Decodificador de Lenguaje Maquina del Procesador Intel 8088

Cynthia Yolanda Sosa Cervantes - jlo.lopez.ortega@gmail.com,
Prof.: M. en C. Israel Rivera Zárate. – irivera@ipn.mx,
Prof.: M. en D. Patricia Pérez Romero. – promerop@ipn.mx
Centro de innovación y desarrollo tecnológico en cómputo - IPN

I. ABSTRACT

El lenguaje ensamblador es el lenguaje de programación utilizado para escribir programas de bajo nivel, mediante una representación de código máquina. El código máquina se encuentra representado en forma binaria, sin embargo, la longitud de los programas hace un tanto complicada la comprensión de cada instrucción, es por esto, que el lenguaje ensamblador utiliza una representación equivalente para cada instrucción.

II.

III. INTRODUCCIÓN

Formato de instrucción 'mov'

Para poder llevar acabo dicho programa, es necesario comprender el formato de la instrucción 'mov'. Tal como se muestra en [2], existen 7 tipos de transferencia de datos, de los cuales se tomaran en cuenta los siguientes:

- Registro/memoria a/de registro
- Inmediato a registro
- Memoria a acumulador
- Acumulador a memoria
- Registro/memoria a registro de segmentación
- Registro de segmentación a registro/memoria

A continuación se muestra el formato para cada una de las instrucciones:

- Registro/memoria a/de registro

1 0 0 0 1 0 d w	mod * reg r/m

- Inmediato a registro

1 0 1 1 w reg	dato	dato si w=1

- Memoria a acumulador

1 0 1 0 0 0 0 w	dirección-baja	dirección-alta

- Acumulador a memoria

1 0 1 0 0 0 1 w	dirección-baja	dirección-alta

- Registro/memoria a registro de segmentación

1 0 0 0 1 1 1 0	mod 0 reg r/m

- Registro de segmentación a registro/memoria

1 0 0 0 1 1 0 0	mod 0 reg r/m

Donde:

'W' nos informa del tamaño de los operandos. Con W = 0 estamos accediendo a 8 bits y con W = 1 a una palabra, es decir 16 bits.

'D'* nos informa si el registro especificado en el campo REG se trata del operando fuente u operando destino de instrucción especificada por el código de operación. (Para nuestro caso 'd' tendrá un valor de 1, y 'mod' de 11, por lo que 'r/m' será tratado como 'reg'.)

Para D = 0, el registro indicado por el campo REG es el operando fuente y para D = 1, el registro indicado por el campo REG es el operando destino.

'Reg' nos indica el registro que se utilizara, para lo cual existen las siguientes combinaciones (tabla 1)

16 bits (w=1)	8 bits (w=0)	Segmento
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Tabla 1. Asignacion de bits del campo 'reg'

Desarrollo

Primer término

Una vez que se analiza el formato de la instrucción 'mov' se puede observar que se cuenta con un patrón en los primeros cuatro bits, teniendo este los siguientes valores:

Binario	Hexadecimal	ASCII (hexa.)
1000	8	38
1010	A	41
1011	B	42

Por lo que la primera parte del código se centra en encontrar estos valores (Figura 1), de no encontrarse se mandara un mensaje de error. La comparación se realiza con el valor en ASCI correspondiente a cada término.

```
;------empieza la comparacion
;-----comparacion del primer termino
compa:
    cmp [cad+SI],42 ; * asterisco
    jz error

    cmp [cad+SI],38h ; 8
    jz seg_ter_8

    cmp [cad+SI],41H ; A
    jz seg_ter_a

    cmp [cad+SI],42h ; B
    jz seg_ter_b

    jnz error
```

Fig.1 Comparación del primer término

Segundo término

Posteriormente se procede a hacer la comparación del segundo término de cada instrucción Fig. 2, empezando de esta forma a separar cada tipo de transferencia de datos.

```
;-----comparacion del segundo termino termino "8"
seg_ter_8:

    add SI,1

    cmp [cad+SI],42 ; asterisco
    jz error

    cmp [cad+SI],41h ; A
    jz ult_ter_rm_8

    cmp [cad+SI],42H ; B
    jz ult_ter_rm_16

    cmp [cad+SI],43h ; C
    jz ult_ter_segre

    cmp [cad+SI],45h ; E
    jz ult_ter_reseg

    jnz error
```

a)

```
;--------Segundo termino de A
seg_ter_a:

    add SI,1
    cmp [cad+SI],42
    jz error

    cmp [cad+SI],30h
    jz acum_mem

    cmp [cad+SI],31h
    jz acum_mem

    cmp [cad+SI],32h
    jz mem_acum

    cmp [cad+SI],33h
    jz mem_acum
    jnz error
```

b)

```
;--------Inmediato a registro
;segundo termino de B
seg_ter_b:
    add SI,1
    cmp [cad+SI],42
    jz error

    cmp [cad+SI],30h
    jz purnum   ; el valor es 0
    js error    ;si el valor es menor que 0

    cmp [cad+SI],39h ;cmp el segundo limite de numeros
    jz purnum
    jns letr ;mas grande que num checar si es letra
    js purnum
letr:
    cmp [cad+SI],41h ;1er limite de letras
    jz vallet
    js error
    cmp [cad+SI],46h
    jz vallet
    jns error
    js vallet
```

c)

a) Segundo término para 8, b) Segundo término para A, c) Segundo término de B

Fig.2 Comparación del segundo término

Debido a que la entrada está dada en hexadecimal y lo que el programa recibe es su correspondiente en ASCII, es necesario hacer una conversión, así como delimitar el rango en el cual está permitido introducir los datos, esto es, valores de 0~9 y A~F (Fig. 3).

```
    cmp aux1,43h ;1er limite de letras
    js error
    cmp aux1,46h
    jz nxt2
    jns error

nxt2:
    sub aux1,37h ;letras
    ;valor puro numeros cuarto termino
    ;empieza comparacion segundo termino
    cmp aux2,30h
    jz val_pu_num   ; el valor es 0

    js error    ;si el valor es menor que 0

    cmp aux2,39h ;cmp el segundo limite de numeros
    jz val_pu_num
    jns ch_letra ;mas grande que num checar si es letra
    js val_pu_num
ch_letra:
    cmp aux2,41h ;1er limite de letras
    jz val_pur_let
    js error
    cmp aux2,46h
    jz val_pur_let
    jns error
    js val_pur_let
```

Fig.3 Delimitación de valores permitidos

Registros

La selección de registros, ya sea de 8 bits o de 16 bits, se realiza por medio de 'w', por lo que se realiza una comparación de dicha variable (Fig. 4).

```
    mov ah,9
    cmp w,1
    jz cero16
    mov dx,offset r8al
    int 21h

    jmp ult_8A
cero16:
    mov dx,offset r16ax
    int 21h
    jmp ult_8A
```

Fig.4 Comparación de 'w'

Siguiente instrucción

Una vez que se termina de evaluar una instrucción es necesario seguir evaluando las consecutivas, en caso de que el carácter siguiente sea '*' se da por concluida la evaluación a la cadena introducida, de no ser así se continuara evaluando (Fig. 5).

```
;--------siguiente instruccion
sig_lin:
    add SI,1

    cmp [cad+SI],42
    jz fin

    mov w,0
    mov b,0
    mov ctrl,0
    mov rs,2
    mov mac,2
    jnz compa
```

Fig.5 Código para seguir evaluando

Código en ensamblador

A continuación se muestra el código empleado.

Resultados

En la siguiente figura (Fig. 6) se puede observar la evaluación de la cadena conformada por las siguientes instrucciones (tabla 2).

Código maquina (hexa)	Instrucción
8BCE	mov CX,SI
B80500	mov AX,0005
B44C	mov AH,4C
A20300	mov AL,0003
8ED8	mov DS,AX

Fig. 6 Evaluación de cadena

IV. CONCLUSIONES

Es necesario conocer el formato de código máquina de la instrucción para de esta manera poder evaluar carácter a carácter la cadena introducida. De igual manera es importante tomar en cuenta los rangos que estos caracteres pueden tomar, es decir, deben ser solo caracteres de 0~9 y A~F, por lo que si se introduce una instrucción incompleta o un carácter que no corresponde un mensaje de error aparcera (Fig. 7). Otro aspecto no menos importante a considerar es el hecho de que el código introducido esta dado en ASCII por lo que es importante trasladar de este valor a su correspondiente hexadecimal.

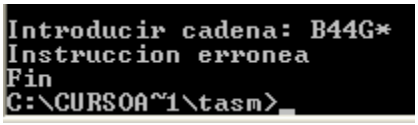

Fig. 6 Instrucción errónea

V. REFERENCIAS

[1] C. L. Morgan y M. Waite, “Introducción al micropocesador 8086/8088 (16 bits)”, 1ra. Edición, McGraw-Hill, 1988.