Lenguaje Ensamblador

Fundamentos del Hardware

Adrián Pisabarro
@adriantxupisi

Lenguaje Máquina

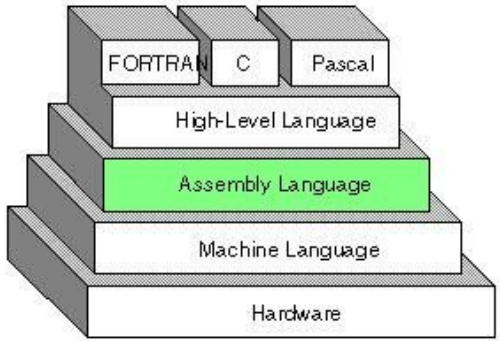
Lenguaje Máquina

- Los procesadores realizan instrucciones con bits (0/1)
- Paradigma de los lenguajes:

Lenguaje interpretado
.java → VMJava → 0/1

Lenguaje compilado
.c → 0/1

Lenguaje Máquina y Lenguaje Ensamblador



Lenguaje Máquina

- Las instrucciones son entendidas por un circuito microprogramable.
- Las instrucciones se basan en un cambio de flujo causado por el propio programa.

```
-u 100 1a
0CFD:0100 BA0B01      MOV     DX,010B
0CFD:0103 B409        MOV     AH,09
0CFD:0105 CD21        INT     21
0CFD:0107 B400        MOV     AH,00
0CFD:0109 CD21        INT     21
-d 10b 13f
0CFD:0100                                48 6F 6C 61 2C          Hola,
0CFD:0110 20 65 73 74 65 20 65 73-20 75 6E 20 70 72 6F 67  este es un prog
0CFD:0120 72 61 6D 61 20 68 65 63-68 6F 20 65 6E 20 61 73  rama hecho en as
0CFD:0130 73 65 6D 62 6C 65 72 20-70 61 72 61 20 6C 61 20  sembler para la
0CFD:0140 57 69 6B 69 70 65 64 69-61 24                    Wikipedia$
```

Wikipedia

Lenguaje Ensamblador

- Derivado del lenguaje a bajo nivel.
- Conjuntos mnemotécnico/“mnemónicos” que representan instrucciones básicas.
- Forma de programar una arquitectura de un procesador.
- Cada arquitectura de procesador tiene su propio lenguaje ensamblador, está definido por cada fabricante.

Algunas Arquitecturas

CISC (Complex Instruction Set Computer)

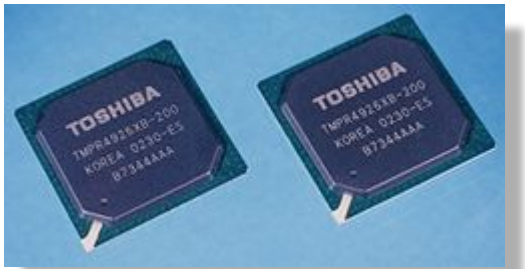
- Se caracteriza por :
  - La rapidez de las operaciones.
  - Realizar operaciones complejas.
  - Operandos registrados en memoria y en registros internos.
  - Incluir RISCs en su interior

CISC pertenece a la Arquitectura x86 utilizada actualmente.

RISC (Reduced Instruction Set Computer)

- Se caracteriza por :
  - Instrucciones de tamaño fijo y reducido formato número de formatos
  - Sólo instrucciones de:
    - Carga
    - Almacenamiento
    - Acceden a memoria de datos.
  - Procesadores con muchos registros (por lo general)

RISC equivale a un “módulo” interno de CISC.



Arquitectura x86

- Utilizado en microarquitecturas de PCs
- Basados en 8, 16, 32, 64 bits...

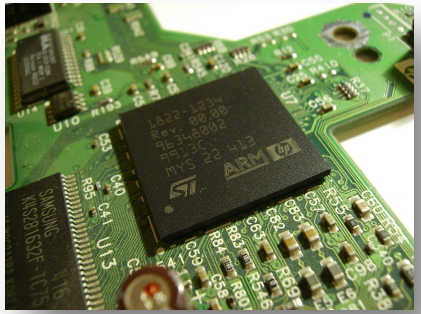
Recurso muy relevante:
Linux Problem 2038 Date

El procesador de tu ordenador ahora mismo.

Arquitectura ARM

- Bits 32, 64.
- Basado en RISC, basado en una cantidad menor de transistores tales como CISC.
- Ideal para aplicaciones de baja potencia.
- Bajo consumo, pequeños y relativamente a bajo costo.

El procesador de tu móvil ahora mismo.
Calculadoras, relojes inteligentes, videoconsolas, ...
Próximamente el procesador: Apple Silicon
Wikipedia



Instrucciones

De declaración

cargar

LD r1, H’0000

(instrucción registro, valor)

LD

Cargar los datos de memoria de este hueco dentro del registro 1, en este caso

almacenar

ST H’0000, r1

(instrucción registro, valor)

ST

Almacena los datos del registro 1, en este caso

carga inmediata baja

LLI r1, H’51

(instrucción registro, valor)

LLI

Carga los datos en el registro 1, en este caso.
Únicamente carga el bit menos significativo.

Example

r1 valdrá → 0051

carga inmediata alta

LHI r1, H'51

(instrucción registro, valor)

LHI

Carga los datos en el registro 1, en este caso.
Únicamente carga el bit más significativo.

Example

r1 valdrá → 5100

Control

puertos de salida

OUT OP01, r1
OUT OP02, r1

(instrucción puerto_de_salidanumsalida, registro)

OP

Muestra el valor del registro, en este caso en la salida 2

Example

r1 valdrá → H'5123

```
OUT OP01, r1
//OP1: 5123
```

entrada estándar

IN r1, IP01

(instrucción registro, puerto de salida)

IN

Introducimos el valor que tenemos en la entrada estándar al registro 1.

Example

```
previo valor → IP1: 4321
       IN r1, IP01
   r1 valor → H'4321
```

salida

HALT
(instrucción)

HALT
Parar ejecución.

terminar script

END
(marcado)

END
Simboliza el final del programa.

Comentarios

Forma de que comprendamos qué estamos haciendo.

forma 1

*

example

* esto es un comentario de una línea
* y con otro asterisco he hecho otro comentario de otra línea

forma 2

sobre la instrucción, al final de la misma línea ;
(no recomendado)

example

```
ADD D0,D1 ; Comentario sobre la instrucción ADD
```

Operaciones aritmético-lógicas

suma

ADDS ra, r2, r3

(instrucción registro final, registro sumando, registro sumando)

ADDS

Suma el valor del r3 y r2 y lo inserta en el ra

Example

r2→ 0001

r3 → 0000

ra → 0001

(bit a bit)

resta

SUBS ra, r2, r3
(instrucción registro final, registro minuendo, registro minuendo)

SUBS
Resta el valor del r3 y r2 y lo inserta en el ra

Example
r2→ 0001
r3 → 0000
ra → 0001
(bit a bit)

NAND

NAND ra, r2, r3
(instrucción registro final, registro, registro)

INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

NAND

Realiza el NOT AND (negación de suma) de bits (0/1)
el valor del r3 y r2 y lo inserta en el ra

Example

r2→ 0001
r3 → 0000
ra → 0001
(bit a bit)

Ejercicios Propuestos

Soluciones:
https://drive.google.com/file/d/1XQbWlcZETVPGFe9u28qwjFZcvHfMttxY/view?usp=sharing

Yo doy el permiso, en cuanto se me pida acceso a la carpeta.

lets chope code! 🤯

01.- Realiza un programa que contenga el registro A, el resultado de la suma del registro 2 y el registro 3.

Nota: el registro hay que introducirlo por la instrucción de bit menos y más significativo.

Registro 2: valor → H'1278
Registro 3: valor → H'1101

lets chope code! 🤯

02.- Realiza un programa que contenga el registro A, el resultado de la resta del registro 2 y el registro 3.

Nota: el registro hay que introducirlo por la instrucción de bit menos y más significativo.

Registro 2: valor → H'1278
Registro 3: valor → H'1101

lets chope code! 🤯

03.- Realiza un programa que muestre por pantalla el registro 1, el resultado de la resta del registro 2 y el registro 3.

Nota: el registro hay que imprimirlo por la salida `OP01`.
Nota: el registro hay que introducirlo por la instrucción de bit menos y más significativo.

Registro 2: valor → `H'1278`
Registro 3: valor → `H'1101`

lets chope code! 🤯

04.- Realiza un programa que muestre por pantalla el registro 1, el resultado de la resta del registro 0 y el registro 2.

Nota: el registro hay que imprimirlo por la salida `OP01.`
Nota: el registro r0 se introducirá con LLI y LHI.
Nota: el registro r2 se introducirá con entrada estándar.

Registro 0: valor → `0101`
Registro 2: valor → `(el que quieras)`

lets chope code! 🤯

05.- Realiza un programa que realice los siguientes pasos:

- Inserta en el registro D: 00 como bits de menos peso.
- Carga en el registro A: 0006 como carga.
- Carga en el registro A: FF como bits de menos peso.
- Almacena en el registro A: 0006 cómo almacena.

¿Cuánto vale ahora el registro A y D?

lets chope code! 🤯

06.- Realiza un programa que contenga el registro A, el resultado de la resta del registro 2 y el registro 3.

Nota: el registro hay que introducirlo por la instrucción de bit menos y más significativo.
Nota: la resta la realizaremos con puertas lógicas.

Registro 2: valor → H'1278
Registro 3: valor → H'1101

lets chope code! 🤯

07.- Realiza un programa que contenga el registro 1, el resultado de la resta del registro 0 y el registro 2.

Nota: los registros hay que introducirlos por la instrucción de bit menos y más significativo.
Nota: la instrucción la podemos hacer con la resta simple.

Registro 0: valor → H'E03B
Registro 3: valor → H'4C54

Nuevas instrucciones

comienzo de programa/dirección

ORG H’0000

(instrucción dirección)

ORG H’XXXX

XXXX equivale a la dirección en la que vamos a comenzar nuestro programa

desplazamiento a la izquierda

SHL r1

(instrucción registro)

SHL rX

Desplaza los números de los registros en hexadecimal hacia la izquierda

Example

0011 → SHL r2 → 1100

desplazamiento a la derecha

SHR r1

(instrucción registro)

SHR rX

Desplaza los números de los registros en hexadecimal hacia la derecha

Example

0111→ SHR r3 → 1011

“subprograma” - subrutina

NOMBRE: SUBS ra, r2, r3

(nombre_sub_programa: instrucción registro final, registro sumando, registro sumando)

NOMBRE: (instrucciones)

Cuando tenemos un programa bastante extenso lo utilizamos.

Example

ETRESTA: SUBS R0,R0,RB

retorno de subrutina

RET

(instrucciÓN)

RET

volver al momento de ejecución del programa donde había hecho un salto ó llamada.

Example

```
        ETFIN:
LLI   RD,lo(SUMP)
LHI  RD,hi(SUMP)
        CALLR
        HALT

SUMP: ADDS RA,RA,R1 ;sumar parcial
        OUT 0,RA
        RET
```

lets chope code! 🤯

08.- Realiza un programa que desplace el registro 0 a la izquierda.

09.- Realiza un programa que desplace el registro 1 a la derecha.

Saltos

Son parecidos a las condiciones
Si para x cosa realizamos un salto de bloque.

retorno

LLI rd, lo(ETRESTA)
LFI rd, hi(ETRESTA)

(instrucciónN)

RET

salto a subprograma (subrutina)

Saltos

"Si se cumple esto hacemos esto"

Example:

1. Cargamos un registro con un valor en memoria
2. Cargamos otro registro con un valor en memoria
3. Si el segundo valor es 0 saltamos al paso 6
4. Dividir el contenido del primer registro entre el segundo registro y colocar el resultado en un tercer registro.
5. Almacena el contenido del tercer registro en memoria
6. Stop

saltos

BR

salto incondicional (no admite condiciones)

BZ

salto si el resultado da 0

BS

salto si el resultado es negativo

BC

salto si el resultado tiene acarreo

BV

salto si hay un desbordamiento (overflow)

Llamadas

Para llamar a ciertos bloques (subrutinas) de mi código

llamadas

CALLR

llamada incondicional a subrutina

*una subrutina es un sub-programa

CALLZ

llamada si el resultado es 0

CALLS

llamada si el resultado es negativo

CALLC

llamadasi el resultado tiene acarreo

CALLV

llamada si hay un desbordamiento (overflow)

lets chope code! 🤯

10.- Realiza un programa utilizando subrutinas que sume el número hasta 9 desde 0.

11.- Realiza un programa que metiendo un 9 en el registro 0, realicemos un bucle hasta que el resultado del registro 0 sea 0.

12.- Realiza una división a base de restas sucesivas
(por ejemplo haz 8 entre 4, el resultado debe de ser 2)

13.- Realiza una multiplicación, por ejemplo 2*4 = 8

14.- Realiza la potencia 2^3 = 8

lets chope code! 🤯

15.- Sigue las siguientes instrucciones:

- Recupera el número almacenado en la dirección de memoria 2000 (no el número 2000, sino el valor contenido en dicha posición) y sitúalo en el registro A.
- Recupera el número almacenado en la dirección de memoria 2001 y sitúalo en un registro B.
- Suma los contenidos de los registros A y B y el resultado lo colocas en un registro C.
- Escribe el valor del registro C en la dirección de memoria 2002.

Resumen

Todas las instrucciones

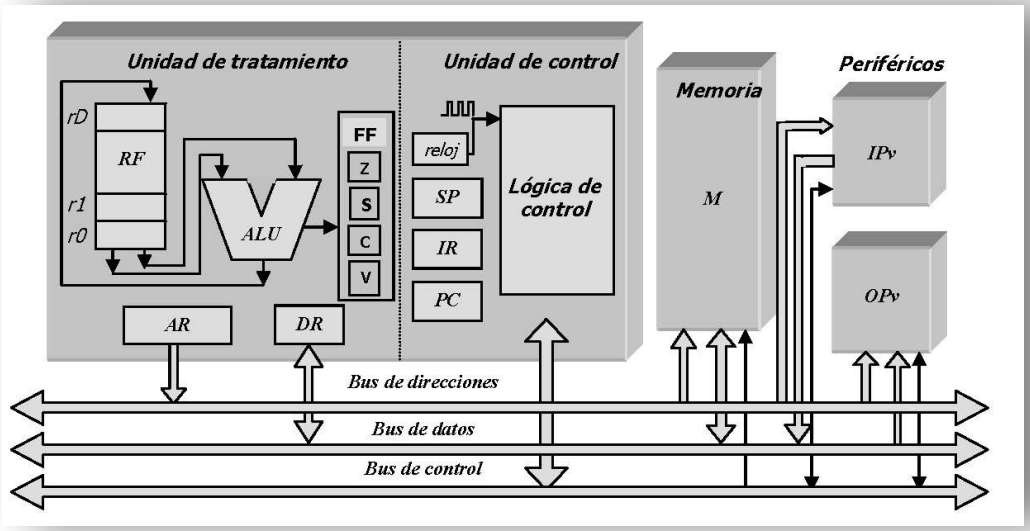
Nombre	Ensamblador	Ejemplo	Significado del ejemplo
Cargar	LD rx,[rD+v]	LD r4,[H'01]	r4 ←M(rD+H'01)
Almacenar	ST [rD+v],rx	ST [H'00],rE	M(rD+H'00)←rE
Carga inmediata baja	LLI rx,v	LLI r7,H'07	r7(15:8)←H'00; r7(7:0)←H'07
Carga inmediata alta	LHI rx,v	LHI r7, H'AB	r7(15:8)←H'AB
Entrada	IN rx,IPv	IN rD,IP1	rD←IP1
Salida	OUT OPv,rx	OUT OPA,r3	OP0D←rx
Suma	ADDS rx,rs,ra	ADDS rD,r3,r4	rD←r3+r4
Resta	SUBS rx,rs,ra	SUBS rF,r1,r0	rF←r1-r0
NAND	NAND rx,rs,ra	NAND r7,r3,r5	r7←(r3·r5)'
Desplaza izquierda	SHL rx	SHL r6	C←r6(15), r6(i)←r6(i-1), i=15,...,1; r6(0)←0
Desplaza derecha	SHR rx	SHR r5	C←r5(0), r5(i)←r5(i+1), i=0,...,14; r5(15)←0
Desplaza arit. dcha.	SHRA rx	SHRA r8	C←r8(0), r8(i)←r8(i+1), i=0,...,14
Salto incondicional	BR	BR	rD←dir_P7; PC←rD
Salto si resultado cero	BZ	BZ	Si Z=1 , rD←dir_alfa; PC←rD
Salto si resultado negativo	BS	BS	Si S=1 , rD←dir_P3; PC←rD
Salto si resultado con acarreo	BC	BC	Si C=1 , rD←dir_a; PC←rD
Salto si desbordamiento	BV	BV	Si V=1 , rD←dir_b; PC←rD
Llamada incondicional a subrutina	CALLR	CALLR	rD←dir_P1, rE←rE-1, M(rE)←PC, PC←rD
Llamada si resultado cero	CALLZ	CALLZ	Si Z=1, rD←dir_a, rE←rE-1, M(rE)←PC, PC←rD
Llamada si resultado negativo	CALLS	CALLS	Si S=1, rD←dir_b, rE←rE-1, M(rE)←PC, PC←rD
Llamada si resultado con acarreo	CALLC	CALLC	Si C=1, rD←dir_f, rE←rE-1, M(rE)←PC, PC←rD
Llamada si desbordamiento	CALLV	CALLV	Si V=1, rD←dir_c, rE←rE-1, M(rE)←PC, PC←rD
Retorno	RET		PC← M(rE); rE←rE+1
Parar	HALT		Parar

Code2

https://atc.ugr.es/docencia/code-2

Code2

- Es un computador didáctico elemental realizado por profesores de la Universidad de Granada.
- Basado en palabras de 16 bits
- Esquema:



Contiene

- **Banco de registros**, compuesto de 16 registros *r0,...,rF*. El registro *rE* se utiliza como **puntero de pila** (*SP*) y el *rD* como **registro de dirección**, para almacenar posiciones de memoria con objeto de realizar direccionamientos indirectos e indexados. A pesar de la dedicación específica de los registros *rE* y *rD*, el programador puede utilizarlos también para otros cometidos.
- **Unidad aritmético lógica** (*ALU*), con la que se pueden realizar las operaciones aritméticas de suma y resta (entero en complemento a 2), la operación lógica *NAND*, desplazamientos lógicos del acumulador a derecha e izquierda y desplazamiento aritmético a la derecha.
- **Biestables indicadores** (*FF*) para gestionar las instrucciones condicionales de salto o de llamadas a subprogramas. Se incluyen **biestables de cero** (*Z*), **signo** (*S*), **acarreo** (*C*) y **desbordamiento** (*V*).
- **Memoria principal** (*M*), de 2E16 = 64 Kpalabras de 16 bits (128 KBytes).
- **Puertos de entrada** (*IP*) y **puertos de salida** (*OP*): admite hasta 256 de cada uno de ellos (*IP00* a *IPFF* y *OP00* a *OPFF*)

Instrucciones

Codop		Nombre	Nemónico	Parámetros	Formato	Explicación	Nº ciclos
binario	Hex						
0000	0	Cargar	LD	rx,[v]	F3	rx←M(rD+v)	9
0001	1	Almacenar	ST	[v],rx	F3	M(rD+v)←rx	9
0010	2	Carga inmediata baja	LLI	rx,v	F3	rx(15:8)←H'00; rx(7:0)←v	6
0011	3	Carga inmediata alta	LHI	rx,v	F3	rx(15:8)←v	8
0100	4	Entrada	IN	rx,IPv	F3	rx←IPv	8
0101	5	Salida	OUT	OPv,rx	F3	OPv←rx	8
0110	6	Suma	ADDS	rx,rs,ra	F4	rx←rs+ra	7
0111	7	Resta	SUBS	rx,rs,ra	F4	rx←rs-ra	7
1000	8	NAND	NAND	rx,rs,ra	F4	rx←(rs·ra)'	7
1001	9	Desplaza izquierda	SHL	rx	F1	C←rx(15), rx(i)←rx(i-1), i=15,...,1; rx(0)←0	6
1010	A	Desplaza derecha	SHR	rx	F1	C←rx(0), rx(i)←rx(i+1), i=0,...,14; rx(15)←0	6
1011	B	Desplaza arit. dcha.	SHRA	rx	F1	C←rx(0), rx(i)←rx(I+1), i=0,...,14	6
1100	C	Salto	B-	cnd	F2	Si cnd se cumple, PC←rD	6
1101	D	Subrutina	CALL-	cnd	F2	Si cnd se cumple, rE←rE-1, M(rE)←PC, PC←rD	6/10
1110	E	Retorno	RET	-	F0	PC← M(rE); rE←rE+1	8
1111	F	Parar	HALT	-	F0	Parar	6

lets chope code! 🤯

Dadas las siguientes instrucciones en hexadecimal tradúcelas a ensamblador (CODE-2)

- `0|1|A7`
- `4|1|02`
- `6|1|D|E`

Solución:

https://drive.google.com/file/d/1FThT25o-t_2JC_zVkZlWH19rCuAAuUMv/view?usp=sharing

Lecturas Recomendadas

Links

- Creación de juegos en Game Boy
- Creación de juegos en Nintendo DS (C)
- Carpeta de Drive de programación de ROM y práctica de juego para NDS en C (proyecto propio para la EHU-Universidad del País Vasco)
  prof.: Iñaki Morlán (@InakiMorlan) ¡recomendado!

Lenguaje Ensamblador

Fundamentos del Hardware