo hace leyendo y escribiendo siempre de a 4 Bytes (el tamaño de la palabra en una arquitectura x86)
Cuando accede a un dato desalineado debe hacer 2 lecturas (dos accesos).
Para la escritura requiere más operaciones, ya que debe leer memoria para salvar el dato que no va a sobreescribir, luego combinarlo con el dato que si va a escribir para la primera porción y luego recuperar el dato de la segunda porción que va a escribir en la segunda porción.

Scanned by CamScanner

Una forma muy facil de arreglar este problema, es poner en cero los últimos 4 Bits como el stack crece hacia menores valores de memoria de FFFFFFFFh a 00000000h Entonces seguro esa posición va a estar por detrás del puntero actual De esta forma se puede hacer un acceso a memoria más eficiente.

```
and esp -16 ; alineación a palabra
```

-16 → 0XFFFFFFF0

Llamados de C a Assembler =

Sabra que existe una razón puntual, como que no existe un compilador, evitaremos programar en Assembler Nuevamente las funciones a fin de cuentas son direcciones, así que es posible llamar funciones desde C hacia assembler. Para que C sepa de que forma, con que argumentos y con que tipo de retorno se va a llamar a una función es necesario especificar su prototipo.

Otros directivas de ASM =

ALIGN <number> : alinea la proxima variable o instrucción a un Byte multiplo de "number".

EQU : definir una constante $-const
↳ Posición relativa

Variables =

Hasta ahora, siempre que hizo falta reservamos espacio en la zona de datos (section .data o .rodata con resb, resw o resd <number>) por ejemplo, para guardar una variable, pero si la función es recursivamente llamada se reservaría dicho espacio multiples veces, por ello C para que las variables automáticas vivan sólo en el scope de función, las guarda en el stack así, cuando se destruye el stack frame se libera la memoria utilizada.

Armado de stack Frame → (MACRO) ENTER
Destrucción de stack Frame → (INSTRUCIÓN) LEAVE

De esta forma las variables no tienen visibilidad para una función externa.

ESI y EDI → trabajar con arreglos
↳ source ↳ Destination

En GS: C busca los canarys del SSP luego invoca _stack_chck_fail.

Scanned by CamScanner

Instrucción: rep "Repeat the next instruction CX times".

stosd: Guarda en [EDI] lo que hay en EAX incrementando o decrementando EDI según el Direction Flag

stos
stosb
stosw
stosd

DF = 0 → EDI incrementa
DF = 1 → EDI decrementa

DF: Direction Flag → En el status

Instrucción CPUID =

Permite conocer detalles del Procesador

EAX = 0 → Vendor ID (string)	EAX = 1 → Signature

Instrucción MOVZX: extiende un Byte o word a dword llenando la parte alta con ceros.

Instrucción TEST: idem AND pero no guarda el resultado

X86_64 ABI

Hasta ahora trabajamos con la ABI de 32 Bits de C y Assembler
Basicamente los conceptos son los mismos, lo unico que cambia es el pasaje de parámetros.
El pasaje de parámetros se realiza de la siguiente forma:

* Se cargan los argumentos en los registros.
* Se llama a la función
* Los argumentos se copian al stack y se referencian desde ahí.

<u>clasificación</u> = Los argumentos se clasifican de la siguiente forma:

* INTEGER: char, short, int, long, longlong, y PUNTEROS.
* SSE: floats y doubles.
* MEMORY: datos mayores a un quadword (8 Bytes) y datos desalineados

Scanned by CamScanner

Pasaje de Parámetros =

(*) Si el dato es INTEGER:

se van ocupando los registros RDI, RSI, RCX, R8 y R9 en orden.

(*) Si el dato es SSE =

se utilizan los registros XMM0 a XMM7 en orden ascendente de ocupación.

(*) Si el dato es MEMORY =

se pasan y devuelven por STACK Igual que en 32 Bits.

(De derecha a izquierda)

Retorno de valores =

(*) Si el dato es INTEGER =

se utiliza RAX y RDX

(*) Si el Dato es SSE =

se retorna por XMM0 y XMM1.

Registros a preservar =

- RBP
- RSP
- R 12
- R 13
- R 15

NOTA = Puede haber funciones con prototipos distintos que pasan igual sus parámetros.

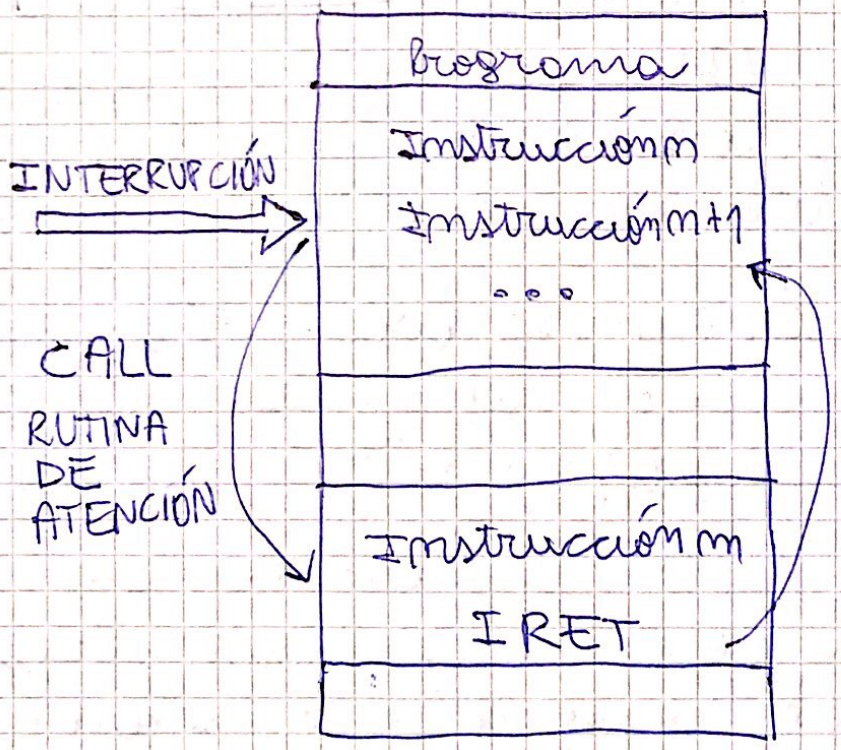
Ej: int foo(int a1, double a2); y int bar(double a1, int a2);

Scanned by CamScanner

Interrupciones =

Una señal externa interrumpe al micro para requerir un servicio de atención.
La aplicación de los mismos a nivel software para el control de flujo tiene un antónimo que es el llamado "Polling" en donde se chequea constantemente (preguntando + muchos if) si se requiere atención.

Rutina de atención de Interrupción =



La interrupción puede ser generada por un evento externo (INTR o NMI) o por la instrucción INT.

LAS INSTRUCCIONES SON **ATÓMICAS**

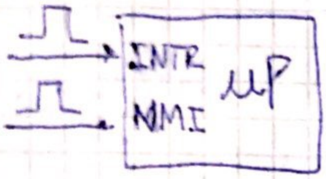
Atomicidad: porción de código que puede ejecutarse sin ser interrumpida.

Scanned by CamScanner

Tipos de Interrupciones =

(*) INTERRUPCIONES DE HARDWARE =

Se interrumpe la ejecución del programa activando alguna de las dos entradas que tiene el microprocesador INTR y NMI



INTR es ENMASCARABLE y NMI NO LO ES

(*) Interrupciones De Software =

Se interrumpe la ejecución del programa al ejecutar la instrucción de assembler INT.

INTERRUPCIONES DE HARDWARE =

El flag IF (interruption Flag) indica si se debe atender a las interrupciones externas si IF=1 (habilitado) si IF=0 (deshabilitado).

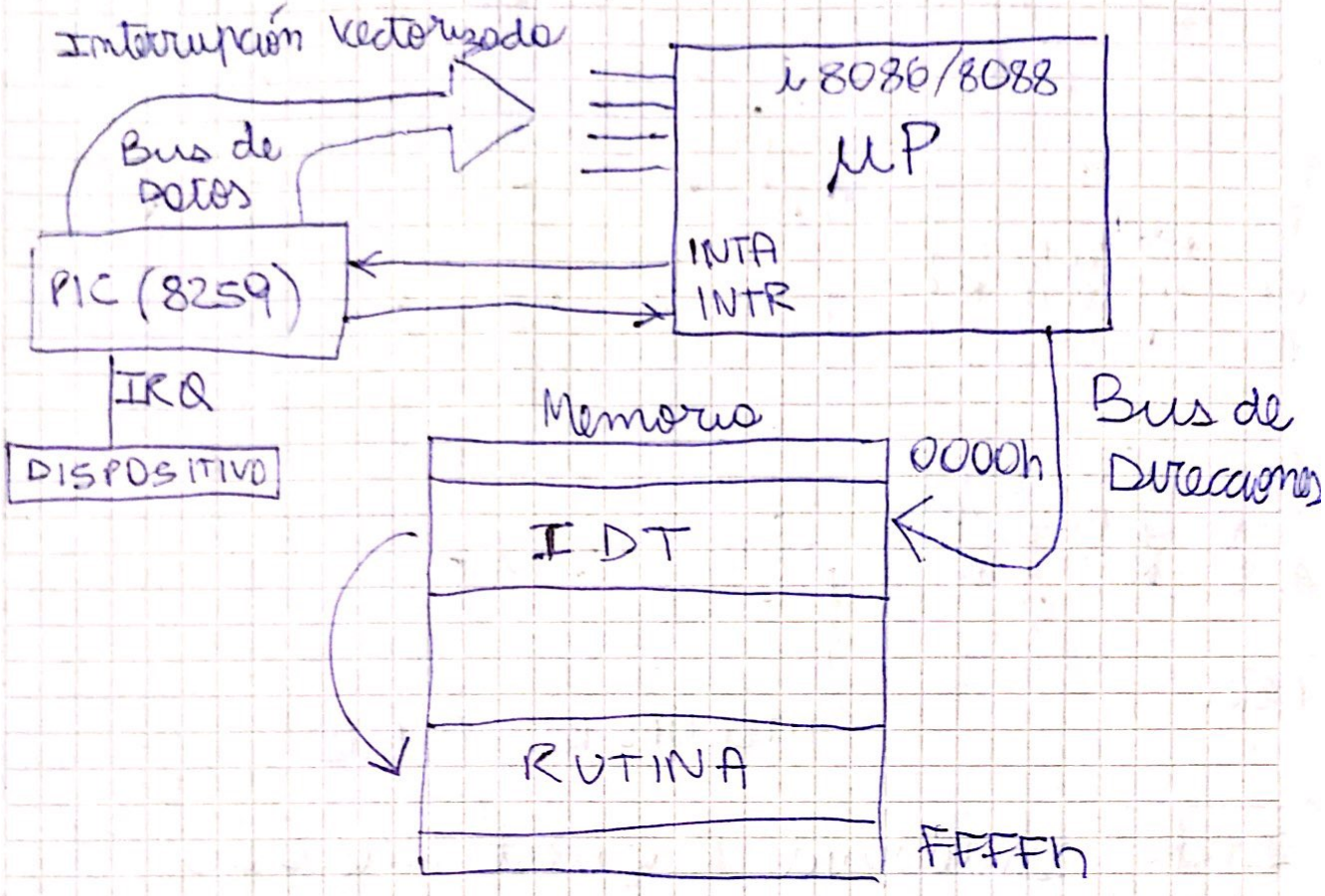
(*) Interrupción Enmascarable = El flag IF se controla con las instrucciones:
- STI (Set interrupts) y - CLI (clear interrupts).

(*) Interrupción NO enmascarable =
las interrupciones que ingresan por la entrada NMI no pueden ser enmascaradas siempre ejecutan la rutina que se encuentra en la posición 2h. del IDT (INT 2h).

NOTA

Scanned by CamScanner

PIC – Controlador Programable de Interrupciones



Es un gestor de interrupciones.
Interrumpe, manda por el Bus de datos la identidad del dispositivo el µP busca en la IDT un puntero a función (el driver) y lo ejecuta.

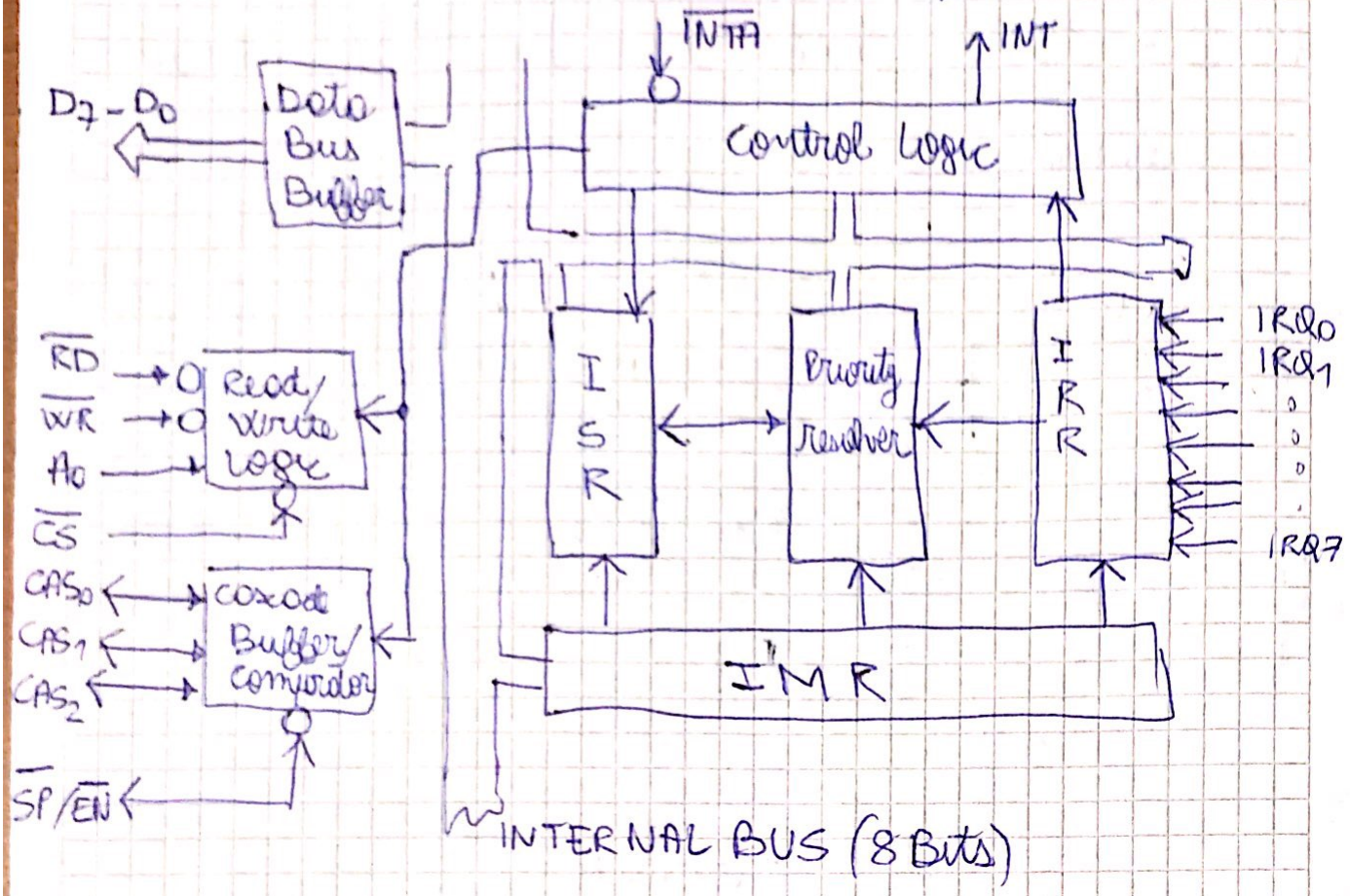
IRQ0	IRQ1	IRQ3	...	IRQ7

8 líneas de solicitud de interrupción de entrada.

NOTA

Scanned by CamScanner

PIC - Diagrama de Bloques (8259)



IRR Interrupt Request Register

ISR In-Service Register

IMR Interrupt Mask Register

→ Los puertos de entrada y salida en la PC son el 20h y el 21h.
El 20h se utiliza para programar el PIC (lo utiliza el BIOS al arrancar el sistema)
En el puerto 21h podemos acceder al registro IMR del 8259 donde podemos setear las interrupciones.

NOTA

Scanned by CamScanner

IMR

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	M7	M6	M5	M4	M3	M2	M1	M0

Interrupt Mask
1: set
0: reset

Acceso al PIC

IN / OUT

```
in al, 21h      ; leo Mascara del PIC
Mov al, 0FEh    ; por Ej solo habilito
                  la Interrupción del
Out 21h, al       Teclado.
```

EOI (End of Interrupt)

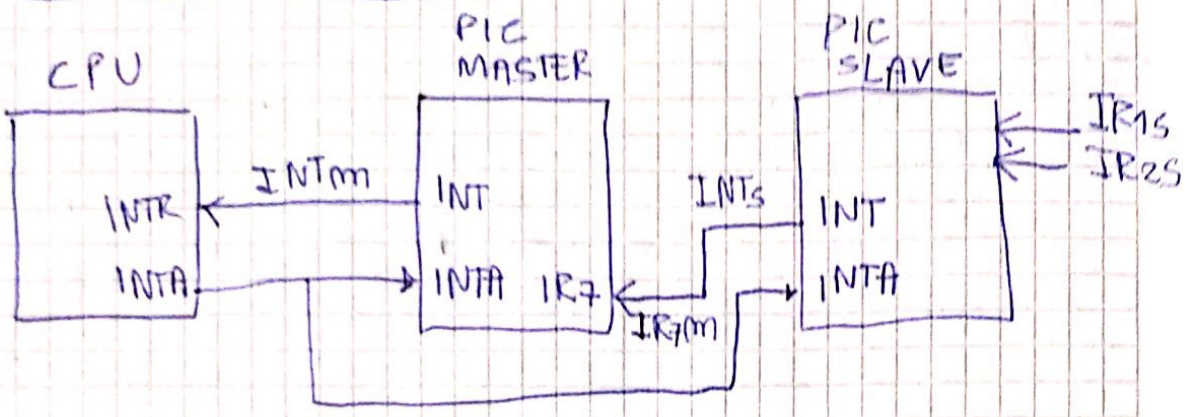
cada rutina de atención de interrupción luego de correr debe avisar al PIC que terminó su ejecución.

se puede especificar la IRQ que terminó o enviar un codigo que indica que finalizo la atención de la ultima interrupción que llego.

Esa palabra se envía al puerto 20h y el valor se envía para indicar que finalizo la atención de la int es el valor 20h.

Scanned by CamScanner

PIC en cascada =



Colocando 2 PICs en configuracion Master - slave o cascada, se amplia la cantidad de interrupciones de hardware que el µP puede controlar
En la PC se utilizo IRQ2 del Master para conectar el PIC slave.

Interrupciones de Software =

Dentro de un código puedo realizar una interrupcion con la instruccion INT

Servicios de Bios = al iniciar la PC

guarda en memoria rutinas básicas para poder empezar a operar.

INT 10h → Rutinas de Video
INT 13h → Rutinas de Disco
INT 14h → Rutinas de Puerto Serie
INT 19h → Rutinas para Bootloader
INT 1Ah → Rutinas para el RTC

RTC: Real Time Clock.

NOTA

Scanned by CamScanner

Interrupciones de Hardware por Default:

IRQ0	INT 08h	Timmer Tick (18,2 veces/seg)
IRQ1	INT 09h	Teclado
IRQ2	INT 0Ah	INT desde 8259 A esclavo
IRQ8	INT 70h	RTC
IRQ9	INT 71h	Redir por soft a IRQ2
IRQ10	INT 72h	Reservado
IRQ11	INT 73h	Reservado
IRQ12	INT 74h	Reservado
IRQ13	INT 75h	Coprocesador Numérico
IRQ14	INT 76h	controlador HDD
IRQ15	INT 77h	Reservado
IRQ3	INT 0Bh	COM2
IRQ4	INT 0Ch	COM 1
IRQ5	INT 0Dh	LPT2
IRQ6	INT 0Eh	FLOPPY
IRQ7	INT 0Fh	LPT1

Interrupciones en Modo Protegido:

En modo protegido cada entrada de la IDT es un descriptor de Interrupción, contiene además de las direcciones de ~~atender~~ atención de ints otros datos como los permisos. Los primeros 32 Descriptores son las Excepciones.

Scanned by CamScanner

EXCEPCIONES

Una Excepción es un evento generado por el procesador cuando detecta una o más condiciones predefinidas al ejecutar una instrucción.

Existen 3 tipos de excepciones =

(*) Fault = Excepción que puede corregirse
El procesador guarda en la pila la dirección de la instrucción que produjo la falla

(*) Trap = Se utilizan para realizar accesos al S.O. (SYSCALL)

(*) ABort = No siempre se puede obtener la instrucción que causo la excepción.
Reporta errores severos

De la tabla de interrupciones (IDT) los primeros 32, son excepciones.
Actualmente se están utilizando 20 de ellos y el resto queda disponible para uso futuro

NOTA

Scanned by CamScanner

Excepciones =

0 Divide Error

1 Debug Exception

2 Nonmaskable interrupt

3 Breakpoint (one-Byte INT3 instruction)

4 Overflow (INTO instruction).

5 Bounds Check (Bound instruction)

6 Invalid Opcode

7 Coprocessor not available.

8 Double Fault

9 (reserved)

10 Invalid TSS

11 Segment not present

12 Stack Exception

13 General Protection

14 Page Fault

15 (reserved).

16 Coprocessor Error

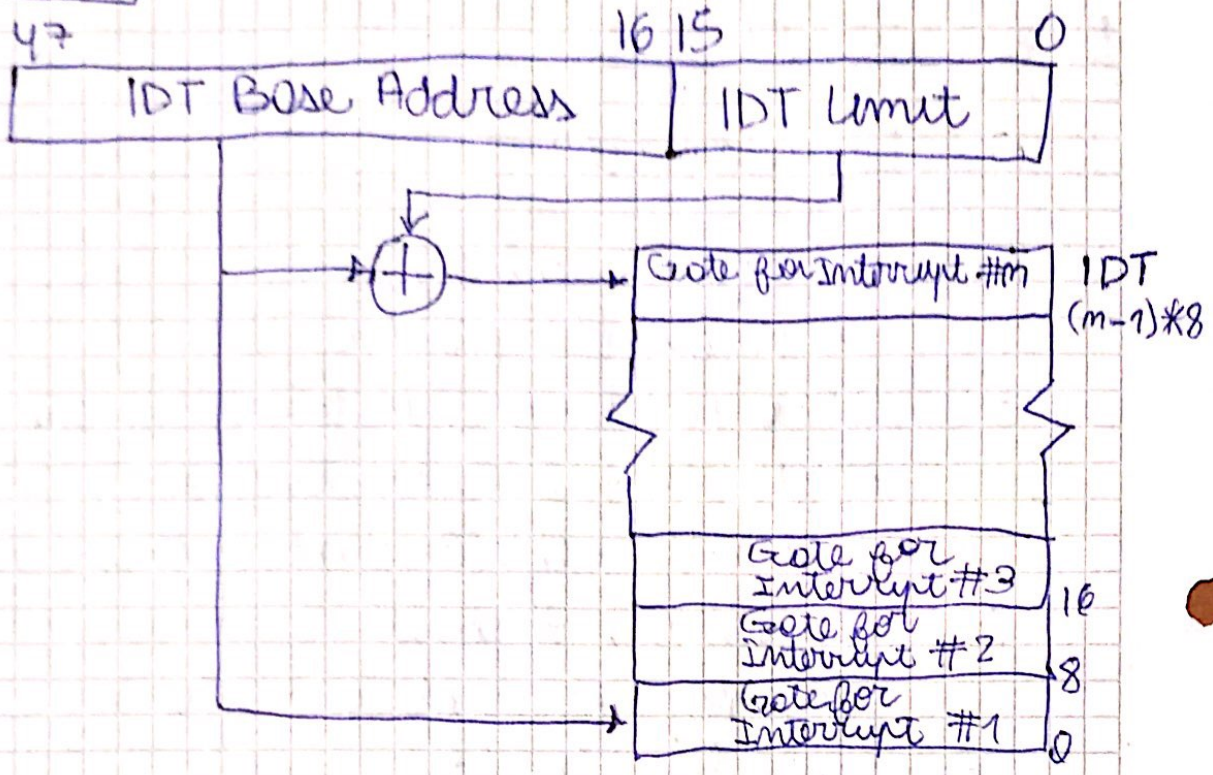
17-31 (reserved)

32-255 Available for external interrupts via INTR pin

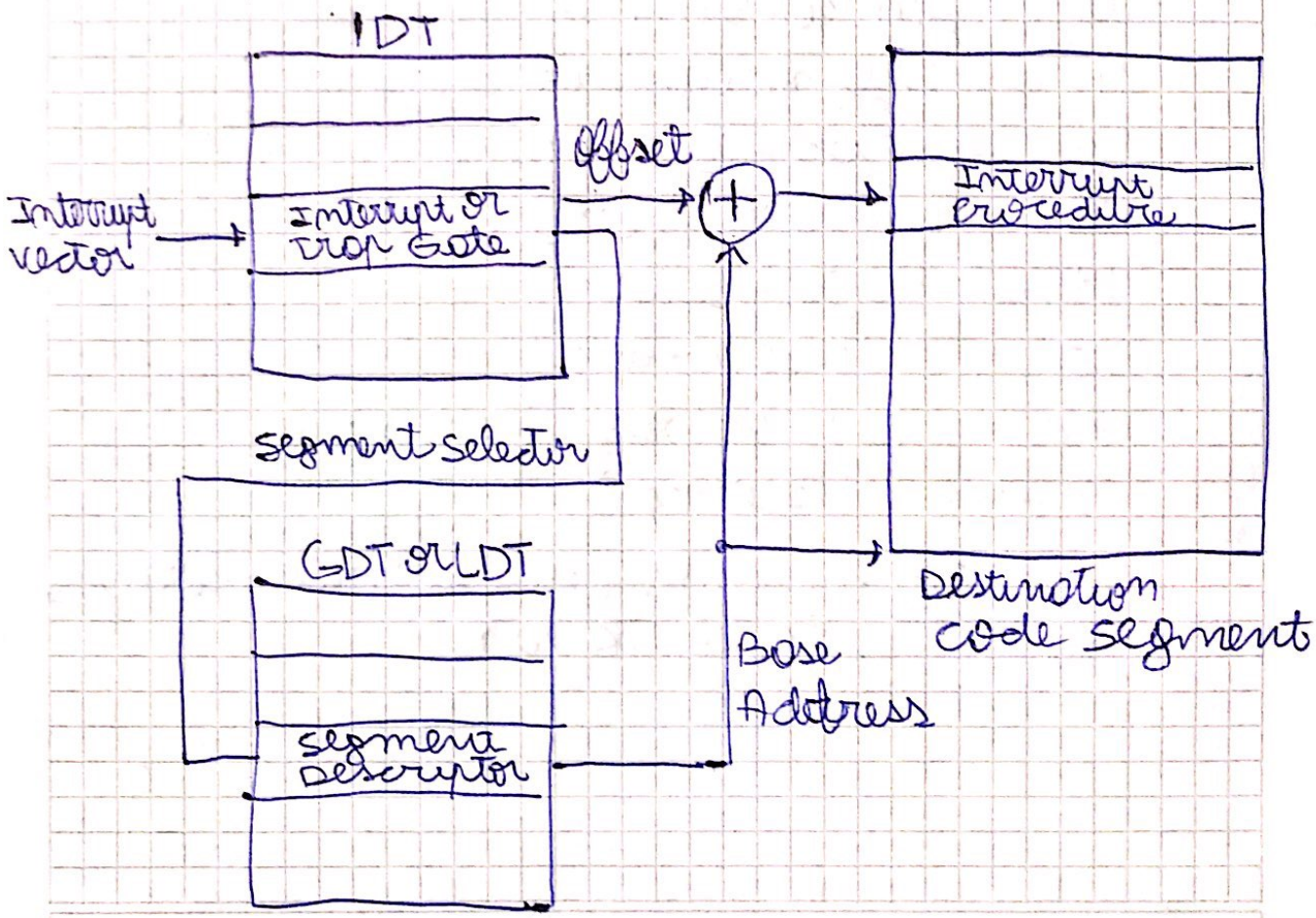
Scanned by CamScanner

IDTR

IDTR Register



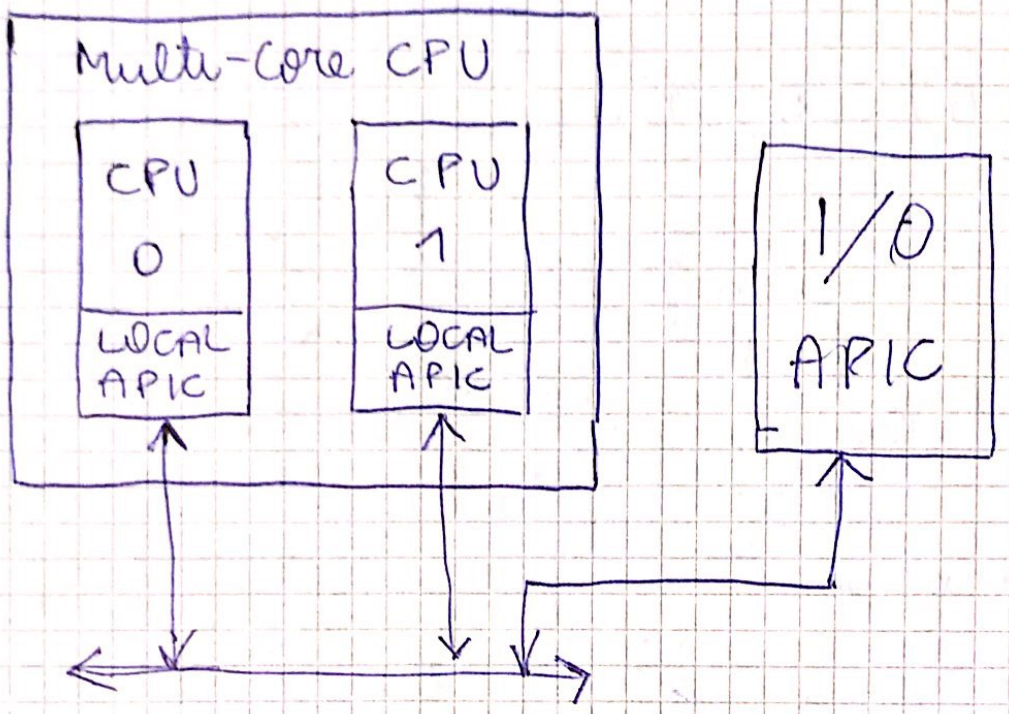
Interrupciones en Modo Protegido =



NOTA

Scanned by CamScanner

Interrupciones en Multi-Core =



El APIC (Advanced Programmable Interrupt controller) realiza el "ruteo" o direccionamiento de los periféricos a los CPU.

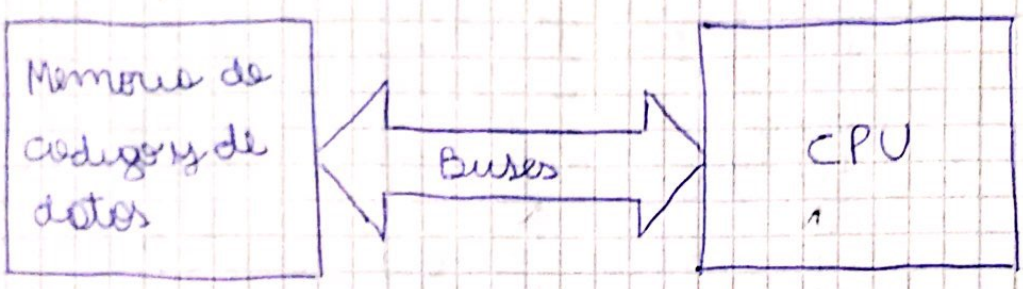
NOTA

Scanned by CamScanner

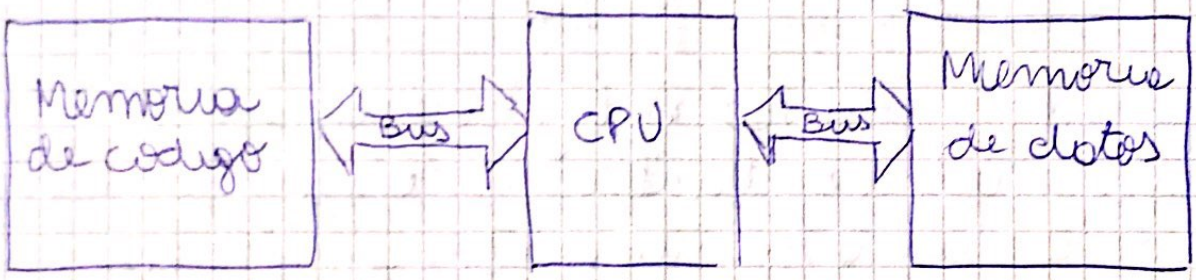
Arquitectura de Procesador

Tipos de Arquitectura =

1 ⊛ Von Neumon =



2 ⊛ Harvard =

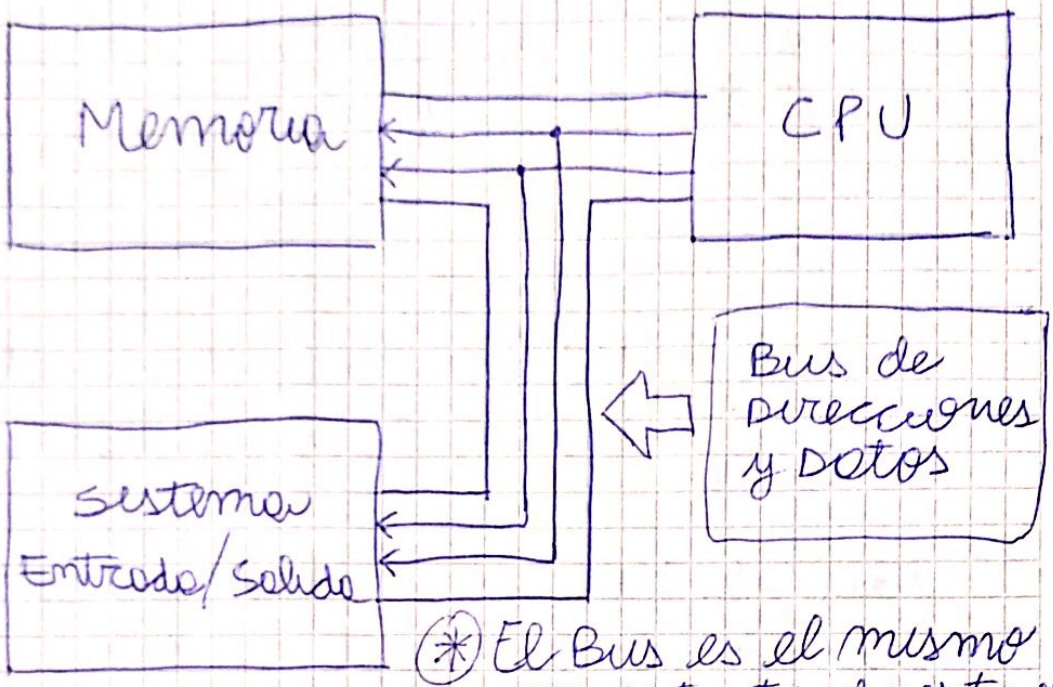


1 más Económico: utilizado en los PC actualmente

2 Paralelizo puede ir a buscar la siguiente instrucción mientras ejecutas, resulta más caro, pero se utiliza en dispositivos moviles: Ej µP ARM.

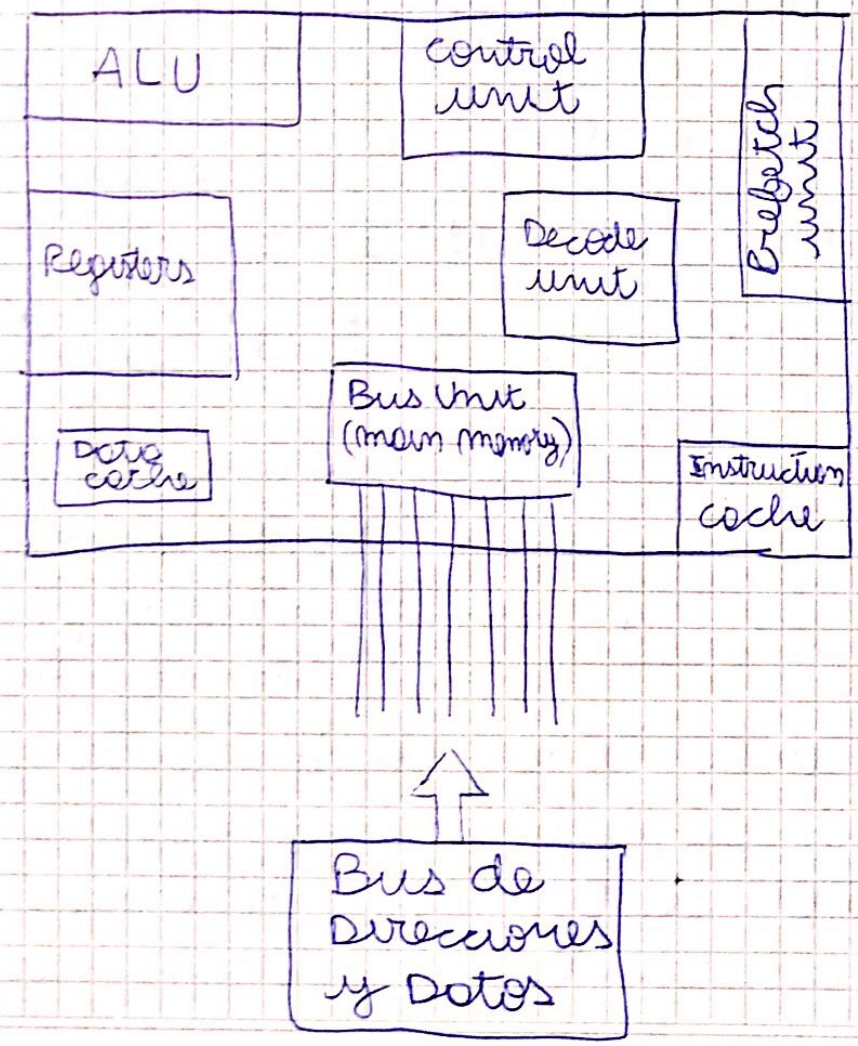
NOTA

Scanned by CamScanner



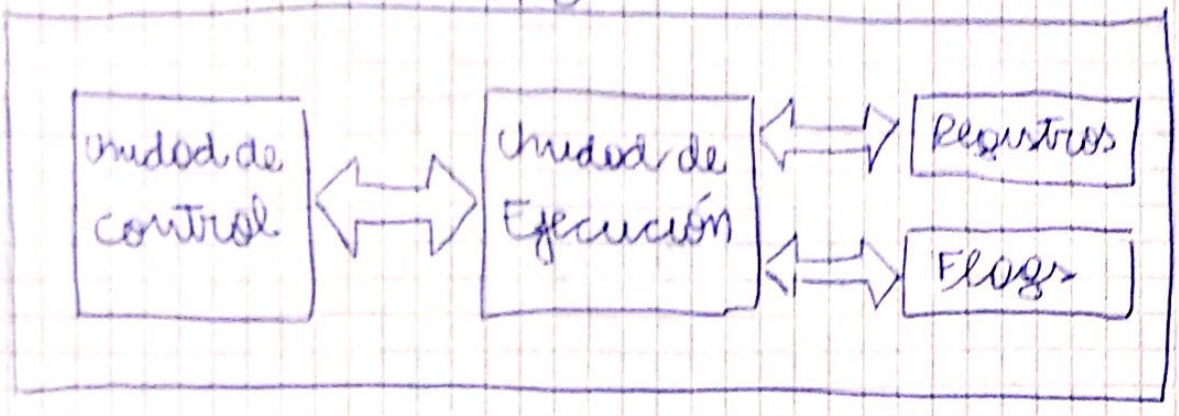
(*) El Bus es el mismo para distintos dispositivos

CPU =



Scanned by CamScanner

CPU



Unidad de control = recupera instrucciones de memoria, las decodifica, escribe en memoria.

Unidad de Ejecución = Llevo a cabo la ejecución de la instrucción.

Registros = memoria interna utilizada como variable.

Flags = indican eventos luego de ejecutar instrucciones.

Registros de INTEL =

(*) IP: Instruction Pointer

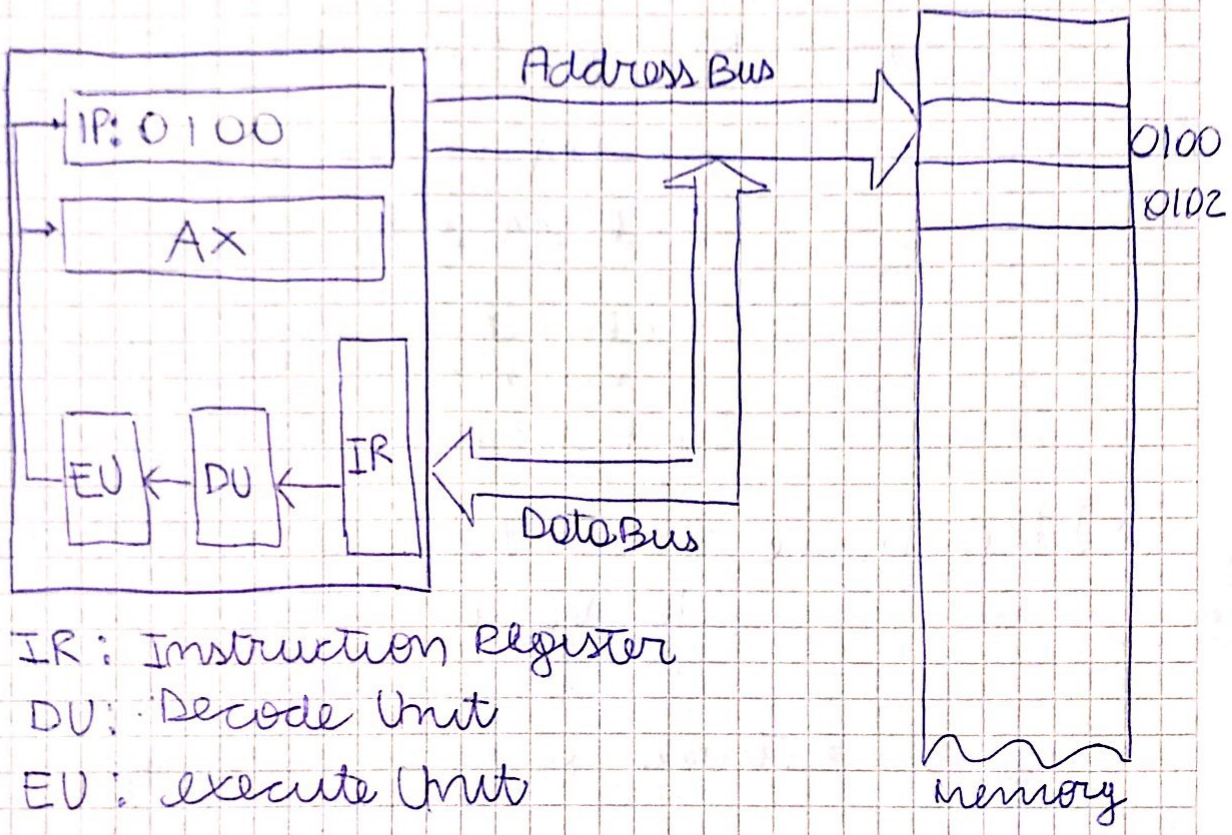
¿Cual es la primera instrucción que ejecuta el micro al encenderse?

IP —En Intel arranca→ FFFØh

Cuando se prende el uP manda por el Bus de direcciones → Hay que configurar la ROM del BIOS. (desde el Motherboard) para levantar memoria en esta ROM

NOTA

¿Que sucede en el Bus de direcciones al ejecutar: mov AX, [100h] ?

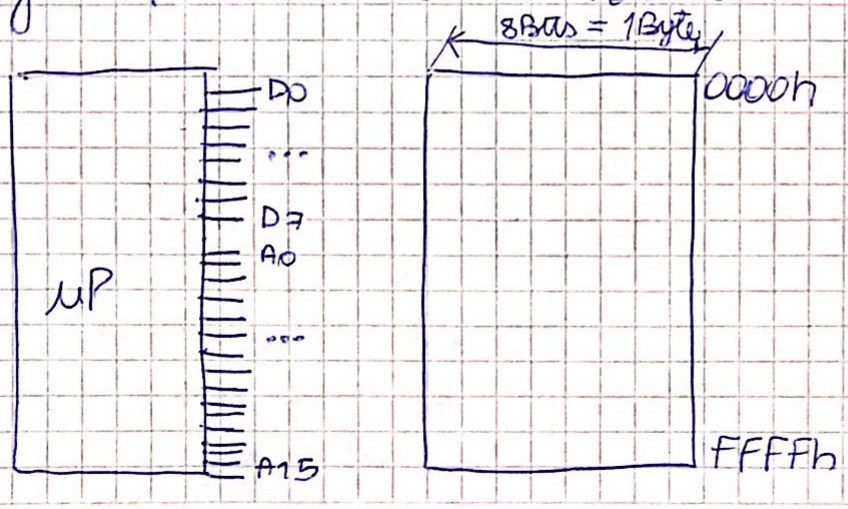


IR: Instruction Register
DU: Decode Unit
EU: execute Unit

Mapa de Memoria =

Representación de todos las direcciones que puede acceder el microprocesador

Ej MP. Bus Direcciones: 16Bits Bus de Datos = 8Bits



B. Dir = 16Bits
=> puedo direccionar 2^{16} lugares de Mem.
$2^6 \cdot 2^{10} = 64K$ direcciones.
B. Datos = 1Byte
∴ el MP puede acceder a ver 64KB de Mem.

Scanned by CamScanner

Memorias =

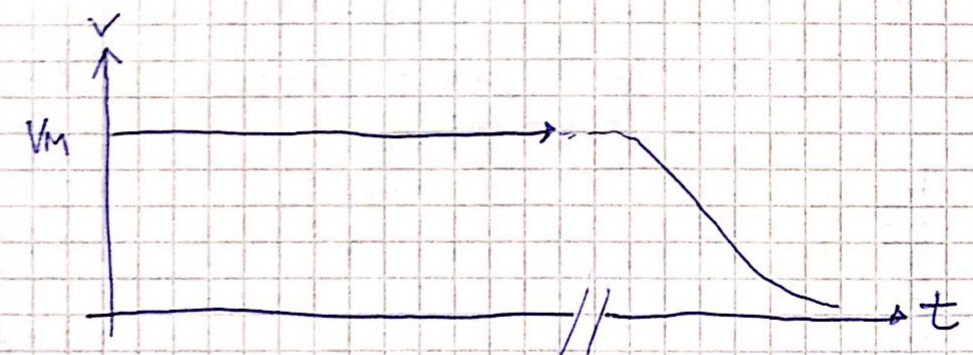
clasificación
- por el MODO en que accede a los Datos
- por las OPERACIONES que acepta
- por la DURACION de los Datos

(*) Memoria ROM (Read Only Memory)
- PROM (Programable ROM)
- EPROM (Erasable PROM)
- EEPROM = E^2PROM (Electric Erasable PROM)
- FLASH

(X) Mantiene su información sin energía
NO VOLÁTIL

(X) La escritura es más lento que la R.A.M

(X) No mantienen la información eternamente



→ Tiempo de Mantención sin refresco (por medio de un capacitor) Es muy grande: 20/30 años.

NOTA

Scanned by CamScanner

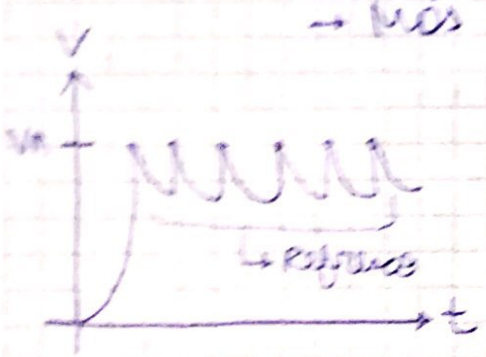
(*) Memoria RAM (Random Access Memory)

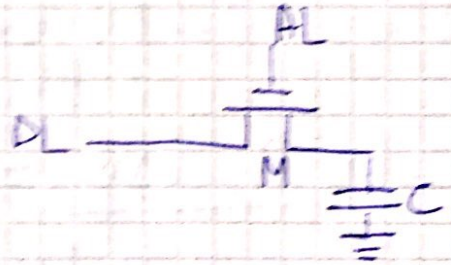
- → DRAM (Dinamic RAM)
- → SRAM (Static RAM)

(*) Pierde su información sin energía **VOLÁTIL**

DRAM =

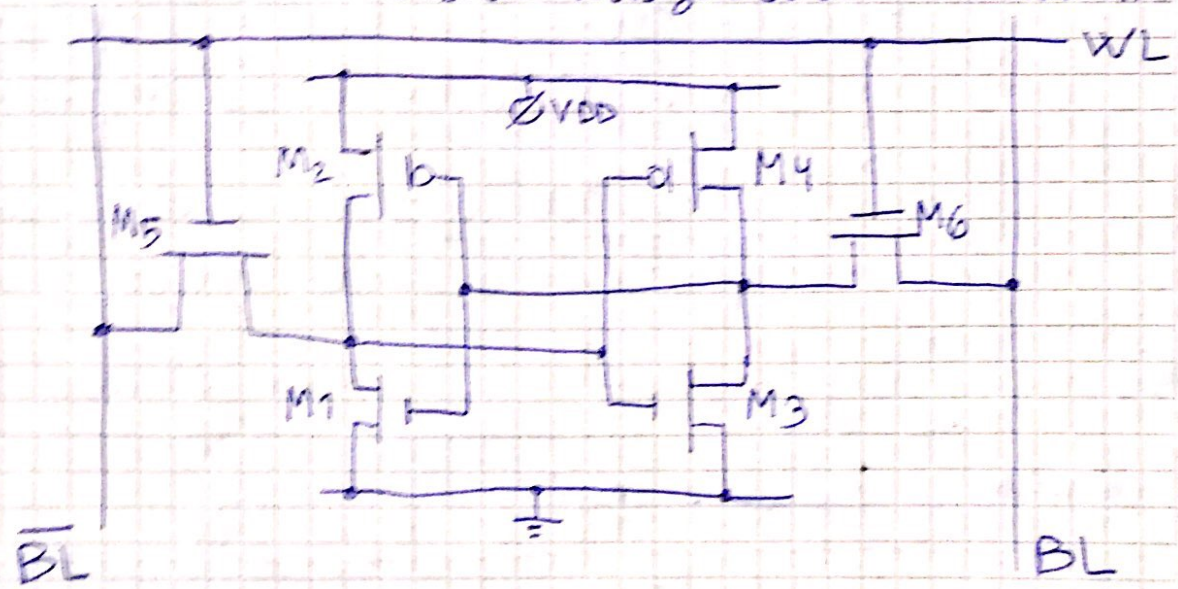
- → Necesita **REFRESCO** de niveles cada n milisegundos
- → Menos compleja ⇒ más económica
- → más lenta





SRAM =

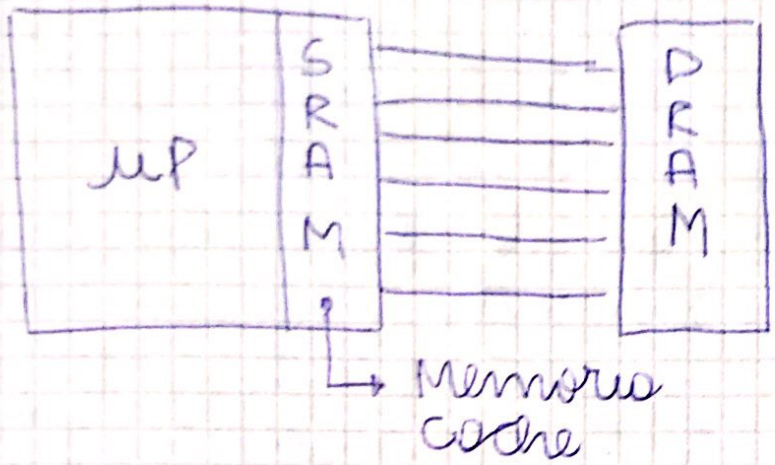
- → **No** necesita refrescar.
- → más compleja ⇒ más cara
- → más rápida
- → se suele utilizar para memoria CACHE



Más compleja: 6 Mosfet más que DRAM.

Scanned by CamScanner

(#) En las memorias el proceso de lectura es mucho más rapido que la escritura pues en este proceso no se involucra carga de capacitores, etc



Tiempo de Acceso =

Es el tiempo que le toma a una memoria RAM para completar un acceso despues de otro.
se compone de 2 tiempo =

(*) Latencia = tiempo que tarda la memoria en devolver el valor

(*) Transferencia = tiempo que tarda en enviar la información por el Bus.

Ej

	LATENCIA	TRANSFERENCIA
DRAM	50 a 150 mS	
SRAM	< 10 mS (menor)	

es el mismo pues depende del Bus

Memorias - Operación =

Las memorias para operar utilizan:

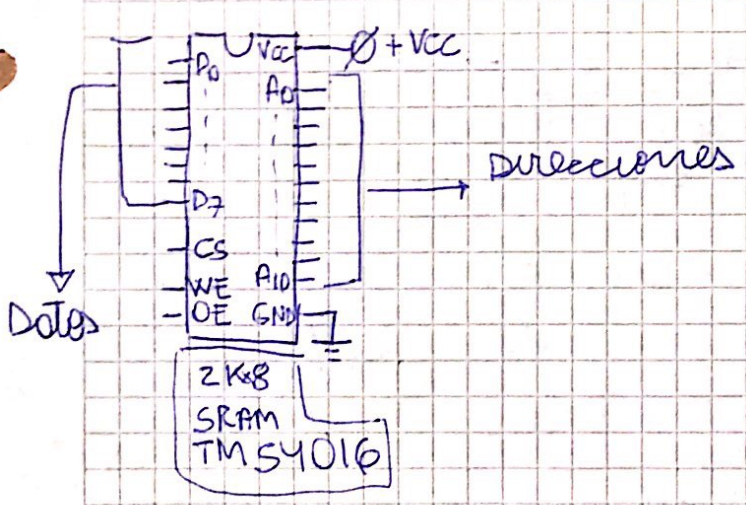
- (*) Acción a realizar (lectura o escritura)
- (*) Dirección de la palabra a acceder.
- (*) Dato (entrante o saliente según la acción)

Estructura de la Memoria =

PALABRA → ancho del Bus de Datos.

(*) Si el µP, como en el caso de $\boxed{\text{Intel}}$, quiere mantener la retrocompatibilidad => permite acceder a la memoria a nivel de $\boxed{\text{Byte}}$

Memoria Comercial =



$\boxed{\text{CS: (o S)}}$: Chip Select

$\boxed{\text{OE/G/RD}}$: Read Enable

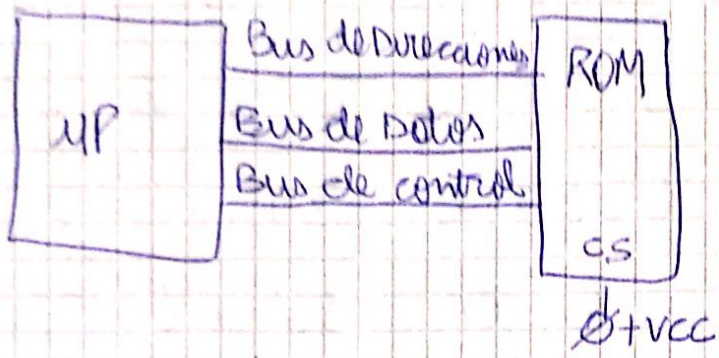
$\boxed{\text{WE/W/WR}}$: Write Enable

El Chip Select en 0 interrumpe toda la actividad. El Bus es común a todos los dispositivos → Solo debe atenderlo el dispositivo con el CS en 1.

Scanned by CamScanner

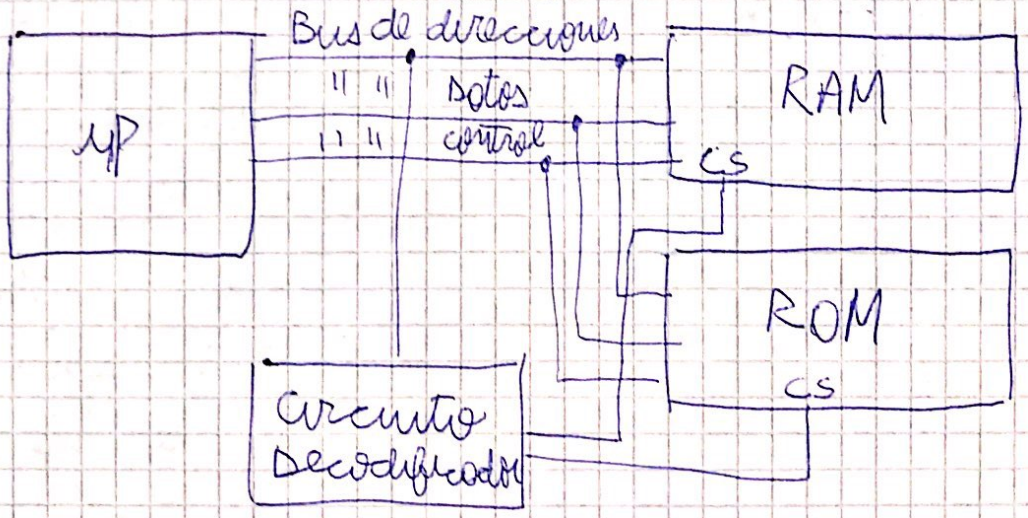
Decodificación de Hardware =

Sistema 1 =



No Hay escritura / o bien la escritura es muy lento
=> este sistema no cuenta con el Hardware necesario para el funcionamiento de una PC con S.O, etc.
Este sistema deja parado al microprocesador como un microcontrolador. (lo limita) a las soluciones que resuelven los µC.

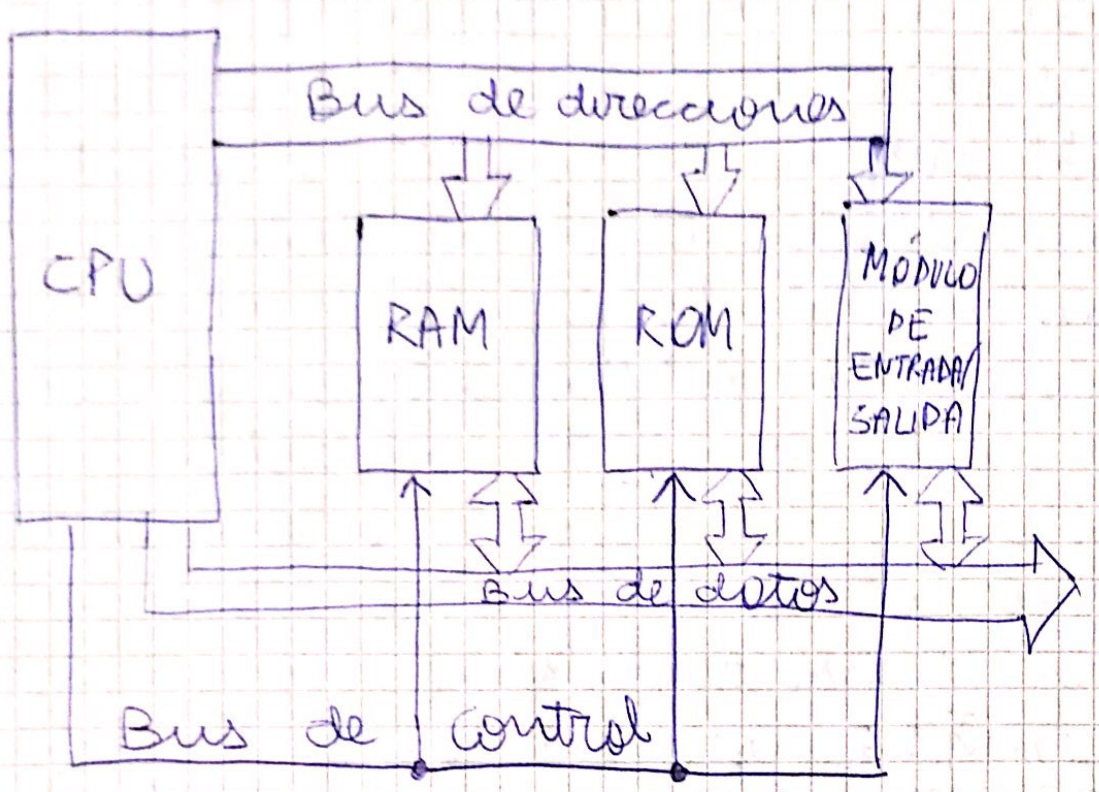
Sistema 2 =



Sumo RAM para poder escribir rapido más apto para soluciones como S.O.

Scanned by CamScanner

Buses



Bus de control =

Además del Bus de Direcciones y el Bus de datos, hay un tercer Bus, llamado Bus de control. Este Bus lleva la información digital de lo que deben hacer los distintos periféricos

Pines Básicos de Control =

$\boxed{R/\overline{W}}$: 1 → Read 0 → Write

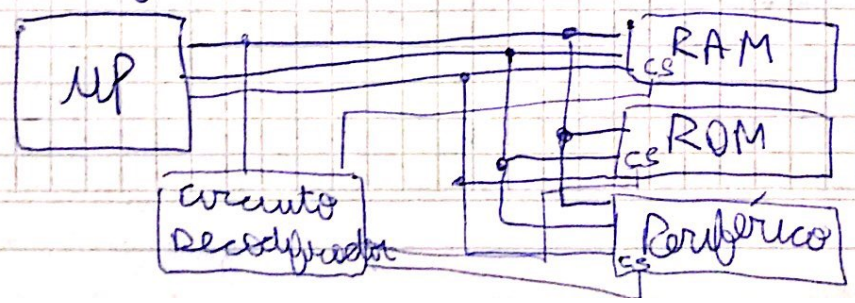
$\boxed{IO/\overline{M}}$: 1 → Operaciones con port In / out 0 → operaciones con la memoria

Scanned by CamScanner

R/$\overline{W}$: esta línea del bus de control indica si se va a leer o escribir un dato. Es decir, el dispositivo que está activado (por ejemplo la memoria) debe saber si debe tomar o escribir un dato en el Bus de datos.
Cuando esta línea esta en 0 significa que el dispositivo activo debe escribir un dato en el Bus de datos.
Si está en 1, significa que debe guardar el dato que esté presente en el Bus de datos.

IO/$\overline{M}$: Existe el mapa de memoria, que es al que accede el procesador cada vez que ejecuta una instrucción de acceso a memoria como la instrucción MOV. Pero también esta el mapa de Entrada/Salida. A este mapa se accede con las instrucciones IN/OUT. Esta línea es 0 si se quiere acceder al mapa de memoria y en 1 al mapa de entrada y salida.

Decodificación de Hardware:



NOTA

Scanned by CamScanner

Decodificación de Hardware

(*) **Decodificación completa** =

Se dice que una decodificación es completa, cuando hay una relación biunívoca entre cada posición de memoria y cada dirección

(*) **Decodificación incompleta** = Por simplicidad o para minimizar la cant de componentes, no se hacen llegar todas las líneas del Bus de direcciones. Por lo tanto aparecen "IMAGENES".

Sistema de Entrada y Salida

E/S Aislada = una señal especial del micro induce la ejecución de una operación de E/S

Interrupción = una señal externa interrumpe al micro para requerir un servicio de asistencia

Acceso Directo a Memoria (DMA) = la información se transfiere directamente a la memoria (sin CPU)

Scanned by CamScanner

Mapeo en Memoria =

se le otorga un sector de la memoria principal al dispositivo

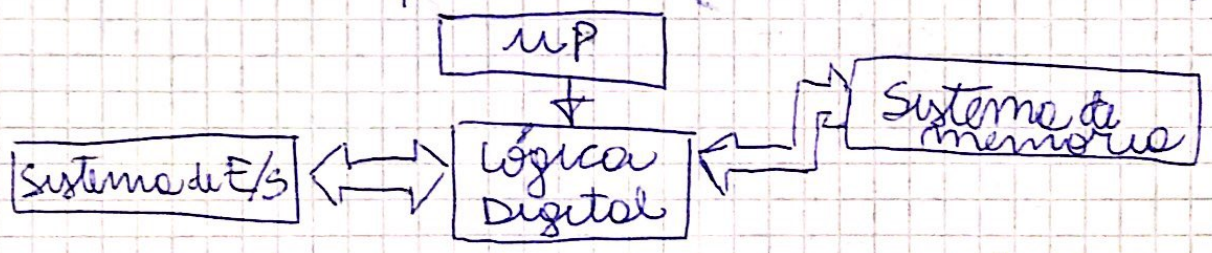
* Un dispositivo es manejado como una o varias posiciones de memoria

* cada posición de Memoria puede ser un registro interno diferente del dispositivo

* Ventaja: se utilizan las instrucciones de acceso a memoria, por lo tanto los alternativas de programación son mayores (mayor cant de instrucciones)

* Ventaja: se modifica directamente los registros del periférico sin necesidad de obtener el valor con las instrucciones IN y OUT.

* Desventaja: Reduce la cantidad de memoria. (El impacto es mínimo relacionado con la cantidad de memoria que tienen los PC actuales).



Se decodifica el periférico en el mapa de memoria, convirtiendo la lógica digital.

NOTA

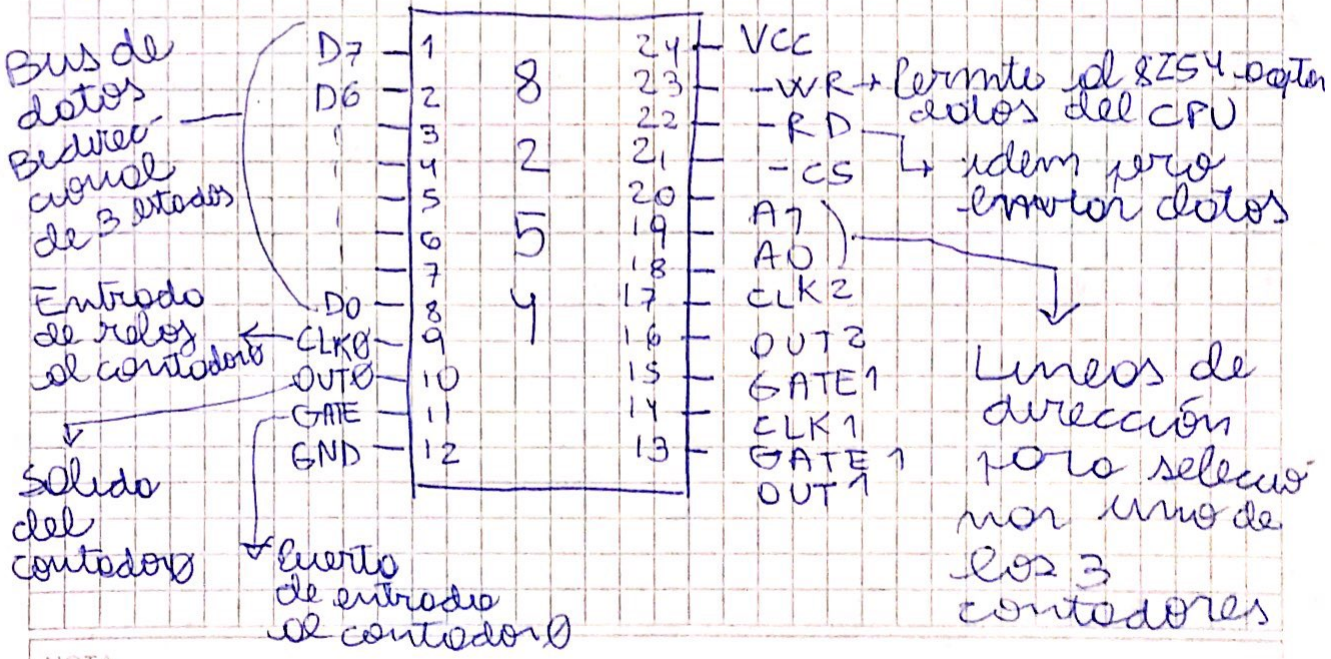
Scanned by CamScanner

Periféricos =

* Timer Programable (PIT)
* Reloj de tiempo real (RTC)
* Teclado.
* Disco Magnético.
* DMA
* BIOS

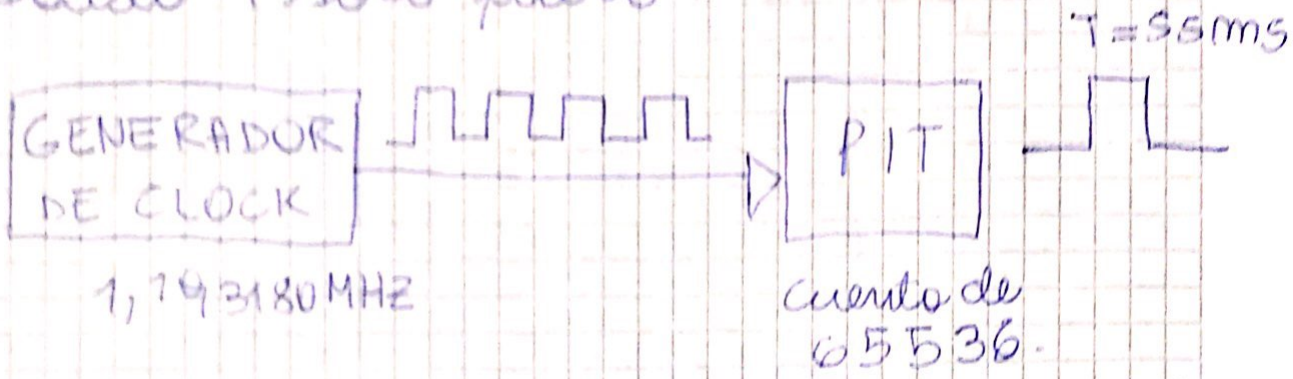
Programmable Interval Timer (PIT) =

En los primeros PCs se usó el 8254/3. Este chip puede usarse como RTC; contador de sucesos, generador de ritmo programable, generador de onda cuadrada, etc.



Scanned by CamScanner

Se programa el PIT con un número de cuenta de los pulsos de entrada al finalizar la cuenta, genera a la salida 1 solo pulso.



En la PC el timer es de vital importancia, se puede acceder a través de los puertos de entrada y salida:

40h	→ counter 1 (Timer Tick IRQ0 del PIC con T= 55ms).
41h	→ counter 2
42h	→ counter 3
43h	→ registro de control para programar el integrado

Al contador 0 se lo denomina Timer Tick y es el patrón de referencia de tiempo de todos los PC sin importar modelo y velocidad de procesador.

NOTA

Scanned by CamScanner

Real Time Clock (RTC)

con la creación de la PCAT se incluye este integrado que dispone de varios registros de configuración y status. Este dispone de una memoria RAM CMOS

(*) Para acceder al RTC se utiliza el modo de acceso indirecto, para evitar consumir demasiadas direcciones de E/S.

- En uno de los puertos se escribe el número del registro que se desea acceder.
- En el otro puerto se envían los datos.

Read

```
mov al, mem
OUT 70h, al
IN  al, 71h
```

Write

```
mov al, mem
OUT 70h, al
mov al, new_content
OUT 71h, al.
```


Scanned by CamScanner

Teclado =

(#) Al presionar una tecla se genera un MAKE CODE

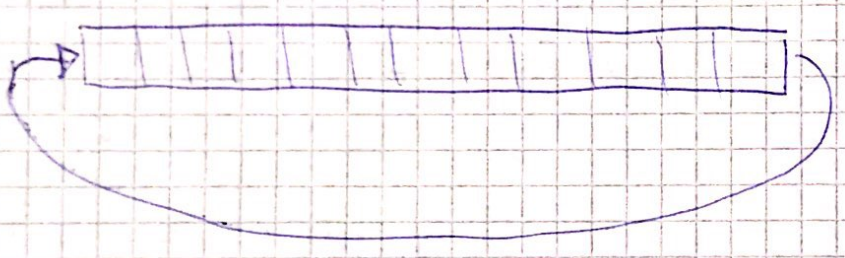
(#) Al liberar una tecla se genera un BREAK CODE

Ambos generan Interrupción

Ej: 'Q': MAKE = 10h → BREAK = 90h.

Para leer el scan code en el driver del Teclado: IN AL, 60h

Buffer circular o Anillo

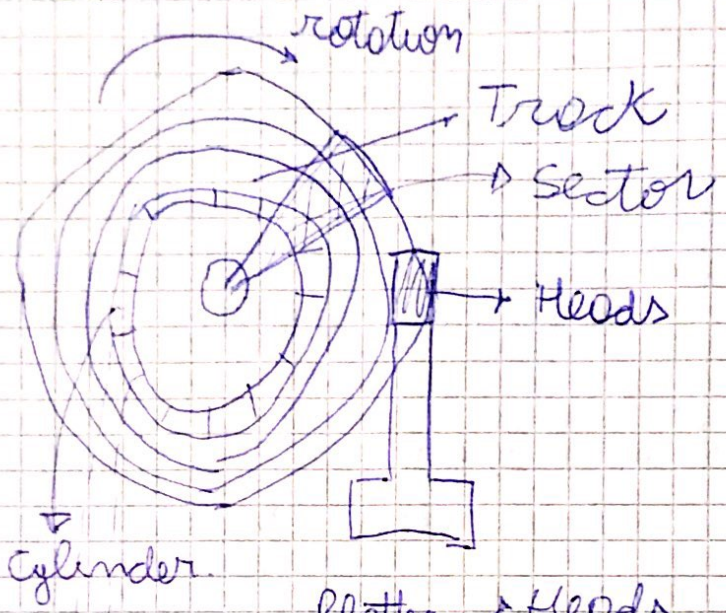


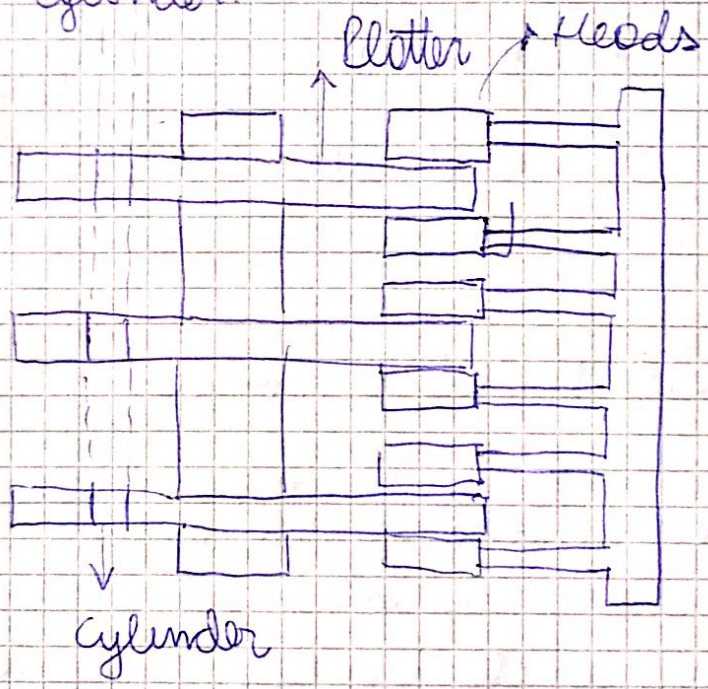
(*) Las teclas modificadoras (BIOS) se encuentran en la BIOS Data Area.

NOTA

Scanned by CamScanner

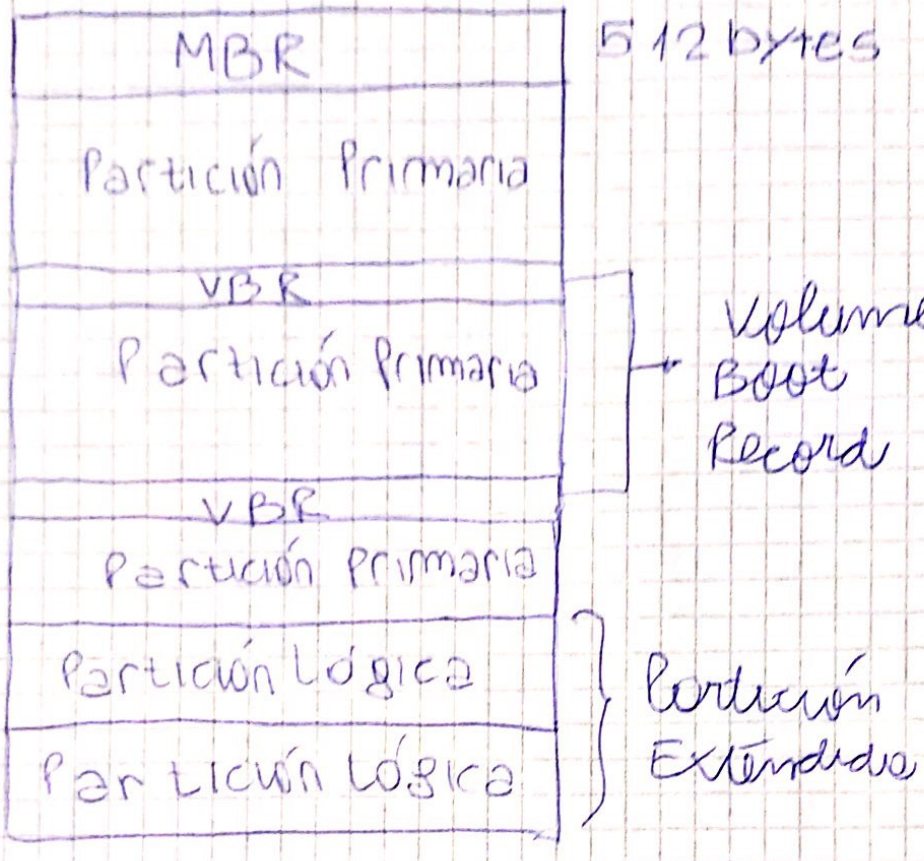
Disco Rígido Magnético





Scanned by CamScanner

Tabla de Particiones =



MBR (Master Boot Record)

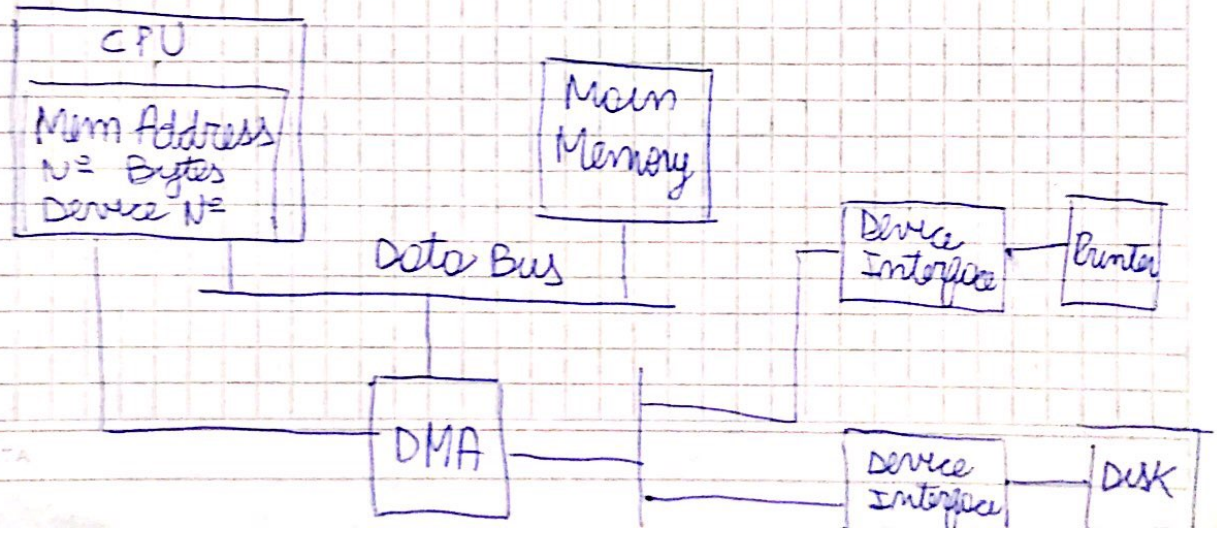
GRUB El GRUB se compone de dos archivos principales:

* Stage 1 (512 Bytes)
  - Básicamente carga el Stage 2 de disco a memoria.
* Stage 2 (30 KB aprox)
  - Interfaz de usuario.
  - Set de comandos de Booteo
  - Levanta Kernel a memoria
  - O paso el control a otro Bootloader (Windows).

NOTA

DMA (Direct Memory Access)

* El controlador de DMA utiliza el bus de direcciones.
* Accede a la memoria cuando el CPU lo permite.
* Puede realizar transacciones:
  - Periférico a memoria
  - Memoria a memoria
* En sus inicios se implementó ISA DMA (8237):
  - 2,4Mb/s como máximo
  - 16 Bits de Bus de direcciones
  - Obsoleto.
* PCI DMA (BusMastering-UDMA)
  - 66Mb/s
  - 64Bits de direccionamiento.



Scanned by CamScanner

BIOS (Basic Input Output System)

El orden que sigue el BIOS para chequear el sistema es:

- comprobación de CPU
- CMOS clock
- DMA
- Teclado
- Los primeros 64Kbytes de RAM
- controlador de interrupciones
- controlador cache de motherboard
- controlador de video
- Memoria superior a los primeros 64KBytes
- Puerto serie y Paralelo
- Disco y Disketera

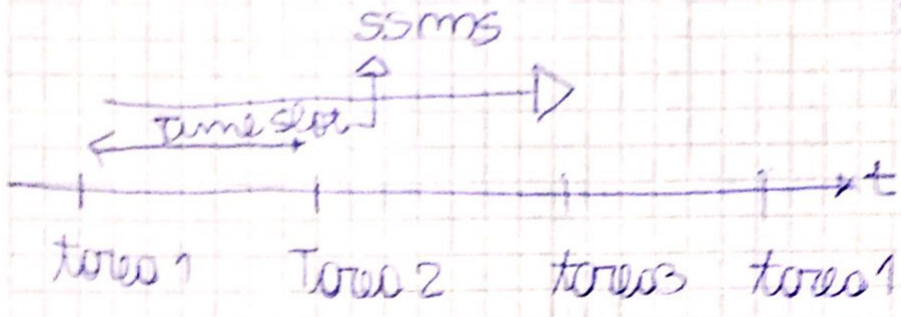
Luego en la etapa POST lo primero que hace es Buscar las extensiones de ROM. Realiza una Busqueda por los slots para Buscar placas que poseen memorias ROM o EPROM. Estas tienen la función de ampliar o sustituir las funciones del BIOS.

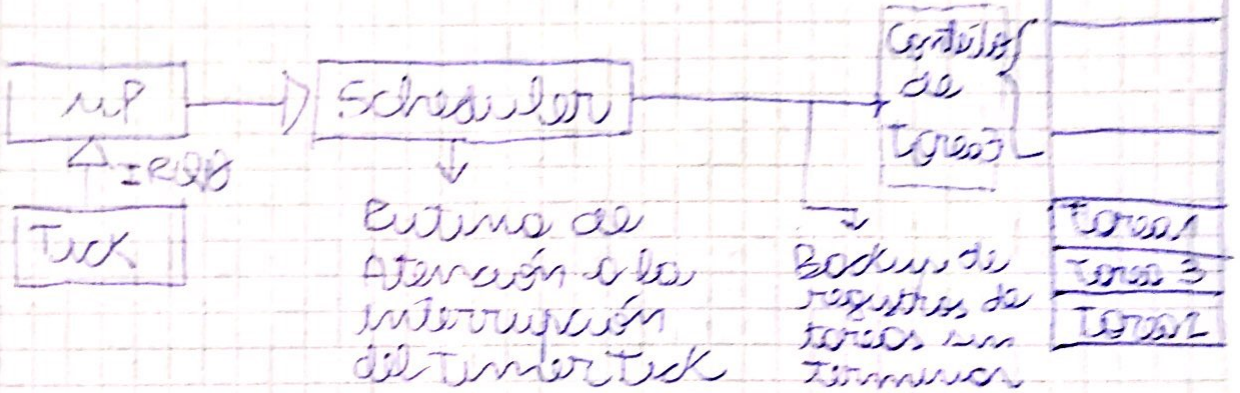
NOTA

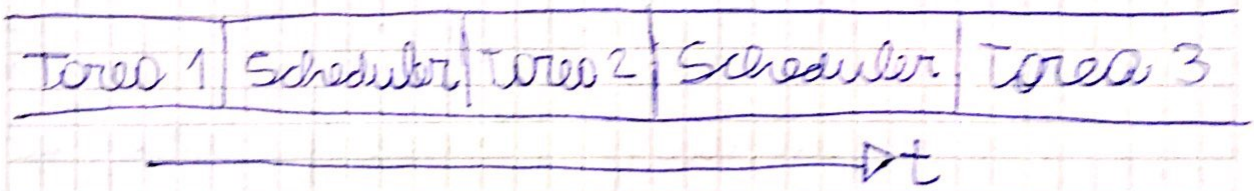
Scanned by CamScanner

Modo Protegido

(*) Conmutación de Tareas = Contexto de Tarea





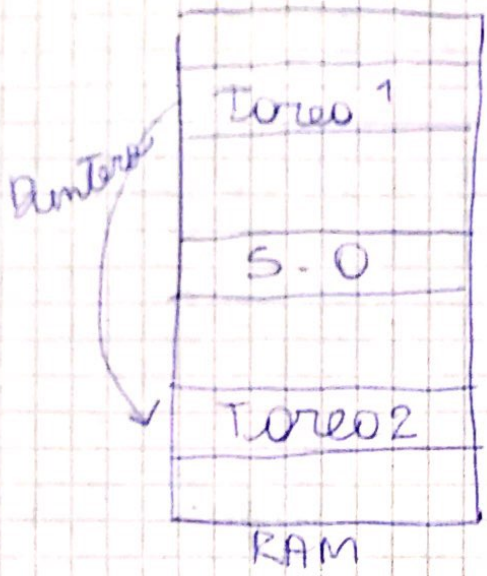


(*) ¿El sistema Operativo es una Tarea?

Sí, sin ir más lejos en el anterior gráfico temporal se ve el scheduler el mismo es una tarea y tiene su time slot

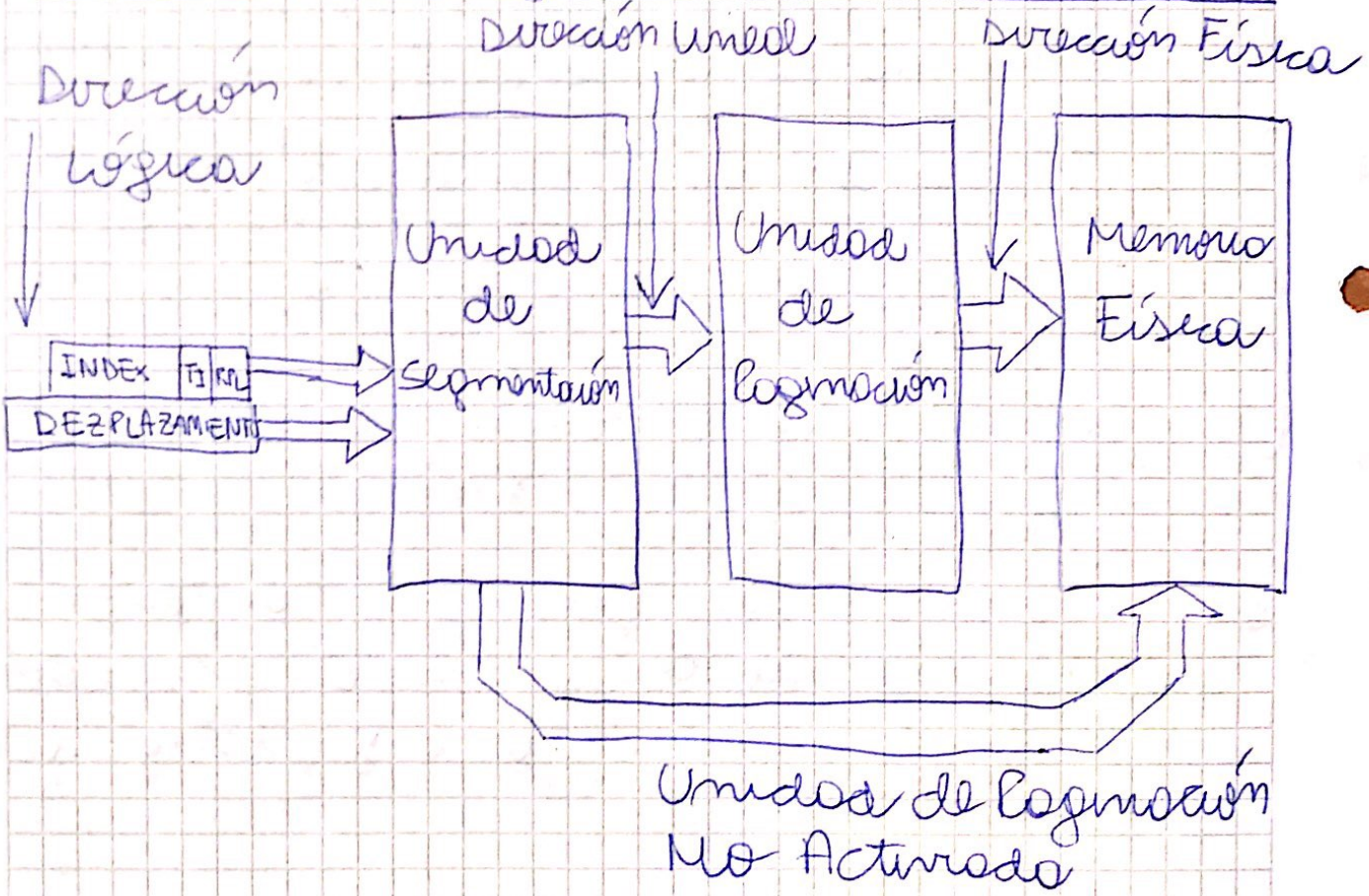
Scanned by CamScanner

Protección de Tareas =



¿Como puede evitar el S.O. este acceso? → Lo evita el µP.

MMU (Memory Management Unit) =



NOTA

Scanned by CamScanner

(*) La unidad de segmentación [No] se puede desahabilitar.

(*) La unidad de paginación [SI] se puede desahabilitar.

Debido a esta limitación de los procesadores Intel, algunos sistemas operativos eligen utilizar la memoria en modo "FLAT" para no utilizar las reglas de segmentación pero si las de paginación.

Dirección Lógica =
La segmentación ofrece un mecanismo para la división del espacio de memoria direccionable por el procesador (llamado Espacio de direccionamiento Lineal.) en espacios de direcciones protegidos más pequeños (llamados segmentos)
Los segmentos pueden ser usados para almacenar CODIGO, DATOS o PILA para un proceso o estructura de datos del sistema. (como TSS's o LDT's).
Si más de un proceso o tarea corre en el uP =>

Scanned by CamScanner

A cada uno de ellos se le puede asignar su propio juego de segmentos

El uP fuerza los límites (Privilegios) entre estos segmentos y asegura que una tarea no interfiera con otra escribiendo en los segmentos de memoria que no les fueron asignados.

Todos los segmentos de un sistema deben ubicarse dentro del espacio de direccionamiento lineal del uP. Para ubicar una posición de memoria (Un Byte: INTEL) en particular, se requiere de una <u>dirección</u> <u>Lógica</u> (Puntero lejano).

Una Dirección Lógica consta de un SELECTOR de segmento y un desplazamiento dentro del mismo (Offset)

Un selector de segmento es un identificador único del segmento, el mismo se lo ubica a través de un desplazamiento de la tabla de descriptores de segmentos (por ejemplo GDT), en dicha posición se encuentra una estructura del sistema, reconocida por el uP como descriptor de segmento, conteniendola Base del Segmento (Ubicación del primer Byte del segmento dentro del espacio direccionable lineal de Memoria.)

Scanned by CamScanner

también su límite (tamaño del segmento), sus permisos de acceso y nivel de privilegio y tipo de segmento.
La segmentación también brinda la posibilidad de tipificar los segmentos para evitar que ciertas operaciones se puedan realizar en ciertos segmentos (lectura, escritura, ejecución).

(*) Si no se utiliza paginación, la dirección lineal es mapeada directamente a la dirección física de memoria.
La dirección física de memoria es el rango de direcciones que el µP es capaz de direccionar (Definido por el Bus de direcciones).
Comúnmente en un sistema multitarea se define un espacio de direccionamiento lineal (Mapa de memoria) más grande conviene poder abarcarlo en espacio físico de direccionamiento, para ello es necesario algún método para "virtualizar" el espacio lineal. (Esta virtualización es manejada por el mecanismo de paginación del procesador)

Ejemplo: DS = 20h offset = 1000h

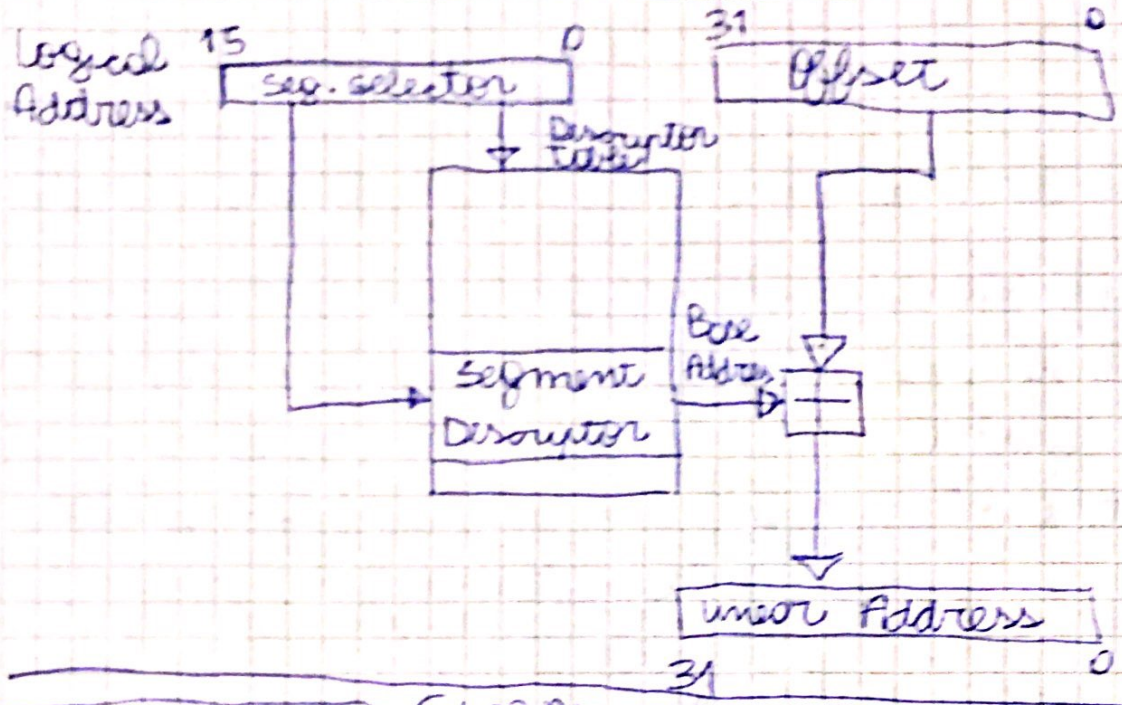
MOV DS: [1000h], EAX

NOTA
(CS, DS, ES, SS, FS o GS) → Registro de segmento

SELECTOR

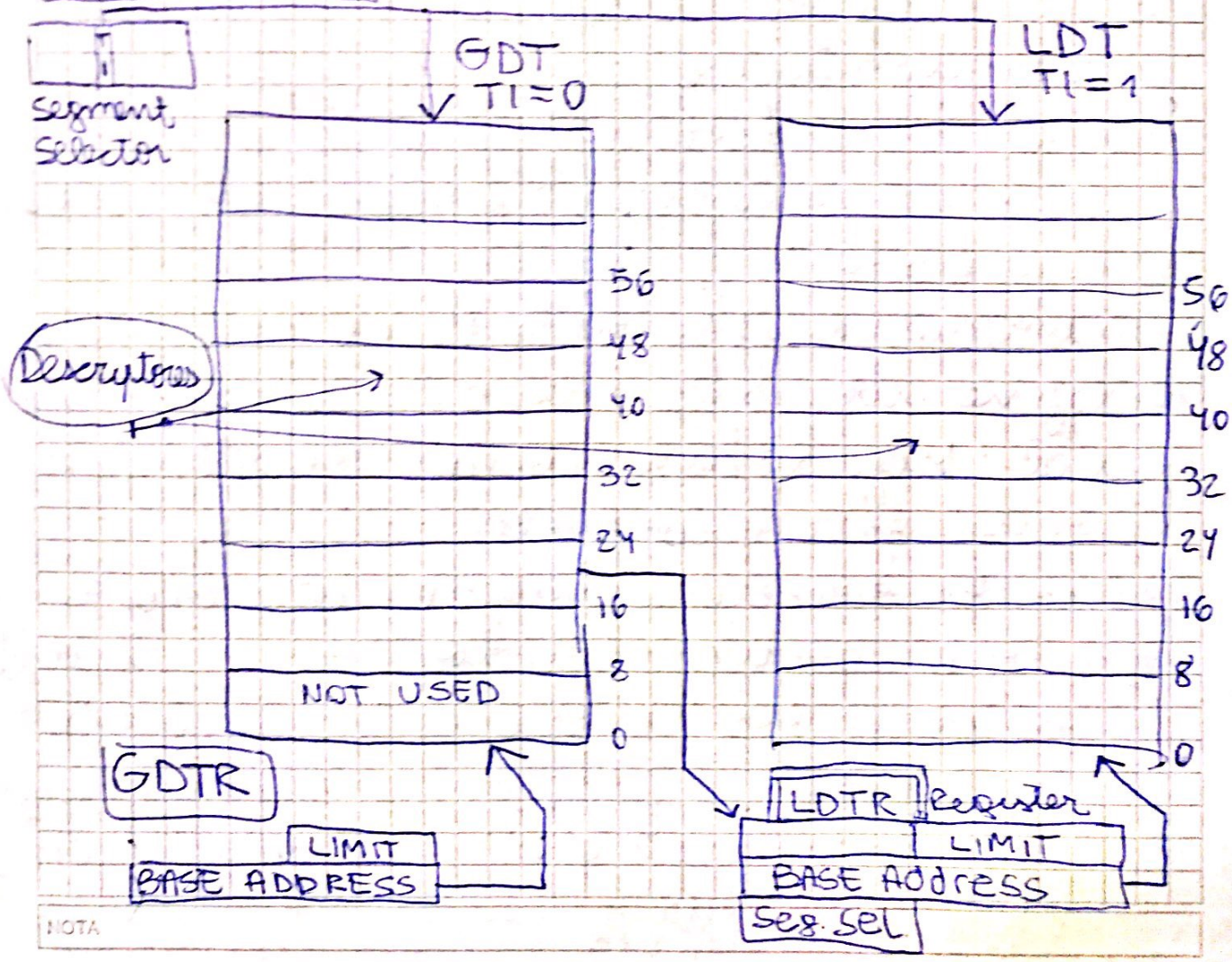
15	2	0
INDEX	TI	RPL

Dirección lógica a Lineal =



GDT y LDT

GLOBAL DESCRIPTOR TABLE y
LOCAL DESCRIPTOR TABLE



NOTA

Scanned by CamScanner

Descriptores

Veamos la estructura genérica de los descriptores de sistemas

31					16						0	
SEGMENT BASE 15...0						SEGMENT LIMIT 15...0						0
BASE 31...24	G	0	0	0	LIMIT 19...16	P	DPL	0	TYPE		BASE 23...16	+4

1 2

TYPE

0 → Invalid

1 → Available 80286 TSS

2 → LDT

3 → BUSY 80286 TSS

4 → 80286 Call Gate

5 → Task Gate

6 → Interrupt Gate

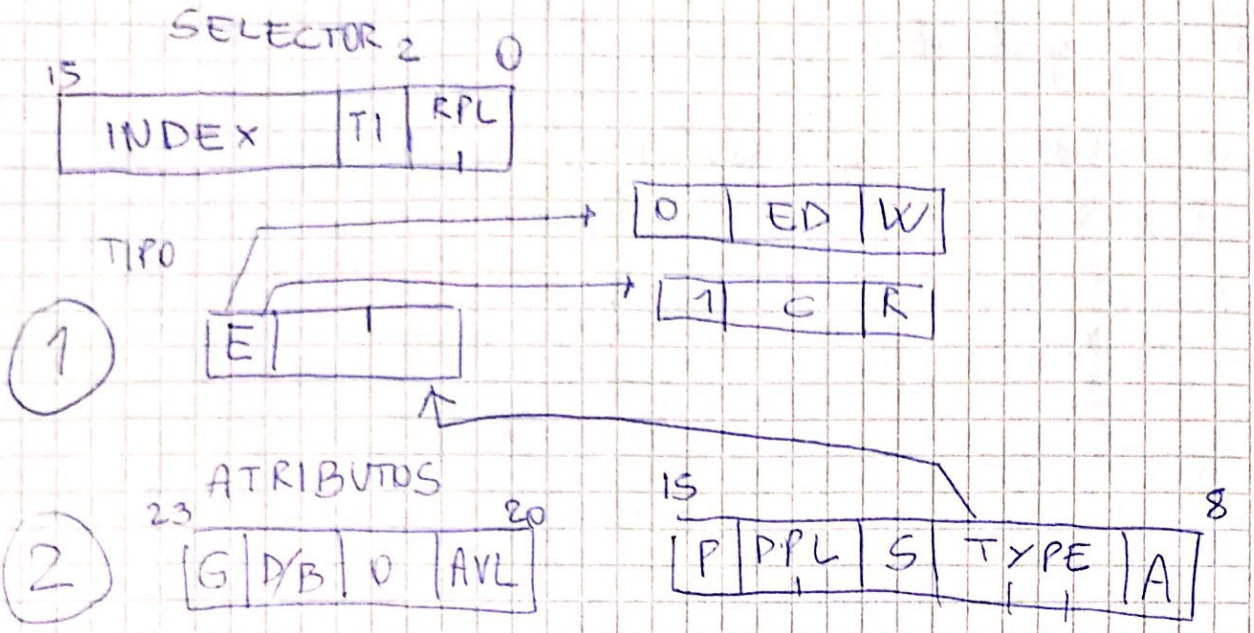
7 → Trap Gate

8 → Invalid

9 →
⋮
F → } Intel 386.

Scanned by CamScanner

Descriptores de Segmento



Atributos =

(*) P (Bit de Presencia)

- Si P=1 el segmento esta cargado en la memoria física.
- Si P=0 el segmento esta ausente

(*) DPL (Nivel de Privilegio)

- Indica el nivel de privilegio del segmento
- 0 = maximo privilegio (Kernel Space). 3 = minimo privilegio (User Space)

(*) S. (tipo de segmento)

- Si S=1 se trata de un segmento normal (codigo, datos o pila)
- Si S=0 se trata de un segmento de sistema (puerto de llamado, TSS, etc).

NOTA

Scanned by CamScanner

(*) Tipo = Es un campo de 3 Bits.

[#] En el caso de un segmento normal:

- Si E (Ejecutable) = 1 => segmento ejecutable
- Si C (Ajustable o conforming) define si al segmento al ser accedido cambia su nivel de privilegio al nivel de privilegio del segmento que lo llama (C = 1).
- Si R (Leible) = 1 => segmento puede ser leído

[#] Si E = 0 => Segmento de Datos

- El segundo Bit ED define si se trata de un segmento de datos normal (ED = 0) o segmento de Pila (ED = 1) ED: Expansión decreciente.
- el tercer Bit W (Escribible).

  W = 1 => se puede escribir

  W = 0 => solo lectura

Dirección Lineal - Ejemplo =

DS: 20h y Offset: 1000h

DS = 0020h = 0000 0000 0010 0000 b

TI → GDT, RPL

RPL: Request Privilege Level.

- Accede al Descriptor y de la GDT
- Obtiene la dirección Base
- Ej Dir Base: 5600 0000h
- Luego le sumo Offset de 1000h

Dirección Lineal: 5600 1000h

Scanned by CamScanner

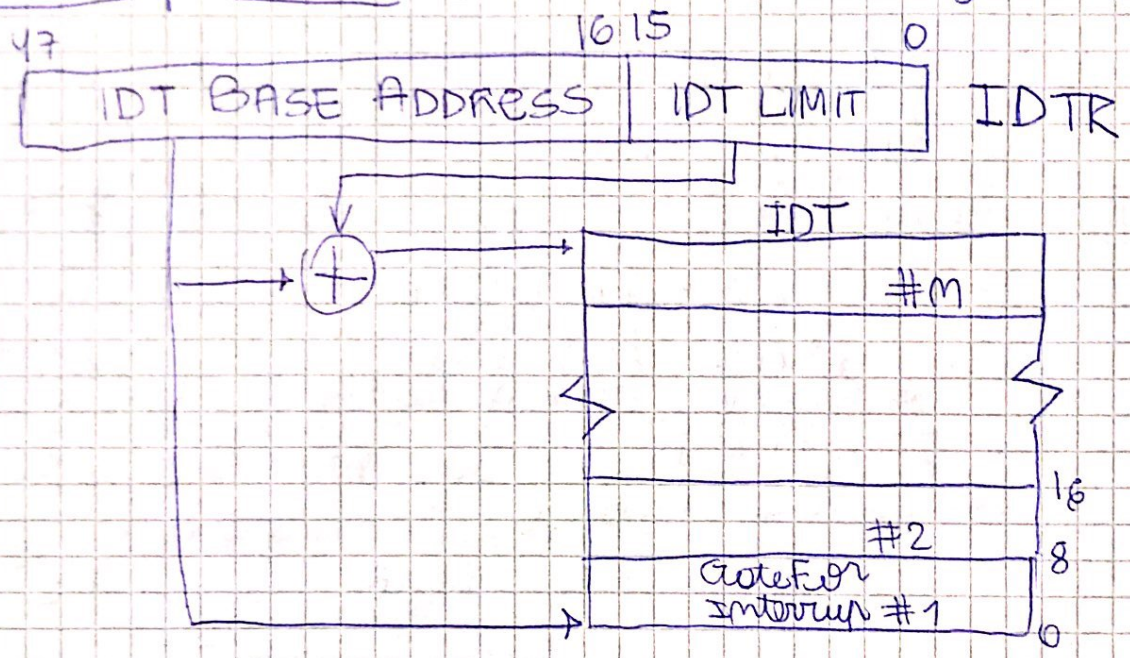
Descriptores de sistema

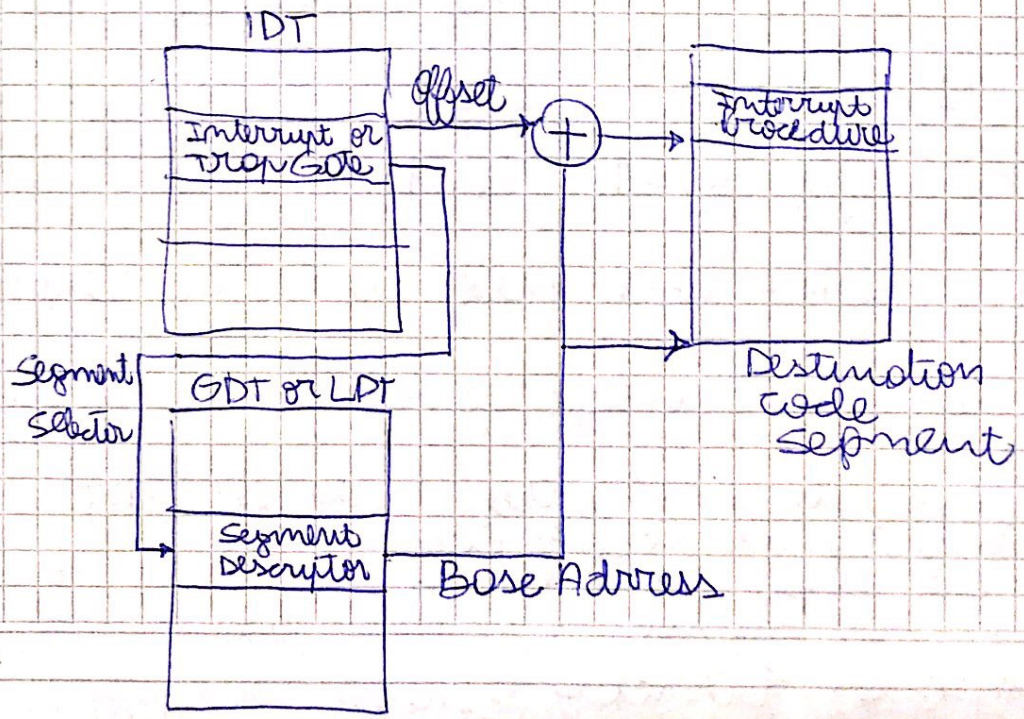
Cuando el Bit S=0 se lo denomina "descriptor de sistema"

Existen diferentes tipos:

- Trap Gate (Puerta de Excepción)
- Interrupt Gate (Puerta de Interrupción).
- TOSK Gate.

Interrupciones en modo Protegido





NOTA

Tipos de descriptores en la IDT =

Task Gate

31 … 16	15	12 … 8	7 … 0	
	P	DPL	00101	4

31 … 16	15 … 0	
TSS segment selector		0

Interrupt Gate

31 … 16	15	12 … 8	7 … 4	0	
Offset 31…16	P	PL	00110	000	4

31 … 16	15 … 0	
Segment selector	Offset 15…0	0

Trap Gate

31 … 16	15	12 … 8	7 … 4	0	
Offset 31…16	P	PL	00111	000	4

31 … 16	15 … 0	
segment selector	Offset 15…0	0

(CPL) Current Privilege Level
Variable interna del µP para comparar con un (RPL) Request Privilege Level.

(*) La tabla de Descriptores se describe a sí misma

Scanned by CamScanner

Manejo de Tareas

El procesador de Intel nos da una herramienta para guardar contextos de la ejecución: se denomina TSS
Task segment state

* Guarda los registros y un puntero a la tarea anterior, etc.
* Tiene un tamaño mínimo de 100 Bytes
* no es utilizado (no tuvo exito)

Privilegios de E/S =

El procesador provee 2 mecanismos.
- Campo de 2 Bits $\boxed{\text{IOPL}}$ en los EFLAGS. Que especifica el mínimo nivel de privilegio que se debe tener para utilizar las instrucciones de E/S
- Mapa de Bits de E/S en el TSS.

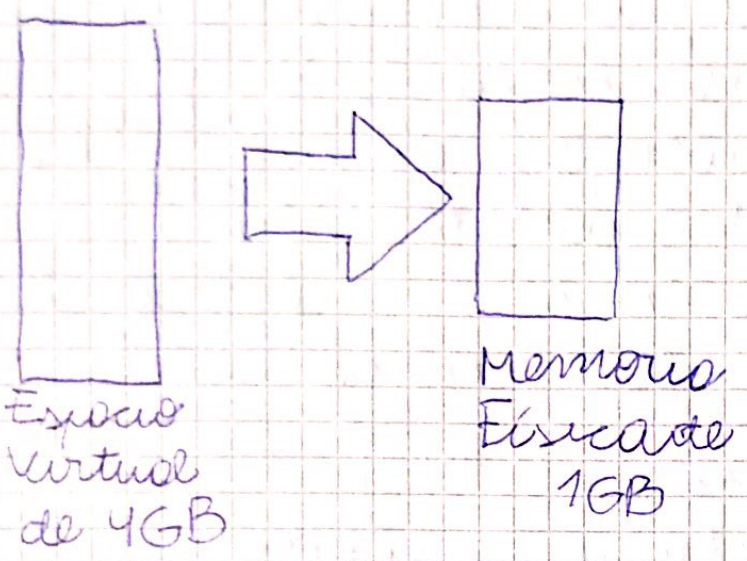
- IN
- OUT
- INS
- OUTS
- CLI
- STI

NOTA

Scanned by CamScanner

Memoria Virtual

Problemática: suponemos un µP de 32Bits en una PC de 1GB de Memoria



La paginación soporta un entorno de "memoria virtual" donde un gran espacio de memoria lineal es simulado con un pequeño espacio de memoria físico (RAM y ROM) y un sector en disco. Cuando se utiliza paginación, los segmentos son divididos en páginas, cada una, comunmente de 4K los cuales son almacenados o en espacio de memoria físico o en disco. El sistema operativo mantiene un directorio de paginas, y un juego de tablas de páginas para tener un seguimiento de los mismos.

NOTA

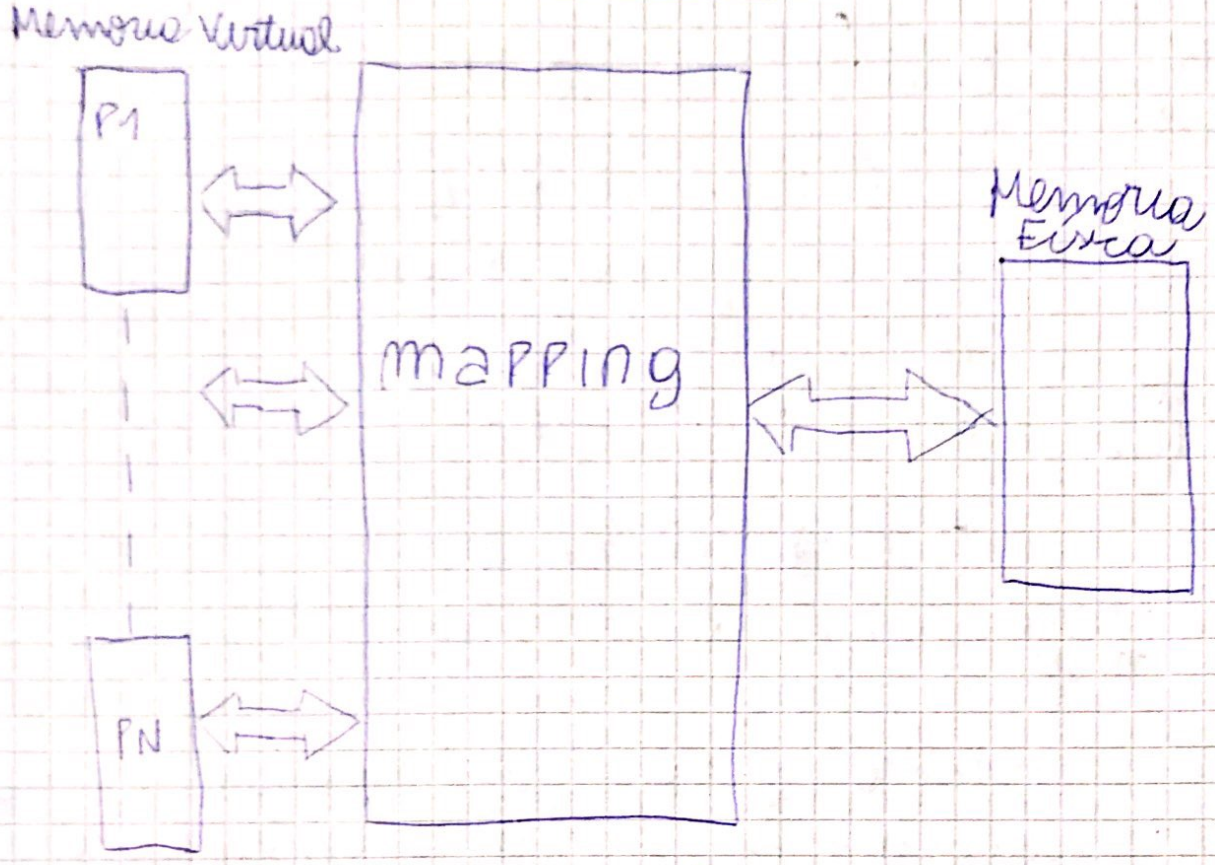
Scanned by CamScanner

Cuando un proceso o una tarea intenta acceder a una dirección del espacio de memoria ~~virtual~~ lineal, el MP consulta el directorio de paginas y las tablas para transformar la dirección lineal en una dirección del espacio físico de memoria y luego realiza la operación requerida (lectura / Escritura) en la ubicación deseada. Si la pagina solicitada no se encuentra en el espacio de direccionamiento físico el MP genera una Exception (Page Fault) El S.O obtiene la pagina del disco y la ubica en el espacio de direccionamiento físico para ~~[illegible]~~ continuar la ejecución interrumpida
Cuando la paginación es implementada correctamente en el sistema operativo el swapping de paginas entre el espacio físico de direccionamiento y el disco, es transparente para el correcto funcionamiento del proceso o tarea.

NOTA

Scanned by CamScanner

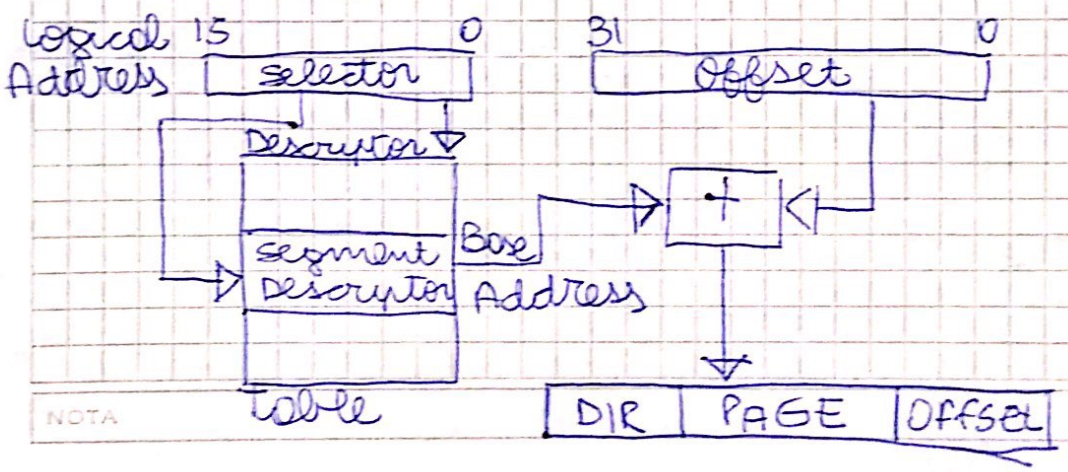
Solución: Indirección =



Cada proceso es dueño de su espacio virtual de memoria.

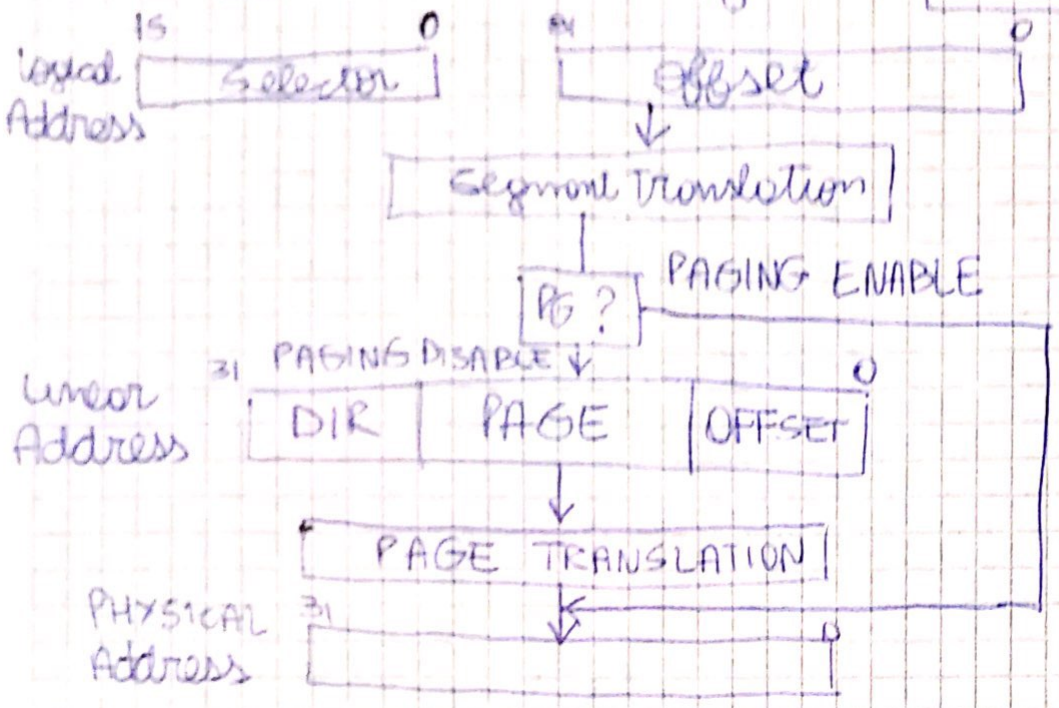
Paginación =

Con un selector y el offset a través de una tabla de descriptores (GDT o LDT) se obtiene una dirección lineal.

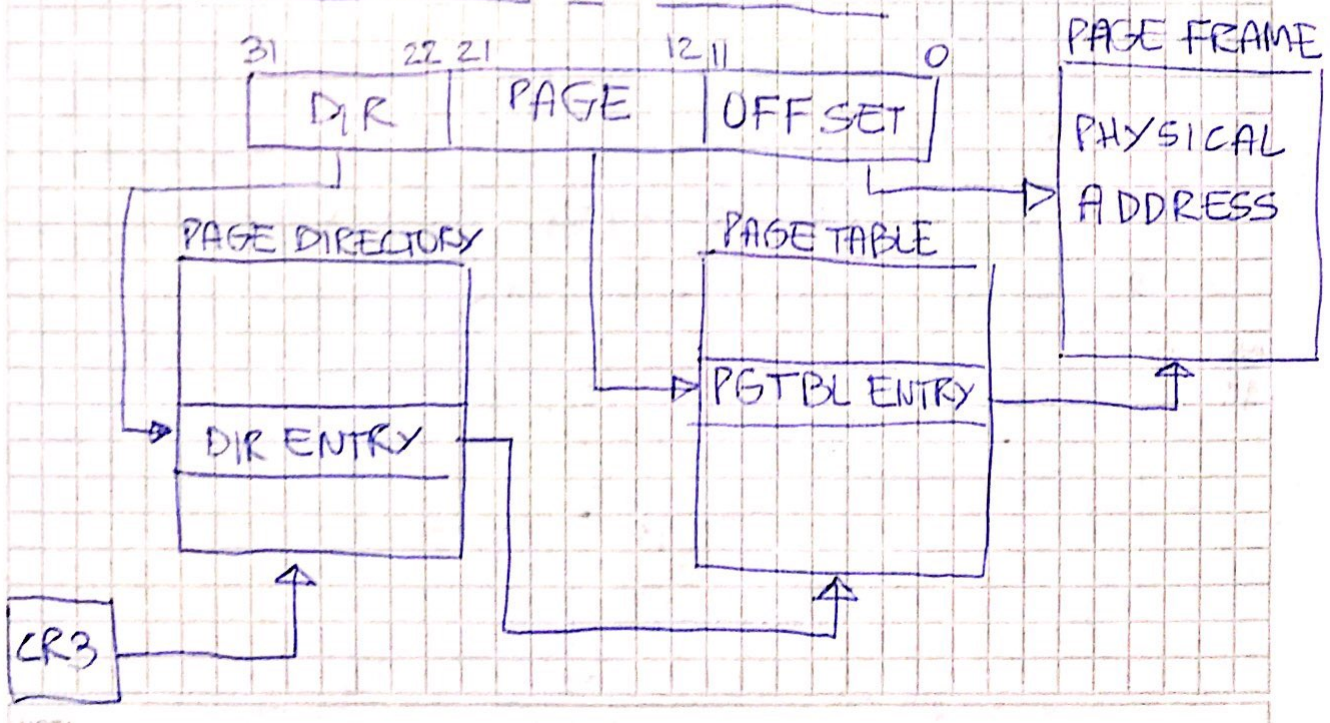


Scanned by CamScanner

(*) Recordemos que la unidad de Paginación se puede deshabilitar, con el Bit [PG] que se encuentra en el registro [CR0]



(*) La dirección lineal al pasar el módulo de paginación se convierte en dirección físico. El registro [CR3] indica la ubicación del Directorio de Paginas.



NOTA

El Directorio de paginas, puede referenciar 1024 Tablas de Paginas.
Luego cada una de esas 1024 Tablas de Paginas puede referenciar 1024 Paginas, en la memoria física
Los elementos de estas tablas tienen el mismo formato:

31 … 12	11	9	8	7	6	5	4	3	2	1	0
PAGEFRAME ADDRESS	AVAIL	0	0	D	A	0	0	U/S	R/W	P

P - Present
R/W - read/Write
U/S - User / supervisor
D - Dirty
AVAIL - Available for systems programmer use.

El Bit P o present Bit, indica cuando tiene valor 1, que esa tabla se puede utilizar para obtener una dirección de memoria. Si tiene valor 0, no se puede utilizar.
Al encontrar un Bit P=0 =>
UP lanza Exception (Page Fault) =>
llevo a memoria pagina faltante (que seguramente estaba en disco)
El Bit U/S divide las paginas en 2 niveles de privilegios (usuario y supervisor).

Scanned by CamScanner

Si no se respetan los accesos se produce una excepción si el UP esta ejecutando en los niveles 0, 1 y 2 es supervisor y en nivel 3 es usuario.

Protección combinada

Cuando la paginación esta habilitada el UP primero chequea los permisos en la segmentación y luego en la paginación.
Por ejemplo, se puede definir un segmento de datos que esta compuesto por paginas, donde algunas son de escritura y otras de solo lectura.
Debido a este manejo de memoria algunos S.O como Unix eligen usar el modelo Flat de memoria.

Modelo de memoria Flat =

Unix declara 2 segmentos de codigo y de datos que referencian a toda la memoria.
Esto le permite manejo más simple y portabilidad de UP sin unidad de segmentación.

Linux = - Utilizaba TSS hasta 2.2
- /usr/src/linux/Arch
- Utiliza Paginación de Hardware

En UP de 64 Bits agrega un nivel extra de paginas (en el medio de los de Hardware) llamado "middle directory

Paging y swapping =

Cuando se necesita una página en memoria y la memoria está completa, Linux elige una página que no ha sido accedida últimamente y realiza un "Page Out", que consiste en guardar esa página en disco.
Generalmente, en una zona especial destinada para ello, puede ser una partición o un archivo en el filesystem luego seteo en la página "presente" P=0 y guardo su dirección en el disco a este concepto se lo denomina "Paging".
El concepto de swapping se refiere a guardar en disco TODAS! las páginas de un proceso.
En Linux existe un Thread llamado "kswapd" que se encargo de hacer este Trabajo.

NOTA

Scanned by CamScanner