

75.03/95.57 Organización del Computador

U4 - CASO DE ESTUDIO INTEL

Caso de estudio Intel

Agenda

⦿ ISA (Instruction Set Architecture)

- Registros

- Direccionamiento

- Tipos de dato

- Memoria

- Endiannes

⦿ Ensamblador

- Estructura de un programa

- Definición y reserva de campos de memoria

⦿ Instrucciones

- Transferencia y Copia

Detalle de la Arquitectura de Programación ISA (Instruction Set Architecture)

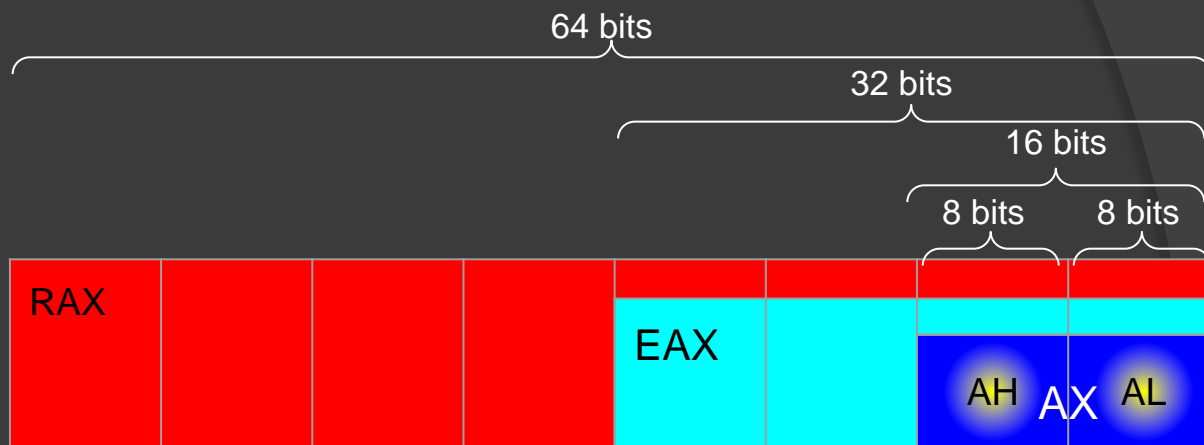
⦿ Registros

- Generales
- Índices
- Pila
- Instrucción
- Control

Registros Generales

Acumulador

operando de instrucciones aritméticas y lógicas



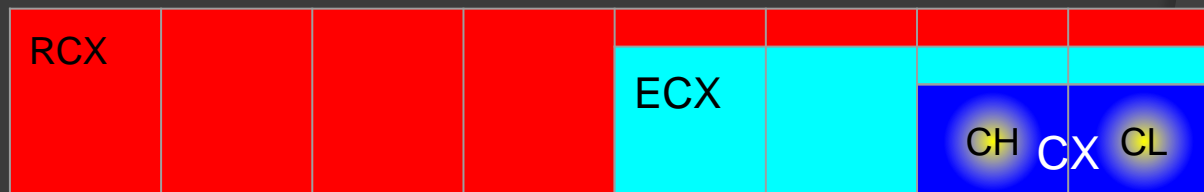
Base

direccionamiento de operandos



Contador

operaciones aritméticas o de string

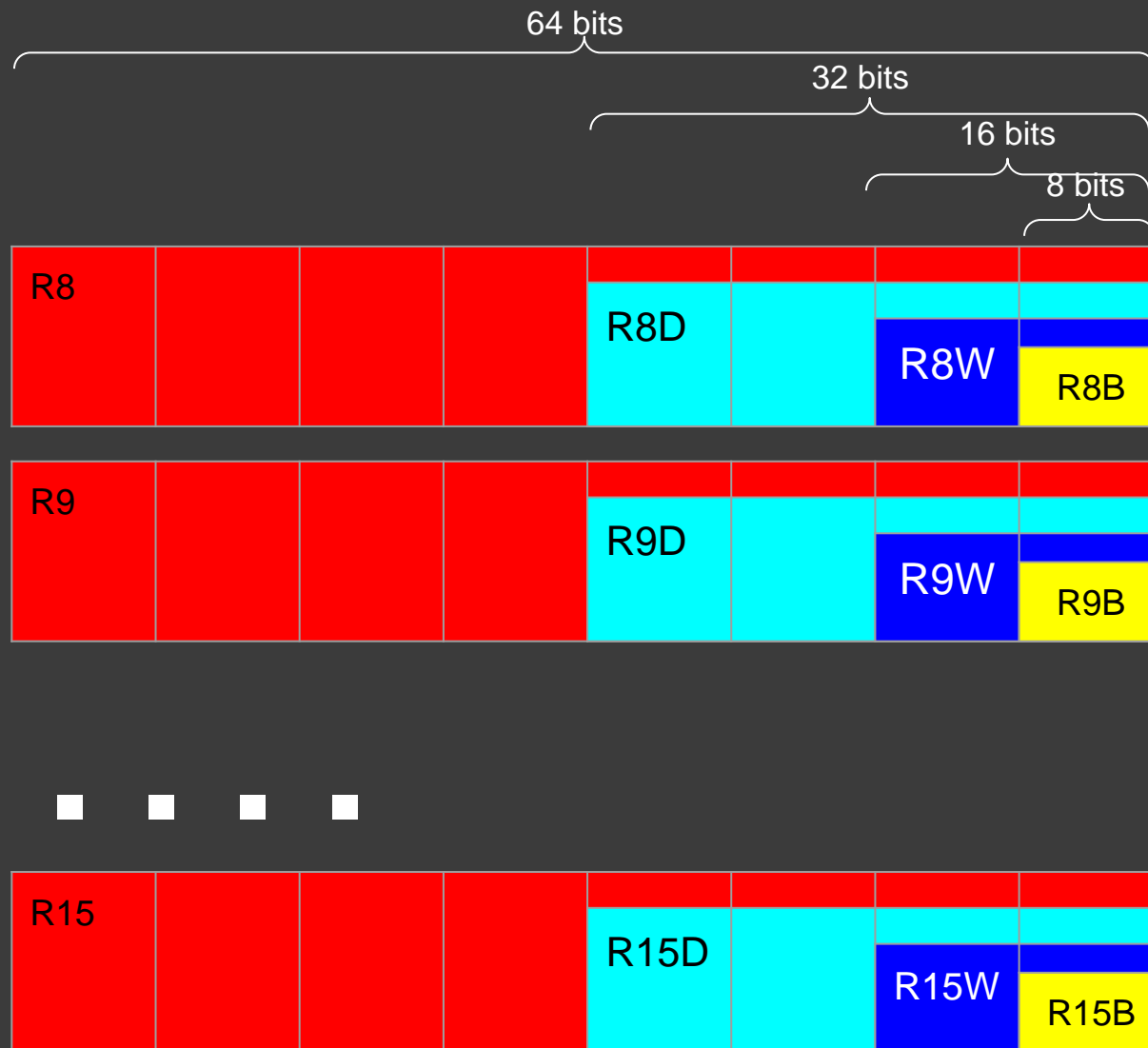


Data

operaciones que requieren duplas de registros



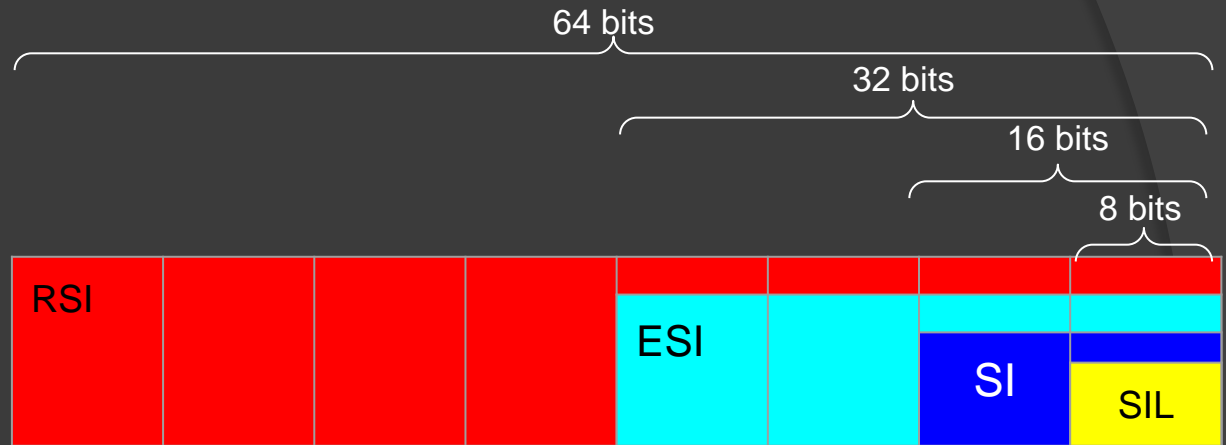
Registros Generales



Registros Indice

Source

operaciones de manejo de cadenas para apuntar al operando "origen"

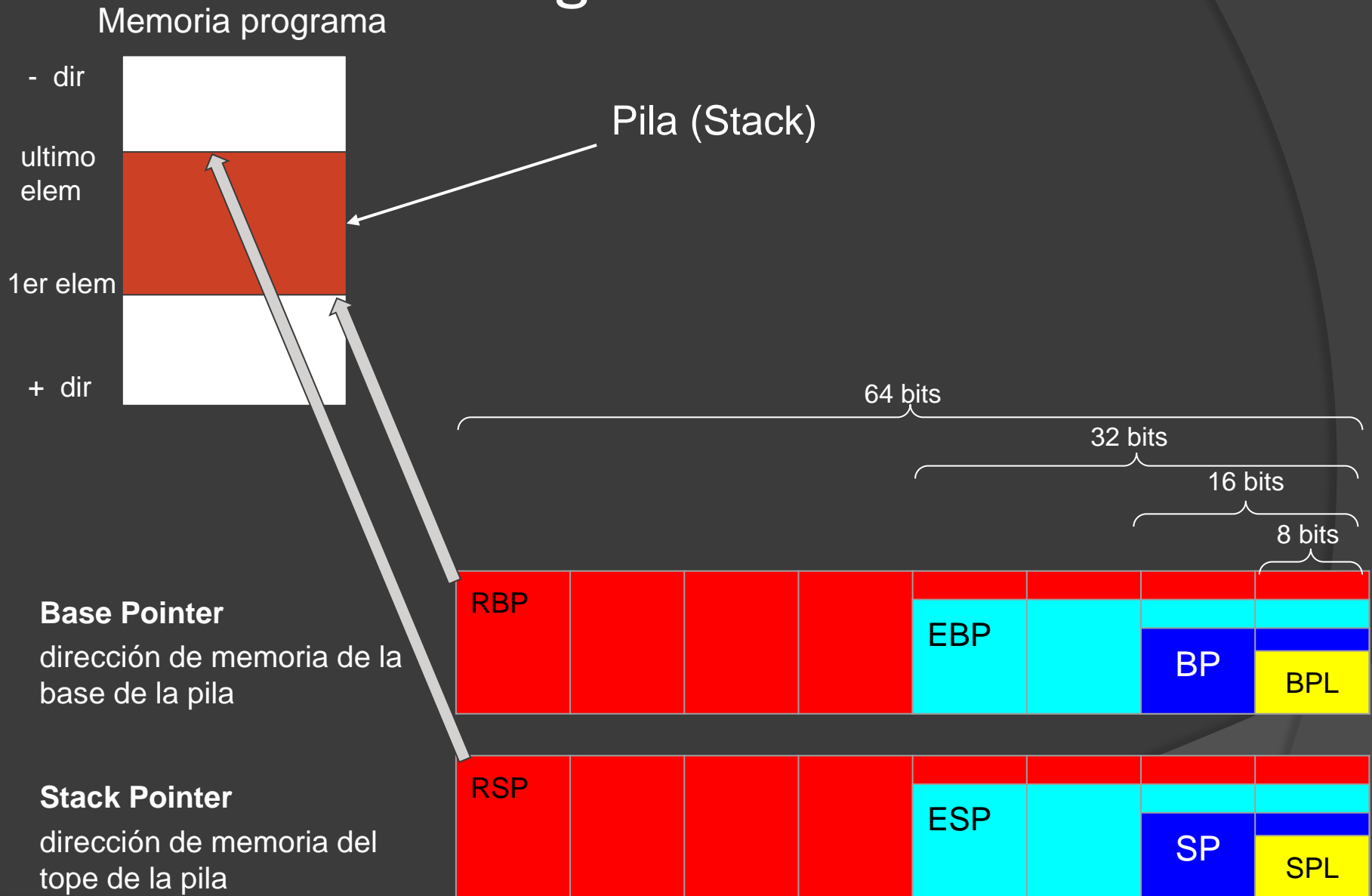


Destination

operaciones de manejo de cadenas para apuntar al operando "destino"



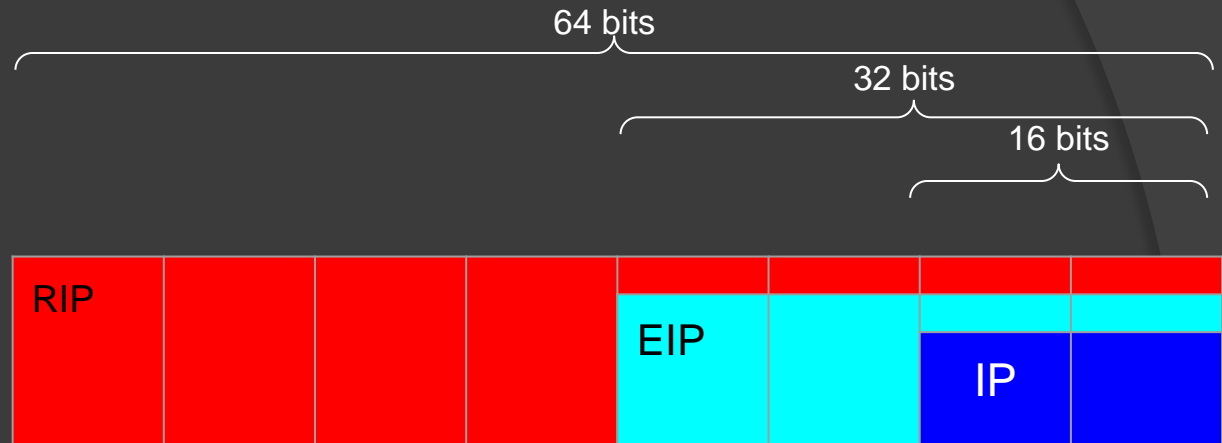
Registros de Pila



Registros de Instrucción y Control

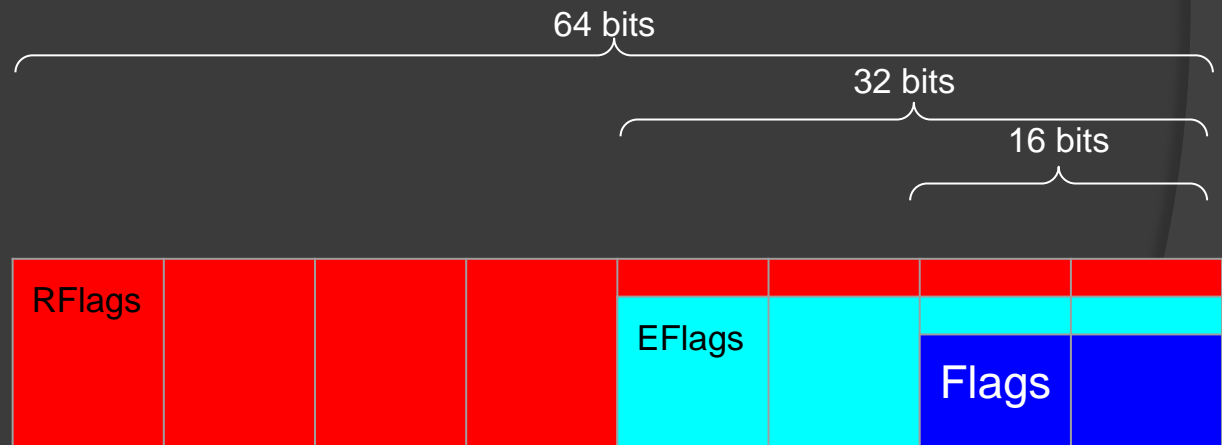
Instruction Pointer

Actúa como registro contador de programa (PC) y contiene la dirección efectiva de la instrucción siguiente que se ha de ejecutar

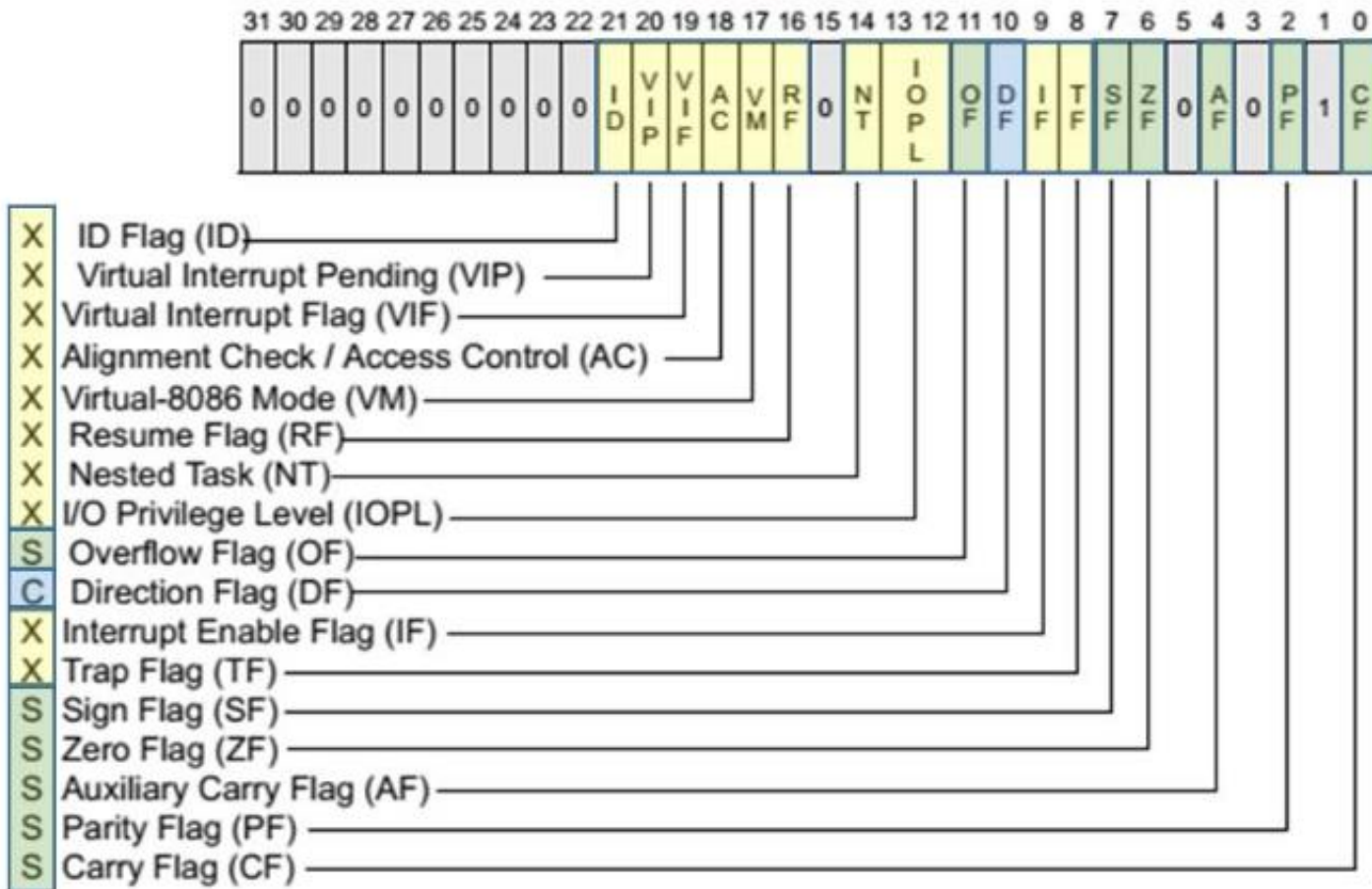


Flags

se usa para almacenar el estado general de la CPU; indica el resultado de la ejecución de cada instrucción



Registro de Control (Flags) - Detalle



- S Indicates a Status Flag
- C Indicates a Control Flag
- X Indicates a System Flag

IA-32 32-Bit EFLAGS Register

Reserved bit positions. DO NOT USE.
Always set to values previously read.

Caso de estudio Intel

Agenda

⦿ ISA (Instruction Set Architecture)

- Registros
- **Direccionamiento**
- Tipos de dato
- Memoria
- Endiannes

⦿ **Ensamblador**

- Estructura de un programa
- Definición y reserva de campos de memoria

⦿ **Instrucciones**

- Transferencia y Copia

Modos de Direccionamiento

- **Implícito:** El dato está implícito en el código de operación.

Ej. CBW

- **Registro:** El dato está en un registro.

Ej. MOV RAX, 5

- **Inmediato:** El dato está dentro de la instrucción

Ej. MOV RAX, 5

- **Directo:** El dato está en memoria referenciado por el nombre de un campo

Ej. MOV RAX, [VARIABLE]

MOV RAX, [VARIABLE + 2]

- **Registro Indirecto:** El dato está en memoria apuntado por un registro base o índice.

Ej. MOV EAX, [EBX]

MOV EAX, [ESI]

Modos de Direccionamiento

- **Registro Relativo:** El dato está en memoria apuntado por un registro base o índice más un desplazamiento.

Ej. MOV RAX, [RBX+4]

MOV RAX, [VECTOR+RBX]

MOV [RDI+3], RAX

- **Base + Índice:** El dato está en memoria apuntado por un registro base más un registro índice.

Ej. MOV [RBX+RDI], CL

- **Base Relativo + Índice:** El dato está en memoria apuntado por un registro base más un registro índice más un desplazamiento.

Ej. MOV RAX, [RBX+RDI+4]

MOV RAX, [VECTOR+RBX+RDI]

Caso de estudio Intel

Agenda

⦿ ISA (Instruction Set Architecture)

- Registros
- Direccionamiento
- Tipos de dato
- Memoria
- Endiannes

⦿ Ensamblador

- Estructura de un programa
- Definición y reserva de campos de memoria

⦿ Instrucciones

- Transferencia y Copia

Tipos de dato

- **Numérico Entero**: Binario de punto fijo con Signo.
- **Numérico Decimal**: Binario de punto Flotante IEEE.
- **Caracteres**: ASCII

Memoria

- **Celda de Memoria**: 1 Byte
- **Palabra** : 2 Bytes
- **Doble Palabra** : 4 Bytes
- **Cuádruple Palabra** : 8 Bytes

Caso de estudio Intel

Agenda

⦿ ISA (Instruction Set Architecture)

- Registros
- Direccionamiento
- Tipos de dato
- Memoria
- Endiannes

⦿ Ensamblador

- Estructura de un programa
- Definición y reserva de campos de memoria

⦿ Instrucciones

- Transferencia y Copia

Endiannes

Es el método aplicado para almacenar datos mayores a un byte en una computadora respecto a la dirección que se le asigna a cada uno de ellos en la memoria.

Existen 2 métodos:

- **Big-Endian**: determina que el orden en la memoria coincide con el orden lógico del dato.

“el dato final en la mayor dirección”

Ej. IBM Mainframe

- **Little-Endian** : es a la inversa, el dato inicial para la lógica se coloca en la mayor dirección y el dato final en la menor.

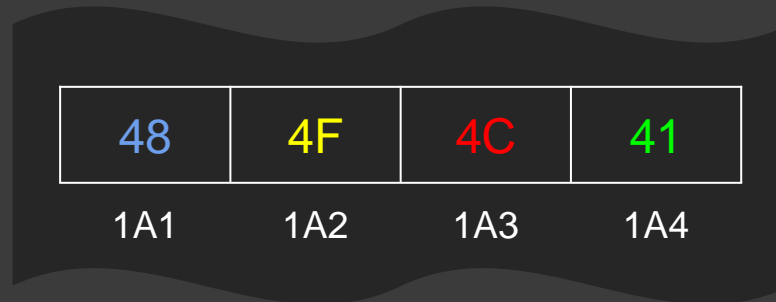
“el dato final en la menor dirección”

Ej. Intel

Endiannes - Ejemplos

- **Caso 1:** Definición de un área de memoria con contenido inicial definido en formato carácter

msg db 'HOLA'

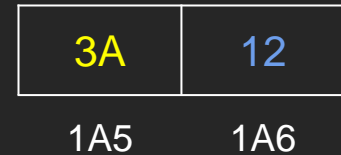


Aquí la posición de memoria que toma cada caracter recibido es la que por intuición uno asume, o sea, la letra 'H' en la dirección menor

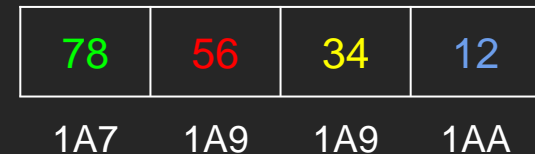
Endiannes - Ejemplos

- **Caso 2:** Definición de un área de memoria con contenido inicial definido en formato numérico

num dw 4666 ; es 123A



num2 dd 12345678h



Aquí es donde se observa la ubicación de los bytes con el método Little-Endian, el byte menos significativo se ubica en la dirección de memoria menor que el byte más significativo

Endiannes - Ejemplos

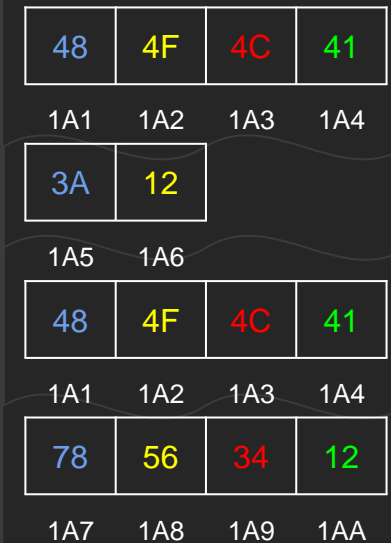
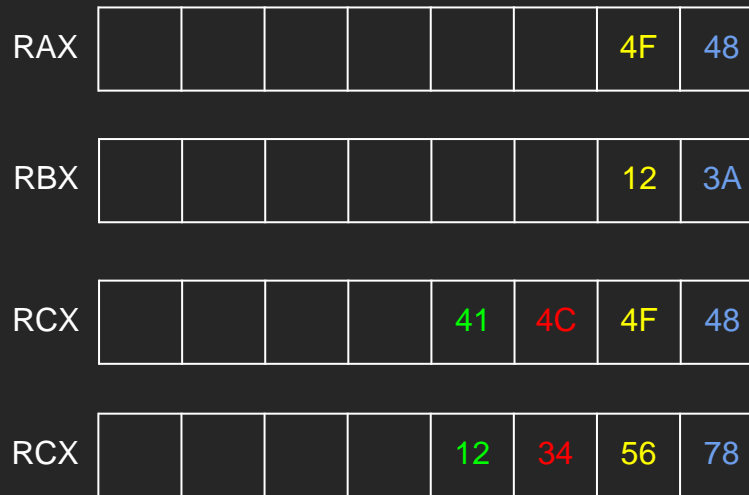
- **Caso 3:** Se ejecuta una copia de memoria a registro

mov AX, [msc]

mov BX, [nur]

mov ECX, [msc]

mov EDX, [nur]



La parte alta del registro contiene el byte de orden superior de memoria, y la parte baja del registro contiene el byte de orden inferior.

Caso de estudio Intel

Agenda

⦿ ISA (Instruction Set Architecture)

- Registros
- Direccionamiento
- Tipos de dato
- Memoria
- Endiannes

⦿ **Ensamblador**

- Estructura de un programa
- Definición y reserva de campos de memoria

⦿ **Instrucciones**

- Transferencia y Copia

Estructura de un programa

```
section    .data

    ;variables con contenido

section    .bss

    ;variables sin contenido

section    .text
main:

    ;código fuente

    ret
```

Definición y reserva de campos en memoria

● Con contenido inicial

<code>db</code>	define byte (1 byte)
<code>dw</code>	define word (2 bytes)
<code>dd</code>	define double (4 bytes)
<code>dq</code>	define quad (8 bytes)
<code>dt</code>	define ten (10 bytes)

```
section .data
```

● Sin contenido inicial

<code>resb</code>	reserve byte (1 byte)
<code>resw</code>	reserve word (2 bytes)
<code>resd</code>	reserve double (4 bytes)
<code>resq</code>	reserve quad (8 bytes)
<code>rest</code>	reserve ten (10 bytes)

```
section .bss
```

Definición y reserva de campos en memoria

● Ejemplos (1/4)

Definición	Bytes reservados
campo1 resb 1	1
campo2 resb 2	2
campo3 resw 1	2
campo4 resw 2	4
campo5 resd 1	4
campo4 resd 2	8
campo5 resq 1	8
campo6 resq 2	16
vector1 times 2 resb 2	4
vector2 times 3 resw 2	12

Definición y reserva de campos en memoria

● Ejemplos (2/4)

Definicion	Bytes reservados	Contenido memoria
decimal1 db 11	1	0B
decimal2 dw -11	2	F5 FF
decimal3 dd 12345	4	39 30 00 00
decimal4 dq -1	8	FF FF FF FF FF FF FF FF
hexa1 db -0Bh	1	F5
hexa2 dw 0Ch	2	0C 00
hexa3 dd FFFFh	4	FF FF 00 00
hexa4 dq 96B43Fh	8	3F B4 96 00 00 00 00 00

Definición y reserva de campos en memoria

● Ejemplos (3/4)

Definicion		Bytes reservados	Contenido memoria
octal1	db 13o	1	0B
octal2	dw 71o	2	39 00
octal3	dd 10o	4	08 00 00 00
binario1	db 1011b	1	0B
binario2	dw 1011b	2	0B 00
binario3	dd -1000b	4	F8 FF FF FF
truncado	db 2571 ;0A0Bh	1	0B
noTruncado	dw 2571 ;0A0Bh	2	0B 0A

Definición y reserva de campos en memoria

● Ejemplos (4/4)

Definicion	Bytes reservados	Contenido memoria
letra db 'A'	1	41
letra2 dw 'A'	2	41 20
letra3 db 'a'	1	61
cadena db 'hola'	4	68 6F 6C 61
cadena2 dw 'ola'	4	6F 6C 61 20
numero db '12'	2	31 32
vector1 times 3 db 'A'	3	41 41 41
vector2 times 3 db 'A',0	6	41 00 41 00 41 00
registro times 0 db 'A'	0	n/a

Caso de estudio Intel

Agenda

⦿ ISA (Instruction Set Architecture)

- Registros
- Direccionamiento
- Tipos de dato
- Memoria
- Endiannes

⦿ **Ensamblador**

- Estructura de un programa
- Definición y reserva de campos de memoria

⦿ **Instrucciones**

- Transferencia y Copia

Instrucciones - Transferencia y Copia

MOV op1, op2

Copia el valor del 2do operando en el primer operando.

Combinaciones	Ejemplos en NASM
MOV <reg>,<reg>	MOV AH,BL MOV AX,BX MOV ECX, EAX MOV RDX,RCX
MOV <reg>,<mem>	MOV CH,[VARIABLE_8] (*) MOV CX, [VARIABLE_16] (*) MOV ECX, [VARIABLE_32] (*) MOV RDX, [VARIABLE_64] (*)
MOV <reg>,<inm>	MOV DL,7o MOV CX,2450h MOV EAX,0h MOV RDX,28h

Instrucciones - Transferencia y Copia

MOV op1, op2

Combinaciones	Ejemplos en NASM
MOV <mem>,<reg>	MOV [VARIABLE_8],AH (*) MOV [VARIABLE_16],AX (*) MOV [VARIABLE_32],EAX (*) MOV [VARIABLE_64],RAX (*) MOV VARIABLE_64,RAX
MOV <long><mem>,<inm>	MOV byte [VARIABLE_8],2Ah MOV word [VARIABLE_16],777o MOV dword [VARIABLE_32],1234 MOV qword [VARIABLE_64],1234 MOV [VARIABLE_64],4321

Instrucciones - Comparación

CMP op1, op2

Compara el contenido del op1 contra el op2 mediante la resta entre los dos operando sin modificarlos.

Combinaciones	Ejemplos en NASM
CMP <reg>,<reg>	CMP AH,BL CMP AX,BX CMP EAX,EBX CMP RCX,RAX
CMP <reg>,<mem>	CMP CH,[VARIABLE_8] (*) CMP CX,[VARIABLE_16] (*) CMP EAX,[VARIABLE_32] (*) CMP RBX,[VARIABLE_64] (*)

Instrucciones - Comparación

CMP op1, op2

Combinaciones	Ejemplos en NASM
CMP <reg>,<inm>	CMP CH,10o CMP CX,2Ah CMP EAX,1000 CMP RBX,432h
CMP <mem>,<reg> (*) Considera la long del REG	CMP VARIABLE,AX CMP [VARIABLE_8],RAX (*) CMP [VARIABLE_8],AH CMP [VARIABLE_64],RCX
CMP <long><mem>,<inm>	CMP VARIABLE,245h CMP byte [VARIABLE_8],2Ah CMP word [VARIABLE_16],2Ah CMP dword [VARIABLE_32],2Ah CMP qword [VARIABLE_64],2Ah

Instrucciones - Saltos/Bifurcaciones

JMP op

Bifurca a la dirección indicada del operando.

Jcc op

Bifurca a la dirección indicada del operando si se cumple la condición.

Combinaciones		Ejemplos en NASM
JMP	<etiqueta>	JMP finalizar
Jcc	<etiqueta>	JE esIgual

Instrucciones - Saltos/Bifurcaciones

Condicionales generales

JE	op	-> por igual (ZF=1)
JNE	op	-> por no igual (ZF=0)
JZ	op	-> por igual a cero (ZF=1)
JNZ	op	-> por distinto a cero (ZF=0)
JRCXZ	op	-> por contenido de RCX igual cero
JC	op	-> por carry flag distinto a
cero (CF=1)		
JO	op	-> por overflow (OF=1)

Instrucciones - Saltos/Bifurcaciones Condicionales con signo

JG SF=OF)	op	-> por mayor (ZF=0 and
JGE	op	-> por mayor o igual (SF=OF)
JL	op	-> por menor (SF<> OF)
JLE OF)	op	-> por menor o igual (ZF=1 or SF<>
JNG	op	-> por no mayor (ZF=1 or SF<>OF)
JNGE	op	-> por no mayor o igual (SF<>OF)
JNL	op	-> por no menor (SF=OF)
JNLE SF=OF)	op	-> por no menor o igual (ZF=0 and

Instrucciones - Saltos/Bifurcaciones

Condicionales sin signo

JA ZF=0)	op	-> por mayor (CF=0 and
JAЕ	op	-> por mayor o igual (CF=0)
JB	op	-> por menor (CF=1)
JBE	op	-> por menor o igual (CF=1 or ZF=1).
JNA	op	-> por no mayor (CF=1 or ZF=1)
JNAЕ	op	-> por no mayor o igual (CF=1)
JNB	op	-> por no menor (CF=0)
JNBE ZF=0)	op	-> por no menor o igual CF=0 and

Instrucciones - Aritmeticas

Suma

ADD op1, op2

Suma los valores de los dos operando (binarios de punto fijo con signo) dejando el resultado en el primero.

Combinaciones	Ejemplos en NASM
ADD <reg>,<reg>	ADD AH,BL ADD AX,BX ADD ECX,EAX ADD RDX,RCX
ADD <reg>,<mem>	ADD AL,[VARIABLE_8] (*) ADD BX,[VARIABLE_16] (*) ADD ECX,[VARIABLE_32] (*) ADD RDX,[VARIABLE_64] (*)

Instrucciones - Aritmeticas

Suma (continuación)

ADD op1, op2

Combinaciones	Ejemplos en NASM
ADD <reg>,<inm>	ADD DL,10b ADD AX,10h ADD ECX,1234 ADD RCX,1234h
ADD <mem>,<reg> (*) Considera la long del REG	ADD VARIABLE_16,AX ADD [VARIABLE_8],AL ADD [VARIABLE_8],BX (*) ADD [VARIABLE_32],ECX ADD [VARIABLE_64],RDX
ADD <long><mem>,<inm>	ADD VARIABLE,123 ADD byte [RDI],245h ADD word [VARIABLE_16],2Ah ADD dword [VARIABLE_32],123 ADD qword [VARIABLE_64],2Ah

Instrucciones - Aritméticas

Resta

SUB op1, op2

Resta los valores de los dos operandos (binarios de punto fijo con signo) dejando el resultado en el primero.

Combinaciones	Ejemplos en NASM
SUB <reg>,<reg>	SUB AH,BL SUB AX,BX SUB ECX,EAX SUB RDX,RBX
SUB <reg>,<mem>	SUB AL,[VARIABLE_8] (*) SUB BX,[VARIABLE_16] (*) SUB ECX,[VARIABLE_32] (*) SUB RDX,[VARIABLE_64] (*)

Instrucciones - Aritméticas

Resta (continuación)

SUB op1, op2

Combinaciones	Ejemplos en NASM
SUB <reg>,<inm>	SUB DL,10b SUB AX,10h SUB ECX,1234 SUB RDX,1234h
SUB <mem>,<reg> (*) Considera la long del REG	SUB VARIABLE,AX SUB [VARIABLE_8],AL SUB [VARIABLE_8],BX(*) SUB [VARIABLE_32],ECX SUB [VARIABLE_64],RDX
SUB <long><mem>,<inm>	SUB VARIABLE,123 SUB byte [EDI],245h SUB word [VARIABLE],2Ah SUB dword [VARIABLE],123 SUB qword [VARIABLE],123h

Instrucciones - Aritmeticas

Incremento y Decremento

INC *op*

Suma uno al operando (binarios de punto fijo con signo)

DEC *op*

Resta uno al operando (binarios de punto fijo con signo)

Combinaciones	Ejemplos en NASM
INC/DEC <reg>	<div>INC/DEC BH <input type="checkbox"/> BH = BH + 1 / BH = BH - 1</div> <div>INC/DEC CX <input type="checkbox"/> CX = CX + 1 / CX = CX - 1</div> <div>INC/DEC EAX <input type="checkbox"/> EAX = EAX + 1 / EAX = EAX - 1</div> <div>INC/DEC RDX <input type="checkbox"/> RDX = RDX + 1 / RDX = RDX - 1</div>
INC/DEC <mem>	<div>INC byte[VAR_8B]</div> <div>INC word[VAR_16B]</div> <div>INC dword[VAR_32B] <input type="checkbox"/> VAR_32B = VAR_32B + 1</div> <div>INC qword[VAR_64B]</div> <div>DEC byte[VAR_8B]</div> <div>DEC word[VAR_16B] <input type="checkbox"/> VAR_16B = VAR_16B - 1</div> <div>DEC dword[VAR_32B]</div> <div>DEC qword[VAR_64B]</div>

Instrucciones - Aritmeticas

Multiplicación - Formato 1 operando

IMUL *op*

Si longitud de *op* es 8 bits:

Multiplica *AL* * *op* y deja el resultado en *AX*

Si longitud de *op* es 16 bits:

Multiplica *AX* * *op* y deja el resultado en *DX:AX*

Si longitud de *op* es 32 bits:

Multiplica *EAX* * *op* y deja el resultado en *EDX:EAX*

Si longitud de *op* es 64 bits:

Multiplica *RAX* * *op* y deja el resultado en *RDX:RAX*

Los operandos son interpretados como binario de punto fijo CON signo

MUL *op*

Igual que **IMUL** pero los operandos son interpretados como binario de punto fijo SIN signo

Instrucciones - Aritméticas

Multiplicación - Formato 1 operando (cont)

IMUL op

MUL op

Combinaciones	Ejemplos en NASM
MUL/IMUL <reg>	<div>MUL/IMUL BH <input type="checkbox"/> AX = (AL)*(BH)</div> <div>MUL/IMUL BX <input type="checkbox"/> DX:AX = (AX)*(BX)</div> <div>MUL/IMUL EBX <input type="checkbox"/> EDX:EAX = (EAX)*(EBX)</div> <div>MUL/IMUL RCX <input type="checkbox"/> RDX:EAX = (EAX)*(RCX)</div>
MUL/IMUL <mem>	<div>MUL/IMUL byte[VAR_8] <input type="checkbox"/> AX = (AL)*(VAR_8)</div> <div>MUL/IMUL word[VAR_16] <input type="checkbox"/> DX:AX = (AX)*(VAR_16)</div> <div>MUL/IMUL dword[VAR_32] <input type="checkbox"/> EDX:EAX = (EAX)*(VAR_32)</div> <div>MUL/IMUL qword[VAR_64] <input type="checkbox"/> RDX:RAX = (RAX)*(VAR_64)</div>

Instrucciones - Aritméticas

Multiplicación - Formato 2 operandos

IMUL op1, op2

Multiplica el contenido de los operandos y almacena el resultado en el primero.

Ambos operandos deben tener la misma longitud

Si el resultado no entra en el operando 1, se trunca.

Los operandos son interpretados como binario de punto fijo CON signo

MUL op1, op2

Igual que IMUL pero los operandos son interpretados como binario de punto fijo SIN signo

Instrucciones - Aritméticas

Multiplicación - Formato 2 operandos (cont)

IMUL op1, op2

MUL op1, op2

Combinaciones	Ejemplos en NASM
IMUL <reg>, <inm>	IMUL CX,4 □ $CX = (CX) * 4$
IMUL <reg>,<reg>	IMUL EAX,EBX □ $EAX = (EAX) * (EBX)$
IMUL <reg>,<log><mem>	IMUL RBX,qword[VAR_64] □ $RBX = (RBX) * (VAR_64)$

Instrucciones - Aritméticas

Multiplicación - Formato 3 operandos

IMUL op1, op2, op3

Multiplica el contenido de los operandos 2 y 3 y almacena el resultado en el operando 1.

El operando 1 es siempre un registro y el operando 3 siempre un valor inmediato.

Si el resultado no entra en el operando 1, se trunca.

Los operandos son interpretados como binario de punto fijo CON signo

MUL op1, op2, op3

Igual que IMUL pero los operandos son interpretados como binario de punto fijo SIN signo

Instrucciones - Aritméticas

Multiplicación - Formato 3 operandos (cont)

IMUL $op1, op2, op3$

MUL $op1, op2, op3$

Combinaciones	Ejemplos en NASM
IMUL <reg>, <reg>,<inm>	IMUL CX,BX,10h □ $CX = (BX) * 10h$
IMUL <reg>,<mem>,<inm>	IMUL RBX,qword[VAR_64],4 □ $RBX = (VAR_64) * 4$

Instrucciones - Aritméticas

División

IDIV *op*

Si longitud de *op* es 8 bits:

AX/*op* resto en *AH* y cociente en *AL*

Si longitud de *op* es 16 bits:

DX:AX/*op* resto en *DX* y cociente en *AX*

Si longitud de *op* es 32 bits:

EDX:EAX/*op* resto en *EDX* y cociente en *EAX*

Si longitud de *op* es 64 bits:

RDX:RAX/*op* resto en *RDX* y cociente en *RAX*

Los operandos son interpretados como binario de punto fijo CON signo

DIV *op*

Igual que *IMUL* pero los operandos son interpretados como binario de punto fijo SIN signo

Instrucciones - Aritméticas

División (cont)

IDIV op

DIV op

Combinaciones	Ejemplos en NASM
DIV/IDIV <reg>	<div>DIV/IDIV BX \square DX:AX / BX<ul style="list-style-type: none">• Cociente = AX• Resto = DX</div> <div>DIV/IDIV EBX \square EDX:EAX / EBX<ul style="list-style-type: none">• Cociente = EAX• Resto = EDX</div> <div>DIV/IDIV RCX \square RDX:RAX / RCX<ul style="list-style-type: none">• Cociente = RAX• Resto = RDX</div>
DIV/IDIV <long><mem>	<div>DIV/IDIV byte[VAR_8] \square AX / [VAR_8]<ul style="list-style-type: none">• Cociente = AL• Resto = AH</div> <div>DIV/IDIV dword[VAR_32] \square EDX:EAX / [VAR_32]<ul style="list-style-type: none">• Cociente = EAX• Resto = EDX</div> <div>DIV/IDIV qword[VAR_64] \square RDX:RAX / [VAR_64]<ul style="list-style-type: none">• Cociente = RAX• Resto = RDX</div>

Instrucciones - Aritméticas

Conversión

CBW

Convierte el byte almacenado en `AL` a una word en `AX`.

CWD

Convierte la word almacenada en `AX` a una double-word en `DX:AX`

CWDE

Convierte la word almacenada en `AX` a una double-word en `EAX`

CDQE

Convierte la doble-word almacenada en `EAX` a una quad-word en `RAX`

Expanden en signo del operando

Ejemplo en NASM

```
MOV AL,byte[VAR_8B]
CBW
CWDE
CDQE
```

```
VAR_8B    db    9Bh □ AL = 9B
AX = FF9B
EAX = FFFFFFF9B
RAX = FFFFFFFFFFFFFFF9B
```

```
MOV AX,word[VAR_16B]
CWD
CWDE
CDQE
```

```
VAR_16B    dw    FF9Bh □ AX = FF9B
DX:AX = FFFFh:FF9B
EAX = FFFFFFF9B
RAX = FFFFFFFFFFFFFFF9B
```

Instrucciones - Aritméticas

Complemento

NEG *op*

Realiza el complemento a 2 del operando, es decir, le cambia el signo.

Combinaciones	Ejemplos en NASM
NEG <reg>	NEG BH NEG AX NEG ECX NEG RDX
NEG <mem>	NEG byte [VAR_8] NEG word [VAR_16] NEG dword [VAR_32] NEG qword [VAR_64]

Instrucciones - Saltos/Bifurcaciones

Loop

LOOP *op*

Resta 1 al contenido del registro RCX y si el resultado es 0, bifurca al punto indicado por el operando, sino continua la ejecución en la instrucción siguiente.

El desplazamiento al punto indicado debe estar en un rango entre -128 a 127 bytes (*near jump*)

```
        mov    rcx, 5
inicio:
    . . .
        . . .
        loop   inicio
    . . .
```

Instrucciones - Saltos/Bifurcaciones

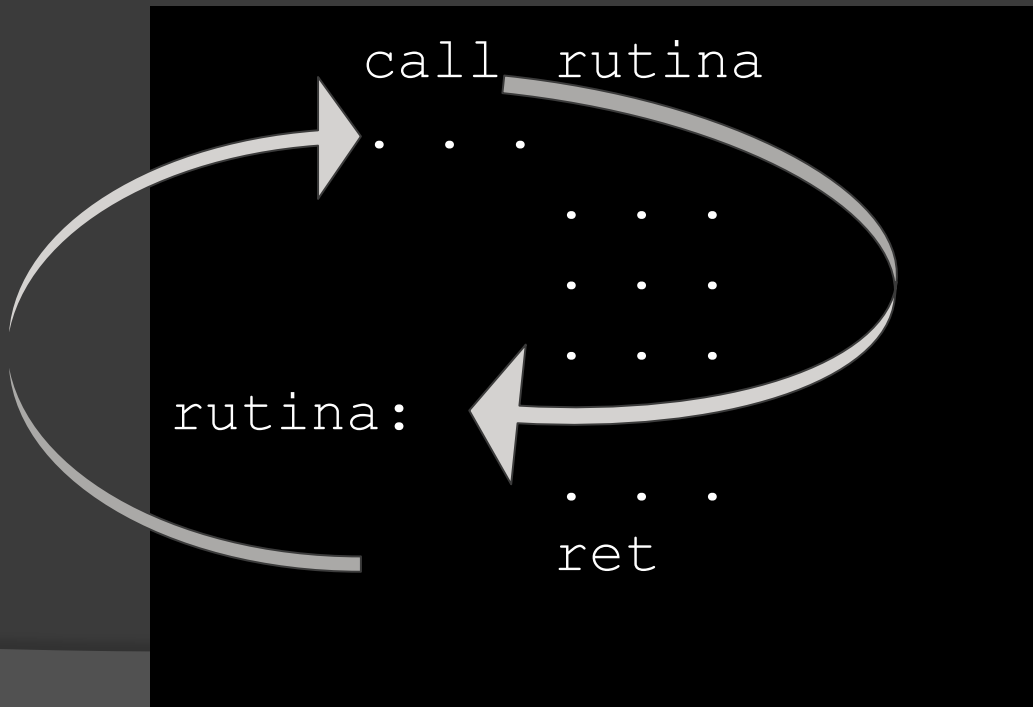
Llamada y Retorno de procedimiento

CALL op

Almacena en la pila la dirección de la instrucción siguiente a la call y bifurca al punto indicado por el operando.

RET

Toma el elemento del tope de la pila que debe ser una dirección de memoria (generalmente cargada por una call) y bifurca hacia la misma.



Tablas

Tira de bytes en memoria destinada a usar como una estructura de Vector o Matriz

```
tabla          times 40 resb 1
vector          times 10 resw
1
matriz          times 25 db
```

``*``

Posicionamiento en el elemento i de un vector

$(i - 1) * longitudElemento$

Posicionamiento en el elemento i,j de una matriz

$(i-1)*longitudFila + (j-1)*longitudElemento$

$longitudFila = longitudElemento * cantidadColumnas$

Validación

Código destinado a verificar que los datos de un programa provenientes del exterior (teclado, archivos) cumplen las condiciones esperadas y/o necesarias.

Clasificación según el contenido del dato:

- **Lógica:** que se adecúe al significado lógico

Ej: Dia de la semana: lunes, martes, miércoles, etc
 Respuesta "S" o "N"

- **Física:** que el dato esté en un formato particular

Ej: Campo en formato empaquetado
 Fecha en formato DD/MM/AA

Clasificación según el mecanismo aplicado

- **Por valor:** comparar contra uno o varios valores válidos
- **Por Rango:** comparar que esté dentro de un rango continuo válido
- **Por tabla:** buscar que exista en una tabla de valores válidos

Instrucciones - Lógicas

And

AND op1, op2

Ejecuta la operación lógica AND entre el operando 1 y 2 dejando el resultado en el 1

Combinaciones	Ejemplos en NASM
AND <reg>,<reg>	AND AH,BL / AND AX,BX / AND ECX,EAX
AND <reg>,<long><mem>	AND AL, byte [VAR_8B] AND ECX, dword [VAR_32B]
AND <reg>,<inm>	AND CH,00h / AND CX,250h / AND CX,3456
AND <long><mem>,<inm>	AND word [VAR_16B], 1010b AND dword [VAR_32B],FFCC00AAh
AND <long><mem>,<reg>	AND byte [VAR_8B],BL AND dword [VAR_32B],EAX

Instrucciones - Lógicas

Or

OR op1, op2

Ejecuta la operación lógica OR entre el operando 1 y 2 dejando el resultado en el 1

Combinaciones	Ejemplos en NASM
OR <reg>,<reg>	OR AH,BL / OR AX,BX / OR ECX,EAX
OR <reg>,<long><mem>	OR AL, byte [VAR_8B] OR ECX, dword [VAR_32B]
OR <reg>,<inm>	OR CH,00h / OR CX,250h / OR CX,3456
OR <long><mem>,<inm>	OR word [VAR_16B], 1010b OR dword [VAR_32B],FFCC00AAh
OR <long><mem>,<reg>	OR byte [VAR_8B],BL OR dword [VAR_32B],EAX

Instrucciones - Lógicas

Exclusive or

XOR op1, op2

Ejecuta la operación lógica EXCLUSIVE OR entre el operando 1 y 2 dejando el resultado en el 1

Combinaciones	Ejemplos en NASM
XOR <reg>,<reg>	XOR AH,BL / XOR AX,BX / XOR ECX,EAX
XOR <reg>,<long><mem>	XOR AL, byte [VAR_8B] XOR ECX, dword [VAR_32B]
XOR <reg>,<inm>	XOR CH,00h / XOR CX,250h / XOR CX,3456
XOR <long><mem>,<inm>	XOR word [VAR_16B], 1010b XOR dword [VAR_32B],FFCC00AAh
XOR <long><mem>,<reg>	XOR byte [VAR_8B],BL XOR dword [VAR_32B],EAX

Instrucciones - Lógicas

Not

NOT *op*

Ejecuta la operación lógica NOT en el operando

Combinaciones	Ejemplos en NASM
NOT <reg>	NOT AH NOT BX NOT ECX
NOT <mem>	NOT byte [VAR_8B] NOT word [VAR_16B] NOT dword [VAR_32B]

Instrucciones - Transferencia y Copia

LEA op1, op2

Copia en el operando 1 (un registro) la dirección de memoria del operando 2.

Combinaciones	Ejemplos en NASM
LEA <reg>,<mem>	LEA RAX,[VARIABLE]
	es equivalente a hacer MOV RAX,VARIABLE ;notar q aca NO hay corchetes

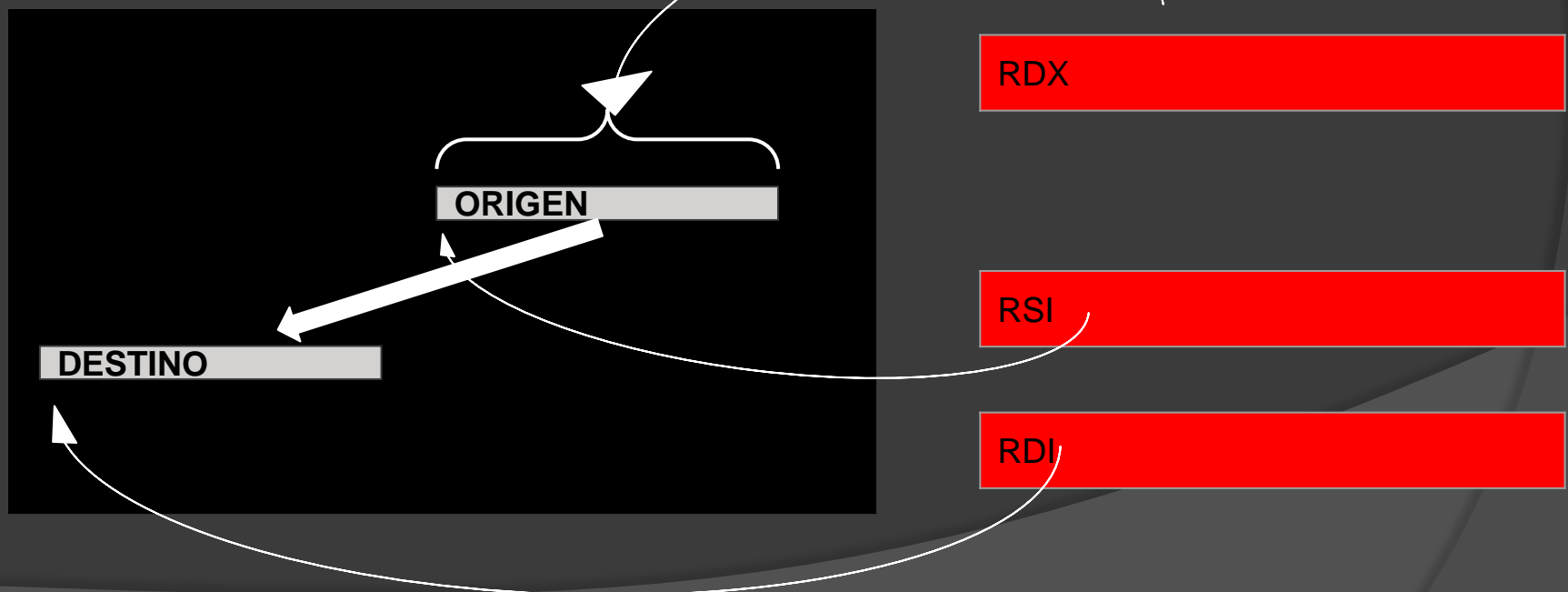
Instrucciones - Transferencia y Copia

Copia de strings

MOVSB

Copia el contenido de memoria apuntado por RSI (origen/source) al apuntado por RDI (destino/destination). Copia tantos bytes como los indicados en el registro RCX

Memoria programa



Instrucciones - Transferencia y Copia

Copia de strings (cont)

MOVSB

```
    . . .  
    MOV    RCX, 4  
    LEA  
RSI, [MSGORI]  
    LEA  
RDI, [MSGDES]  
REP MOVSB  
    . . .
```


Instrucciones - Comparación

Comparación de strings

CMPSB

Compara el contenido de memoria apuntado por RSI (origen/source) con el apuntado por RDI (destino/destination).
Compara tantos bytes como los indicados en el registro RCX

```
. . .  
MOV    RCX, 4  
LEA    RSI, [MSG1]  
LEA    RDI, [MSG2]  
REPE CMPSB  
JE  
IGUALES  
. . .
```

Instrucciones - Transferencia y Copia

Manejo de la pila (stack)

PUSH *op*

Inserta el operando (de 64 bits) en la pila. Decrementa (resta 1) el contenido del registro RSP

POP *op*

Elimina el último elemento insertado en la pila (de 64 bits) y lo copia en el operando. Incrementa (suma 1) al contenido del registro RSP

Combinaciones	Ejemplos en NASM
PUSH / POP <reg>	PUSH / POP RDX
PUSH / POP qword <mem>	PUSH / POP qword [VARIABLE]

● Manejo de Pila

- La pila es usada para almacenar transitoriamente datos, direcciones de retornos de subrutinas o pasar parámetros a funciones o subrutinas.
- El ultimo que entra es el primero que sale □ LIFO.
- PUSH sirve para poner el dato en la pila mientras que POP se usa para recuperar el dato.
- El stack Pointer (SP) se incrementa con el POP y se decrementa con el PUSH.
- El operando puede ser un registro o posición de memoria(ambos 64bits).

Combinaciones	Ejemplos en NASM
PUSH / POP <reg>	PUSH / POP RDX
PUSH / POP qword <mem>	PUSH / POP qword [VARIABLE]

