Organización del Computador II

Orga2

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

19-03-2019

Bienvenidos a ORGA II

Profesor

- Alejandro Furfaro

JTPs

- David Gonzalez Marquez

Ay1

- Rodolfo Sumoza

Ay2

- Daniel Kundro
- Facundo Ruiz
- Gonzalo Fernández Florio
- Belén Ticona







Clases

Teóricas

Jueves de 17 a 19 hs
Aula 8

Prácticas

Martes de 17 a 22 hs
Aula 8
Jueves de 19 a 22 hs
Laboratorio 6 y Turing

Evaluación

Trabajos Prácticos

TP1: Indiv. - 11/04 - 30/04
TP2: Grupal - 09/05 - 11/06
TP3: Grupal - 20/06 - 16/07

Parciales

1er Parcial: 07/05
2do Parcial: 18/06
1er Recuperatorio: 02/07
2do Recuperatorio: 11/07

Régimen de Aprobación

Parciales

Calificaciones: I (0 a 59), A- (60 a 64) y A (65 a 100)
No pueden aprobar con A- ambos parciales
Los recuperatorios tienen 2 notas: I (0 a 64) y A (65 a 100)

Trabajos Prácticos

Calificaciones: I, A
TP1 → individual (sin informe)
TP2 y TP3 → (con informe) grupos de 3 personas
Entregas mediante GIT

Aprobar Trabajos Prácticos

Aprobar parciales
Aprobar TPs

$\Rightarrow$

Materia

Aprobar Final

Regímenes especiales

Trabajo Práctico Final

Más de 70 en ambos parciales (no recuperatorios)
y habiendo aprobado los trabajos prácticos en primera instancia
Posibilidad de hacer un tp final.

Extensión de Aprobación de TPs

Tener aprobados los 3 TPs
Mediante solicitud y coloquio individual
Se salvan los Tps **por un sólo cuatrimestre**

Medios de Comunicación

- **Página de la materia**
  `https://campus.exactas.uba.ar/`

- **Lista de docentes**
  `orga2-doc@dc.uba.ar`
  Consultas, sugerencias, quejas, agradecimientos, insultos, etc

- **Lista de alumnos**
  `orga2-alu@dc.uba.ar`
  Uso casi exclusivo para envío de mensajes a los alumnos
  Vale el "busco grupo" o "el sabado por la noche sale tp"

- **No oficial**
  `#Orga2 @ freenode.net` (IRC)
  `https://t.me/joinchat/Cy7ktORKuAeYfsyffvTR2A` (Telegram)

Bibliografía

Arquitectura

- Manuales de Intel
  (se los pueden bajar de la página de la materia)
- The Unabridged Pentium 4: IA32 Processor Genealogy
  MindShare, Tom Shamley, INC. Addison-Wesley
- Computer Architecture: A Quantitative Approach, 4th Edition
  John L. Hennessy , David A. Patterson

Interacción con lenguajes de alto nivel

- Thinking in C, Volumen 1; Bruce Eckel; Mindview, Inc.
- Programming Languages: Design and Implementation, 4/E; Terrence W. Pratt, Marvin V. Zelkowitz

¡Bienvenidos!

Ejercicio

Describa en sus palabras las funciones de las siguientes aplicaciones:

- Compilador
- Ensamblador
- Linker

Ejercicio

Describa en sus palabras las funciones de las siguientes aplicaciones:

- Compilador
- Ensamblador
- Linker

- **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.

Ejercicio

Muestre cómo se almacenan en memoria los siguientes datos en procesadores Big-Endian y Little-Endian:

```
DB 12h                DD 12345678h
DB 12h, 34h           DD 12345678h, 9ABCDEF1h
DW 1234h              DQ 123456789ABCDEF1h
DW 1234h, 5678h       DB '1234'
```

Ejercicio

Muestre cómo se almacenan en memoria los siguientes datos en procesadores Big-Endian y Little-Endian:

```
DB 12h                DD 12345678h
DB 12h, 34h           DD 12345678h, 9ABCDEF1h
DW 1234h              DQ 123456789ABCDEF1h
DW 1234h, 5678h       DB '1234'
```

`DB`, `DW`, `DD`, `DQ`: Pseudo-instrucciones para el ensamblador que indica cómo definir datos en el archivo objeto.
NO se ejecutan por la CPU, las interpreta el ensamblador.

Big Endian: el byte más significativo en la posición de memoria menos significativa.
Little Endian: el byte más significativo en la posición de memoria mas significativa.

`DB 12h`	`-|12|-` big endian `-|12|-` little endian
`DB 12h, 34h`	`-|12|34|-` big endian `-|12|34|-` little endian
`DW 1234h`	`-|12|34|-` big endian `-|34|12|-` little endian
`DW 1234h, 5678h`	`-|12|34|56|78|-` big endian `-|34|12|78|56|-` little endian
`DD 12345678h`	`-|12|34|56|78|-` big endian `-|78|56|34|12|-` little endian
`DD 12345678h, 9ABCDEF1h`	`-|12|34|56|78|9A|BC|DE|F1|-` big endian `-|78|56|34|12|F1|DE|BC|9A|-` little endian
`DQ 123456789ABCDEF1h`	`-|12|34|56|78|9A|BC|DE|F1|-` big endian `-|F1|DE|BC|9A|78|56|34|12|-` little endian
`DB '1234'`	`-|31|32|33|34|-` big endian `-|31|32|33|34|-` little endian

Ejercicio

¿Cuál es el rango de representación de los números enteros sin signo con 8, 16 y 32 bits de precisión? ¿Cuál es el rango de representación de los números enteros en complemento a dos con 8, 16 y 32 bits de precisión?

Ejercicio

¿Cuál es el rango de representación de los números enteros sin signo con 8, 16 y 32 bits de precisión? ¿Cuál es el rango de representación de los números enteros en complemento a dos con 8, 16 y 32 bits de precisión?

Sin signo	0 a $2^n - 1$
Con signo	-2^{n-1} a $2^{n-1} - 1$

	Sin signo	Con signo
8	0 a 255	−128 a 127
16	0 a 65535	−32768 a 32767
32	0 a 4294967295	−2147483648 a 2147483647

Ejercicio

Exprese los números 133 y 123 en notación binaria con 8 bits de precisión (notación sin signo), y realice la suma de estos dos números bit a bit. Luego, exprese los números -123 y 123 en notación complemento a dos con 8 bits de precisión y realice la suma de estos dos números bit a bit. ¿Qué conclusión puede sacar al observar el resultados de ambas operaciones?

Ejercicio

Exprese los números 133 y 123 en notación binaria con 8 bits de precisión (notación sin signo), y realice la suma de estos dos números bit a bit. Luego, exprese los números -123 y 123 en notación complemento a dos con 8 bits de precisión y realice la suma de estos dos números bit a bit. ¿Qué conclusión puede sacar al observar el resultados de ambas operaciones?

```
  123 = 01111011
 -123 = 10000101
-----------------
      100000000
```

```
  123 = 01111011
  133 = 10000101
-----------------
      100000000
```

Esa es la razón por la cual no hay dos ADD/SUB,
sino uno solo tanto para números con signo como sin signo.

Es responsabilidad del programador saber con qué tipo de números se está operando, y prestar atención a los flags correctos.

Ejercicio

Explique qué indican y cuándo se setean los flags de paridad (PF), de cero (ZF) y de signo (SF). Explique las diferencias entre el flag de carry (CF) y el flag de overflow (OF).

Importante: Los flags se setean dependiendo de la operación. La interpretación depende del programador.

Ejercicio

Explique qué indican y cuándo se setean los flags de paridad (PF), de cero (ZF) y de signo (SF). Explique las diferencias entre el flag de carry (CF) y el flag de overflow (OF).

Importante: Los flags se setean dependiendo de la operación. La interpretación depende del programador.

CF = 1	Bit más significativo en la suma. En la resta si hay *borrow*.
CF = 0	cualquier otro caso
OF = 1	Si hay overflow (el resultado esta fuera de la representación)
OF = 0	cualquier otro caso
PF = 1	Si el byte menos significativo tiene un número par de 1s
PF = 0	cualquier otro caso
SF = 1	Si el bit más significativo es 1
SF = 0	cualquer otro caso
ZF = 1	Si el resultado es cero
ZF = 0	cualquier otro caso

Ejercicio

Indique cuáles son las condiciones para que se activen las siguientes instrucciones de salto: JA, JAE, JB, JBE, JE, JG, JGE, JL, JLE y JZ.

Ejercicio

Indique cuáles son las condiciones para que se activen las siguientes instrucciones de salto: JA, JAE, JB, JBE, JE, JG, JGE, JL, JLE y JZ.

JA	CF=0 and ZF=0	Above
JAE	CF=0	Above or Equal
JB	CF=1	Below
JBE	CF=1 or ZF=1	Below or Equal
JE	ZF=1	Equal
JG	ZF=0 and SF=OF	Greater (signed)
JGE	SF=OF	Greater or Equal (signed)
JL	SF != OF	Less (signed)
JLE	ZF=1 or SF != OF	Less or Equal (signed)
JZ	ZF=1	Zero

Instrucciones y Registros

Operaciones

`ADD`, `SUB`, `MOV`, `SHL`, `JMP` ...

Instrucciones y Registros

Operaciones

`ADD`, `SUB`, `MOV`, `SHL`, `JMP` ... (ver manual)

Instrucciones y Registros

Operaciones

`ADD`, `SUB`, `MOV`, `SHL`, `JMP` ... (ver manual)

Registros

8 bits:	AL	BL	CL	DL	DIL	SIL	BPL	SPL	R8B	...	R15B
16 bits:	AX	BX	CX	DX	DI	SI	BP	SP	R8W	...	R15W
32 bits:	EAX	EBX	ECX	EDX	EDI	ESI	EBP	ESP	R8D	...	R15D
64 bits:	RAX	RBX	RCX	RDX	RSI	RDI	RBP	RSP	R8	...	R15

128 bits: XMM0, ... , XMM15

Instrucciones y Registros

Operaciones

`ADD`, `SUB`, `MOV`, `SHL`, `JMP` ... (ver manual)

Registros

8 bits:	AL	BL	CL	DL	DIL	SIL	BPL	SPL	R8B	...	R15B
16 bits:	AX	BX	CX	DX	DI	SI	BP	SP	R8W	...	R15W
32 bits:	EAX	EBX	ECX	EDX	EDI	ESI	EBP	ESP	R8D	...	R15D
64 bits:	RAX	RBX	RCX	RDX	RSI	RDI	RBP	RSP	R8	...	R15

128 bits: XMM0, ... , XMM15

Direccionamiento

$$\left[\ \underbrace{\begin{matrix}\texttt{Base}\\ \hline \texttt{RAX}\\ \texttt{...}\\ \texttt{R15}\end{matrix}}\ +\ \left(\ \underbrace{\begin{matrix}\texttt{Index}\\ \hline \texttt{RAX}\\ \texttt{...}\\ \texttt{R15}\end{matrix}}\ *\ \underbrace{\begin{matrix}\texttt{Scale}\\ \hline 1\\ 2\\ 4\\ 8\end{matrix}}\ \right)\ +\ \underbrace{\begin{matrix}\texttt{Displacement}\\ \hline \texttt{Cte. 32 bits}\end{matrix}}\ \right]$$

Hola Mundo ...

Ejercicio

Escriba un programa en lenguaje ensamblador que imprima por pantalla:

```
Hola Mundo
```

Hola Mundo ...

Ejercicio

Escriba un programa en lenguaje ensamblador que imprima por pantalla:

```
Hola Mundo
```

¿Cómo?

Secciones, etiquetas y símbolos

Un programa assembler se separa en 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.

Secciones, etiquetas y símbolos

Un programa assembler se separa en 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.

Etiquetas y símbolos

- `global`: Define un símbolo que va a ser visto externamente
- `_start`: Punto de entrada de un programa en linux

Pseudoinstrucciones

Son 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.

Pseudoinstrucciones

Son 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 siguiente.

Llamadas al sistema operativo (syscalls)

Utilizando la famosa `int 0x80` (en Linux) solicitamos al Sistema Operativo que haga algo por nosotros.

Su interfaz es:

1- El número de función que queremos en rax

2- Los parámetros en rbx, rcx, rdx, rsi, rdi y rbp; en ese orden

3- Llamamos a la interrupción del sistema operativo (int 0x80)

4- En general, la respuesta está en rax

Llamadas al sistema operativo (syscalls)

Utilizando la famosa `int 0x80` (en Linux) solicitamos al Sistema Operativo que haga algo por nosotros.

Su interfaz es:

1- El número de función que queremos en rax

2- Los parámetros en rbx, rcx, rdx, rsi, rdi y rbp; en ese orden

3- Llamamos a la interrupción del sistema operativo (int 0x80)

4- En general, la respuesta está en rax

- Mostrar por pantalla (**sys_write**):
  Función **4**
  Parámetro 1: **¿donde?** (1 = stdout)
  Parámetro 2: **Dirección de memoria del mensaje**
  Parámetro 3: **Longitud del mensaje** (en bytes)
- Terminar programa (**exit**):
  Función **1**
  Parámetro 1: **código de retorno** (0 = sin error)

Hola Mundo... solución

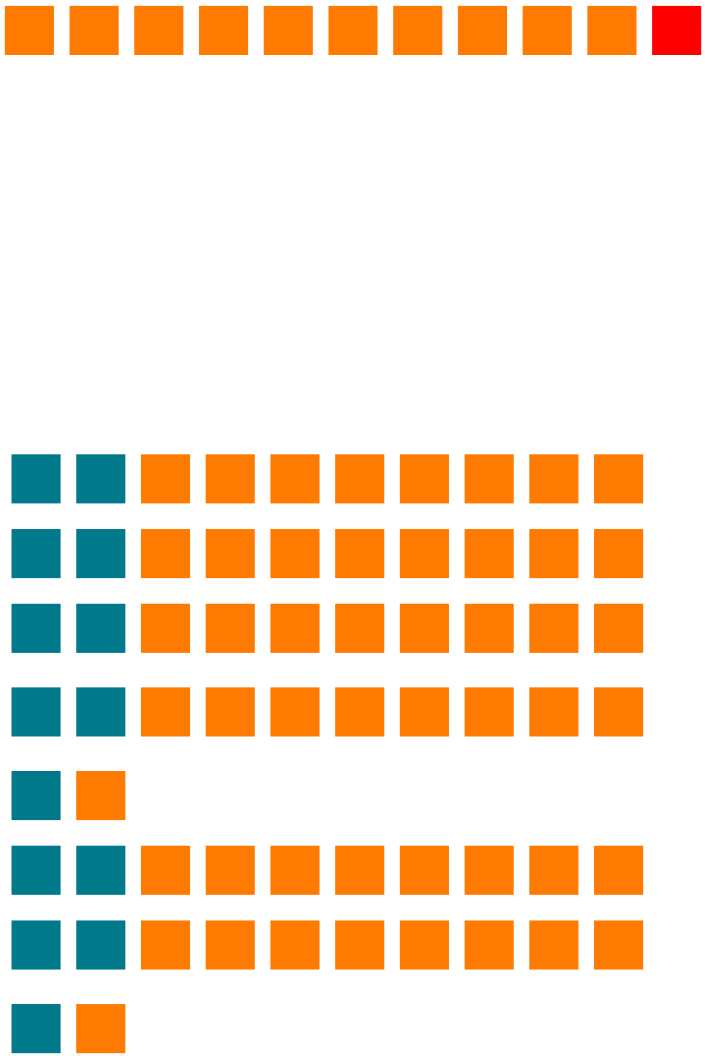
```
section .data
  msg: DB 'Hola Mundo', 10
  largo EQU $ - msg

  global _start
section .text
  _start:
    mov rax, 4      ; funcion 4
    mov rbx, 1      ; stdout
    mov rcx, msg    ; mensaje
    mov rdx, largo  ; longitud
    int 0x80
    mov rax, 1      ; funcion 1
    mov rbx, 0      ; codigo
    int 0x80
```

Hola Mundo... solución

```
section .data
  msg: DB 'Hola Mundo', 10
  largo EQU $ - msg

  global _start
section .text
  _start:
    mov rax, 4       ; funcion 4
    mov rbx, 1       ; stdout
    mov rcx, msg     ; mensaje
    mov rdx, largo   ; longitud
    int 0x80
    mov rax, 1       ; funcion 1
    mov rbx, 0       ; codigo
    int 0x80
```



Ensamblando y linkeando

Ensamblamos:

```
nasm -f elf64 holamundo.asm
```

Linkeamos:

```
ld -o holamundo holamundo.o
```

Ejecutamos:

```
./holamundo
```

Comandos Basicos

`r | run`	Ejecuta el programa hasta el primer break
`b | break FILE:LINE`	Breakpoint en la línea
`b | break FUNCTION`	Breakpoint en la función
`info breakpoints`	Muestra información sobre los breakpoints
`c | continue`	Continúa con la ejecución
`s | step`	Siguiente línea (Into)
`n | next`	Siguiente línea (Over)
`si | stepi`	Siguiente instrucción asm (Into)
`ni | nexti`	Siguiente instrucción asm (Over)
`x/Nuf ADDR`	Muestra los datos en memoria

`N` = Cantidad (bytes)

`u` = Unidad `b|h|w|g`

`b:byte, h:word, w:dword, g:qword`

`f` = Formato `x|d|u|o|f|a`

`x`:hex, `d`:decimal, `u`:decimal sin signo, `o`:octal, `f`:float, `a`:direcciones

GDB

Configuración de GDB:

```
~/.gdbinit
```

Para usar sintaxis intel y guardar historial de comandos:

```
set disassembly-flavor intel
set history save
```

Correr GDB con argumentos:

```
gdb --args <ejecutable> <arg1> <arg2> ...
```

Ejercicio

Escriba un programa en lenguaje ensamblador que imprima por pantalla:

```
¡me quiero ir!
me voy en 1
me voy en 2
me voy en 3
me voy en 4
me voy en 5
me voy en 6
me voy en 7
me voy en 8
me voy en 9
¡¡¡CHAU!!!
```

(OJO! haciendo un ciclo...)

Solución... (incompleta)

```
section .data
    msg: DB 'en 10 me voy ... 9',10
    largo EQU $ - msg
    global _start
section .text
  _start:
    mov esi, 10
    ciclo:
      mov rax, 4      ; funcion 4
      mov rbx, 1      ; stdout
      mov rcx, msg    ; mensaje
      mov rdx, largo  ; longitud
      int 0x80
      dec byte [msg+largo-2]
      dec esi
      cmp esi, 0
    jnz ciclo
    mov rax, 1
    mov rbx, 0
    int 0x80
```

¿Preguntas?

¡Gracias!