Universidad Centroamericana "José Simeón Cañas"

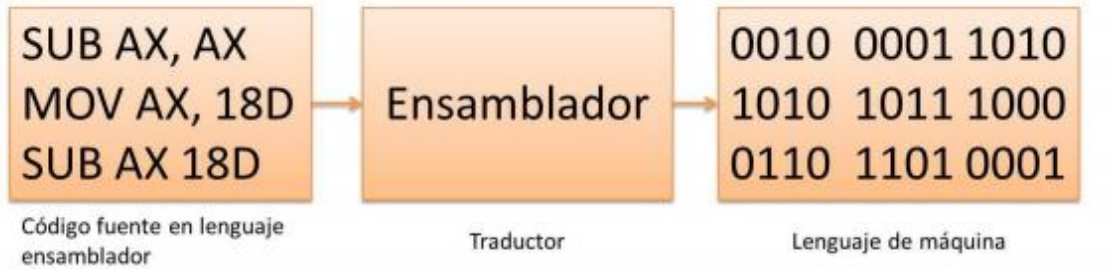
Arquitectura de Computadoras 12/04/2021
Guía 2: **Registros ensamblador**
Ing. Gabriela Carolina Reynosa Pérez <greynosa@uca.edu.sv>
Ricardo Alberto Gómez Quijano <00108517@uca.edu.sv>
Juan Humberto Gallardo Hernández <00091813@uca.edu.sv>

Objetivo

- ✓ Familiarizar al estudiante con la programación en lenguaje ensamblador.

Teoría

Recapitulando lo esencial, el lenguaje ensamblador es el lenguaje de programación utilizado para escribir programas de bajo nivel, y constituye la representación más directa del código de máquina específico para cada arquitectura de computadoras legible por un programador. Se utiliza en el siguiente proceso:



Debido a que ya configuramos nuestro entorno de desarrollo, tenemos todos los elementos necesarios: Un editor de texto para escribir nuestro código fuente, NASM para ensamblarlo y el entorno de DOSBox para correr nuestro código. Debido a que hemos abstraído tanto el concepto de programación en ensamblador es importante entender bien que es el código de máquina, es un lenguaje fácil de comprender e interpretar para un ordenador (sus instrucciones son construidas utilizando el lenguaje binario que consta solo de representaciones binarias), el cual es muy difícil de programar a este nivel para una persona; Estas instrucciones directas son las que pueden realizar acciones concretas debido que son interpretadas por el sistema operativo implementado en la arquitectura de microprocesador deseada.

La principal ventaja del lenguaje ensamblador es que proporciona la oportunidad de conocer más a fondo la operación de su PC, lo que permite el desarrollo de software de una manera más consistente. Bueno de esto que partimos al primer concepto que hay que comprender de ensamblador:

Registros

Las principales herramientas para escribir programas en el ensamblaje x86 son los registros del procesador. Los registros son como variables dispuestas por el procesador. El uso de registros en lugar de memoria para almacenar valores hace que el proceso sea más rápido y limpio. Tenemos varios tipos:

Registros de uso general

- AX: Acumulador (AL:AH)
- BX: Registro base (BL:BH)
- CX: Registro contador (CL:CH)
- DX: Registro de datos (DL:DH)

Registros de segmento (Solo se pueden usar para los usos mencionados a excepción de ES):

- DS: Registro del segmento de datos
- ES: Registro del segmento extra
- SS: Registro del segmento de pila
- CS: Registro del segmento de código

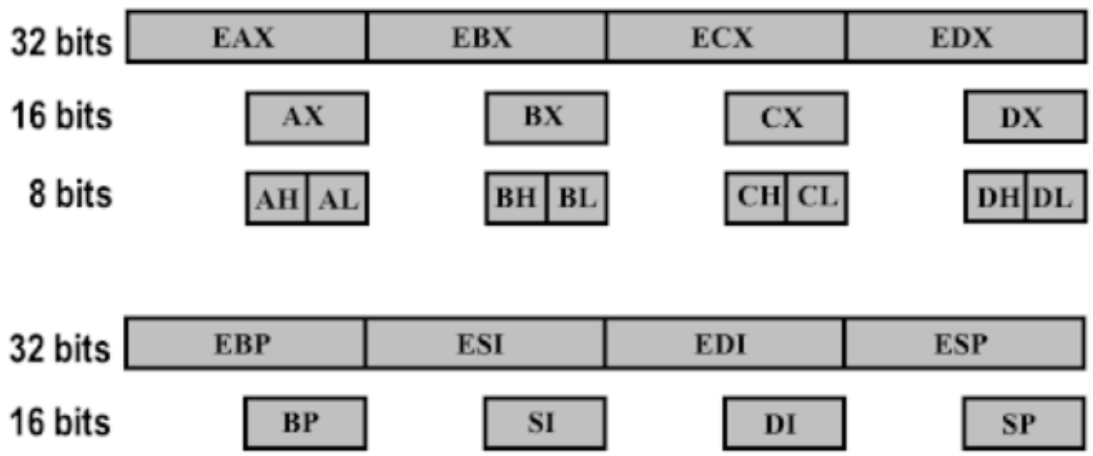
Registros punteros (También pueden tener uso general):

- BP: Registro de apuntadores base
- SI: Registro índice fuente
- DI: Registro índice destino

Registros especiales (Solo se pueden usar para los usos mencionados):

- SP: Registro apuntador de la pila
- IP: Registro apuntador de la siguiente instrucción
- F: Registro de banderas (8 bits)

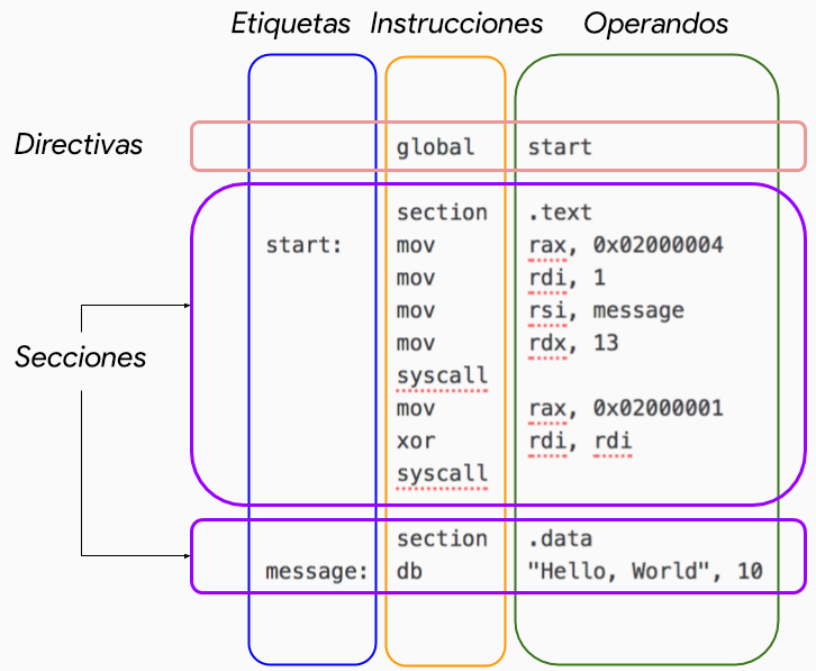
La parte baja del registro AX se llama AL y la parte alta AH. La parte baja del registro BX se llama BL y la parte alta BH, y también ocurre lo mismo con el registro CX y DX.



Como podemos observar en la imagen los registros EAX, EBX, ECX, EDX son registros de 32 bits, AX, BX, CX Y DX son registros de 16 bits y dentro de cada registro tenemos su parte alta y su parte baja, por ejemplo: AH (H de HIGH) y AL (L de LOW).

Estructura lenguaje ensamblador

La sintaxis de los mnemónicos se descompone de la siguiente manera:



Comenzamos con las directivas, que son las instrucciones que se ejecutan las precondiciones antes de ejecutar el bloque principal de código, normalmente solo utilizaremos una, la cual es *ORG 100h que* indica en la dirección de memoria en la que comenzamos a trabajar.

Después vemos que el programa está dividido en dos secciones, una de “data” donde solamente se define una variable llamada message, la otra es “text” el bloque principal de código donde están las operaciones que harán posible que el programa imprima en consola, normalmente las secciones son tacitas porque solo se utiliza la sección de “text”.

Si nos fijamos, cada instrucción tiene sus operandos con los que interactúa, los que se están utilizando son registros de memoria, son variables que nos sirven para guardar un valor temporalmente y a su vez, interactúan de ciertas maneras con algunas instrucciones que iremos aprendiendo en las siguientes sesiones.

Lo último que podemos notar es que estamos utilizando notación hexadecimal para denotar números, es la practica más común en el lenguaje de ensamblador usar esta notación o directamente binario. En un sistema operativo GNU/Linux como Ubuntu, se corre de la siguiente manera:

```
$ nasm -felf64 hello.asm && ld hello.o && ./a.out
Hello, World
```

IMPORTANTE

Para ensamblar un archivo .asm debemos ejecutar el siguiente comando:

nasm -f bin <*nombre*>**.asm -o** <*nombre*>**.com**

Ya que estamos utilizando dosbox es necesario iniciar el emulador y montar una partición y buscar la ruta del archivo. Por suerte, con un simple comando desde la terminal de Windows podemos montar y ubicarnos en la ruta donde estemos ejecutando el comando:

dosbox .

Al escribir esto en la terminal de Windows nos abrirá el emulador DOSBOX en la ruta donde ejecutemos este comando.

Dentro del emulador DOSBOX debemos utilizar el siguiente comando:

DEBUG.EXE <*nombre*>**.COM**

Por lo que siempre debemos tener nuestro archivo “debug.exe” en la carpeta junto con el archivo.com

Por último, los comandos utilizados en la práctica para debugger son los siguientes:

- d [registro]: nos permite visualizar los datos contenidos en el registro especificado.
- t: permite ejecutar línea por línea las instrucciones del programa.
- r: muestra los registros generales del 8086 y además muestra la siguiente instrucción a ejecutar.
- q: termina la ejecución del programa debugger.

Estos no son los únicos comandos que podemos utilizar, para mayor detalle revisar la guía básica de DEBUG en los recursos adicionales

Recursos adicionales

1. Guía básica de DEBUG: https://thestarman.pcministry.com/asm/debug/debug2.htm#CMDS
2. Registros de la arquitectura x86: http://www.eecg.toronto.edu/~amza/www.mindsec.com/files/x86regs.html
3. Maneras de direccionamiento de memoria en x86: http://www.c-jump.com/CIS77/ASM/Memory/lecture.html
4. Interrupciones de la arquitectura x86: https://jbwyatt.com/253/emu/8086_bios_and_dos_interrupts.html
5. Unidad de Punto Flotante (FPU): http://www.website.masmforum.com/tutorials/fptute/#intro
6. Ejemplos de instrucciones de la Unidad de Punto Flotante: https://gist.github.com/nikAizuddin/0e307cac142792dcdeba