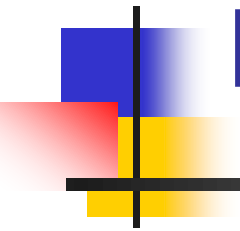
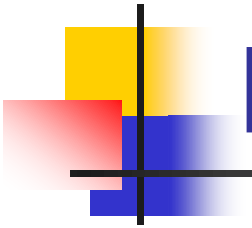


Sistemas de Procesamiento de Datos

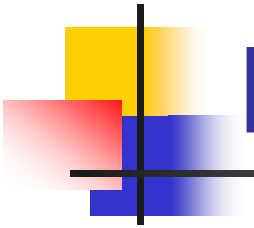
Desarrollo de programas Assembler (parte 1)

Profesor: Fabio Bruschetti
Aydte: Pedro Iriso
Ver 2019



Desarrollo de Programas

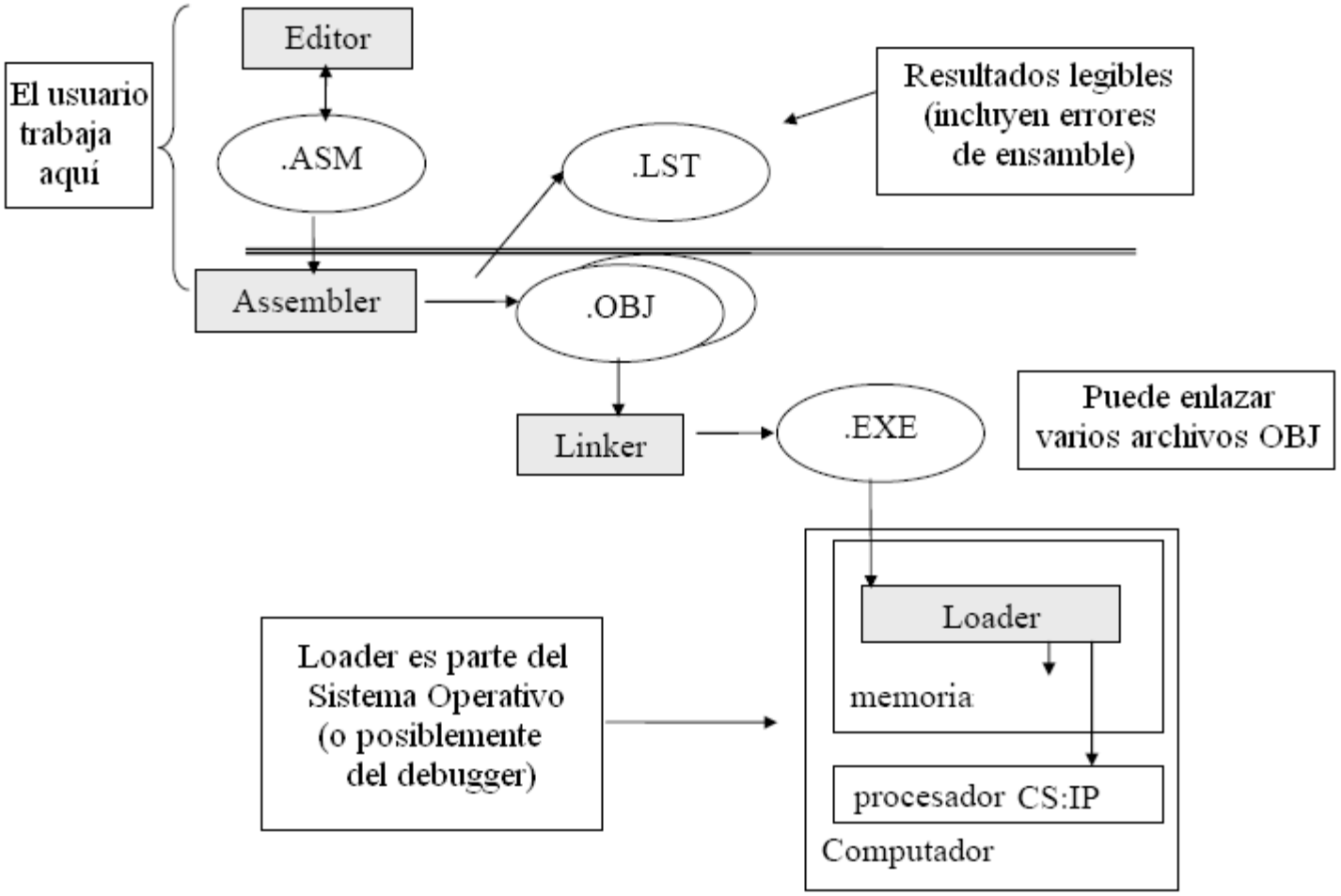
- Problema: Se debe convertir ideas (pensamiento humano) en un programa ejecutable (imagen binaria en memoria)
- El proceso de Desarrollo de Programas usa un conjunto de herramientas user-friendly para escribir programas y luego convertirlos en imagen binaria.
- Lenguaje de Programación:
  - **Sintaxis**: Conjunto de símbolos + reglas de gramática para construir sentencias usando símbolos.
  - **Semántica**: Cual es el significado de las sentencias, o cual es el resultado de la ejecución
  - **Lenguaje Assembler**: Un lenguaje legible que mapea uno a uno con las instrucciones de máquina (con las operaciones que son soportadas por la CPU)

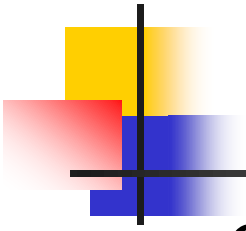


Desarrollo de Programas

- Assembler o Ensamblador:
  - Se trata de un programa que convierte el lenguaje assembler al formato Objeto.
    - El código objeto es el programa ejecutable en formato de máquina (formato binario)
    - El código objeto puede contener Referencias No Resueltas.
- Linker o Enlazador:
  - Se trata de un programa que conviena archivos en formato objeto y genera un solo archivo ejecutable
    - Con todas las referencias resueltas
- Loader o Cargador:
  - Se trata de un programa que carga el archivo ejecutable en memoria y puede inicializar algunos registros (Ej. IP) e inicia la ejecución.
- Debugger o Depurador:
  - Similar al Loader pero controla la ejecución del programa con el objetivo de ver y modificar el estado de las variables durante la ejecución.

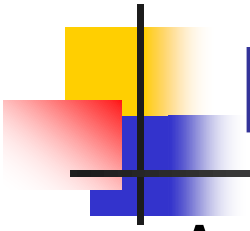
Desarrollo de Programas





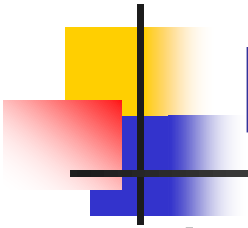
Desarrollo de Programas

- Código Fuente
  - Es un programa escrito en lenguaje assembler o de alto nivel almacenado en un archivo “fuente”.
- Código Objeto
  - Es la salida de un ensamblador o compilador
  - Es el programa ejecutable en formato binario (instrucciones de máquina)
  - Referencias externas no resueltas (Linker resuelve estas referencias y genera el archivo ejecutable)
- Código Ejecutable
  - El programa ejecutable completo en formato binario con la estructura impuesta por el sistema operativo.



Desarrollo de Programas

- Archivo
  - Se trata de una estructura del sistema operativo que permite almacenar información en algún soporte, con alguna organización interna.
    - Puede estar organizado en registros
    - Puede tener un formato especial reconocible por el sistema operativo
- Registro
  - Se trata de un conjunto de campos en un orden dado que se repite en todos los registros del archivo
- Campo
  - Se trata de un conjunto de bytes que contiene un tipo de información diferenciable.
- Un archivo es una secuencia de bytes de información.

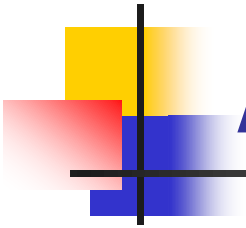


Lenguaje Assembler

- La sintaxis del lenguaje assembler debe cubrir todos los aspectos del proceso de programación y desarrollo.
  - Definir Constantes
  - Reservar memoria para uso de variables
  - Escritura de instrucciones: operaciones & operandos
  - Especificar Modos de direccionamiento
  - Directivas a herramientas en el proceso de desarrollo

Assembler – Constantes

- Valores Binarios : consisten solamente de 0's y 1's
  - Terminan con 'B'o 'b'
  - ej. 10101110b
- Valores Hexadecimales: comienzan con 0 .. 9
  - Puede incluir 0 . . 9, A .. F (a . . f)
  - Terminan con 'H' o 'h'
  - Requieren un cero antes de poner A..F como primer dígito
  - ej. 0FFH (valor hex de 8-bit)
- Valores Decimales:
  - Formato por default – no necesitan "calificador"
  - consisten de dígitos entre 0 . . 9
- String: secuencia de caracters codificados como bytes ASCII:
  - Se encierran entre comillas simples ` '
  - ej. 'Hola Ma' – 7 bytes
  - caracter: string de longitud = 1
  - D.O.S. Strings DEBEN TERMINAR SIEMPRE con '$'



Assembler – Etiquetas

- Las etiquetas son nombres definidos por el usuario que representan direcciones
  - Permiten al programador referirse a direcciones usando nombres lógicos sin necesidad de saber los valores hexadecimales
  - Deja todo eso a decisión del emsamblador o compilador.
- Las direcciones definidas por las etiquetas son usadas:
  - Por el flujo de control: Identificar la dirección destino.
  - Por variables de memoria: Identificar la dirección donde está el dato almacenado
- Siendo más específicos, las etiquetas identifican el offset de la dirección
  - Para el flujo de control: Se combina con CS (Code Segment).
  - Para variables de memoria: Se combina con DS (Data Segment).
- Las etiquetas tiene dos roles: Definición & Referencia.

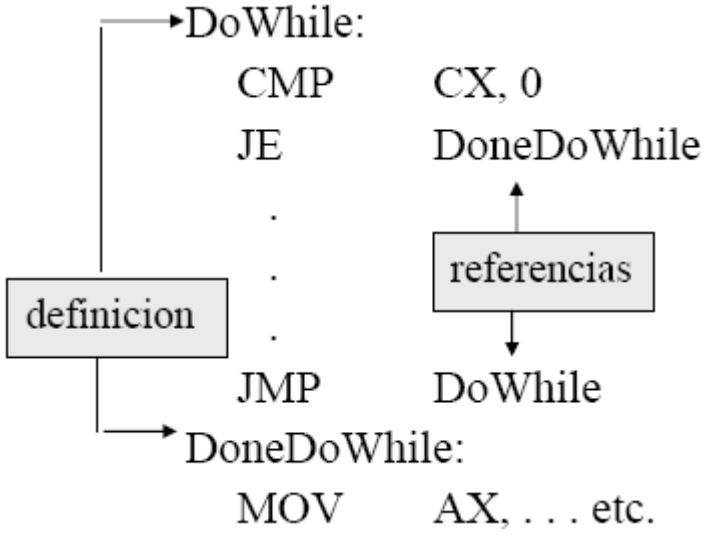
Assembler – Etiquetas

- Definición de Etiquetas
  - Las etiquetas definen el offset de la dirección correspondiente al primer byte siguiente.
  - Son usadas por el compilador para decidir la dirección exacta.
  - Deben ser el primer texto no blanco en la línea
  - Deben comenzar con una letra y continuar con alfanuméricos
  - Si se trata de un destino de control debe agregársele `:'
  - No pueden redefinir palabras reservadas (instrucciones)
  - Ejemplo de etiqueta destino de control
    - Continuar:
    - Otro_p_Seguir:
  - Ejemplo de etiqueta de dato
    - Hacer: MOV AX, BX

Hacer: corresponde la dirección del primer byte de la instrucción MOV

Assembler – Etiquetas

- Referencia de Etiquetas
  - La referencia es el uso de la etiqueta como operando de una instrucción
  - Hará referencia a la dirección asignada por el compilador
  - No se le agrega ":"
  - Ejemplo de Control
    - asumiendo que CX contiene el contador de ciclos:



- Se indica el destino como etiqueta
- El ensamblador o compilador asigna la dirección y calcula los offset (desplazamientos) relativos

Assembler – Memoria

- Declaración de Memoria
  - Se reserva memoria para las variables
  - Hay dos tamaños comunes en la arquitectura 8086
- DB reserva un byte de memoria
- DW reserva un word (2 bytes consecutivos) de memoria
- Se puede indicar opcionalmente la incialización de dichos espacios de memoria como un operando.
- No se usan `:' en la definición de variables

	DB	?	Reserva un byte
X	DB	?	Reserva un byte – etiqueta X apunta al byte
Y	DB	3	Reserva un byte – etiqueta Y – con valor 3
	DW	?	Reserva dos bytes consecutivos
W	DW	256	Reserva dos bytes – etiqueta W – con valor 255
HUH	DW	W	Reserva dos bytes – etiqueta W – con dir de W
	DB	'C'	Reserva un byte con valor 43h

Assembler – Memoria

- Usando constantes para inicializar memoria
  - Ambigüedades
    - DW 8000h
    - DW 0FFFFh

    No significa que se usen 20 bits, sino que son 16 bits pero FFFF es un valor no una referencia a una etiqueta.
  - ¿Cuál es más fácil de leer?
    - DW -1
    - DW 0FFFFH
    - DW 1111111111111111B
  - Múltiples declaraciones en una línea
    - Se separan por comas y se ubican en locaciones consecutivas
    - DB 3,6,9
    - Array1 DW -1, -1, -1, 0
    - Array2 DB 5 dup(0)
    - Array3 DW 3 dup(?)

Assembler – Memoria

- Usando constantes para inicializar memoria
  - Para declarar strings
    - Encerrar entre comillas dobles
    - Los caracteres ASCII se almacenan en bytes consecutivos

    - Mensaje DB 'Hola Todos'
    - MensajecNull DB 'Hola Todos',0
    - MensajeDOS DB 'Hola Todos', '$'

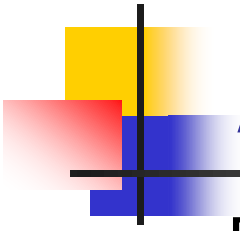
  - Todo string que va a ser impreso por funciones del sistema operativo D.O.S. deben terminar con el signo $

Assembler – Directivas

- Directivas son sentencias que están destinadas a otras herramientas como ser el compilador
  - No son ensambladas o traducidas directamente a instrucciones o declaraciones de memoria.
- Se usan para
  - Identificar segmentos al ensamblador
  - Terminar el emsablador
  - Definir símbolos para el ensamblador
- Directivas:
  - END
  - De Programa: .8086 // .model
  - SEGMENT, ENDS y ASSUME
  - Model SMALL: .code // .data // .stack size
  - EQU

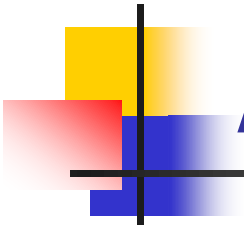
Assembler – Directivas

```
.8086
.model small
MYSTACK SEGMENT STACK
    db 100h dup(?)
MYSTACK ENDS
DATA SEGMENT
message db "Hello, world!",0dh,0ah,'$`
DATA ENDS
CODE SEGMENT
main proc
    ASSUME  DS:DATA, SS:MYSTACK, CS:CODE
    mov ax, @data              ; Inicializa DS
    mov ds,ax
    …
    mov ax, 4C00h              ; llamada a la función exit de DOS
    int 21h
main endp
CODE ENDS
end main
```



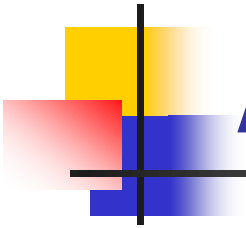
Assembler – Directivas

- Directiva END
  - Es usada por dos herramientas:
    - Ensamblador (Assembler)
    - Cargador (Loader)
  - Ensamblador
    - Usa esta directiva para saber cuando termina la lectura del archivo .ASM
      - Cualquier otra sentencia después del END es ignorada
  - Tiene un operando opcional que de estar presente debe ser una referencia a una etiqueta de control de flujo
  - Cargador
    - El loader utilizará el operando opcional como la dirección de la primera instrucción a ser ejecutada.
  - Sintaxis:
    - END [refencia_a_etiqueta]



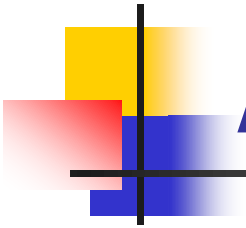
Assembler – Directivas

- Directivas de programa
  - Estas directivas indican a las herramientas el tipo de máquina donde se ejecutará el programa
    - Miembros distintos de la familia 80x86 tienen diferentes instrucciones aunque todos tengan el conjunto básico de instrucciones del 8086
    - Miembros distintos de la familia 80x86 tienen diferentes espacios de direcciones, permitiendo diferentes tamaños y configuraciones de programas a ejecutar.
  - .8086 limita el conjunto de instrucciones al procesador 8086
  - .model
    - Permite a las herramientas hacer simplificaciones en la organización de los datos
    - A lo sumo: los programas usarán un segmento de código y otro de datos
    - No se necesita manejo de control entre segmentos
    - No se necesita modificar DS y CS una vez inicializados



Assembler – Directivas

- Directivas SEGMENT, ENDS y ASSUME
  - **Ejemplo**: Suponga que un programa requiere 20 bytes de datos, 137 bytes de código y 100 bytes de stack.
    - Esto puede caber en un solo segmento de 64K
    - Para una organización optima, CS, DS y SS se pueden solapar
  - **Ejemplo**: Suponga que un programa requiere 80K bytes de datos, 47K bytes de código y 10K bytes de stack.
    - Necesitaremos un segmento no solapado para cada uno y para datos necesitaremos dos segmentos
    - El manejo de segmentos es muy complicado y necesita mas tiempo de procesamiento
- En el curso, escribiremos pequeños programas para 8086 usando modelo small
  - La cantidad de memoria reservada para código < 64k
  - La cantidad de memoria reservada para datos < 64k
  - Un segmento para código y un segmento para datos



Assembler – Directivas

- Directiva .code
  - Identifica el comienzo del segmento de código
  - Las herramientas se asegurarán de reservar suficiente memoria para la codificación de las instrucciones
- Directiva .data
  - Identifica el comienzo del segmento de datos
- Directiva .stack tamaño
  - Reserva tantos bytes como tamaño para ser usado por el stack en tiempo de ejecución
  - Mas sobre el stack próximamente

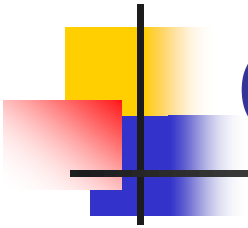
Programa “Hello world!”

```
.8086
.model small
.stack 100h
.data
message db "Hello, world!",0dh,0ah,'$'
.code
main proc
    mov ax,@data          ; Initalize DS
    mov ds,ax
    mov ah,9              ; DOS Function call to print a message
    mov dx, offset message
    int 21h
    mov ax,4C00h          ; DOS Function call to exit back to DOS
    int 21h
main endp
endmain
```

Assembler – Directivas

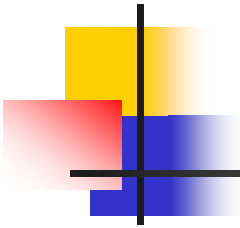
- Directiva EQU
  - Es una directiva que permite definir un nombre simbólico a un número
  - Ese nombre simbólico puede ser usado en cualquier lugar donde se quiera usar ese número
  - El ensamblador reemplazará el nombre por el número antes de realizar el ensamblado.
  - No tiene efecto sobre la memoria.

VAL EQU 0FFFFh	
X DW VAL	MOV BX, VAL ;Inmediato
V DW VAL	CMP AX, VAL ;Inmediato
	MOV DX, X ;Directo Memoria



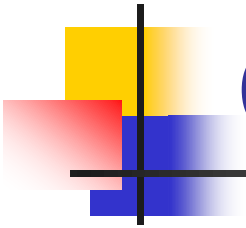
Generación de Código

- Trabajo del Assembler
  - Lee el archivo .asm línea por línea
  - Usa una aproximación de dos pasadas
    - Una pasada: Procesar todas las sentencias del .asm en forma secuencias desde el principio al final.



Generación de Código

- Primera Pasada:
  - Chequeo de sintaxis
  - Aloca la memoria necesaria para la imagen
    - Para las declaraciones de memoria
    - Para cada instrucción: bytes suficientes pasa el Opcode + Operandos
    - Para las directivas:?
  - Para cada definición de etiqueta, le asigna un valor y lo almacena en la Tabla de Símbolos
- Si hay errores de sintaxis en la primer pasada, los escribe en el .LST y para, sino...

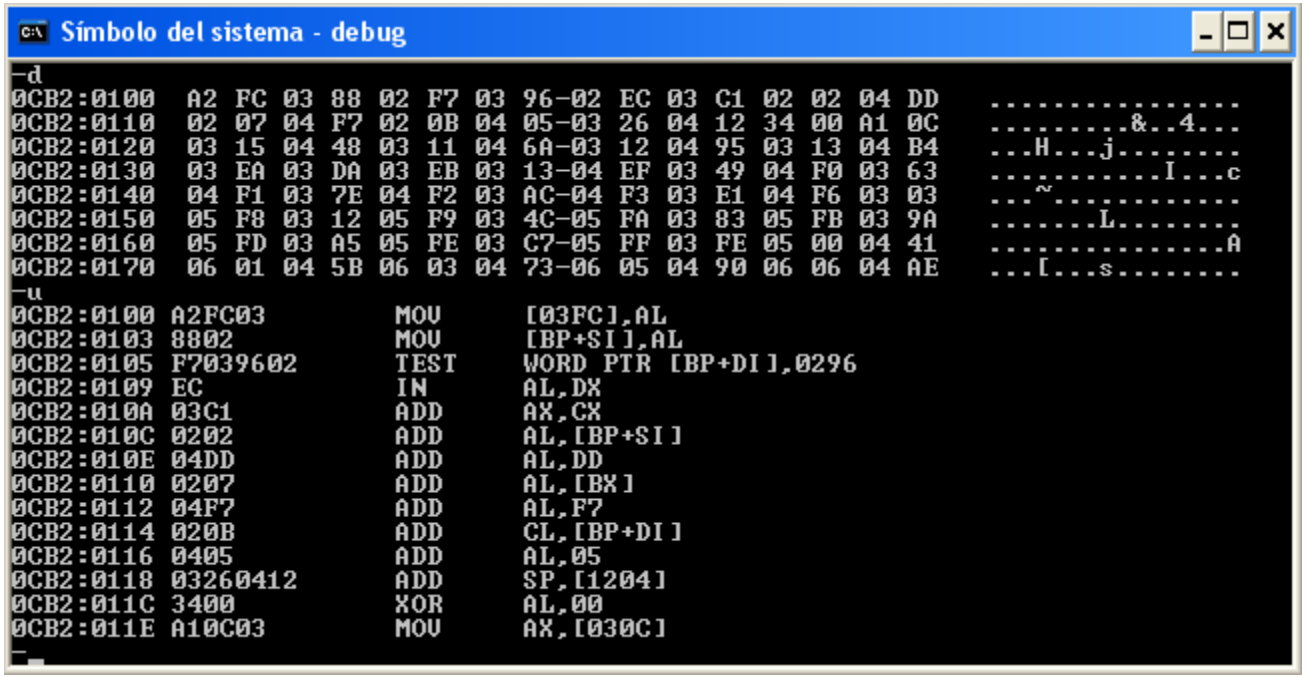


Generación de Código

- Segunda Pasada
  - Genera la imagen binaria (.obj)
    - Para cada instrucción, completa la codificación de la instrucción.
    - Pueden ocurrir errores en el cálculo de los offsets
      - Ej: Tratar de saltar demasiado lejos para un salto condicional
    - Si hay errores, los escribe en el .LST
    - Sino, genera el archivo .OBJ

Depuración de Código

- Una forma de depuración de código se puede llevar a cabo con el comando debug.exe



Depuración de Código

- Debug
  - d muestra memoria de datos
  - a permite ingresar instrucciones assembler
  - u muestra las instrucciones en memoria
  - n nombre del archivo
  - w escribe en el archivo la cantidad de bytes
  - p procesa paso a paso las instrucciones
  - e escribe posiciones de memoria
  - t ejecución instrucción por instrucción
  - g ejecuta el programa completo (hasta encontrar INT 20)
  - r muestra el contenido de los registros
    - rip permite cambiar el contenido del registro ip
    - rcx permite cambiar el contenido del registro cx
    - rf permite cambiar el contenido de los flags
  - Como almacenar un programa
    - -n c:\temp\prg.bin
    - -rcx
      - CX 0000
      - :44
    - -w 0

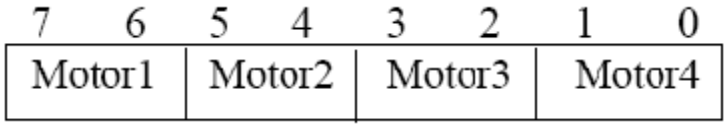
Código Assembler – Programas

- Escriba un fragmento de código para testear si una variable es divisible por 4, dejando el resultado lógico en AX
- Solución: Un número divisible por 4 debería tener sus dos bits menos significativos iguales a 0.

```
FALSE       equ     0
TRUE        equ     1
.data
variable    dw      1922h
.code
        MOV     AX, variable
        AND     AX,03h
        JZ      si
        MOV     AX, FALSE
        JMP     continuar
si:     MOV     AX, TRUE
continuar:
```

Código Assembler – Programas

- Suponga un robot con cuatro motores, que pueden encenderse y apagarse en dirección hacia delante o hacia atrás. El estado de los motores son escritos por el robot en un byte de estado llamado "motores" con la siguiente estructura de bits.



- Donde los dos bits correspondientes a cada motor se setea de acuerdo a:
  - 01 = adelante
  - 10 = atrás
  - 11 = apagado
- Escriba el fragmento de código que espere hasta que el motor 1 se apague para continuar.

Solución:

```
.data
motores db      ?
.code
espera: MOV     AL, motores
        AND     AL, 0C0h
        CMP     AL, 0C0h
        JNZ     espera
        ....
```