

COMPONENTES LÓGICOS-DIGITALES

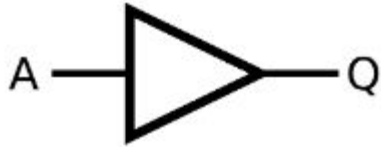
UNIDAD II

A dark blue diagonal gradient bar that starts from the bottom left corner and extends towards the top right corner, covering the lower half of the slide.

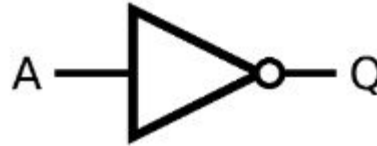
46 BASIC GATES



46.1 Basic Gates



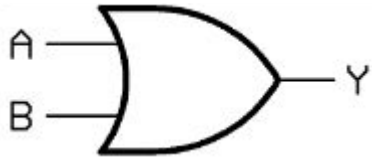
BUFFER



NOT



AND



OR

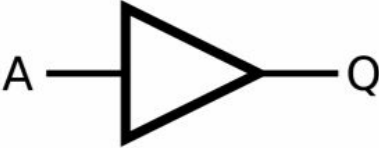
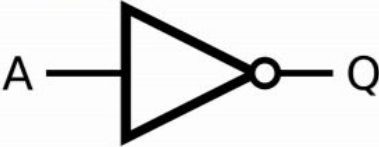

Existen 5 compuertas lógicas básicas que hacen operaciones lógicas en electrónica digital.

Cada una tiene un símbolo, una expresión matemática y una tabla de verdad.



XOR

46.2 Five Basic Logic Gates

Digital gates	Symbol	Logic Operation	Mathematic Expression
BUFFER	 <p>Figure 194 Cổng Tiếp (Buffer Gate)</p>	$Y = \text{BUFFER } A$	$Y = A$
NOT	 <p>Figure 195</p>	$Y = \text{NOT } A$	$Y = \bar{A}$
AND	 <p>Figure 196</p>	$Y = A \text{ AND } B$	$Y = A \cdot B$

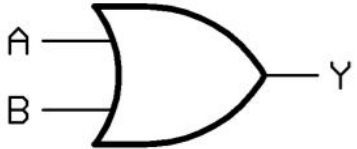

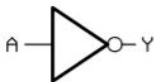
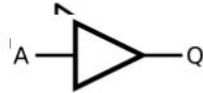
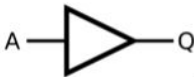
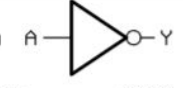


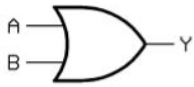


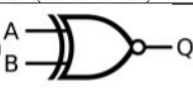
OR	 <p>Figure 197</p>	$Y = A \text{ OR } B$	$Y = A + B$
XOR	 <p>Figure 198</p>	$Y = A \text{ XOR } B$	$Y = A \oplus B$

Tabla de verdad

A	B	$Q = A$	$Q = \text{NOT } A$	$Q = A \text{ AND } B$	$Q = A \text{ OR } B$	$Q = A \text{ XOR } B$
0	0	0	1	0	0	0
0	1	0	1	0	1	1
1	0	1	0	0	1	1
1	1	1	0	1	1	0

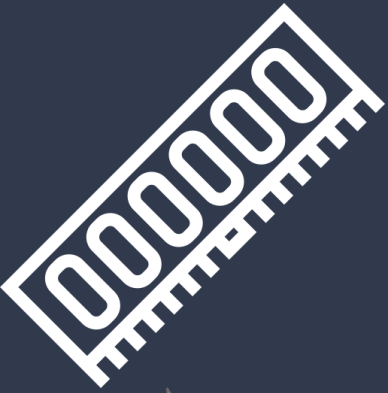
46.3 Complement of Basic Logic gates

Basic Gates	Combination Gates	Symbol	Mathematical Expression
BUFFER	 Figure 199	 Figure 200, Công NOT (NOT Gate)	$Q = \text{is NOT NOT } A$ $Y = A$
NOT	 Figure 202	 Figure 203, Công NOT (NOT Gate)	$Y = \text{is NOT } A$
NAND	 Figure 205	 Figure 206, Công NOT (NOT Gate)	$Q = \text{NOT } A \text{ AND } B$
NOR	 Figure 208	 Figure 209, Công NOT (NOT Gate)	$Y = \text{NOT } A \text{ OR } B$
XNOR	 Figure 211	 Figure 212, Công NOT (NOT Gate)	$Q = \text{NOT } A \text{ XOR } B$

46.3 Complement of Basic Logic gates

A	B	$Q = A$	$Q = \text{NOT } A$	$Q = A \text{ NAND } B$	$Q = A \text{ NOR } B$	$Q = A \text{ XNOR } B$
0	0	0	1	1	1	1
0	1	0	1	1	0	0
1	0	1	0	1	0	0
1	1	1	0	0	0	1

1.3 REGISTERS



?

Instrucciones,
operandos,
resultados



- Son más caros
- Instrucciones con código más largo
- Circuitos más complejos

Patrones de programación
+
Reutilización de datos
=
Localidad de referencia

1.3.1 General Purpose Registers

Nombre	Alias	Descripción
r0	rax	Tipo de "acumulador", utilizado en instrucciones aritméticas. Por ejemplo, una instrucción div se usa para dividir dos enteros. Acepta un operando y usa rax implícitamente como el segundo. Después de ejecutar div rcx, un gran número de 128 bits de ancho, almacenado en partes en dos registros rdx y rax se divide por rcx y el resultado se almacena nuevamente en rax.
r3	rbx	Base de registro. Se utilizó para el direccionamiento base en los primeros modelos de procesador.
r1	rcx	Se usa para ciclos (por ejemplo, en bucle).
r2	rdx	Almacena datos durante las operaciones de entrada / salida.
r4	rsp	Almacena la dirección del elemento superior en la pila de hardware. Consulte la sección 1.5 "Pila de hardware".

1.3.1 General Purpose Registers

Nombre	Alias	Descripción
r5	rbp	Base del marco de la pila. Consulte la sección 14.1.2 "Convención de llamada".
r6	rsi	Índice de origen en comandos de manipulación de cadenas (como movsd)
r7	rdi	Índice de destino en comandos de manipulación de cadenas (como movsd)
r8		
r9...r15	no	Apareció más tarde. Se usa principalmente para almacenar variables temporales (pero a veces se usa implícitamente, como r10, que guarda los indicadores de la CPU cuando se ejecuta la instrucción syscall. Consulte el Capítulo 6 "Interrupciones y llamadas al sistema").

1.3.2 Other Registers



Other Registers tienen un significado especial. Algunos tienen importancia en todo el sistema y por ello no pueden ser modificados excepto por el sistema operativo.

Un programador tiene acceso al registro rip. Es un registro de 64 bits, que siempre almacena una dirección de siguiente instrucción a ejecutar.

Las instrucciones de ramificación (por ejemplo, jmp) de hecho lo están modificando. Entonces, cada vez que la instrucción se está ejecutando, rip almacena la dirección de la siguiente instrucción que se ejecutará.

Otro registro accesible se llama rflags. Almacena marcas, que reflejan el estado actual del programa.

Por ejemplo, cuál fue el resultado de la última instrucción aritmética: si fue negativa, si ocurrió un overflow. Sus partes más pequeñas se llaman eflags (32 bits) y flags (16 bits).

- Además de estos registros centrales, también hay registros utilizados por instrucciones que trabajan con float o instrucciones paralelas especiales capaces de realizar acciones similares en múltiples pares de operandos al mismo tiempo.
- Estas instrucciones se usan a menudo con fines multimedia (ayudan a acelerar algoritmos de decodificación multimedia).
- Los registros correspondientes son de 128 bits de ancho y se denominan xmm0 - xmm15.

1.3.3 System Registers

Algunos registros están diseñados específicamente para uso del S.O. Estos registros no guardan ningún valor computacional, estos registros guardan información requerida para el sistema.

Es muy importante que estos registros no los pueda modificar ninguna aplicación.

Registros:

- cr0,cr4= Guardan “flags” relacionado a diferentes modos del procesador y memoria virtual
- cr2,cr3= Usados para soportar memoria virtual
- cr8(trp)= Realiza una afinación del mecanismo de interruptores

Registros:

- efer= Registro de “flags” que controlan el modo del procesador
- idtr= Almacena la dirección de memoria del interruptor descripción tabla
- gdtr y ldtr= Almacena la dirección de memoria de las tablas descriptivas
- cs,ds,ss,es,gs,fs= “Segment Register” se implementan en el modo privilegiado del procesador.

Descomposición de los registros rsi y rdi

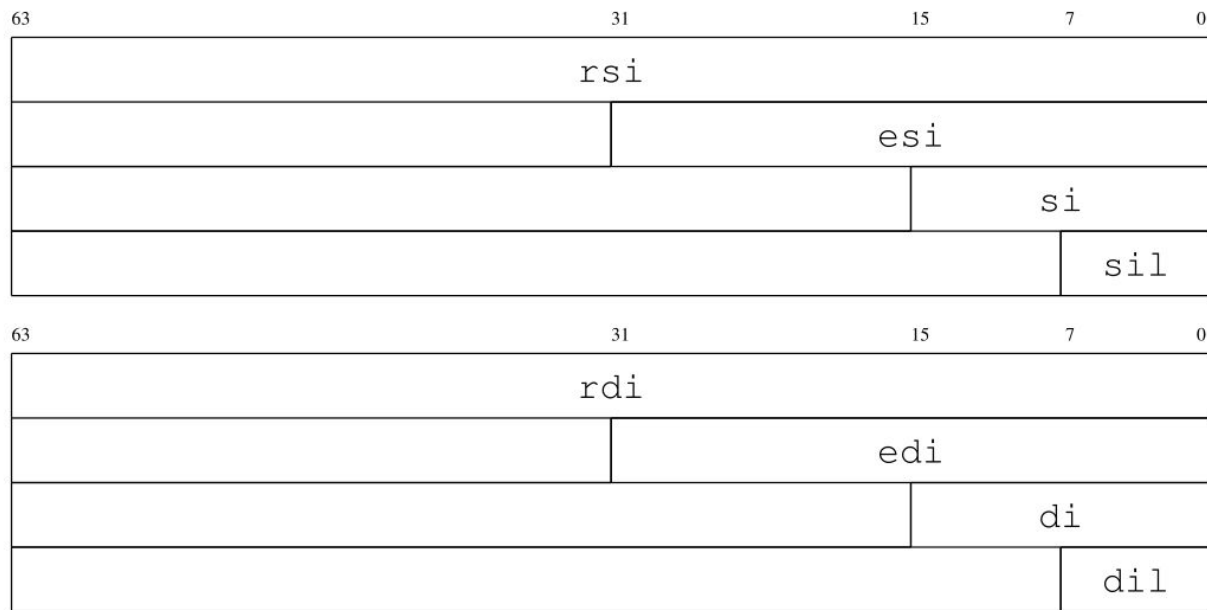


Figure 1-5. *rsi and rdi decomposition*

Descomposición de los registros rsp y rbp

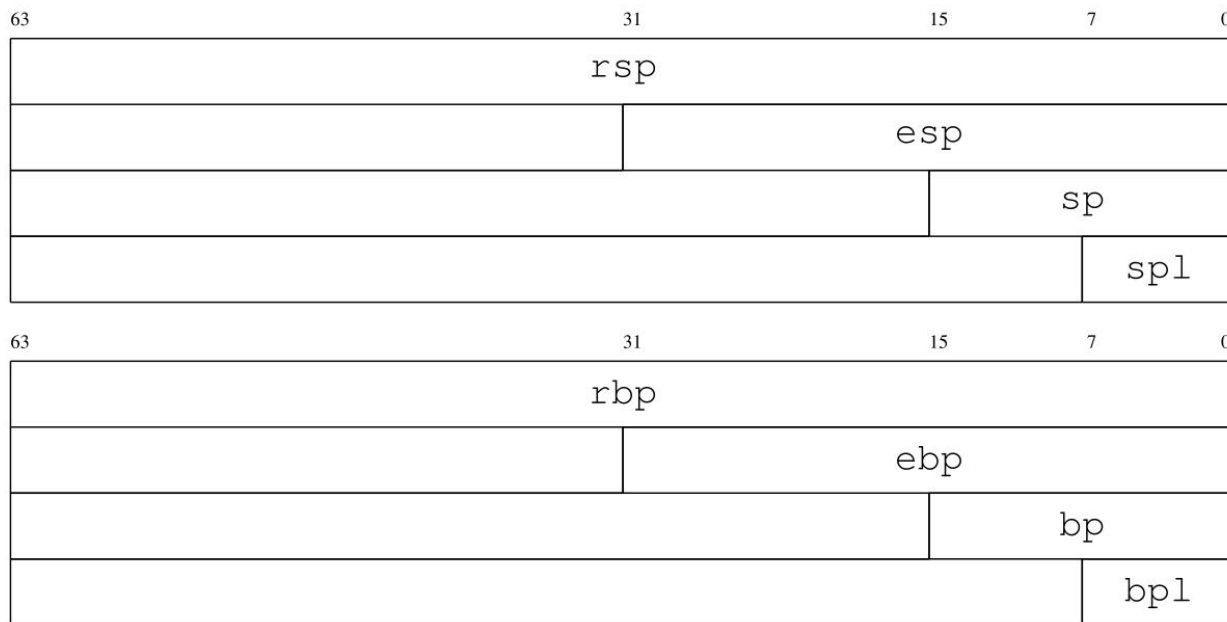


Figure 1-6. *rsp and rbp decomposition*

4 VIRTUAL MEMORY

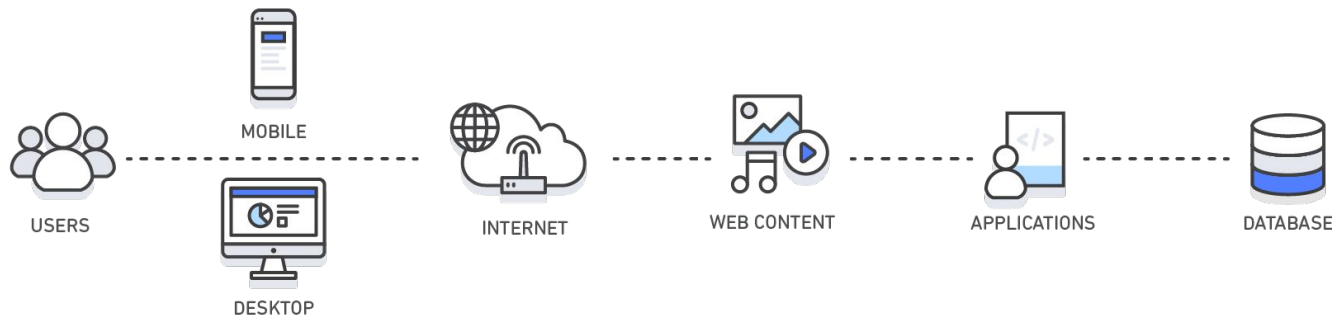


4.1 Caching

Internet es un gran almacenamiento de datos. Puede acceder a cualquier parte, pero el retraso después de realizar una consulta puede ser significativo y para suavizar su experiencia de navegación el navegador web almacena en caché las páginas web y sus elementos (imágenes, hojas de estilo, etc.).

El navegador guarda los datos en el disco duro o en la RAM (memoria de acceso aleatorio) para dar mucho acceso más rápido a una copia local.

El mecanismo de memoria virtual nos permite, entre otras cosas, utilizar la memoria física como caché para fragmentos de código de programa y datos.



4.2 Motivation

En un sistema multitarea, se requiere un marco que soporte un ejecución de múltiples programas en paralelo. En este caso, un sistema operativo necesita algún tipo de gestión de la memoria para hacer frente a los siguientes desafíos:

- Ejecutar programas de tamaño arbitrario (incluso mayor que el tamaño de la memoria física).
- Tener varios programas en la memoria al mismo tiempo.
- Almacenar programas en cualquier lugar de la memoria física

4.2 Motivation

- Liberar a los programadores de las tareas de administración de memoria tanto como sea posible.
- Uso efectivo de datos y códigos compartidos.

4.3 Address Spaces

El espacio de direcciones es un rango de direcciones. Vemos dos tipos de espacios de direcciones:

- Dirección física, que se utiliza para acceder a los bytes en el hardware real. Hay una cierta capacidad de memoria que un procesador no puede exceder.

Por ejemplo, un sistema de 32 bits no puede abordar más de 4 GB de memoria por proceso, porque 2^{32} direcciones diferentes corresponden aproximadamente a 4 GB de memoria direccionada.

4.3 Address Spaces

- Dirección lógica es la dirección tal como la ve una aplicación.

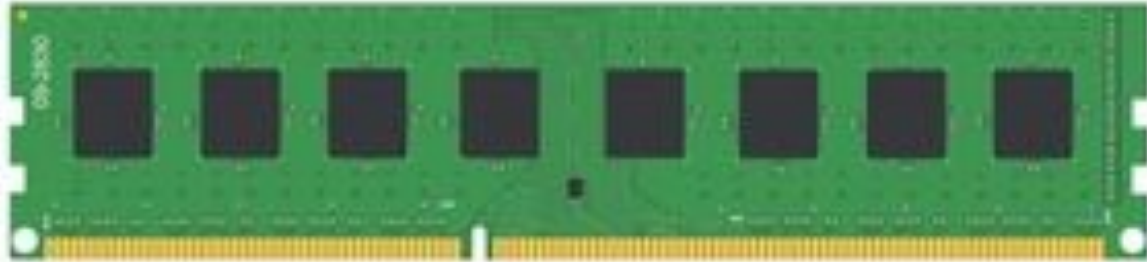
En la instrucción `mov rax, [0x10bfd]` hay una dirección lógica: `0x10bfd`.

Un programador cree que él es el único usuario de la memoria, pero nunca ve datos o instrucciones de otros programas que se ejecutan en paralelo.

La traducción entre estos dos tipos de direcciones es realizada por una entidad de hardware llamada Memory Management Unit (MMU) con ayuda de múltiples tablas de traducción, que residen en la memoria.

4.4 Features

- Podemos comunicarnos con dispositivos externos mediante la entrada/ Salida de memoria asignada
- Algunas páginas se pueden compartir entre varios procesos.
- La mayoría de las direcciones están prohibidas: su valor no está definido.
- Algunas páginas corresponden a archivos, tomados del almacenamiento



4.4 Features

Si la memoria física libre se termina, algunas páginas se moverán al almacenamiento externo, ó directamente será descartado.

En Windows el archivo llamado “**PageFile.sys**” en “*nix” systems la partición es usualmente guardada en el disco.

La elección de paginas para ser cambiada es descrita por las **estrategias de reemplazo** tales como:

- El menos usado frecuentemente
- El último usado
- Aleatorio

4.6 Efficiency

Efficiency V

Capacidad para realizar o cumplir adecuadamente una función.

1. Gracias a la localidad, la necesidad de cargar páginas adicionales ocurre raramente.

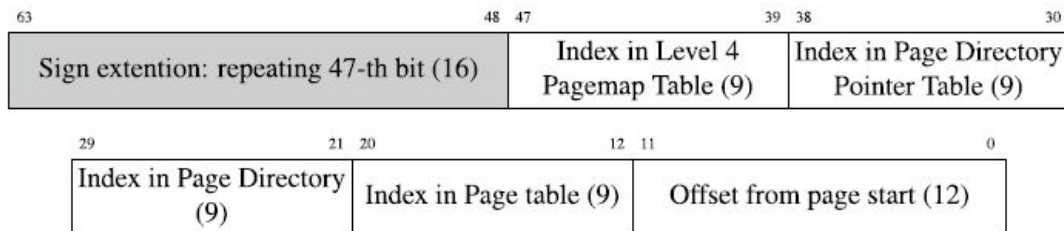
TLB (Traducción
Lookaside Buffer)

2. Está claro que la eficiencia no podría lograrse sin la ayuda de especialistas hardware.



4.7.1 Virtual Address Structure

Cada dirección virtual de 64 bits (por ejemplo, las que estamos usando en nuestros programas) consta de varios campos,



La dirección en sí tiene, de hecho, solo 48 bits de ancho; está extendido por un signo a una dirección canónica de 64 bits.

La característica es que sus 17 bits izquierdos son iguales. Si la condición no se cumple, la dirección se rechaza inmediatamente cuando se usa.

Luego, 48 bits de dirección virtual se transforman en 52 bits de dirección física

4.7.1 Virtual Address Structure

■■ Error de bus: Cuando ocasionalmente use una dirección no canónica, verá otro mensaje de error:
Error de bus

El espacio de direcciones físicas se divide en ranuras para ser llenado con páginas virtuales. Estas ranuras se llaman página de marcos.

No hay espacios entre ellos, por lo que siempre comienzan desde una dirección que termina con 12 bits cero. Los 12 bits menos significativos de dirección virtual y de página física corresponden al desplazamiento de la dirección página interior, por lo que son iguales.

Las otras cuatro partes de la dirección virtual representan índices en las tablas de traducción. Cada mesa ocupa exactamente 4KB para llenar una página de memoria completa.

Cada registro tiene 64 bits de ancho; almacena una parte de la siguiente tabla dirección de inicio, así como algunas banderas de servicio.

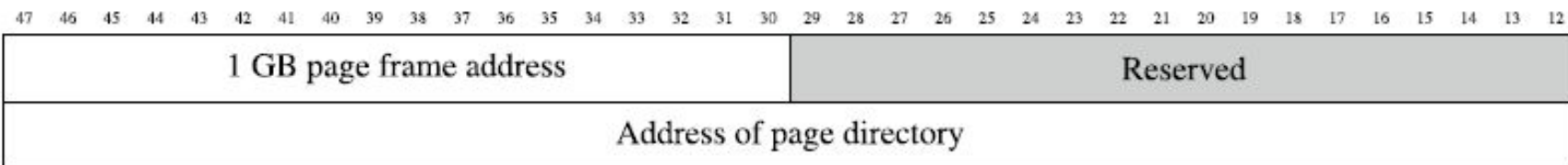
4.7.2 Address Translation in Depth

4.7.3 Page Sizes

- El tamaño de la página puede ajustarse para que sea 4KB, 2MB o 1GB.
- Dependiendo de la estructura, esta jerarquía puede reducirse a un mínimo de dos niveles.
- En este caso, PDP funcionará como una tabla de páginas y almacenará parte de un marco de 1GB.

4.7.3 Page Sizes

PDP partial format: either maps 1GB pages or transfers to the next level



PD partial format: either maps 2MB pages or transfers to the next level

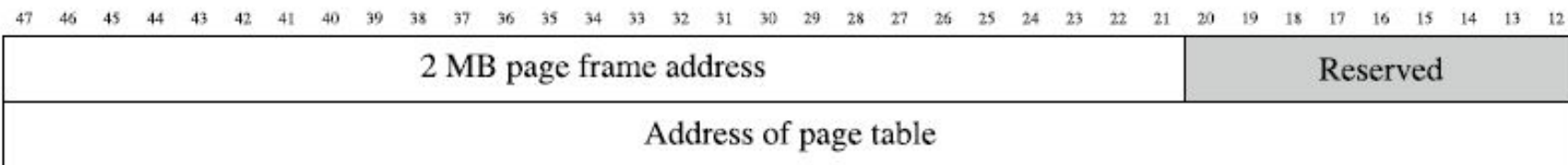


Figure 4-4. Page Directory Pointer table and Page Directory table entry format

4.8 Memory Mapping

Mapping significa "proyección", haciendo correspondencia entre entidades (archivos, dispositivos, memoria física) y regiones de memoria virtual.

Cuando el cargador llena el espacio de direcciones del proceso, cuando un proceso solicita páginas del sistema operativo o cuando el sistema operativo proyecta archivos desde un disco en los espacios de direcciones de los procesos, estos son ejemplos de mapeo de memoria.

Los mapas de memoria suelen ser creados usualmente por el firmware para dar información al núcleo del sistema operativo sobre cómo está distribuida la memoria.

REGISTRO	VALOR	SIGNIFICADO
rax	9	Identificador de llamada del sistema
rdi	addr	comienza desde esta dirección específica.
rsi	len	Tamaño de la región.
rdx	prot	Banderas de protección (r, w, x.)
r10	flags	Indicadores de utilidad.
r8	fd	Descriptor opcional de un archivo mapeado.
r9	offset	Desplazamiento en archivo.

4.9 Example: Mapping File into Memory

Necesitamos otra llamada al sistema, en este caso, abierto. Se utiliza para abrir un archivo por nombre y para adquirir su descriptor.

Registro	Valor	Significado
rax	2	Identificador de llamada al sistema.
rdi	File name	Apuntador a una cadena terminada en string. name.holding file.
rsi	Flags	Una combinación de marcas de permiso (solo lectura, solo escritura o ambas).
rdx	mode	Si se llama al sistema abierto para crear un archivo, tendrá los permisos del sistema de archivos.

El archivo de mapeo en la memoria se realiza en tres simples pasos:

- Abrir el archivo usando la llamada al sistema abierto. rax contendrá el descriptor de archivo.
- Llame a mmap con argumentos relevantes. Uno de ellos será el descriptor de archivo, adquirido en el paso 1.
- Utilice la rutina print_string que hemos creado en el Capítulo 2. Por razones de brevedad, omitimos el cierre de archivos y las verificaciones de errores.

4.9.1 Mnemonic Names for Constants

$$\int u \cdot dv = uv - \int v du$$

Uniforme

de

vestido

integrar

menos

valor

una



```
#include <stdio.h>
int main(){
```

```
int a,b,c;
```

```
scanf("%d %d ",&a,&b);
```

```
    c=a+d;
```

```
    printf("%d",c);
```

```
return 0;
}
```

```
#include <stdio.h>
int main(){
```

```
int sum1,sum2,res1;
```

```
scanf("%d %d ",&sum1,&sum2);
```

```
    res1=sum1+sum2;
```

```
    printf("%d",res1);
```

```
return 0;
}
```

4.9.2 Complete Example

No vino Rocha :v

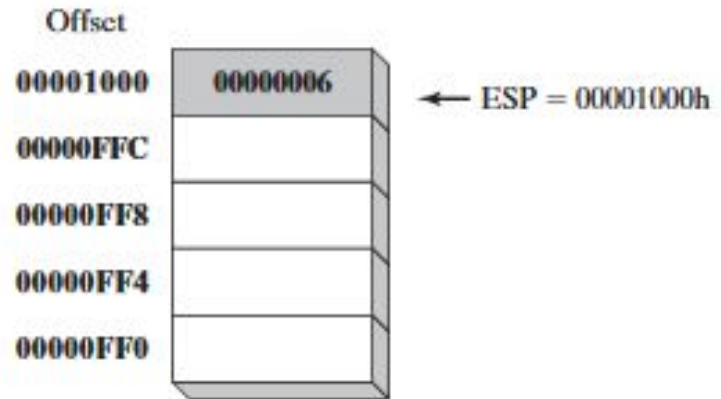
5 PROCEDURES



5.1.1 Runtime Stack (32-bit mode)

La pila de tiempo de ejecución es una matriz de memoria administrada directamente por la CPU, utilizando el registro ESP (puntero de pila extendido), conocido como el registro de puntero de pila.

Una pila crea un área de guardado temporal conveniente para los registros cuando se usan para más de un propósito.



5.1.2 PUSH and POP Instructions

PUSH

La instrucción PUSH es una instrucción que disminuye el ESP y copia el operando fuente en una pila.

`PUSH reg/mem16`

`PUSH reg/mem32`

`PUSH imm32`

POP

La introducción POP primero copia el contenido de las pilas que están apuntadas por el ESP en un operando de 16/32 bits y después incrementa el ESP.

`POP reg/mem16`

`POP reg/mem32`

PUSHD :

“pushes” el las “32-bit EFLAGS registers” en una pila.

POPFD :

“pops” la pila en las EFLAGS.

```
pushfd                ; save the flags
;
; any sequence of statements here...
;
popfd                 ; restore the flags
```



MOV



PUSHFD

PUSHFD Y POPFD

PUSHAD



EAX ECX EDX EBX ESP EBP ESI EDI



POPAD

PUSHA



AX CX DX BX SP BP SI DI



POPA

5.2.1 PROC Directive

Las directivas PROC y ENDP definen el inicio y el final de un procedimiento, que es un conjunto de sentencias a las que se puede acceder directamente llamando al procedimiento. Un procedimiento posee un nombre, el cual es utilizado para ser llamado.

- El parámetro 'atributo' es opcional, y puede ser NEAR ó FAR (por defecto es NEAR). Si el procedimiento es el principal de un programa, ha de utilizarse el atributo FAR.
- Para que un procedimiento en un módulo sea accesible desde otros módulos se ha de especificar la directiva PUBLIC.

5.2.1 PROC Directive

EJEMPLO:

```
PUBLIC  PROC1          ; Se define como público
PROC1  PROC  FAR       ; comienzo del procedimiento (lejano)
      (instrucciones) ; Instrucciones del procedimiento
      RET             ; Instrucción para retornar
PROC1  ENDP           ; Final del procedimiento
```

NOTA:

Para llamar a un procedimiento se utiliza la instrucción

CALL: CALL nombre_procedimiento

5.2.2 CALL and RET Instructions

La instrucción CALL llama a un procedimiento directamente del procesador para que comience la ejecución en una nueva ubicación de memoria

El procedimiento utiliza una instrucción RET para volver al punto del programa justo antes de haberse hecho el CALL.

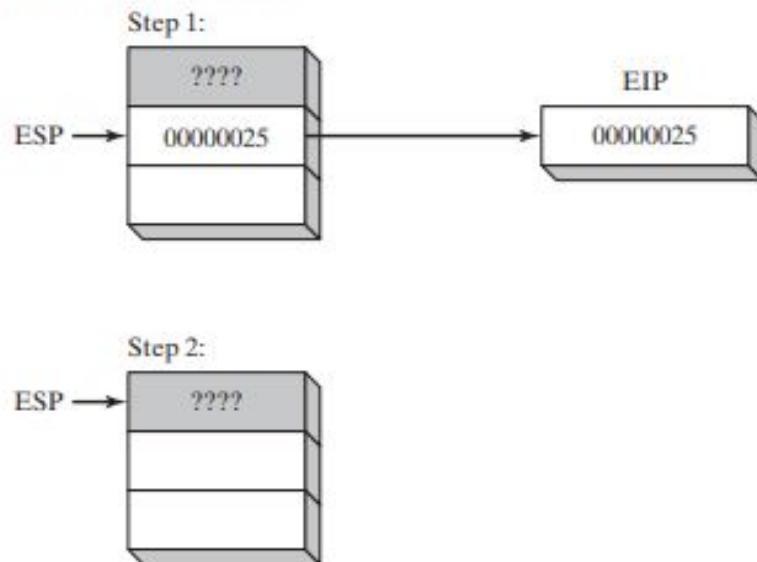
La instrucción CALL empuja su dirección de retorno en la pila y copia la dirección del procedimiento CALL en el instruction pointer.

Cuando el procedimiento está listo para regresar, la instrucción RET carga la dirección de retorno de la pila al instruction pointer.

FIGURE 5-6 Executing a CALL instruction.

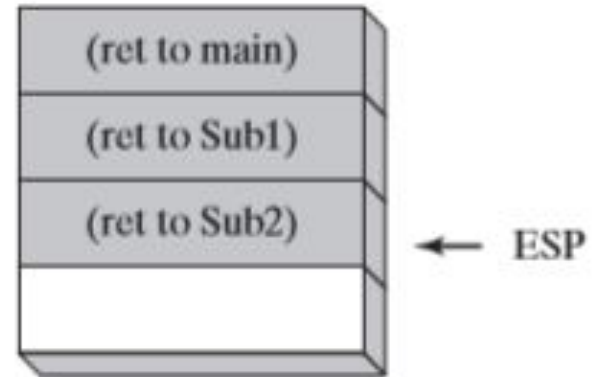


FIGURE 5-7 Executing the RET instruction.



5.2.3 Nested Procedure Calls

Nested Procedure calls o llamadas a procedimientos anidadas ocurren cuando otro procedimiento llama (CALL) a otro antes de que éste regrese (RET).



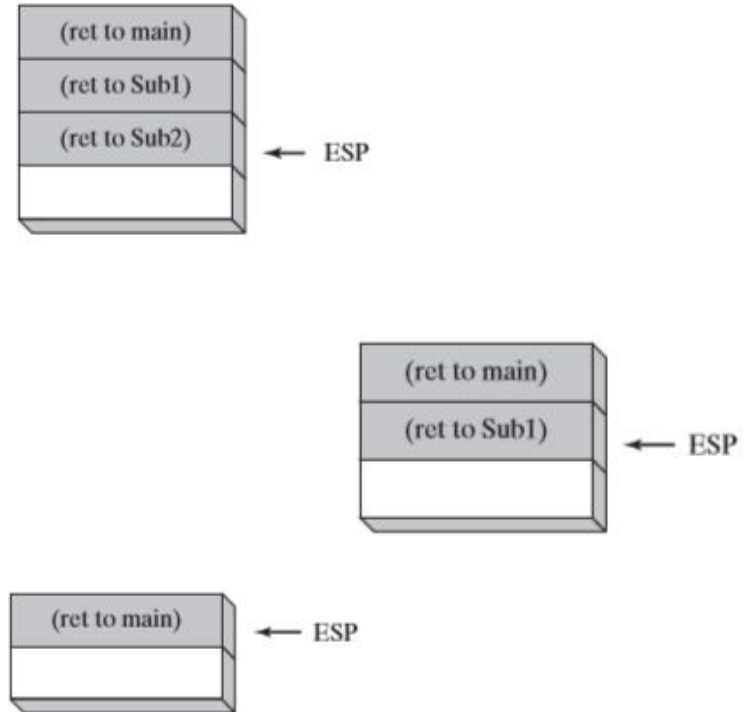
5.2.3 Nested Procedure Calls

Ejemplo:

Supongamos que main llama a una función llamada Sub 1 y mientras esta se ejecuta llama a un procedimiento Sub 2 y a su vez está llama a Sub 3.

Cuando Sub 3 termina de ejecutarse y llega a la instrucción RET; esto hace que saque el valor del stack pointer register hacia el instruction pointer.

Después el stack pointer register apunta a la siguiente entrada más alta de la pila (Sub2) y se repetirá el mismo proceso hasta regresar a main.



5.2.4 Passing Register Arguments to Procedures

Un mejor enfoque es pasar el desplazamiento de una matriz al procedimiento y pasa un número entero que especifica el número de elementos de la matriz. Llamamos a estos argumentos (o parámetros de entrada). En lenguaje ensamblador, es común que pasar argumentos dentro de registros de propósito general.

En la sección anterior creamos un procedimiento simple llamado **SumOf** que agregó los enteros en los registros EAX, EBX y ECX. En **main** , antes de llamar a **SumOf** , asignamos valores a EAX, EBX y ECX

```
.data
theSum  DWORD  ?
.code
main PROC
    mov  eax,10000h      ; argument
    mov  ebx,20000h      ; argument
    mov  ecx,30000h      ; argument
    call Sumof           ; EAX = (EAX + EBX + ECX)
    mov  theSum,eax      ; save the sum
```

Después de la declaración **call**, tenemos la opción de copiar la suma en EAX a una variable

5.2.5 Example: Summing an Integer Array

```
;-----  
; ArraySum  
;  
; Calculates the sum of an array of 32-bit integers.  
; Receives: ESI = the array offset  
;           ECX = number of elements in the array  
; Returns:  EAX = sum of the array elements  
;-----  
ArraySum PROC  
    push esi                ; save ESI, ECX  
    push ecx  
    mov  eax,0              ; set the sum to zero  
L1:  add  eax,[esi]          ; add each integer to sum  
    add  esi,TYPE DWORD     ; point to next integer  
    loop L1                 ; repeat for array size  
    pop  ecx                ; restore ECX, ESI  
    pop  esi  
    ret                    ; sum is in EAX  
ArraySum ENDP
```


5.2.6 Saving and Restoring Registers

Arrays, ECX Y ESI son utilizados para aquellos procedimientos en el que se modifican los registros. Aunque siempre debemos guardar nuestros registros para que no se sobrescriben.

Operadores como USES Y PROC Permiten enumerar los registros que han sido modificados. y para ello se realiza dos cosas:

1. Generar PUSH que es el que guarda los registros en la pila
2. Generar instrucciones POP que restauran los valores del registro al final de procedimiento.

Sin embargo hay una regla de guardar registros y sucede cuando en un procedimiento se devuelven valores en un registro.

EJEMPLO:

Name	Operation	Operand
BEGIN	SAVE	(14,12)
	USING	BEGIN,15
	:	
	ST	13,SAVEBLK+4
	LA	13,SAVEBLK
	:	
program	function	source statements
	:	
	L	13,SAVEBLK+4
	RETURN	(14,12)
SAVEBLK	DC	18F'0'
	:	
	END	

5.3.1 Background Information

Supongamos que un programa muestra una cadena en la ventana de la consola llamando a un procedimiento (o función) llamada `WriteString`. La fuente del programa debe contener una directiva `PROTO` que identifique el procedimiento `WriteString`:

`WriteString proto`

A continuación, una instrucción `CALL` ejecuta `WriteString`:

`call WriteString`

5.3.1 Background Information

Opciones de comando de vinculador (Linker Command Options)

La utilidad de enlace (linker utility) combina el archivo de objetos de un programa con uno o más archivos de objetos y bibliotecas de enlaces. El siguiente comando, por ejemplo, vincula hello.obj a las bibliotecas irvine32.lib y kernel32.lib:

```
enlace hello.obj irvine32.lib kernel32.lib
```

5.4.1 Motivation for Creating the Library

No existe una biblioteca estándar aprobada por Microsoft para la programación en lenguaje ensamblador.

Los programadores comenzaron a escribir lenguaje ensamblador para procesadores x86 a principios de la década de 1980, MSDOS era el sistema operativo más utilizado. Estos programas de 16 bits pudieron llamar a las funciones de MSDOS (conocidas como servicios INT 21h) para realizar una entrada / salida simple.

5.4.2 Overview

- Ventana de Comandos:
 - Es una ventana solo de texto.
 - Es una forma de realizar acciones avanzadas mediante comandos de texto.
 - Usando el comando “mode” se puede modificar el número de líneas y columnas.
mode con cols=40 lines=30
- File handle:
 - Es un entero de 32 bits utilizado por el sistema operativo Windows para identificar un archivo que está abierto actualmente.
 - Cuando un programa llama a un servicio de Windows para abrir o crear un archivo, el sistema operativo crea un nuevo identificador de archivo y lo pone a disposición del programa.
 - Cada vez que llame a un método de servicio del sistema operativo para leer o escribir en el archivo, debe pasar el mismo identificador de archivo como parámetro al método de servicio.
 - Si un programa llama a procedimientos en la biblioteca Irvine32, siempre debe insertar valores de 32 bits en la pila de tiempo de ejecución; si no lo hace, las funciones de la consola Win32 llamadas por la biblioteca no funcionarán correctamente.

5.4.3 Individual Procedure Descriptions

Uso de los procedimientos en la biblioteca Irvine32, se omiten los más avanzados. se llaman con **call**.

CloseFile: cierra un archivo previamente creado o abierto. El retorno al Acumulador Extendido **NO** será cero.

Clscr: limpia la consola. Se llama al inicio y al final de un programa.

CreateOutputFile: Crea un archivo en disco y lo abre para escritura.

Crlf: (por CarriageReturn LineFeed) Esto avanza el cursor al principio de la siguiente línea en la consola.

Delay: Pausa el programa por el número indicado de milisegundos.

DumpMem: Escribe un intervalo de memoria en la ventana de consola en hexadecimal.

DumpRegs: muestra un volcado hexadecimal de los registros EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EIP, y EFL.

GetCommandTail: copia la línea de comandos del programa a una cadena terminada en “\0”, si no hubo comandos se activa una bandera de carry, o se desactiva por lo contrario.

GetMaxXY: recupera el valor del tamaño máximo del buffer de la ventana de consola, si es mayor que el area visible, entonces, una barra desplazadora aparecerá.

GetMseconds: recupera el número de milisegundos desde la medianoche hasta el tiempo actual.

GetTextColor: Regresa un color a 4 bits más alto que AL y 4 bits más bajos que el fondo de la consola

Gotoxy: ubicará al cursor en las coordenadas designadas que se pongan en DH (y) y DL (x).

5.4.3 Individual Procedure Descriptions

MsgBox: despliega un cuadro de diálogo con contenido opcional.

MsgBoxAsk: El mismo despliegue que el anterior pero con botones de “Sí” y “No”.

OpenInputFile: Abre un archivo existente para usarlo como entrada.

ParseDecimal32: Convierte una cadena de decimal entero sin signo a un binario de 32-bit.

ParseInteger32: Convierte una cadena de decimal entero **CON** signo a un binario de 32-bit.

Random32: Genera un entero aleatorio de 32-bit en el Registro Extendido.

Randomize: inicializa el valor de la semilla de **Random32** y **RandomRange**.

RandomRange: produce un entero aleatorio en el rango de 0 a $n - 1$, n es un parámetro en EAX.

ReadChar: lee un caracter del teclado y regresa el caracter en el registro AL. No hace echo.

ReadDec: lee un entero decimal sin signo de 32-bit desde el teclado y regresa el valor en EAX. No “ ” ni Afa.

ReadFromFile: Lee un archivo del disco hacia un buffer de memoria.

ReadHex: Lee un entero hexadecimal de 32-bit y regresa el valor binario que corresponda en EAX.

ReadInt: Lee un entero con signo de 32-bit desde el teclado y regresa el valor en EAX. Acepta el “+” o “-”

ReadKey: Inspecciona el buffer del teclado: si NO hubo presión de tecla la Zero flag se activa, y viceversa pero adicionalmente al registro AL se le asigna un cero o un código ASCII. si AL ya tenía cero, puede que el usuario haya presionado una tecla especial.

5.4.3 Individual Procedure Descriptions

ReadString:

SetTextColor:

Str_length

WaitMsg

WriteBin

WriteBinB

WriteChar

WriteDec

WriteHex

WriteHexB

WriteInt

WriteString

WriteToFile

WriteWindowsMsg

5.4.4 Library Test Programs

5.5.1 The Irvine64 Library

- **crlf**: *escribe una secuencia end-of-line en la consola*
- **random64**: *genera un INT pseudo randomizado de 64 bits*
- **randomize**: *planta un generador de número aleatorios con un valor único*
- **readint64**: *lee un INT de 64 bits del teclado*
- **readstring**: *lee una cadena de teclado*
- **str_compare**: *compara 2 cadenas de la misma forma que CMP*
- **stt_copy**: *copia una cadena a un destino específico*

5.5.2 Calling 64-Bit Subroutines

Si desea llamar a una subrutina que ha creado, o una subrutina en la biblioteca Irvine64, todo lo que debe hacer es colocar los parámetros de entrada en los registros y ejecutar la instrucción CALL.

por ejemplo:

```
mov rax,12345678h  
call WriteHex64
```

Hay otra pequeña cosa que debes hacer, que es usar la directiva PROTO en la parte superior de su programa para identificar cada procedimiento que planea llamar fuera de su propio programa:

ExitProcess PROTO ; located in the Windows API

WriteHex64 PROTO ; located in the Irvine64 library

5.5.3 The calling convention

Microsoft sigue un esquema consistente para pasar parámetros y procedimientos de llamada en 64 bits programas conocidos como la Convención de llamadas de Microsoft x64. Esta convención es utilizada por C / C ++, así como por la interfaz de programación de aplicaciones (API) de Windows. Estas son algunas de las características básicas de este convención de llamadas:

1. La instrucción CALL resta 8 del registro RSP (puntero de pila), ya que las direcciones son 64 bits de largo.
2. Los primeros cuatro parámetros pasados a un procedimiento se colocan en RCX, RDX, R8 y R9, registros, en ese orden. Si solo se pasa un parámetro, se colocará en RCX. Si hay un segundo parámetro, se colocará en RDX, y así sucesivamente. Se introducen parámetros adicionales la pila, en orden de izquierda a derecha.
3. Es responsabilidad de la persona que llama asignar al menos 32 bytes de espacio en la sombra en el tiempo de ejecución stack, por lo que los procedimientos llamados pueden guardar opcionalmente los parámetros de registro en esta área.
4. Al llamar a una subrutina, el puntero de la pila (RSP) debe estar alineado en un límite de 16 bytes (un múltiplo de 16). La instrucción CALL empuja una dirección de retorno de 8 bytes en la pila, por lo que el programa de llamada debe restar 8 del puntero de la pila, además de los 32 que ya resta para el espacio de sombra

5.5.4 Sample Program that Calls a Procedure

Se crea un pequeño programa que puede llamar una subrutina llamada **ADDFOUR**. Dicha subrutina añade valores a los 4 parámetros de registro (RCX, RDX, R8, R9) y guarda la suma realizada en RAX el cual solo acepta valores enteros.

Podríamos decir que esta subrutina es una función, ya que recibe 4 inputs y lanza solo 1 output.

En vez de llamar ExitProcess para terminar el programa, podemos usar las siguientes líneas de código. Sin embargo, requeriría que reiniciemos el puntero usado a su valor inicial.

```
21:    add    rsp,28          ; restore the stack pointer
22:    mov     ecx,0          ; process return code
23:    ret
```

```
1: ; Calling a subroutine in 64-bit mode    (CallProc_64.asm)
2: ; Chapter 5 example
3:
4: ExitProcess PROTO
5: WriteInt64 PROTO          ; Irvine64 library
6: Crlf PROTO                ; Irvine64 library
7:
8: .code
9: main PROC
10:    sub     rsp,8           ; align the stack pointer
11:    sub     rsp,20h         ; reserve 32 bytes for shadow params
12:
13:    mov     rcx,1           ; pass four parameters, in order
14:    mov     rdx,2
15:    mov     r8,3
16:    mov     r9,4
17:    call    AddFour         ; look for return value in RAX
18:    call    WriteInt64      ; display the number
19:    call    Crlf            ; output a CR/LF
20:
21:    mov     ecx,0
22:    call    ExitProcess
23: main ENDP
24:
25: AddFour PROC
26:    mov     rax,rcx
27:    add     rax,rdx
28:    add     rax,r8
29:    add     rax,r9          ; sum is in RAX
30:    ret
31: AddFour ENDP
32:
33: END
```