Ensamblador para Z80

Make Classic Games

Índice

1. Introducción

Hoy en día hay muchos lenguajes de programación que nos ayudan a trabajar tanto con diferentes arquitecturas o diferentes entornos.

Sin embargo, en algunas ocasiones, es necesario poder conocer tanto la arquitectura del procesador en el que estamos trabajando (x86, ARM, Z80, 6502,....) como el conjunto de instrucciones que éste tiene.

Es por ello, que en algunos sistemas es importante utilizar lenguajes de bajo nivel de abstracción, como puede ser el lenguaje ensamblador.

1. Introducción

Abstracción

Llamamos nivel de abstracción de un lenguaje, a cuanto de cerca esta del lenguaje natural. Podemos encontrar 2-3 niveles (dependiendo del autor).

- Alto nivel: trata de abstraernos de las instrucciones del lenguaje máquina y parecerse lo más posible al lenguaje natural. Ejemplo: Java, Python,etc…
- Bajo Nivel: Se acerca bastante al lenguaje máquina ya que depende de su arquitectura y del conjunto de instrucciones que contenga. Ejemplo: Ensamblador.
- “Medio nivel”: Tiene características tanto de alto nivel, pero también pueden usarse instrucciones de bajo nivel. Ejemplo: C.

1. Introducción

El lenguaje ensamblador, es un lenguaje nemotécnico que se basa en el conjunto de instrucciones del procesador.

Es dependiente de la arquitectura del procesador.

La arquitectura del procesador, incluye todos los componentes que lo componen (registros, ALU, instrucciones,etc…)

```
ORG 100h
MOV CX, 5
MOV BX, 1
MOV DL, 2
comienzo:
 MOV AX, BX
 MUL DX
 MOV BX, AX
 LOOP comienzo
 RET
```

2. Procesador Z80

El procesador Zilog Z80, es un procesador de 8 bits con capacidad de trabajar con 16 bits (direcciones). Pudiendo direccionar hasta 16MB.

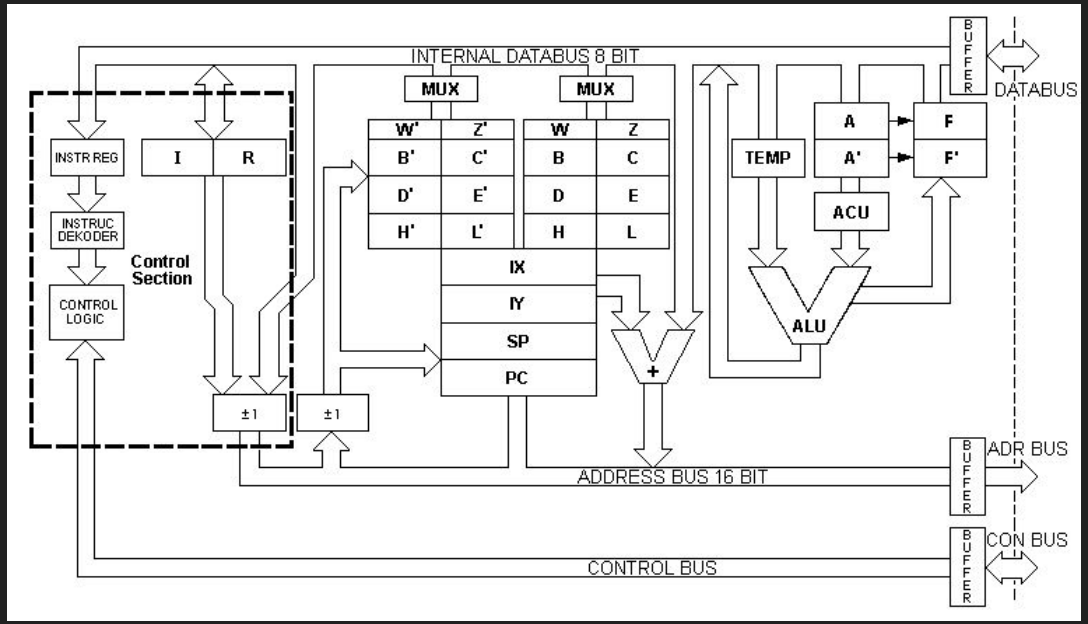
Este procesador fue una mejora del conjunto de instrucciones del 8080.

Fue comercializado en el año 1976, y fue muy utilizado por diferentes máquinas como ordenadores, como consolas e incluso calculadoras.

En el año 2024 (sí; este año); dejó de fabricarse.

2. Procesador Z80

Arquitectura del procesador Z80:



2. Procesador Z80

Este procesador, tiene una ALU, acumulador y una serie de registros (Memoria estática muy rápida de poco tamaño) de propósito general:

- 10 registros de 8 bits de propósito general:
  - A: Registro Acumulador.
  - F: Flags. Se utiliza normalmente para comprobar condiciones.
  - B:
  - C:
  - D:
  - E:
  - H:
  - L:

  } Registros de propósito general.
  - I: Registro de Interrupción.
  - R: Registro de refresco de memoria.

2. Procesador Z80

Registro F (Flags)

El registro F es un registro especial que se modifica cuando hay una operación indicando condiciones o estado del resultado. Cada Bit indica un estado.

- *Bit 0* : Acarreo. Indica 1 cuando “nos llevamos 1” (Acarreo de la suma).
- *Bit 1*: Resta. Indica 1 cuando la última operación fue una resta.
- *Bit 2*: Indica paridad o desbordamiento.
- *Bit 3*: No se utiliza.
- *Bit 4*: Flag H (acarreo BCD). Indica 1 cuando en operaciones BCD existe un acarreo del bit 3 a 4.
- *Bit 5*: No se utiliza.
- *Bit 6*: Flag Z (Cero). Indica 1 si el resultado es igual a 0.
- *Bit 7*: Flag S (Signo). Indica 1 si el complemento a 2 anterior da como resultado un número negativo.

2. Procesador Z80

Este procesador, tiene una serie de registros de propósito general:

- Además, tiene 10 registros alternativos de 8 bits:
  - A’
  - F’
  - B’
  - C’
  - D’
  - E’
  - H’
  - L’

Registros alternativos de propósito general.

2. Procesador Z80

Este procesador, tiene una serie de registros de propósito general:

- También tiene una serie de registros de 16 bits:
  - AF
  - BC
  - DE
  - HL

  } Registros de propósito general.
  - IX: Acceso indexado a memoria.
  - IY: Acceso indexado a memoria.
  - SP: Puntero a pila.
  - PC: Contador de programa.

2. Procesador Z80

Computadoras y consolas que utilizaban el procesador Zilog Z80

- Game Boy /Game Boy Color.
- Sega Master System/Game Gear. (Además de la Sega Mega Drive para orquestación de sonido o modo Master System).
- Amstrad CPC.
- Sinclair ZX Spectrum.
- Calculadoras Texas Instruments.

3. Conjunto de Instrucciones del Z80

Una vez vista la arquitectura, vamos a ver el conjunto de instrucciones que nos provee este procesador.

```
LD A,($4000)
```

Cada instrucción tendrá 1 o varias formas dependiendo si se utiliza cada operando registros, direcciones de memoria, constantes, etc…

```
LD A,($4000)
LD HL,($6000)

LD A,(HL)
```

NOTA: Se puede usar el caracter ‘;’ para comenzar un comentario.

3. Conjunto de Instrucciones del Z80

Hexadecimal y BCD

Habrás podido ver que algunas instrucciones utilizan números de 4 cifras. Estos números están escritos en notación hexadecimal(números del 0 al 9 y letras de la A a la F).

Además como hemos comentado, el procesador Z80 es un procesador que muestra los números en notación “Little Endian”. Esto significa que se muestra siempre los bits menos significativos primero. Además se muestra con notación BC (de 4 en 4 bits). Por ejemplo:

$CCFF se almacena en la memoria en 2 posiciones: $FF en la primera y $CC en la segunda.

3. Conjunto de Instrucciones del Z80

Podemos dividir las instrucciones del procesador Z80 en las siguientes categorías:

- Memoria (Carga).
- Aritméticas.
- Lógicas.
- Incremento y Decremento.
- Salto.
- LLamadas y retorno.
- Entrada/salida.
- Constantes,variables y memoria.
- Inicio y fin de programa.

3. Conjunto de Instrucciones del Z80

Instrucciones de Carga de memoria

Para cargar un dato desde memoria (o una constante), podemos utilizar la instrucción LD. Recordamos que el procesador utilizará la memoria disponible para poder cargar no solo el código a ejecutar; sino también los datos.

Sintaxis:

LD destino, origen

Donde destino puede ser una dirección de memoria o un registro; mientras origen puede ser un registro, posición de memoria o un valor (constante).

Ejemplo:

```
LD hl, $40000

LD (hl), $ff
```

3. Conjunto de Instrucciones del Z80

Instrucciones RST

Las instrucciones RST, permiten ejecutar una dirección de memoria en concreto

Sintaxis:

RST destino

Donde en destino, establecemos la posición de memoria que queremos ejecutar.

Ejemplo:

```
RST $10 ;ejecutamos
        ;la instrucción
        ; en la posición 16.
```

3. Conjunto de Instrucciones del Z80

Instrucciones aritméticas

Las instrucciones aritméticas permiten realizar operaciones con registros o utilizando constantes.

- *ADD:* Suma.
- *SUB*: Resta.
- *ADC*: Suma incluyendo el acarreo.
- *SBC*: Suma del segundo operando y el acarreo, es restado al primer operando.
- *NEG*: Niega el contenido del registro A. A*-1.
- *DAA*: Ajuste decimal del registro A utilizando BCD.

Ejemplos:

```
ADD B,$CC
```

3. Conjunto de Instrucciones del Z80

Instrucciones Lógicas

Las instrucciones Lógicas, permiten realizar las siguientes instrucciones teniendo un origen y como destino siempre el registro A:

- *AND:* Realiza la operación AND a nivel de Bit (solo es 1 si ambos son 1).
- *OR*: Realiza la operación OR a nivel de Bit(Solo es 1 si algún operando es 1).
- *XOR*: Or exclusivo. (Si ambos operandos son iguales devuelve 0; 1 en caso contrario).

Ejemplos:

```
OR H
```

3. Conjunto de Instrucciones del Z80

Instrucciones Incremento y Decremento

Las instrucciones de incremento y decremento, permiten incrementar o decrementar en 1 el valor almacenado en un registro.

- *INC:* Realiza un incremento el valor almacenado en un destino.
- *DEC*: Realiza un decremento el valor almacenado en un destino.

Donde destino, puede ser un registro, posición de memoria, desplazamiento; algunas de estas operaciones pueden ser:

```
INC/DEC r        ;incremento 8bits.
INC/DEC rr       ;incremento 16 bits.
INC/DEC (HL)     ;incremento  de la posición en HL.
INC/DEC (IX+n)   ;incremento de la posición en IX más un desplazamiento.
```

NOTA: cuando se realiza incremento o decremento de 16 bits no cambia el registro F. Si cambia cuando se realiza en operaciones de 8 bits.

3. Conjunto de Instrucciones del Z80

Instrucciones de Salto

Las instrucciones de salto, permiten saltar a una instrucción en concreto dependiendo de una condición; normalmente dependiendo de algún bit del registro F. Podemos encontrar de 2 tipos

- *JP:* Salto absoluto. Salta a una dirección en concreto.
- *JR*: Salto relativo. salta de forma relativa a n bytes en adelante o hacia atras.

Dependiendo del tipo de salto podemos encontrar. Primero para el salto absoluto:

- `JP  nn ; Salto a una dirección en concreto.`
- `JP (HL) ; Salto a la dirección que tiene HL.`
- `JP (Registro índice) ;Salto a la dirección que tiene IX o IY.`
- `JP NZ/Z, nn ;salta a la dirección` *`nn`* `si el flag de Z (cero) no esta a 0 o 1 respectivamente.`
- `JP NC/C, nn ; salta a la dirección de memoria` *`nn`* `si el flag C (Acarreo) está a 0 o 1 respectivamente.`
- `JP PE/PO, nn; salta a la dirección` *`nn`* `si el flag P/V esta a 0 o 1 respectivamente.`
- `JP P/M, nn ; salta a la dirección` *`nn`* `si el flag S (signo) está a 0 o 1 respectivamente.`

3. Conjunto de Instrucciones del Z80

Instrucciones de Salto

Salto relativo:

- `JR  n       ;Salto a una dirección relativa  n bytes.`
- `JR NZ/Z, n  ;Salta a la dirección relativa` *`n`* `bytes si el flag de Z (cero) no está a 0 o 1 respectivamente.`
- `JR NC/C, nn ;Salta a la dirección de memoria` *`n bytes`* `si el flag C (Acarreo) está a 0 o 1 respectivamente.`

3. Conjunto de Instrucciones del Z80

Subrutinas

En muchas ocasiones, el sistema a través de las direcciones de memoria de solo lectura (ROM) o incluso de código que ya hayamos creado, se pueden llamar a subrutinas.

Esto se realiza a través de la instrucción CALL y utiliza el registro del puntero a pila para poder continuar posteriormente con la ejecución.

- `CALL nn ;llama a la subrutina en la dirección nn.`
- `RET     ;finaliza la subrutina y devuelve la ejecución al programa que llamó a dicha subrutina.`

3. Conjunto de Instrucciones del Z80

Entrada/salida

Para poder utilizar los diferentes periféricos como puede ser el teclado o incluso la pantalla o impresora, se pueden utilizar diferentes instrucciones que provee el Z80.

Alguna de ellas son:

- `IN A,nn  ;Carga información desde un registro, hasta un puerto.`
- `OUT (nn),A;Carga desde un puerto hata el registro A.`

Donde *nn*, puede ser una constante que apunte a la dirección de un puerto hardware (por ejemplo joystick, teclado,etc…).

3. Conjunto de Instrucciones del Z80

Constantes, variables y mensajes.

Otro aspecto importante a la hora de trabajar con direcciones de memoria o datos, es que estos se pueden almacenar en memoria y podemos utilizar variables o constantes para tener una etiqueta asociada a cada uno. Estas etiquetas son utilizadas por el programa ensamblador y no afectan al programa.

- *EQU*: crea una constante. Sintaxis: `nombre EQU valor.`
- *DB/DEFB*: define bytes. sintaxis: `nombre DB 1,$FF,%01010101.`
- *DM/DEFM*: define un mensaje. Sintaxis: `nombre DM ‘Hola Mundo’.`
- *DW/DEFW*: define una palabra. Sintaxis: `nombre DW 0040.`
- *DS/DEFS*: define un valor. Sintaxis: `nombre DS $00.`

Normalmente, se definen al final del programa.

3. Conjunto de Instrucciones del Z80

Inicio y fin de programa

Por último y no por ello menos importante, para iniciar y finalizar un programa ensamblador, las instrucciones para iniciar y finalizar un programa.

- *ORG nn*: indica la dirección de inicio del programa.
- *END nn*: indica donde finaliza el programa.

4. Entorno de desarrollo

Una vez visto el procesador Z80, su arquitectura y el conjunto de instrucciones, vamos a hablar del conjunto de herramientas que podemos necesitar para trabajar con este.

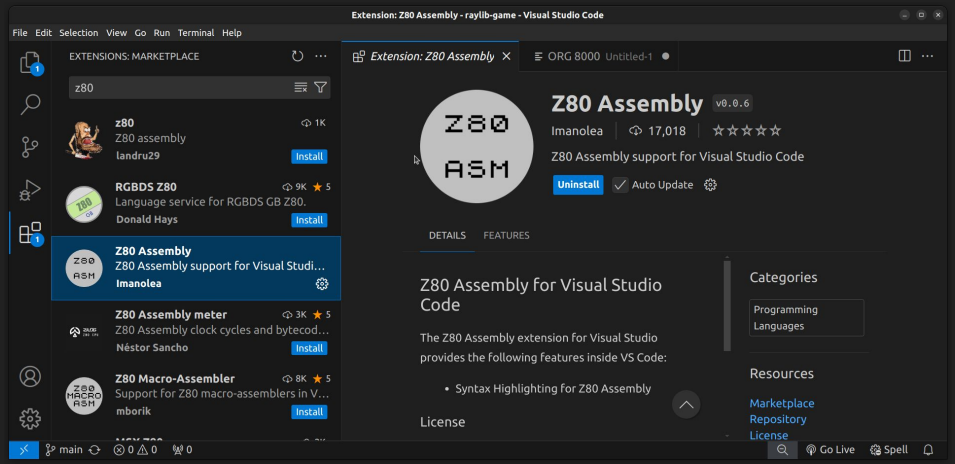
- Editor de código.
- Programa ensamblador.
- Emulador

4. Entorno de desarrollo

Editor de Código

Podemos utilizar cualquier editor de código para poder escribir el programa en lenguaje ensamblador.

En este caso, nosotros recomendamos Visual Studio Code con la extensión *Z80 Assembly.*



4. Entorno de desarrollo

Programa ensamblador

Un programa ensamblador (assembler), es el programa que lee el código ensamblador y lo transforma a binario para que sea entendido por el propio procesador; generando un binario.

Uno de los más utilizados para Z80, es *pasmo*; que es multiplataforma, y se utiliza por línea de comandos.

Puedes encontrar más información en la siguiente dirección:

https://pasmo.speccy.org/

4. Entorno de desarrollo

Emulador

Aunque esto dependerá de la máquina que fuéramos a emular, es útil un emulador que nos provea no sólo la forma de ejecutar nuestros programas sino incluso contenga herramientas de depuración.

Por ejemplo para ZX Spectrum, podemos encontrar el emulador *ZesarUx*

Puedes encontrar más información en la siguiente dirección:

https://github.com/chernandezba/zesarux

5. Bibliografía

- Z80 user Manual
- Z80 Wikipedia
- Ensamblador para ZX Spectrum ¿Hacemos un juego? Juan Antonio Rubio García. 2023.
- https://www.speccy.org/
- https://pasmo.speccy.org/