**Motivación:**

Assembler sirve para el tema de compiladores, es lo más rápido y optimo que se puede hacer, aunque cuesta tanto, al final vale la pena (no es tan difícil) Es muy fácil

**OFFSET:** En lugar de apuntar a un dato, apunta a la dirección de ese dato (es inmediato) ya que directamente el compilador lee la dirección de memoria

Byte ptr Sirve para especificar si se usan dos bytes

Word ptr Sirve para especificar si se usan dos bytes

**Load** carga la variable en contexto (para poder usarla)

**Store** almacena del contexto a una variable

**Add**, **sub** y **div** son operaciones

Cada código de operación ocupa 6 bits y las direcciones de memoria ocupan 10 bits

8 bits es 1 byte

1 byte son 2 dígitos en hexadecimal

**AL**: uso en registro en la selda baja nada mas

**AH**: uso el registro en la selda alta nada mas

**Ax**: uso el registro completo que serian ambos

**NO PUEDO HACER**

**Mov num1, num2** ya que son dos direcciones de memoria que no estan en los registros

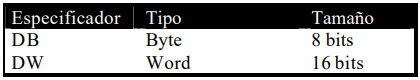
Despues puedo hacer la combinacion que sea menos la que se encuentra arriba

Lo que puedo hacer es :

Nombre\_variable especificador\_tipo valor\_inicial

BX Es el único que puede servir como puntero

**?:** El valor inicial no se especifica usando este carácter



DB 8bits o 1 byte

DW 16 bits o 2 bytes

**h**: hexadecimal cuando termina el número, en caso de no tener, es un decimal

nombre EQU valor definimos constantes

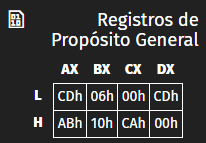
nombre\_variable especificador\_tipo valores definir variables

tabla DB: 1, 2, 4, 8, 16, 32, 64, 128

string DB “Esto es un String”

cantidad DUP (Valores) cantidad indica la cantidad de veces que se repite

**ORG:** dirección Permite cambiar la dirección una vez que el programa se ejecuto



**-1**=FF en hexadecimal

**Etiqueta**: el nombre que le damos a una línea de código

Los negativos están en Ca2 en el simulador

Los podemos tratar como registros de 16 bit o como un par de registros de 8 bits, tomando la parte baja separada del alta. (L: low o bajo y H: high o alto)

**BL** Registro en bajo; BX Registro en alto

**MOV destino, origen** Abreviatura de mover MOV BL, 1 == BL=1

**ADD** (Sumar) SUB (Restar)

**ADD:** operando1, operando2 (operando1 + operando2)

**SUB:** operando1, operando2 (operando1 - operando2)

**ADC**: operando1, operando2 (operando1 + operando2) + C (carry)

**SBB:** operando1, operando2 (operando1 - operando2) - C (carry)

**INC:** operando 🡨 operando + 1 Incrementar

**DEC:** operando 🡨 operando – 1 Decrementar

**AND** operando1, operando2

**OR** operando1, operando2

**XOR** operando1, operando2

**NOT** operando

**NEG** operando

**CMP** op1, op2 op1-op2 pero no modifica nada, solo los flags Compare: Comparar

**JMP** dirección ; Salta siempre Jump = Salto en ingles

**JZ** dirección ; Salta si el flag Z=1

**JNZ** dirección ; Salta si el flag Z=0

**JS** dirección ; Salta si el flag S=1

**JNS** dirección ; Salta si el flag S=0

**JC** dirección ; Salta si el flag C=1

**JNC** dirección ; Salta si el flag C=0

**JO** dirección ; Salta si el flag O=1

**JNO** dirección ; Salta si el flag O=0

**HLT** Halt: detener en inglés (Detiene el cpu)

**NOP** No hace nada

**PUSH** Poner un dato (Siempre uso registros completos como BX)

**POP** Sacar un dato

**SP** STACK POINTER (puntero estático)

**CALL** Llamamos a la sub rutina

**RET** Vuelvo al programa principal

**IP** (Instruction Pointer) Indica que funcion se tiene que ejecutar

Muestra la dirección de memoria de lo que tiene que ejecutar

**PUSH**: Cuando lo hago piso al instruction pointer (Se restan dos posiciones)

**PUSH** es pone un dato en la pila, decrementa el SP en 1

**PUSH** y **POP** Manejan 16 bit (dos celdas de memoria)

**CALL**: Guarda la dirección de la siguiente instrucción en la pila (PUSH IP), asigna la dirección de la subrutina a IP para poder ejecutarla. Sigue la ejecución como siempre, solo que ahora esta en la dirección de la subrutina.

**3000h**: Ponemos las subrutinas

**RET**: Desapila la instrucción que habíamos salvado en la pila y la guarda en IP(POP IP). Listo, sigue la ejecución como siempre. Solo que ahora está la dirección donde nos habíamos quedado.

**CALL**: Sin que nosotros le digamos nada a la máquina, sola hace un push para guardarlo en la pila (SP-2)

**RET**: Es como si hiciera un POP (SP+2)

NO SE PUEDE HACER UN PUSH EN UNA SUBRUTINA, Si hago un push, hago un pop para limpiar