

# Práctica 1: Entorno de desarrollo GNU

Gustavo Romero López

Arquitectura y Tecnología de Computadores

30 de septiembre de 2013

# Índice

- 1 Objetivos
- 2 Introducción
- 3 Esqueleto
- 4 Ejemplos
  - Ensamblador
  - Makefile
  - C++
  - 64 bits
  - Ensamblador y C++
- 5 Enlaces

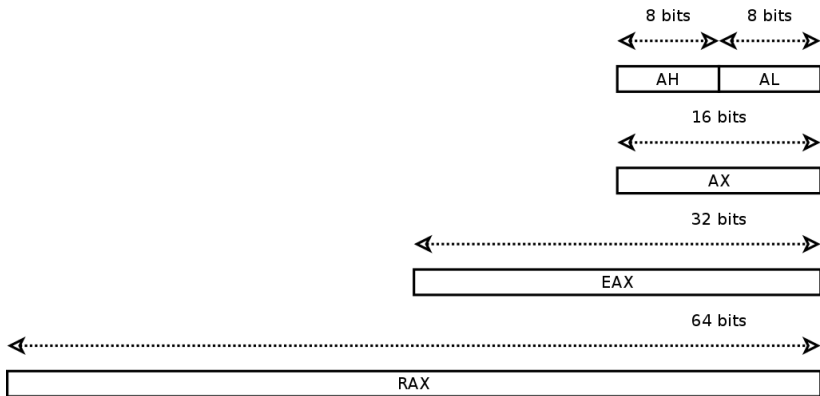
# Objetivos

- **Programar en ensamblador.**
- Linux es tu amigo: si no sabes algo pregunta (**man**).
- Hoy estudiaremos varias cosas:
  - **Esqueleto** de un programa básico en ensamblador.
  - Como aprender de un maestro: **gcc**.
  - Herramientas del entorno de programación:
    - **make**: hará el trabajo sucio y rutinario por nosotros.
    - **as**: el ensamblador.
    - **ld**: el enlazador.
    - **gcc**: el compilador.
    - **nm**: lista los símbolos de un fichero.
    - **objdump**: el desensamblador.
    - **gdb** y **ddd** (gdb con cirugía estética): los depuradores.

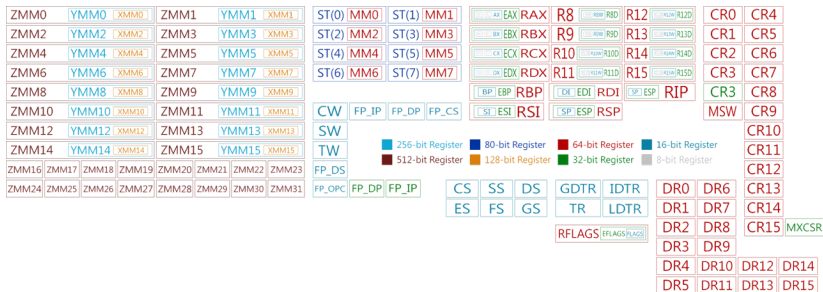
# Ensamblador 80x86

- Los **80x86** son una familia de procesadores.
- Junto con los procesadores tipo ARM son los más utilizados.
- En estas prácticas vamos a centrarnos en su **lenguaje ensamblador** (inglés).
- El lenguaje ensamblador es el más básico, tras el binario, con el que podemos escribir programas utilizando las **instrucciones** que entiende el procesador.
- Cualquier estructura de un lenguaje de alto nivel pueden conseguirse mediante instrucciones sencillas.
- Normalmente es utilizado para poder acceder partes que los lenguajes de alto nivel nos ocultan o hacen de forma que no nos interesa.

# Arquitectura 80x86: registros



# Arquitectura 80x86: registros completos



# Arquitectura 80x86: banderas

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	ID	VIP	VIF	AC	VM	RF	0	NT	IOPL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

X ID Flag (ID)

X Virtual Interrupt Pending (VIP)

X Virtual Interrupt Flag (VIF)

X Alignment Check (AC)

X Virtual-8086 Mode (VM)

X Resume Flag (RF)

X Nested Task (NT)

X I/O Privilege Level (IOPL)

S Overflow Flag (OF)

C Direction Flag (DF)

X Interrupt Enable Flag (IF)

X Trap Flag (TF)

S Sign Flag (SF)

S Zero Flag (ZF)

S Auxiliary Carry Flag (AF)

S Parity Flag (PF)

S Carry Flag (CF)

# Ensamblador desde 0: secciones de un programa

```
1      .data                      # sección de datos
2
3      .text                      # sección de código
```



# Ensamblador desde 0: punto de entrada

```
5  .text                                # sección de código
6      .globl _start                    # punto de entrada
7
8  _start:                              # etiqueta de entrada
```

# Ensamblador desde 0: variables

```
1  .data                                # sección de datos
2      msg: .string "hola, mundo!\n"
3      tam: .int  . - msg
```

# Ensamblador desde 0: código

```
5  .text                                # sección de código
6      .globl _start                    # punto de entrada
7
8  _start:                              # etiqueta de entrada
9      movl    $4, %eax                 # write
10     movl    $1, %ebx                 # salida estandar
11     movl    $msg, %ecx               # cadena
12     movl    tam, %edx                # longitud
13     int     $0x80                    # llamada al sistema
14
15     movl    $1, %eax                 # exit
16     xorl    %ebx, %ebx               # 0
17     int     $0x80                    # llamada al sistema
```

# Ensamblador desde 0: ejemplo básico hola.s

```
1  .data                                # sección de datos
2      msg: .string "hola, mundo!\n"
3      tam: .int  . - msg
4
5  .text                                # sección de código
6      .globl _start                    # punto de entrada
7
8  _start:                              # etiqueta de entrada
9      movl    $4, %eax                 # write
10     movl    $1, %ebx                 # salida estandar
11     movl    $msg, %ecx               # cadena
12     movl    tam, %edx                # longitud
13     int     $0x80                    # llamada al sistema
14
15     movl    $1, %eax                 # exit
16     xorl    %ebx, %ebx               # 0
17     int     $0x80                    # llamada al sistema
```

# ¿Cómo hacer ejecutable mi programa?

¿Cómo hacer ejecutable el código anterior?

- opción a: ensamblar + enlazar
  - `as hola.s -o hola.o`
  - `ld hola.o -o hola`
- opción b: compilar = ensamblar + enlazar
  - `gcc -nostdlib hola.s -o hola`
- opción c: que lo haga alguien por mi → `make`
  - Makefile: fichero con dos tipos de líneas: definiciones y objetivos.

Ejercicios:

- 1 Cree un ejecutable a partir de `hola.s`.
- 2 Use `file` para ver el tipo de cada fichero.
- 3 Descargue el fichero `Makefile` y modifique la forma de compilación de los ficheros ensamblador.
- 4 Examine el código ejecutable con `objdump -C -D hola`

# Makefile

<http://pccito.ugr.es/~gustavo/ec/practicas/01/Makefile>

```
4 S    = $(filter-out printf.s printf2.s, $(
    wildcard *.s))
5 OBJ = $(S:.s=.o)
6 EXE = $(basename $(S) $(SRC) printf printf2)
7
8 all: $(OBJ) $(EXE)
9
10 clean:
11     $(RM) $(EXE) $(OBJ) core.* *~
12
13 %.o: %.s
14     $(AS) $< -o $@
15
16 %: %.o
17     $(LD) $< -o $@
```

# Ejemplo en C++: hola2.cc

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     cout << "hola, mundo!" << endl;
8 }
```

- ¿Qué hace gcc con mi programa?
- La única forma de saberlo es desensamblarlo:
  - Sintaxis AT&T: `objdump -C -D hola2`
  - Sintaxis Intel: `objdump -C -D hola2 -M intel`

Ejercicios:

- 5 Muestre el código de la función `main()`.

# Depuración: hola64.s

```
8  _start:  mov     $0x0a202020216f646e, %rax # "ndo!\n  "  
9          push    %rax  
10         mov     $0x756d202c616c6f68, %rax # "hola, mu"  
11         push    %rax  
12         xor     %rax, %rax  
13         mov     %rsp, %rsi  
14         mov     $1, %rdi  
15         mov     $16, %rdx  
16         call    write
```

## Ejercicios:

- 6 Descargue hola64.s. Ejecute el programa instrucción por instrucción con el ddd hasta comprender como funciona. Podemos destacar: código de 64 bits, uso de “*little endian*”, llamada a subrutina, uso de la pila y codificación de caracteres.



# Mezclando lenguajes: ensamblador y C $\longrightarrow$ printf.s

```
16 main:    pushl   %ebp           # conserva ebp
17          movl    %esp, %ebp     # copia pila
18
19          pushl   i              # apila i
20          pushl   $f             # apila f
21          call    printf         # llamada a función
22          addl    $8, %esp       # restaura pila
```

## Ejercicios:

- 1 Descargue printf.s. Compile siguiendo las instrucciones de su interior o modifique el fichero Makefile.
- 2 Modifique el programa de forma que finalice llamando a la función `exit()` de la biblioteca de C en lugar de usar directamente la llamada al sistema `exit` de Linux. Con `man 3 exit` puede obtener información sobre `exit()`.

# Enlaces de interés

## Manuales:

- Hardware:
  - AMD
  - Intel
- Software:
  - AS
  - NASM

## Programación:

- Programming from the ground up
- Linux Assembly

## Chuletas:

- Chuleta del 8086
- Chuleta del GDB