



Práctica de Organización del Computador II

Programación en Arquitecturas Intel 64 e IA-32

22 de marzo de 2022

Organización del Computador II
DC - UBA

1. Intel® 64 and IA-32 Architectures Software Developer's Manual
Volume 1: Basic Architecture
2. Intel® 64 and IA-32 Architectures Software Developer's Manual
Volume 2: Instruction Set Reference, A-Z

1. Intel® 64 and IA-32 Architectures Software Developer's Manual
Volume 1: Basic Architecture
2. Intel® 64 and IA-32 Architectures Software Developer's Manual
Volume 2: Instruction Set Reference, A-Z
3. Intel® 64 and IA-32 Architectures Software Developer's Manual
Volume 3: System Programming Guide

- Los manuales especifican detalladamente todo lo que debe saber un desarrollador para programar a bajo nivel en procesadores Intel de 32 bits y 64 bits

- Los manuales especifican detalladamente todo lo que debe saber un desarrollador para programar a bajo nivel en procesadores Intel de 32 bits y 64 bits
- Lamentablemente, sólo están disponibles en idioma inglés.

- Los manuales especifican detalladamente todo lo que debe saber un desarrollador para programar a bajo nivel en procesadores Intel de 32 bits y 64 bits
- Lamentablemente, sólo están disponibles en idioma inglés.
- El primer volumen nos cuenta la arquitectura básica

- Los manuales especifican detalladamente todo lo que debe saber un desarrollador para programar a bajo nivel en procesadores Intel de 32 bits y 64 bits
- Lamentablemente, sólo están disponibles en idioma inglés.
- El primer volumen nos cuenta la arquitectura básica
- El segundo volumen es una referencia detallada a cada una de las instrucciones

- Los manuales especifican detalladamente todo lo que debe saber un desarrollador para programar a bajo nivel en procesadores Intel de 32 bits y 64 bits
- Lamentablemente, sólo están disponibles en idioma inglés.
- El primer volumen nos cuenta la arquitectura básica
- El segundo volumen es una referencia detallada a cada una de las instrucciones
- El tercer volumen explica todo lo requerido para la programación del Sistema (2da parte de la materia).

- Los manuales especifican detalladamente todo lo que debe saber un desarrollador para programar a bajo nivel en procesadores Intel de 32 bits y 64 bits
- Lamentablemente, sólo están disponibles en idioma inglés.
- El primer volumen nos cuenta la arquitectura básica
- El segundo volumen es una referencia detallada a cada una de las instrucciones
- El tercer volumen explica todo lo requerido para la programación del Sistema (2da parte de la materia).
- Son necesarios para consulta pero no nos va a interesar todo lo que dicen los manuales

BASIC EXECUTION ENVIRONMENT

The special uses of general-purpose registers by instructions are described in Chapter 5, "Instruction Set Summary," in this volume. See also: Chapter 3, Chapter 4 and Chapter 5 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B & 2C*. The following is a summary of special uses:

- **EAX** — Accumulator for operands and results data
- **EBX** — Pointer to data in the DS segment
- **ECX** — Counter for string and loop operations
- **EDX** — I/O pointer
- **ESI** — Pointer to data in the segment pointed to by the DS register; source pointer for string operations
- **EDI** — Pointer to data (or destination) in the segment pointed to by the ES register; destination pointer for string operations
- **ESP** — Stack pointer (in the SS segment)
- **EBP** — Pointer to data on the stack (in the SS segment)

As shown in Figure 3-5, the lower 16 bits of the general-purpose registers map directly to the register set found in the 8086 and Intel 286 processors and can be referenced with the names AX, BX, CX, DX, BP, SI, DI, and SP. Each of the lower two bytes of the EAX, EBX, ECX, and EDX registers can be referenced by the names AH, BH, CH, and DH (high bytes) and AL, BL, CL, and DL (low bytes).

General-Purpose Registers					
31	16	15	8	7	0
			AH	AL	AX EAX
			BH	BL	BX EBX
			CH	CL	CX ECX
			DH	DL	DX EDX
			BP		EBP
			SI		ESI
			DI		EDI
			SP		ESP

Figure 3-5. Alternate General-Purpose Register Names

3.4.1.1 General-Purpose Registers in 64-Bit Mode

In 64-bit mode, there are 16 general purpose registers and the default operand size is 32 bits. However, general-purpose registers are able to work with either 32-bit or 64-bit operands. If a 32-bit operand size is specified: EAX,

Figura 1: Los registros en el manual de Arquitectura Básica

ADD—Add

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
04 <i>ib</i>	ADD AL, <i>imm8</i>	I	Valid	Valid	Add <i>imm8</i> to AL.
05 <i>iw</i>	ADD AX, <i>imm16</i>	I	Valid	Valid	Add <i>imm16</i> to AX.
05 <i>id</i>	ADD EAX, <i>imm32</i>	I	Valid	Valid	Add <i>imm32</i> to EAX.
REX.W + 05 <i>id</i>	ADD RAX, <i>imm32</i>	I	Valid	N.E.	Add <i>imm32</i> sign-extended to 64-bits to RAX.
80 /0 <i>ib</i>	ADD r/m8, <i>imm8</i>	MI	Valid	Valid	Add <i>imm8</i> to r/m8.
REX + 80 /0 <i>ib</i>	ADD r/m8*, <i>imm8</i>	MI	Valid	N.E.	Add sign-extended <i>imm8</i> to r/m8.
81 /0 <i>iw</i>	ADD r/m16, <i>imm16</i>	MI	Valid	Valid	Add <i>imm16</i> to r/m16.
81 /0 <i>id</i>	ADD r/m32, <i>imm32</i>	MI	Valid	Valid	Add <i>imm32</i> to r/m32.
REX.W + 81 /0 <i>id</i>	ADD r/m64, <i>imm32</i>	MI	Valid	N.E.	Add <i>imm32</i> sign-extended to 64-bits to r/m64.
83 /0 <i>ib</i>	ADD r/m16, <i>imm8</i>	MI	Valid	Valid	Add sign-extended <i>imm8</i> to r/m16.

Figura 2: ADD en el manual de Referencia

INSTRUCTION SET REFERENCE, M-U

Operation

DEST := SRC;

Loading a segment register while in protected mode results in special checks and actions, as described in the following listing. These checks are performed on the segment selector and the segment descriptor to which it points.

IF SS is loaded

THEN

IF segment selector is NULL

THEN #GP(0); FI;

IF segment selector index is outside descriptor table limits

OR segment selector's RPL \neq CPL

OR segment is not a writable data segment

OR DPL \neq CPL

THEN #GP(selector); FI;

IF segment not marked present

THEN #SS(selector);

ELSE

SS := segment selector;

SS := segment descriptor; FI;

FI;

IF DS, ES, FS, or GS is loaded with non-NULL selector

THEN

Figura 3: MOV en el manual de Referencia

- En esta primer actividad, vamos a **buscar los datos** previos necesarios sobre la arquitectura y que vamos a usar durante la segunda actividad de programación en Assembler.

- En esta primer actividad, vamos a **buscar los datos** previos necesarios sobre la arquitectura y que vamos a usar durante la segunda actividad de programación en Assembler.
- Usaremos el **volumen 1 de los manuales Intel**

- En esta primer actividad, vamos a **buscar los datos** previos necesarios sobre la arquitectura y que vamos a usar durante la segunda actividad de programación en Assembler.
- Usaremos el **volumen 1 de los manuales Intel**
- Nos vamos a organizar en grupos. Cada grupo va a elegir responder uno de los dos enunciados subidos en el campus.

- En esta primer actividad, vamos a **buscar los datos** previos necesarios sobre la arquitectura y que vamos a usar durante la segunda actividad de programación en Assembler.
- Usaremos el **volumen 1 de los manuales Intel**
- Nos vamos a organizar en grupos. Cada grupo va a elegir responder uno de los dos enunciados subidos en el campus.
- El trabajo consiste en ir respondiendo las preguntas que figuran en el enunciado y corrigiendo en los checkpoints.

- En esta primer actividad, vamos a **buscar los datos** previos necesarios sobre la arquitectura y que vamos a usar durante la segunda actividad de programación en Assembler.
- Usaremos el **volumen 1 de los manuales Intel**
- Nos vamos a organizar en grupos. Cada grupo va a elegir responder uno de los dos enunciados subidos en el campus.
- El trabajo consiste en ir respondiendo las preguntas que figuran en el enunciado y corrigiendo en los checkpoints.
- Pueden llamarnos a los grupos para responder preguntas y dialogar sobre lo que vayan viendo.

- En esta primer actividad, vamos a **buscar los datos** previos necesarios sobre la arquitectura y que vamos a usar durante la segunda actividad de programación en Assembler.
- Usaremos el **volumen 1 de los manuales Intel**
- Nos vamos a organizar en grupos. Cada grupo va a elegir responder uno de los dos enunciados subidos en el campus.
- El trabajo consiste en ir respondiendo las preguntas que figuran en el enunciado y corrigiendo en los checkpoints.
- Pueden llamarnos a los grupos para responder preguntas y dialogar sobre lo que vayan viendo.
- Al finalizar, haremos una puesta en común o cierre.

- En esta primer actividad, vamos a **buscar los datos** previos necesarios sobre la arquitectura y que vamos a usar durante la segunda actividad de programación en Assembler.
- Usaremos el **volumen 1 de los manuales Intel**
- Nos vamos a organizar en grupos. Cada grupo va a elegir responder uno de los dos enunciados subidos en el campus.
- El trabajo consiste en ir respondiendo las preguntas que figuran en el enunciado y corrigiendo en los checkpoints.
- Pueden llamarnos a los grupos para responder preguntas y dialogar sobre lo que vayan viendo.
- Al finalizar, haremos una puesta en común o cierre.
- Aprender de los compañeros y docentes mediante el diálogo va a ser fundamental en la materia. Queremos que sean capaces de explicar lo que van comprendiendo y colaborar con sus pares.

Actividad: Arquitectura Intel 64 e IA-32

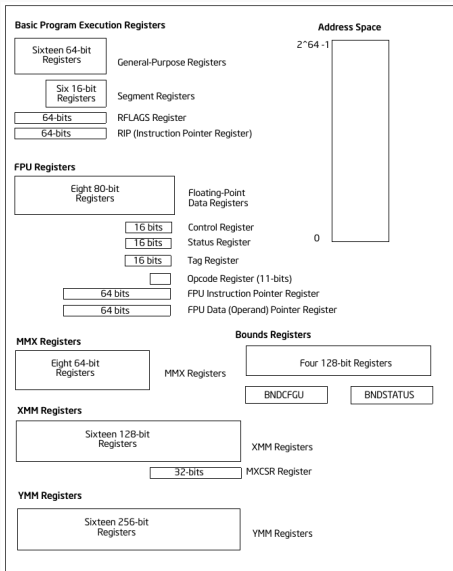


Figure 3-2. 64-Bit Mode Execution Environment

Registros de Propósito General

IA-32

31

0

eax
ebx
ecx
edx
esi
edi
ebp
esp

Nombres para acceder a los bits del registro en las posiciones

31-0 (32 bits)	15-0 (16 bits)	15-8 (8 bits)	7-0 (8 bits)
eax	ax	ah	al
ebx	bx	bh	bl
ecx	cx	ch	cl
edx	dx	dh	dl
esi	si		
edi	di		
ebp	bp		
esp	sp		

Registros de propósito general en Intel de 64 bits

Registros de Propósito General

Intel 64

63	0
rax	
rbx	
rcx	
rdx	
rsi	
rdi	
rbp	
rsp	
r8	
r9	
r10	
r11	
r12	
r13	
r14	
r15	

Nombres para acceder a los bits del registro en las posiciones

63-0 (64 bits)	31-0 (32 bits)	15-0 (16 bits)	15-8 (8 bits)	7-0 (8 bits)
rax	eax	ax	ah	al
rbx	ebx	bx	bh	bl
rcx	ecx	cx	ch	cl
rdx	edx	dx	dh	dl
rsi	esi	si		sil
rdi	edi	di		dil
rbp	ebp	bp		bpl
rsp	esp	sp		spl
r8	r8d	r8w		r8b
r9	r9d	r9w		r9b
r10	r10d	r10w		r10b
r11	r11d	r11w		r11b
r12	r12d	r12w		r12b
r13	r13d	r13w		r13b
r14	r14d	r14w		r14b
r15	r15d	r15w		r15b

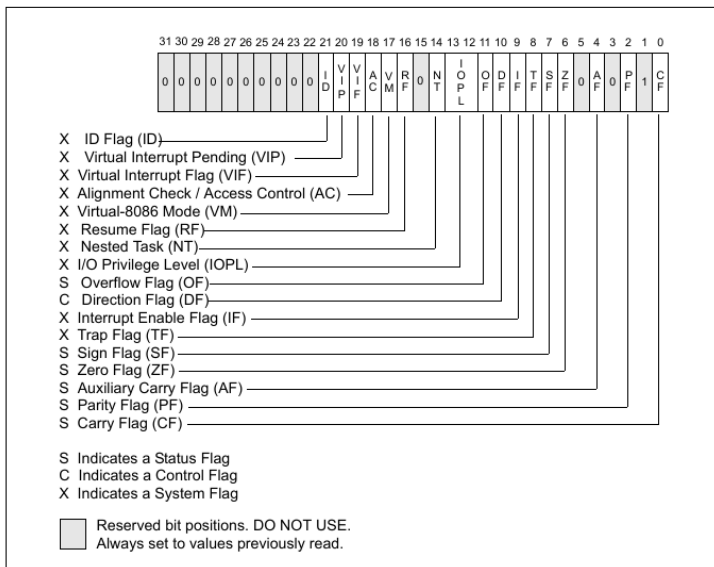


Figure 3-8. EFLAGS Register

Durante la materia, pueden usar cualquier instrucción de Intel. Estas son sólo algunas.

- ADD
- SUB
- INC
- DEC
- OR
- AND
- NOT
- XOR
- POP
- PUSH
- CALL
- RET
- MOV
- SHL
- SHR
- JE
- JGE
- JZ
- JMP
- CMP
- DIV
- MUL

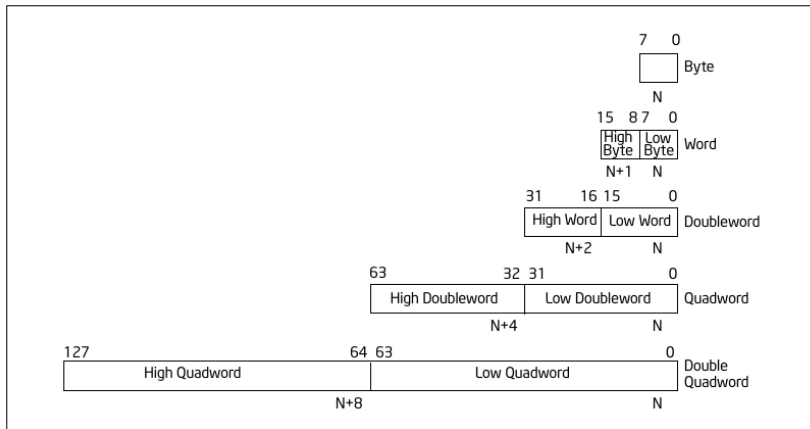


Figure 4-1. Fundamental Data Types

Table 3-2. Addressable General Purpose Registers

Register Type	Without REX	With REX
Byte Registers	AL, BL, CL, DL, AH, BH, CH, DH	AL, BL, CL, DL, DIL, SIL, BPL, SPL, R8B - R15B
Word Registers	AX, BX, CX, DX, DI, SI, BP, SP	AX, BX, CX, DX, DI, SI, BP, SP, R8W - R15W
Doubleword Registers	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D - R15D
Quadword Registers	N.A.	RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8 - R15

**¿Cómo programar, ensamblar y
ejecutar un programa en Assembler
Intel64 e IA-32?**



- **Compilador:** Toma código en un lenguaje de alto nivel y lo transforma a código ensamblador de alguna arquitectura.



- **Compilador:** Toma código en un lenguaje de alto nivel y lo transforma a código ensamblador de alguna arquitectura.
- **Ensamblador:** Toma código en lenguaje ensamblador y lo traduce a código de máquina, generando un archivo objeto. Resuelve nombres, simbólicos y traduce los mnemónicos.



- **Compilador:** Toma código en un lenguaje de alto nivel y lo transforma a código ensamblador de alguna arquitectura.
- **Ensamblador:** Toma código en lenguaje ensamblador y lo traduce a código de máquina, generando un archivo objeto. Resuelve nombres, simbólicos y traduce los mnemónicos.
- **Linker:** Toma varios archivos objeto y los transforma en un ejecutable.



- Usaremos el **NASM** que es un ensamblador libre para escribir programas de 16-bit, 32-bit (IA-32) y 64-bit (x86-64).



- Usaremos el **NASM** que es un ensamblador libre para escribir programas de 16-bit, 32-bit (IA-32) y 64-bit (x86-64).
- NASM es un ensamblador portable diseñado para varios sistemas operativos y distintas versiones de ANSI-C por eso, hay varios formatos de salida.

- Usaremos el **NASM** que es un ensamblador libre para escribir programas de 16-bit, 32-bit (IA-32) y 64-bit (x86-64).
- NASM es un ensamblador portable diseñado para varios sistemas operativos y distintas versiones de ANSI-C por eso, hay varios formatos de salida.
- Dado que vamos a trabajar en **Linux**, vamos a utilizar elfx32 , elf32 y elf64, que generan **formatos de salida ELF32 y ELF64** (Executable and Linkable Format) en los archivos objetos. Más información:
<https://www.nasm.us/xdoc/2.15.05/html/nasmdoc8.html#section-8.9.2>

- El **formato ELF** define que un programa en general se separa en las secciones:

- El **formato ELF** define que un programa en general se separa en las secciones:
 - **.data**: Donde declarar variables globales inicializadas. (DB, DW, DD y DQ).
 - **.rodata**: Donde declarar constantes globales inicializadas. (DB, DW, DD y DQ).
 - **.bss**: Donde declarar variables globales no inicializadas. (RESB, RESW, RESD y RESQ).
 - **.text**: Es donde se escribe el código.

- El **formato ELF** define que un programa en general se separa en las secciones:
 - **.data**: Donde declarar variables globales inicializadas. (DB, DW, DD y DQ).
 - **.rodata**: Donde declarar constantes globales inicializadas. (DB, DW, DD y DQ).
 - **.bss**: Donde declarar variables globales no inicializadas. (RESB, RESW, RESD y RESQ).
 - **.text**: Es donde se escribe el código.
- Y las siguientes etiquetas y símbolos especiales:
 - **global**: Modificador que define un símbolo que va a ser visto externamente.
 - **start**: Símbolo utilizando como punto de entrada de un programa.

Programa de ejemplo

```
section    .text
    global _start
_start:
    mov     edx, len
    mov     ecx, msg
    mov     ebx, 1
    mov     eax, 4
    int     0x80
    mov     eax, 1
    int     0x80
section    .data
    msg     db     'Hello, world!',0xa
    len     equ     $ - msg
```

```
section    .text
```

```
    global _start
```

```
_start:
```

```
    mov     edx, len
```

```
    mov     ecx, msg
```

```
    mov     ebx, 1
```

```
    mov     eax, 4
```

```
    int     0x80
```

```
    mov     eax, 1
```

```
    int     0x80
```

```
section    .data
```

```
msg    db    'Hello, world! ',0xa
```

```
len    equ    $ - msg
```



```
section    .text
```

```
    global _start
```

```
_start:
```

```
    mov     edx, len
```

```
    mov     ecx, msg
```

```
    mov     ebx, 1
```

```
    mov     eax, 4
```

```
    int     0x80
```

```
    mov     eax, 1
```

```
    int     0x80
```

```
section    .data
```

```
msg        db    'Hello, world!',0xa
```

```
len        equ    $ - msg
```

expresión \$ se evalúa en la posición en memoria al principio de la línea que contiene la expresión.

```
section    .text
global _start
_start:
    mov     edx, len
    mov     ecx, msg
    mov     ebx, 1
    mov     eax, 4
    int     0x80
    mov     eax, 1
    int     0x80
section    .data
msg        db    'Hello, world! ', 0xa
len        equ   $ - msg
```

sys_write

parámetros de la llamada al sistema operativo (syscall)

llamada al sistema operativo (syscall)

```
section    .text
global _start
_start:
    mov     edx, len
    mov     ecx, msg
    mov     ebx, 1
    mov     eax, 4
    int     0x80
    mov     eax, 1
    int     0x80
section    .data
msg        db    'Hello, world! ', 0xa
len        equ   $ - msg
```

Diagram illustrating the assembly code for printing to the screen, showing the relationship between the instructions and the system call:

- The instruction `mov eax, 1` sets the system call number to 1.
- The instruction `int 0x80` triggers the system call.
- The parameter 1 is used to call `sys_exit`.
- The instruction `int 0x80` is the system call (syscall) used to invoke the kernel.

Comandos e instrucciones para el ensamblador

- DB, DW, DD, DQ, RESB, RESW, RESD y RESQ.
- expresión \$ se evalúa en la posición en memoria al principio de la línea que contiene la expresión.
- comando EQU, para definir constantes que después no quedan en el archivo objeto.
- comando INCBIN, incluye un binario en un archivo assembler.
- prefijo TIMES, repite una cantidad de veces la instrucción que le sigue.

Cuando trabajen en sus computadoras con el NASM para correr el programa deberán correr desde la terminal:

- Ensamblar: `nasm -f elf64 -g -F DWARF holamundo.asm`

Cuando trabajen en sus computadoras con el NASM para correr el programa deberán correr desde la terminal:

- Ensamblar: `nasm -f elf64 -g -F DWARF holamundo.asm`
- Linkear: `ld -o holamundo holamundo.o`

Cuando trabajen en sus computadoras con el NASM para correr el programa deberán correr desde la terminal:

- Ensamblar: `nasm -f elf64 -g -F DWARF holamundo.asm`
- Linkear: `ld -o holamundo holamundo.o`
- Ejecutar: `./holamundo`

Para la próxima semana

1. Completar las actividades vistas en la clase ambos enunciados
2. Armar una máquina con sistema operativo Linux (pueden tener booteo dual o usar máquina virtual)
3. Completar los ejercicios individuales de programación en Assembler de tarea y subirlos al Git