La Máquina ORGA1

Arquitectura y Seguimiento

Orga I

Verano 2018

Arquitectura

La máquina ORGA1 tiene arquitectura de Von Neummann.

Arquitectura

La máquina ORGA1 tiene arquitectura de Von Neummann.

- Procesador (CPU)

Arquitectura

La máquina ORGA1 tiene arquitectura de Von Neummann.

- Procesador (CPU)
- Memoria

Arquitectura

La máquina ORGA1 tiene arquitectura de Von Neummann.

- Procesador (CPU)
- Memoria
- Dispositivos de Entrada/Salida

Arquitectura

CPU

- Unidad Aritmético-Lógica (ALU)
- Flags: Z, N, C, V
- Registros
  - 8 registros de **propósito general** de 16 bits (R0 a R7)
  - 3 registros de **propósito específico** de 16 bits
    - PC (program counter)
    - SP (stack pointer)
    - IR (instruction register)

Arquitectura

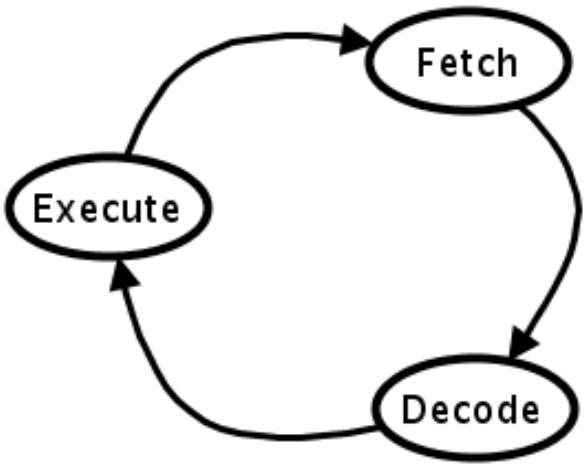
CPU

- Unidad Aritmético-Lógica (ALU)
- Flags: Z, N, C, V
- Registros
  - 8 registros de **propósito general** de 16 bits (R0 a R7)
  - 3 registros de **propósito específico** de 16 bits
    - PC (program counter)
    - SP (stack pointer)
    - IR (instruction register)

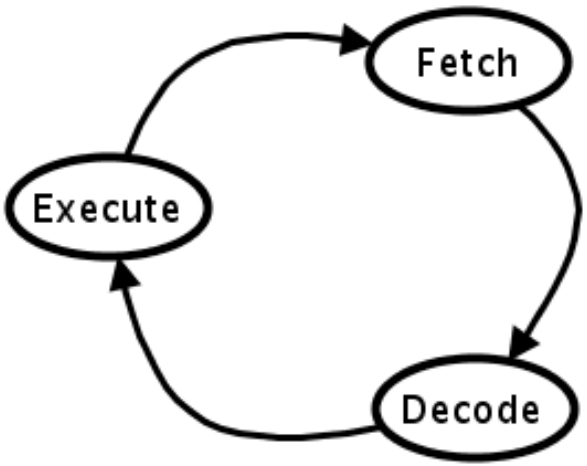
Memoria

- Direcciones de 16 bits (de `0x0000` a `0xFFFF`)
- Palabras de 16 bits
- Direccionamiento a palabra (65536 palabras)

Ciclo de Ejecución

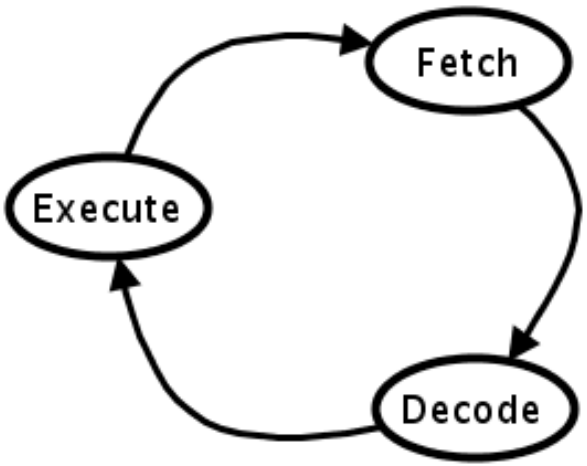


Ciclo de Ejecución



Fetch La UC (unidad de control) obtiene la instrucción de la posición de memoria a la que apunta el PC (y lo incrementa)

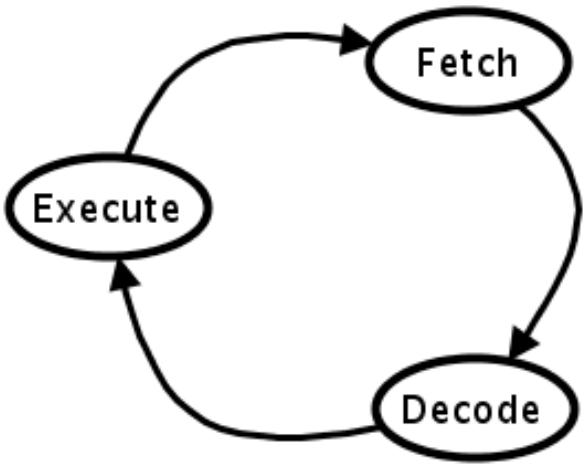
Ciclo de Ejecución



Fetch La UC (unidad de control) obtiene la instrucción de la posición de memoria a la que apunta el PC (y lo incrementa)

Decode La UC decodifica la instrucción

Ciclo de Ejecución



Fetch La UC (unidad de control) obtiene la instrucción de la posición de memoria a la que apunta el PC (y lo incrementa)

Decode La UC decodifica la instrucción

Execute La UC ejecuta la instrucción

Ciclo de Ejecución

1. UC obtiene primer palabra de la instrucción de memoria (de la dirección apuntada por el PC) e incrementa el PC
2. UC decodifica la primer palabra de la instrucción

   Si es necesario: busca más palabras de la instrucción (usando el PC) e incrementa el PC
3. UC decodifica la instrucción completa
4. UC ejecuta la instrucción
5. Ir a Paso 1

Formato de instrucción

Tipo 1: Instrucciones de dos operandos

4 bits	*6 bits*	*6 bits*	*16 bits*	*16 bits*
cod. op.	destino	fuente	constante destino (opcional)	constante fuente (opcional)

operación	cod. op.	efecto	modifica flags
MOV d, f	0001	$d \leftarrow f$	no
ADD d, f	0010	$d \leftarrow d + f$ (suma binaria)	sí
SUB d, f	0011	$d \leftarrow d - f$ (resta binaria)	sí
AND d, f	0100	$d \leftarrow d$ and f	sí (⋆)
OR d, f	0101	$d \leftarrow d$ or f	sí (⋆)
CMP d, f	0110	Modifica los *flags* según el resultado de $d - f$.	sí
ADDC d, f	1101	$d \leftarrow d + f +$ *carry* (suma binaria)	sí

(⋆) dejan el flag de *carry* (C) y el de *overflow* (V) en cero.

Modo	Codificación	Resultado
Inmediato	000000	c16
Directo	001000	[c16]
Indirecto	011000	[[c16]]
Registro	100rrr	Rrrr
Indirecto registro	110rrr	[Rrrr]
Indexado	111rrr	[Rrrr + c16]

c16 es una constante de *16 bits*.

Rrrr es el registro indicado por los últimos tres *bits* del código de operando.

Las instrucciones que tienen como destino un operando de tipo *inmediato* son consideradas como inválidas por el procesador, excepto el CMP.

Formato de instrucción

Tipo 2: Instrucciones de un operando

Tipo 2a: Instrucciones de un operando destino.

4 bits	*6 bits*	*6 bits*	*16 bits*
cod. op.	destino	000000	constante destino (opcional)

operación	cod. op.	efecto	modifica flags
NEG d	1000	$d \leftarrow$ el inverso aditivo de d	s
NOT d	1001	$d \leftarrow$ not d (bit a bit)	s ($\star$)

($\star$) deja el *flag* de *carry* (C) y el de *overflow* (V) en cero.

Tipo 2b: Instrucciones de un operando fuente.

4 bits	*6 bits*	*6 bits*	*16 bits*
cod. op.	000000	fuente	constante fuente (opcional)

operación	cod. op.	efecto	modifica flags
JMP f	1010	PC $\leftarrow f$	no
CALL f	1011	[SP] $\leftarrow$ PC, SP $\leftarrow$ SP $-$ 1, PC $\leftarrow f$	no

Formato de instrucción

Tipo 3: Instrucciones sin operandos

4 bits	*6 bits*	*6 bits*
cod. op.	000000	000000

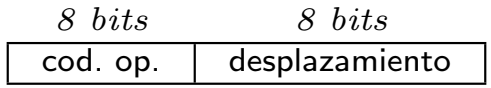
operación	cod. op.	efecto
RET	1100	PC $\leftarrow$ [SP+1], SP $\leftarrow$ SP + 1

- No modifica *flags*.

Formato de instrucción

Tipo 4: Saltos relativos condicionales

- el salto se produce si se cumple la *condición de salto* correspondiente.
- PC $\leftarrow$ PC + *desplazamiento*
- el desplazamiento se representa en *complemento a 2* de *8 bits*.
- No modifican *flags*.



Codop	Operación	Descripción	Condición de Salto
1111 0001	JE	Igual / Cero	Z
1111 1001	JNE	Distinto	not Z
1111 0010	JLE	Menor o igual	Z or (N xor V)
1111 1010	JG	Mayor	not (Z or (N xor V))
1111 0011	JL	Menor	N xor V
1111 1011	JGE	Mayor o igual	not (N xor V)
1111 0100	JLEU	Menor o igual sin signo	C or Z
1111 1100	JGU	Mayor sin signo	not (C or Z)
1111 0101	JCS	Carry / Menor sin signo	C
1111 0110	JNEG	Negativo	N
1111 0111	JVS	Overflow	V

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x1800` *en binario:* `0001 1000 0000 0000`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x1800` *en binario:* `0001 1000 0000 0000`

Decodificamos los bits:

`0001` es un `MOV`

`1000 00` el operando destino es un registro (100) y es el número 000, es decir: `R0`

`00 0000` el operando fuente es un inmediato (son los próximos 16 bits)

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x0002` *en binario:* `0000 0000 0000 0010`

Este es el operando fuente.

Entonces la primer instrucción es `MOV R0, 0x0002`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x1840` *en binario:* `0001 1000 0100 0000`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x1840` *en binario:* `0001 1000 0100 0000`

Decodificamos los bits:

`0001` es un `MOV`

`1000 01` el operando destino es un registro (`100`) y es el número `001`, es decir: `R1`

`00 0000` el operando fuente es un inmediato (son los próximos 16 bits)

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x0001` *en binario:* `0000 0000 0000 0001`
Este es el operando fuente.

Entonces la instrucción es `MOV R1, 0x0001`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x1880` *en binario:* `0001 1000 1000 0000`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x1880` *en binario:* `0001 1000 1000 0000`

Decodificamos los bits:

`0001` es un `MOV`

`1000 10` el operando destino es un registro (`100`) y es el número `010`, es decir: `R2`

`00 0000` el operando fuente es un inmediato (son los próximos 16 bits)

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x0001` *en binario:* `0000 0000 0000 0001`

Este es el operando fuente.

Entonces la instrucción es `MOV R2, 0x0001`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x18E2` *en binario:* `0001 1000 1110 0010`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x18E2` *en binario:* `0001 1000 1110 0010`

Decodificamos los bits:

`0001` es un `MOV`

`1000 11` el operando destino es un registro (100) y es el número 011, es decir: `R3`

`10 0010` el operando fuente es un registro (100) y es el número 010, es decir: `R2`

Entonces la instrucción es `MOV R3, R2`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x28E1` *en binario:* `0010 1000 1110 0001`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x28E1` *en binario:* `0010 1000 1110 0001`

Decodificamos los bits:

`0010` es un `ADD`

`1000 11` el operando destino es un registro (100) y es el número 011, es decir: `R3`

`10 0001` el operando fuente es un registro (100) y es el número 001, es decir: `R1`

Entonces la instrucción es `ADD R3, R1`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x1862` *en binario:* `0001 1000 0110 0010`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x1862` *en binario:* `0001 1000 0110 0010`

Decodificamos los bits:

`0001` es un `MOV`

`1000 01` el operando destino es un registro (100) y es el número 001, es decir: `R1`

`10 0010` el operando fuente es un registro (100) y es el número 010, es decir: `R2`

Entonces la instrucción es `MOV R1, R2`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x18A3` *en binario:* `0001 1000 1010 0011`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x18A3` *en binario:* `0001 1000 1010 0011`

Decodificamos los bits:

`0001` es un `MOV`

`1000 10` el operando destino es un registro (100) y es el número 010, es decir: `R2`

`10 0011` el operando fuente es un registro (100) y es el número 011, es decir: `R3`

Entonces la instrucción es `MOV R2, R3`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x3800` *en binario:* `0011 1000 0000 0000`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x3800` *en binario:* `0011 1000 0000 0000`

Decodificamos los bits:

- `0011` es un `SUB`
- `1000 00` el operando destino es un registro (100) y es el número 000, es decir: `R0`
- `00 0000` el operando fuente es un inmediato (son los próximos 16 bits)

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x0001` *en binario:* `0000 0000 0000 0001`

Este es el operando fuente.

Entonces la instrucción es `SUB R0, 0x0001`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0xFAF9` *en binario:* `1111 1010 1111 1001`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0xFAF9` *en binario:* `1111 1010 1111 1001`

Decodificamos los bits:

`1111 1010` por `1111` sabemos que es un salto condicional, y por `1010` que es un `JG` (mayor)

`1111 1001` es el desplazamiento escrito en 8 bits en complemento a 2. `0xF9` significa -7. Con lo cual esto es un `JG F9`, si salta el `PC` deberá disminuir en 7.

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

`0x0000` *en binario:* `0000 0000 0000 0000`

De memoria...

0000	0001	0002	0003	0004	0005	0006
1800	0002	1840	0001	1880	0001	18E2
0007	0008	0009	000A	000B	000C	000D
28E1	1862	18A3	3800	0001	FAF9	0000

0x0000 *en binario:* 0000 0000 0000 0000

Decodificamos los bits:

0000 es una instrucción inválida. No se corresponde con ningún código de operación.

Entonces es una instrucción inválida.

El código

```
MOV R0, 0x0002
MOV R1, 0x0001
MOV R2, 0x0001
MOV R3, R2
ADD R3, R1
MOV R1, R2
MOV R2, R3
SUB R0, 0x0001
JG F9
```

Si al ejecutar `JG F9` se produce el salto, ¿cuánto vale PC después de ejecutar la instrucción?

El código

```
MOV R0, 0x0002
MOV R1, 0x0001
MOV R2, 0x0001
MOV R3, R2
ADD R3, R1
MOV R1, R2
MOV R2, R3
SUB R0, 0x0001
JG F9
```

Si al ejecutar `JG F9` se produce el salto, ¿cuánto vale PC después de ejecutar la instrucción?

El código

```
        MOV R0, 0x0002
        MOV R1, 0x0001
        MOV R2, 0x0001
do:     MOV R3, R2
        ADD R3, R1
        MOV R1, R2
        MOV R2, R3
        SUB R0, 0x0001
        JG do
```

Podemos reemplazar la dirección por una etiqueta.

El ensamblador después se encargará de calcular las etiquetas, los saltos y traducir

Ensamblador (*Assembler*)

- **Lenguaje** ensamblador $\neq$ **Programa** ensamblador

Ensamblador (*Assembler*)

- **Lenguaje** ensamblador $\neq$ **Programa** ensamblador
- **Lenguaje** ensamblador
  - Lenguaje en el que escribimos programas para Orga1
  - Textual (no confundir con el código de máquina)
  - Permite usar etiquetas

Ensamblador (*Assembler*)

- **Lenguaje** ensamblador $\neq$ **Programa** ensamblador
- **Lenguaje** ensamblador
  - Lenguaje en el que escribimos programas para Orga1
  - Textual (no confundir con el código de máquina)
  - Permite usar etiquetas

- **Programa** ensamblador
  - Traduce programas en lenguaje ensamblador (texto) a código máquina (unos y ceros)
  - También calcula el valor de etiquetas y desplazamientos

El código II

```
        MOV R0, 0x0002
        MOV R1, 0x0001
        MOV R2, 0x0001
do:     MOV R3, R2
        ADD R3, R1
        MOV R1, R2
        MOV R2, R3
        SUB R0, 0x0001
        JG do
```

El código II

```
        MOV R0, 0x0002
        MOV R1, 0x0001
        MOV R2, 0x0001
do:     MOV R3, R2
        ADD R3, R1
        MOV R1, R2
        MOV R2, R3
        SUB R0, 0x0001
        JG do
```

```
a = 2;
b = 1;
c = 1;
do {        d = c;
            d = d + b;
            b = c;
            c = d;
            a = a - 1;
} while     (a > 0);
```

Planilla de seguimiento

PC	Flags				SP	[SP+1]	IR	Instrucción (1er palabra en bits)	PC	fetch 2da palabra	PC	fetch 3ra palabra	Instrucción decodificada	Ejecución
	Z	C	V	N										

0. Anoto el PC de inicio, inicializo flags, SP y [SP+1]

---

1. Cargo IR (IR $\leftarrow$ [PC])
2. Tacho PC (primera columna)
3. Anoto PC+1 (segunda columna PC)
4. Anoto instrucción en *bits*
5. Si necesito otra palabra,
   - 5.1 lo anoto en fetch 2da palabra
   - 5.2 tacho PC (segunda columna)
   - 5.3 anoto PC+1 en tercera columna
6. Si necesito levantar otro operando... (5 bis)
7. Anoto instrucción decodificada
8. Anoto consecuencia de la ejecución en Ejecución, Flags, SP y [SP+1]
9. Tacho último PC anotado y lo copio en la siguiente línea
10. Vuelvo a 1

Ensamblando

Codifiquemos el siguiente programa y carguémoslo desde la posición de memoria 0x0000:

```
main:   MOV R1,[[et1]]
        ADD [R1],0x6000
        JMP main
  et1:  DW 0x0007
  et2:  DW 0x0004
```

Ensamblando

Codifiquemos el siguiente programa y carguémoslo desde la posición de memoria 0x0000:

```
main:   MOV R1,[[et1]]
        ADD [R1],0x6000
        JMP main
  et1:  DW 0x0007
  et2:  DW 0x0004
```

`DW` *(Define Word)*: directiva al ensamblador que provoca que en la posición de memoria que le corresponde, aparezca el valor indicado.

Ensamblando

Codifiquemos el siguiente programa y carguémoslo desde la posición de memoria 0x0000:

```
main:   MOV R1,[[et1]]
        ADD [R1],0x6000
        JMP main
  et1:  DW 0x0007
  et2:  DW 0x0004
```

Tenemos que:

- ver cuántas palabras necesita cada instrucción
- calcular los valores de las etiquetas

`DW` *(Define Word)*: directiva al ensamblador que provoca que en la posición de memoria que le corresponde, aparezca el valor indicado.

Ensamblando

dirección ocupa

```
main:   MOV R1,[[et1]]
        ADD [R1],0x6000
        JMP main
  et1:  DW 0x0007
  et2:  DW 0x0004
```

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	
	ADD [R1],0x6000		
	JMP main		
et1:	DW 0x0007		
et2:	DW 0x0004		

Ensamblando

```
main:   MOV R1,[[et1]]
        ADD [R1],0x6000
        JMP main
  et1:  DW 0x0007
  et2:  DW 0x0004
```

dirección	ocupa
0x0000	dos palabras

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	
	JMP main		
et1:	DW 0x0007		
et2:	DW 0x0004		

Ensamblando

```
main:   MOV R1,[[et1]]
        ADD [R1],0x6000
        JMP main
  et1:  DW 0x0007
  et2:  DW 0x0004
```

dirección	ocupa
0x0000	dos palabras
0x0002	dos palabras

Ensamblando

```
main:   MOV R1,[[et1]]
        ADD [R1],0x6000
        JMP main
  et1:  DW 0x0007
  et2:  DW 0x0004
```

	dirección	ocupa
main:	0x0000	dos palabras
	0x0002	dos palabras
	0x0004	
et1:		
et2:		

Ensamblando

```
main:   MOV R1,[[et1]]
        ADD [R1],0x6000
        JMP main
  et1:  DW 0x0007
  et2:  DW 0x0004
```

dirección	ocupa
0x0000	dos palabras
0x0002	dos palabras
0x0004	dos palabras

Ensamblando

```
main:   MOV R1,[[et1]]
        ADD [R1],0x6000
        JMP main
  et1:  DW 0x0007
  et2:  DW 0x0004
```

	dirección	ocupa
main: MOV R1,[[et1]]	0x0000	dos palabras
ADD [R1],0x6000	0x0002	dos palabras
JMP main	0x0004	dos palabras
et1: DW 0x0007	0x0006	
et2: DW 0x0004		

Ensamblando

```
main:   MOV R1,[[et1]]
        ADD [R1],0x6000
        JMP main
  et1:  DW 0x0007
  et2:  DW 0x0004
```

dirección	ocupa
0x0000	dos palabras
0x0002	dos palabras
0x0004	dos palabras
0x0006	una palabra

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

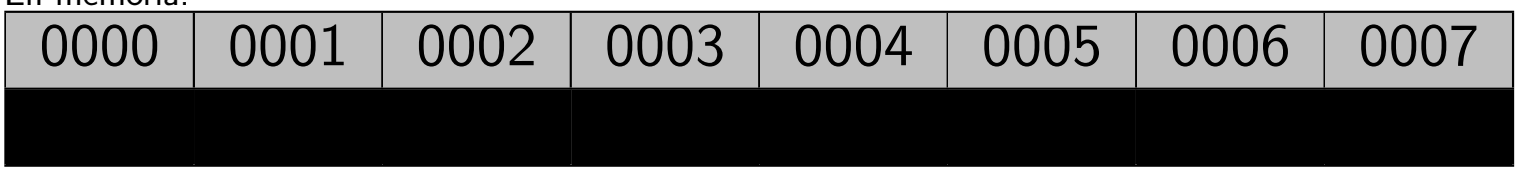
Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

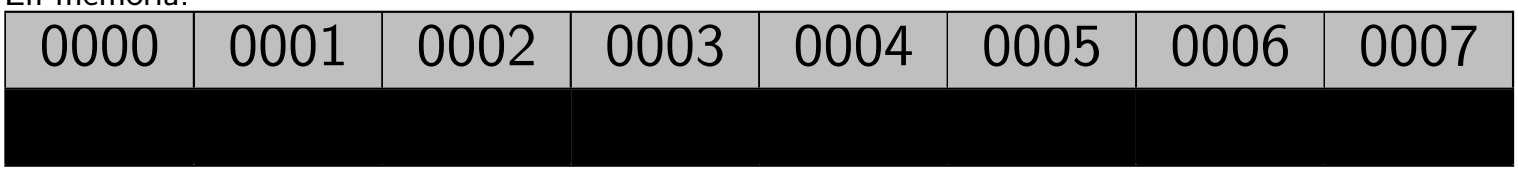
En memoria:



Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

En memoria:



Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006	0007
1858	0006						

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006	0007
1858	0006						

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006	0007
1858	0006	2C40	6000				

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006	0007
1858	0006	2C40	6000				

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006	0007
1858	0006	2C40	6000	A000	0000		

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006	0007
1858	0006	2C40	6000	A000	0000		

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006	0007
1858	0006	2C40	6000	A000	0000	0007	

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006	0007
1858	0006	2C40	6000	A000	0000	0007	

Ensamblando

		dirección	ocupa
main:	MOV R1,[[et1]]	0x0000	dos palabras
	ADD [R1],0x6000	0x0002	dos palabras
	JMP main	0x0004	dos palabras
et1:	DW 0x0007	0x0006	una palabra
et2:	DW 0x0004	0x0007	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006	0007
1858	0006	2C40	6000	A000	0000	0007	0004

Te seguimos...

Empezamos con el PC en la posición de memoria `0x0000`.

0000	0001	0002	0003	0004	0005	0006	0007
1858	0006	2C40	6000	A000	0000	0007	0004

Te seguimos...

Empezamos con el PC en la posición de memoria 0x0000.

0000	0001	0002	0003	0004	0005	0006	0007
1858	0006	2C40	6000	A000	0000	0007	0004

PC	Flags				IR	Instrucción (en bits)	PC	fetch 2da palabra	PC	Instrucción decodificada	Ejecución
	Z	C	V	N							
0000	0	0	0	0	1858	0001 1000 0101 1000	0001	0006	0002	MOV R1, [[OxOOO6]]	R1=[[6]]=[7]=4
0002					2C40	0010 1100 0100 0000	0003	6000	0004	ADD [R1], Ox6OO0	[4]=OxA000+0x6000=0
0004	1	1	0	0	0000	0000 0000 0000 0000	0005	-	-	Instrucción invalida	Fin ejecución

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

```
main:   MOV R7, [0x0004]
        CALL adder
        DW 0x0FE0
adder:  ADD R6, R7
        RET
```

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

		dirección	ocupa
main:	MOV R7, [0x0004]	0x0000	
	CALL adder		
	DW 0x0FE0		
adder:	ADD R6, R7		
	RET		

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

```
main:   MOV R7, [0x0004]
        CALL adder
        DW 0x0FE0
adder:  ADD R6, R7
        RET
```

	dirección	ocupa
	0x0000	dos palabras

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

```
main:   MOV R7, [0x0004]
        CALL adder
        DW 0x0FE0
adder:  ADD R6, R7
        RET
```

	dirección	ocupa
main: MOV R7, [0x0004]	0x0000	dos palabras
CALL adder	0x0002	
DW 0x0FE0		
adder: ADD R6, R7		
RET		

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

```
main:   MOV R7, [0x0004]
        CALL adder
        DW 0x0FE0
adder:  ADD R6, R7
        RET
```

	dirección	ocupa
main: MOV R7, [0x0004]	0x0000	dos palabras
CALL adder	0x0002	dos palabras
DW 0x0FE0		
adder: ADD R6, R7		
RET		

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

```
main:   MOV R7, [0x0004]
        CALL adder
        DW 0x0FE0
adder:  ADD R6, R7
        RET
```

dirección	ocupa
0x0000	dos palabras
0x0002	dos palabras
0x0004	

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

```
main:   MOV R7, [0x0004]
        CALL adder
        DW 0x0FE0
adder:  ADD R6, R7
        RET
```

dirección	ocupa
0x0000	dos palabras
0x0002	dos palabras
0x0004	una palabra

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

```
main:   MOV R7, [0x0004]
        CALL adder
        DW 0x0FE0
adder:  ADD R6, R7
        RET
```

	dirección	ocupa
main: MOV R7, [0x0004]	0x0000	dos palabras
CALL adder	0x0002	dos palabras
DW 0x0FE0	0x0004	una palabra
adder: ADD R6, R7	0x0005	
RET		

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

```
main:   MOV R7, [0x0004]
        CALL adder
        DW 0x0FE0
adder:  ADD R6, R7
        RET
```

	dirección	ocupa
main: MOV R7, [0x0004]	0x0000	dos palabras
CALL adder	0x0002	dos palabras
DW 0x0FE0	0x0004	una palabra
adder: ADD R6, R7	0x0005	una palabra
RET		

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

```
main:   MOV R7, [0x0004]
        CALL adder
        DW 0x0FE0
adder:  ADD R6, R7
        RET
```

dirección	ocupa
0x0000	dos palabras
0x0002	dos palabras
0x0004	una palabra
0x0005	una palabra
0x0006	

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

```
main:   MOV R7, [0x0004]
        CALL adder
        DW 0x0FE0
adder:  ADD R6, R7
        RET
```

dirección	ocupa
0x0000	dos palabras
0x0002	dos palabras
0x0004	una palabra
0x0005	una palabra
0x0006	una palabra

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

```
main:   MOV R7, [0x0004]
        CALL adder
        DW 0x0FE0
adder:  ADD R6, R7
        RET
```

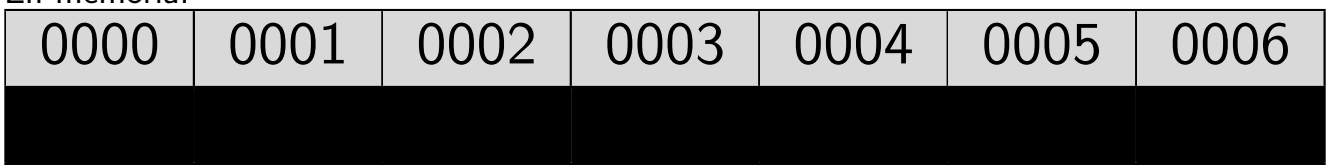
dirección	ocupa
0x0000	dos palabras
0x0002	dos palabras
0x0004	una palabra
0x0005	una palabra
0x0006	una palabra

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

		dirección	ocupa
main:	MOV R7, [0x0004]	0x0000	dos palabras
	CALL adder	0x0002	dos palabras
	DW 0x0FE0	0x0004	una palabra
adder:	ADD R6, R7	0x0005	una palabra
	RET	0x0006	una palabra

En memoria:

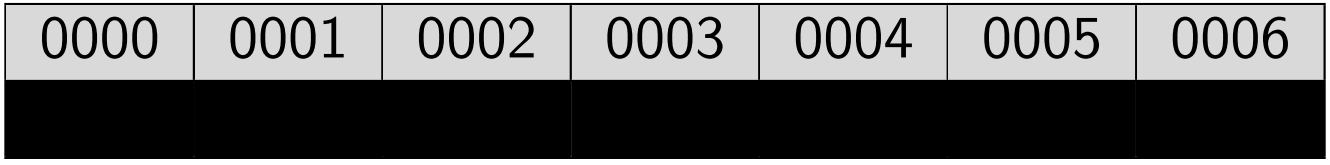


Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

		dirección	ocupa
main:	MOV R7, [0x0004]	0x0000	dos palabras
	CALL adder	0x0002	dos palabras
	DW 0x0FE0	0x0004	una palabra
adder:	ADD R6, R7	0x0005	una palabra
	RET	0x0006	una palabra

En memoria:



Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

		dirección	ocupa
main:	MOV R7, [0x0004]	0x0000	dos palabras
	CALL adder	0x0002	dos palabras
	DW 0x0FE0	0x0004	una palabra
adder:	ADD R6, R7	0x0005	una palabra
	RET	0x0006	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006
19C8	0004					

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

		dirección	ocupa
main:	MOV R7, [0x0004]	0x0000	dos palabras
	CALL adder	0x0002	dos palabras
	DW 0x0FE0	0x0004	una palabra
adder:	ADD R6, R7	0x0005	una palabra
	RET	0x0006	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006
19C8	0004					

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

		dirección	ocupa
main:	MOV R7, [0x0004]	0x0000	dos palabras
	CALL adder	0x0002	dos palabras
	DW 0x0FE0	0x0004	una palabra
adder:	ADD R6, R7	0x0005	una palabra
	RET	0x0006	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006
19C8	0004	B000	0005			

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

		dirección	ocupa
main:	MOV R7, [0x0004]	0x0000	dos palabras
	CALL adder	0x0002	dos palabras
	DW 0x0FE0	0x0004	una palabra
adder:	ADD R6, R7	0x0005	una palabra
	RET	0x0006	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006
19C8	0004	B000	0005			

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

		dirección	ocupa
main:	MOV R7, [0x0004]	0x0000	dos palabras
	CALL adder	0x0002	dos palabras
	DW 0x0FE0	0x0004	una palabra
adder:	ADD R6, R7	0x0005	una palabra
	RET	0x0006	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006
19C8	0004	B000	0005	0FE0		

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

		dirección	ocupa
main:	MOV R7, [0x0004]	0x0000	dos palabras
	CALL adder	0x0002	dos palabras
	DW 0x0FE0	0x0004	una palabra
adder:	ADD R6, R7	0x0005	una palabra
	RET	0x0006	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006
19C8	0004	B000	0005	0FE0		

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

		dirección	ocupa
main:	MOV R7, [0x0004]	0x0000	dos palabras
	CALL adder	0x0002	dos palabras
	DW 0x0FE0	0x0004	una palabra
adder:	ADD R6, R7	0x0005	una palabra
	RET	0x0006	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006
19C8	0004	B000	0005	0FE0	29A7	

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

		dirección	ocupa
main:	MOV R7, [0x0004]	0x0000	dos palabras
	CALL adder	0x0002	dos palabras
	DW 0x0FE0	0x0004	una palabra
adder:	ADD R6, R7	0x0005	una palabra
	RET	0x0006	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006
19C8	0004	B000	0005	0FE0	29A7	

Ensamblando

Codifiquemos el siguiente código y carguémoslo desde 0:

		dirección	ocupa
main:	MOV R7, [0x0004]	0x0000	dos palabras
	CALL adder	0x0002	dos palabras
	DW 0x0FE0	0x0004	una palabra
adder:	ADD R6, R7	0x0005	una palabra
	RET	0x0006	una palabra

En memoria:

0000	0001	0002	0003	0004	0005	0006
19C8	0004	B000	0005	0FE0	29A7	C000

Te seguimos...

Empezamos con el PC en la etiqueta main

es decir, en la posición de memoria `0x0000`.

0000	0001	0002	0003	0004	0005	0006	0007
19C8	0004	B000	0005	0FE0	29A7	C000	0000

Te seguimos...

Empezamos con el PC en la etiqueta main

es decir, en la posición de memoria 0x0000.

0000	0001	0002	0003	0004	0005	0006	0007
19C8	0004	B000	0005	0FE0	29A7	C000	0000

PC	Flags				SP	[SP+1]	IR	Instrucción (en bits)	PC	fetch 2da palabra	PC	Instrucción decodificada	Ejecución
	Z	C	V	N									
0000	0	0	0	0	FFEF	0000	19C8	0001 1001 1100 1000	0001	0004	0002	MOV R7, [0x0004]	R7=[4]=0x0FE0
0002							B000	1011 0000 0000 0000	0003	0005	0004	CALL 0x0005	[SP]=[0xFFEF]=4, SP=0xFFEE, PC=5
0005					FFEE	0004	29A7	0010 1001 1010 0111	0006	-	-	ADD R6, R7	R6=0x0FE0+0=0x0FE0
0006	0	0	0	0			C000	1100 0000 0000 0000	0007	-	-	RET	PC=[0xFFEF]=4, SP=0xFFEF
0004					FFEF	0000	0FE0	0000 1111 1110 0000	0005	-	-	Instrucción inválida	Fin ejecución

Resumen

Hoy:

- arquitectura Orga1
  - ISA: formato de instrucción, modos de direccionamiento
- codificamos, decodificamos, ejecutamos
  - seguimientos

- ciclo de ejecución
- lenguaje ensamblador vs. programa ensamblador
- uso de etiquetas
- directivas

Cómo seguimos

Próxima clase: Taller de ciclo de instrucción.

¡Eso es casi todo amigos!

¿Preguntas?

Ahora sí, ¡Eso es todo amigos!