Seminario de Solución de problemas de Traductores de Lenguajes I

Centro Universitario de Ciencias Exactas en ingenierías

Universidad de Guadalajara

Maestro: Tonatiuh Hernandez Casas

Juan Antonio Pérez Juárez
Código: 215660996
Carrera: INCO

Actividad 4: Parte I

Carga del segmento de datos

El siguiente programa tiene un error, ejecútalo y determina cuál es el problema. Después Corrija y vuelva a ejecutarlo.

```
Unset
PAGE 60,132

TITLE SEGMENTO_DE_DATOS
MODEL SMALL
STACK 64

DATA
DAT1    DB      05H
DAT2     DB      10H
DAT3    DB      ?
TIME    EQU     10

.CODE
MAIN PROC FAR
    MOV AH, DAT1
    ADD AH, DAT2
    MOV DAT3, AH
    MOV AH, TIME
    MOV AX, 4C00H
    INT 21H
    MAIN ENDP
END MAIN
```

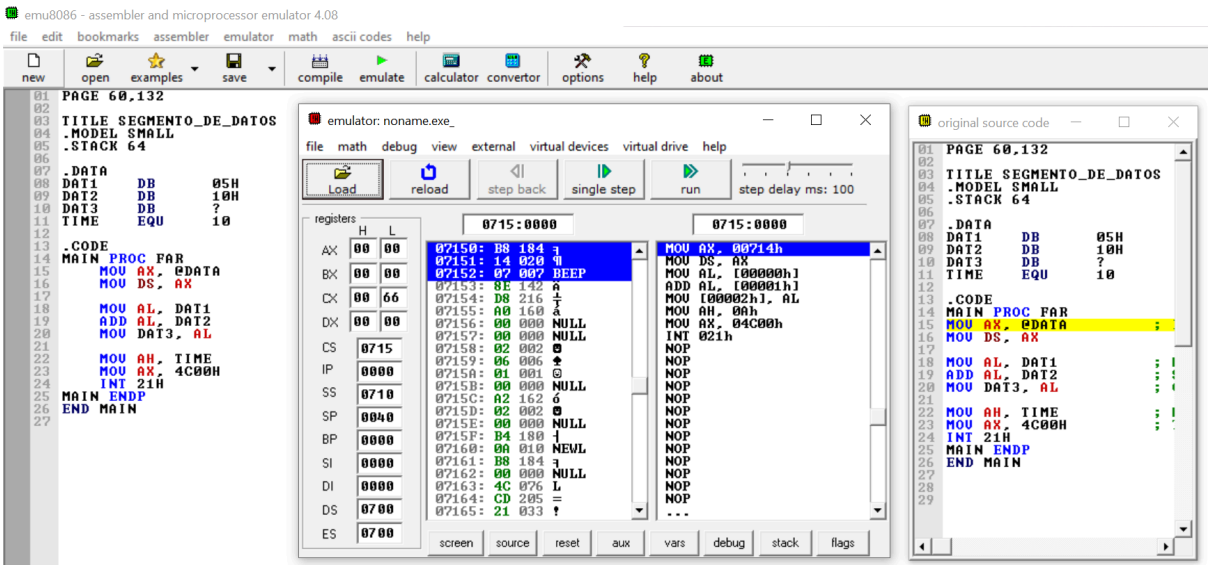
Veamos los errores de este programa.
El error es que estamos tratando de mover el valor de DAT1 a AH con MOV AH, DAT1, pero al hacerlo estamos tratando de mover el contenido de una dirección de memoria directa a un registro.
El problema aquí es que DAT1 es una variable definida en la memoria (una dirección de memoria), y no se puede mover directamente un valor de memoria a un registro de 8 bits como AH sin especificar.

Necesitamos hacer referencia al valor de la dirección de memoria, no al nombre directamente. Debemos utilizar la sintaxis adecuada para acceder a los valores en memoria.
En ensamblador, la CPU no puede acceder directamente a la memoria y hacer operaciones con ella. Lo que se debe hacer es mover el contenido de esa posición de memoria a un registro (como AL o AH) para poder procesarlo. En el caso de la arquitectura x86, los registros AX, BX, CX, y DX son registros de 16 bits, mientras que

sus mitades inferiores (AL, BL, CL, DL) y superiores (AH, BH, CH, DH) son registros de 8 bits.

El código corregido sería algo así:

```
Unset
PAGE 60,132

TITLE SEGMENTO_DE_DATOS
MODEL SMALL
STACK 64

.DATA
DAT1    DB      05H
DAT2    DB      10H
DAT3    DB      ?          ; Lugar para almacenar el resultado
TIME    EQU     10         ; Valor constante

.CODE
MAIN PROC FAR
    MOV AX, @DATA          ; Inicializa el segmento de datos
    MOV DS, AX

    MOV AL, DAT1           ; Mueve el valor de DAT1 a AL (registro de 8 bits)
    ADD AL, DAT2           ; Suma DAT2 al contenido de AL
    MOV DAT3, AL           ; Guarda el resultado en DAT3

    MOV AH, TIME           ; Mueve el valor constante de TIME a AH
    MOV AX, 4C00H          ; Termina el programa
    INT 21H
MAIN ENDP
END MAIN
```



Actividad 4: Parte II

Instrucción Jump

Investiga el salto incondicional (JMP) y los saltos condicionales (JC, JNC, JE, JNE, JP, JNP, etc).
Realiza un programa que calcule el factorial de un número, usando las instrucciones de salto.

Instrucciones de salto Son utilizadas para transferir el flujo del proceso al operador indicado.

Instrucción **JMP**
Propósito: Salto incondicional.

Sintaxis:
JMP destino

Esta instrucción se utiliza para desviar el flujo de un programa sin tomar en cuenta las condiciones actuales de las banderas ni de los datos.

Instrucción **JA (JNBE)**
Propósito: Brinco condicional

Sintaxis:
JA Etiqueta

Después de una comparación este comando salta si está arriba o salta si no está abajo o si no es igual.
Esto significa que el salto se realiza solo si la bandera CF está desactivada o si la bandera ZF está desactivada (que alguna de las dos sea igual a cero).

Instrucción **JAE (JNB)**
Propósito: salto condicional

Sintaxis:
JAE etiqueta

Salta si está arriba o si es igual o salta si no está abajo.
El salto se efectúa si CF está desactivada.

Instrucción **JB (JNAE)**
Propósito: salto condicional

Sintaxis:
JB etiqueta

Salta si está abajo o salta si no está arriba o si no es igual.
Se efectúa el salto si CF está activada.

Instrucción **JBE (JNA)**
Propósito: salto condicional

Sintaxis:
JBE etiqueta

Salta si está abajo o si es igual o salta si no está arriba.
El salto se efectúa si CF está activado o si ZF está activado (que cualquiera sea igual a 1).

Instrucción **JE (JZ)**
Propósito: salto condicional

Sintaxis:
JE etiqueta

Salta si es igual o salta si es cero.
El salto se realiza si ZF está activada.

Instrucción **JNE (JNZ)**
Propósito: salto condicional

Sintaxis:
JNE etiqueta

Salta si no es igual o salta si no es cero.
El salto se efectúa si ZF está desactivada.

Instrucción **JG (JNLE)**
Propósito: salto condicional, se toma en cuenta el signo.

Sintaxis:
JG etiqueta

Salta si es más grande o salta si no es menor o igual.
El salto ocurre si ZF = 0 u OF = SF.

Instrucción **JGE (JNL)**
Propósito: salto condicional, se toma en cuenta el signo.

Sintaxis:
JGE etiqueta

Salta si es más grande o igual o salta si no es menor que.

El salto se realiza si SF = OF

Instrucción **JL (JNGE)**
Propósito: salto condicional, se toma en cuenta el signo.

Sintaxis:
JL etiqueta

Salta si es menor que o salta si no es mayor o igual.
El salto se efectúa si SF es diferente a OF.

Instrucción **JLE (JNG)**
Propósito: salto condicional, se toma en cuenta el signo.

Sintaxis:
JLE etiqueta

Salta si es menor o igual o salta si no es más grande.
El salto se realiza si ZF = 1 o si SF es diferente a OF.

Instrucción **JC**
Propósito: salto condicional, se toman en cuenta las banderas.

Sintaxis:
JC etiqueta

Salta si hay acarreo.
El salto se realiza si CF = 1

Instrucción **JNC**
Propósito: salto condicional, se toma en cuenta el estado de las banderas.

Sintaxis:
JNC etiqueta

Salta si no hay acarreo.
El salto se efectúa si CF = 0.

Instrucción **JNO**
Propósito: salto condicional, se toma en cuenta el estado de las banderas.

Sintaxis:
JNO etiqueta

Salta si no hay desbordamiento.
El salto se efectúa si OF = 0.

Instrucción **JNP (JPO)**
Propósito: salto condicional, toma en cuenta el estado de las banderas.

Sintaxis:
JNP etiqueta

Salta si no hay paridad o salta si la paridad es non.
El salto ocurre si PF = 0.

Instrucción **JNS**
Propósito: salto condicional, toma en cuenta el estado de las banderas.

Sintaxis:
JNP etiqueta

Salta si el signo está desactivado.
El salto se efectúa si SF = 0.

Instrucción **JO**
Propósito: salto condicional, toma en cuenta el estado de las banderas.

Sintaxis:
JO etiqueta

Salta si hay desbordamiento (overflow).
El salto se realiza si OF = 1.

Instrucción **JP (JPE)**
Propósito: salto condicional, toma en cuenta el estado de las banderas.

Sintaxis:
JP etiqueta

Salta si hay paridad o salta si la paridad es par.
El salto se efectúa si PF = 1.

Instrucción **JS**
Propósito: salto condicional, toma en cuenta el estado de las banderas.

Sintaxis:
JS etiqueta

Salta si el signo está prendido.
El salto se efectúa si SF = 1.

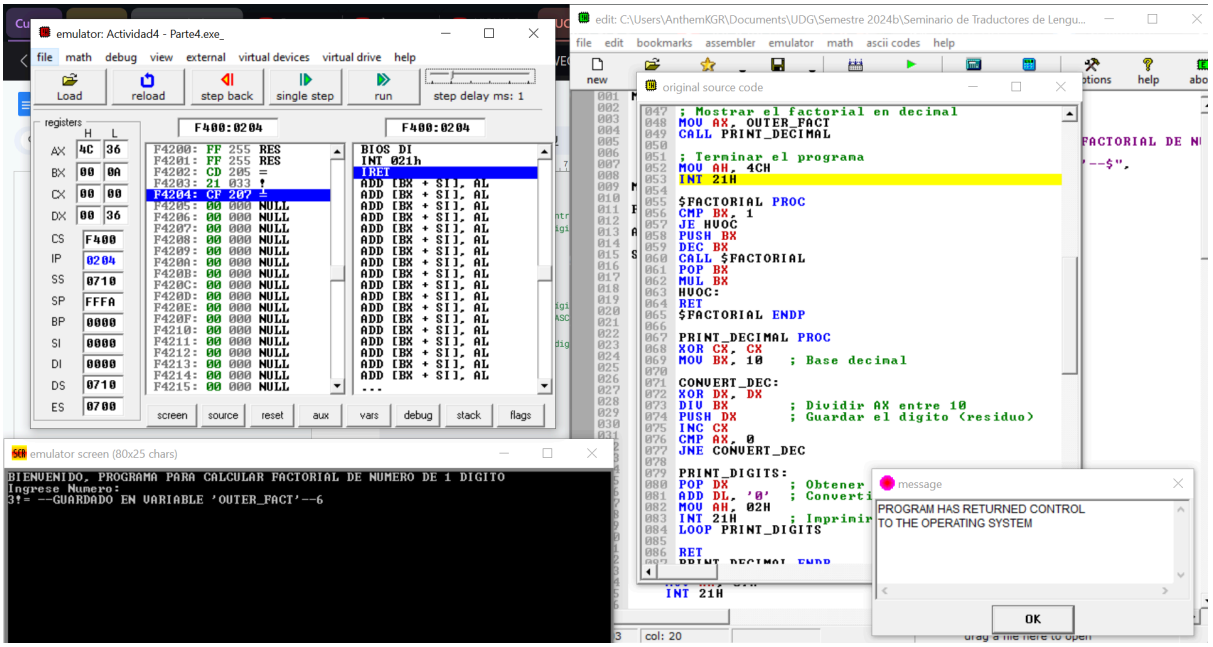
Programa para calcular el factorial de un número:

```
Unset
M_XDATA SEGMENT

      ALGDGADU DB ? ; Numero ingresado por el usuario
```

```
      OUTER_FACT DW ? ; Resultado del factorial
      WELCOME DB "BIENVENIDO, PROGRAMA PARA CALCULAR FACTORIAL DE NUMERO DE
1 DIGITO",10,13,24H
      WOW64 DB "Ingrese Numero: ",10,13,24H
      EQUAL DB "!= --GUARDADO EN VARIABLE 'OUTER_FACT'--$",

M_XDATA ENDS

FULLCODE SEGMENT

ASSUME CS:FULLCODE, DS:M_XDATA

START:
    MOV AX, M_XDATA
    MOV DS, AX

    MOV DX, OFFSET WELCOME
    MOV AH, 09H
    INT 21H

    MOV DX, OFFSET WOW64
    MOV AH, 09H
    INT 21H

    MOV AH, 1H
    INT 21H
    MOV BX, OFFSET ALGDGADU
    SUB AL, 30H
    MOV [BX], AL

    CMP AL, 01H
    JBE NEXTQ

    MOV AX, 1
    MOV BL, ALGDGADU
    MOV BH, 0H

    CALL $FACTORIAL
    MOV OUTER_FACT, AX  ; Guardar el resultado en OUTER_FACT

    LEA DX, EQUAL
    MOV AH, 09H
    INT 21H

    ; Mostrar el factorial en decimal
    MOV AX, OUTER_FACT
    CALL PRINT_DECIMAL
```

```
    ; Terminar el programa
    MOV AH, 4CH
    INT 21H

$FACTORIAL PROC
    CMP BX, 1
    JE HVOC
    PUSH BX
    DEC BX
    CALL $FACTORIAL
    POP BX
    MUL BX
HVOC:
    RET
$FACTORIAL ENDP

PRINT_DECIMAL PROC
    XOR CX, CX
    MOV BX, 10   ; Base decimal

CONVERT_DEC:
    XOR DX, DX
    DIV BX       ; Dividir AX entre 10
    PUSH DX      ; Guardar el digito (residuo)
    INC CX
    CMP AX, 0
    JNE CONVERT_DEC

PRINT_DIGITS:
    POP DX       ; Obtener el dígito
    ADD DL, '0'  ; Convertir a ASCII
    MOV AH, 02H
    INT 21H      ; Imprimir el digito
    LOOP PRINT_DIGITS

    RET
PRINT_DECIMAL ENDP

FULLCODE ENDS

NEXTQ:
    MOV AH, 02H
    MOV DL, '!'
    INT 21H
    MOV AH, 02H
    MOV DL, '='
    INT 21H
    MOV AH, 02H
```

```
    MOV DL, '1'
    INT 21H

END START
```



Actividad 4: Parte III

Instrucción LOOP

Investigue el funcionamiento de la instrucción LOOP y su relación con el registro CX. Realice un programa que calcule la suma de los primeros N(Que en mi caso serán los primeros 4) números enteros.

Como en cualquier otro lenguaje de programación, hay ocasiones en las que es necesario hacer que el programa no siga una secuencia lineal, sino que repita varias veces una misma instrucción o bloque de instrucciones antes de continuar con el resto del programa, es para esto que se utilizan los ciclos.
Instrucción LOOP
Propósito: Generar un ciclo en el programa.

Sintaxis:
LOOP etiqueta

La instrucción loop decrementa CX en 1, y transfiere el flujo del programa a la etiqueta dada como operando si CX es diferente a 1.

Instrucción LOOPE
Propósito: Generar un ciclo en el programa considerando el estado de ZF

Sintaxis:
LOOPE etiqueta

Esta instrucción decrementa CX en 1. Si CX es diferente a cero y ZF es igual a 1, entonces el flujo del programa se transfiere a la etiqueta indicada como operando.

Instrucción LOOPNE
Propósito: Generar un ciclo en el programa, considerando el estado de ZF

Sintaxis:
LOOPNE etiqueta

Esta instrucción decrementa en uno a CX y transfiere el flujo del programa solo si ZF es diferente a 0.

Los ciclos predefinidos de ensamblador son los siguientes:

LOOP:

Esta función decrementa el valor del registro contador CX, si el valor contenido en CX es cero ejecuta la siguiente instrucción, en caso contrario transfiere el control a la ubicación definida por la etiqueta utilizada al momento de declarar el ciclo.

Ejemplo:
mov cx,25 : Número de veces que se repetirá el ciclo, en este caso 25.
Ciclo: Etiqueta que se utilizará como referencia para el ciclo loop.
int 21h: Instrucción contenida dentro del ciclo (puede contener más de una instrucción).

loop: Ciclo loop que transferirá el control a la línea de la etiqueta ciclo en caso de que CX no sea cero.

LOOPE:
Esta función decrementa el valor del registro contador CX, si el valor contenido en CX es cero y ZF es diferente de uno ejecuta la siguiente instrucción, en caso contrario transfiere el control a la ubicación definida por la etiqueta utilizada al momento de declarar el ciclo.

Ejemplo:

Ciclo: Etiqueta que se utilizará como referencia para el ciclo loope.
int 21h: Instrucción contenida dentro del ciclo (puede contener más de una instrucción).
loope: Ciclo loope que transferirá el control a la línea de la etiqueta ciclo en caso de que CX no sea cero y ZF sea igual a uno.

LOOPNE:
Esta función decrementa el valor del registro contador CX, si el valor contenido en CX es cero y ZF es diferente de cero ejecuta la siguiente instrucción, en caso contrario transfiere el control a la ubicación definida por la etiqueta utilizada al momento de declarar el ciclo, esta es la operación contraria a loope.

Ejemplo:
Ciclo: Etiqueta que se utilizará como referencia para el ciclo loopne.
int 21h: Instrucción contenida dentro del ciclo (puede contener más de una instrucción).
loopne: Ciclo loopne que transferirá el control a la línea de la etiqueta ciclo en caso de que CX no sea cero y ZF sea igual a cero.

LOOPZ:
Esta función decrementa el valor del registro contador CX, si el valor contenido en CX es cero y ZF es diferente de uno ejecuta la siguiente instrucción, en caso contrario transfiere el control a la ubicación definida por la etiqueta utilizada al momento de declarar el ciclo.

Ejemplo:
Ciclo: Etiqueta que se utilizará como referencia para el ciclo loopz.
int 21h: Instrucción contenida dentro del ciclo (puede contener más de una instrucción).
loopz: Ciclo loopz que transferirá el control a la línea de la etiqueta ciclo en caso de que CX no sea cero y ZF sea igual a uno.

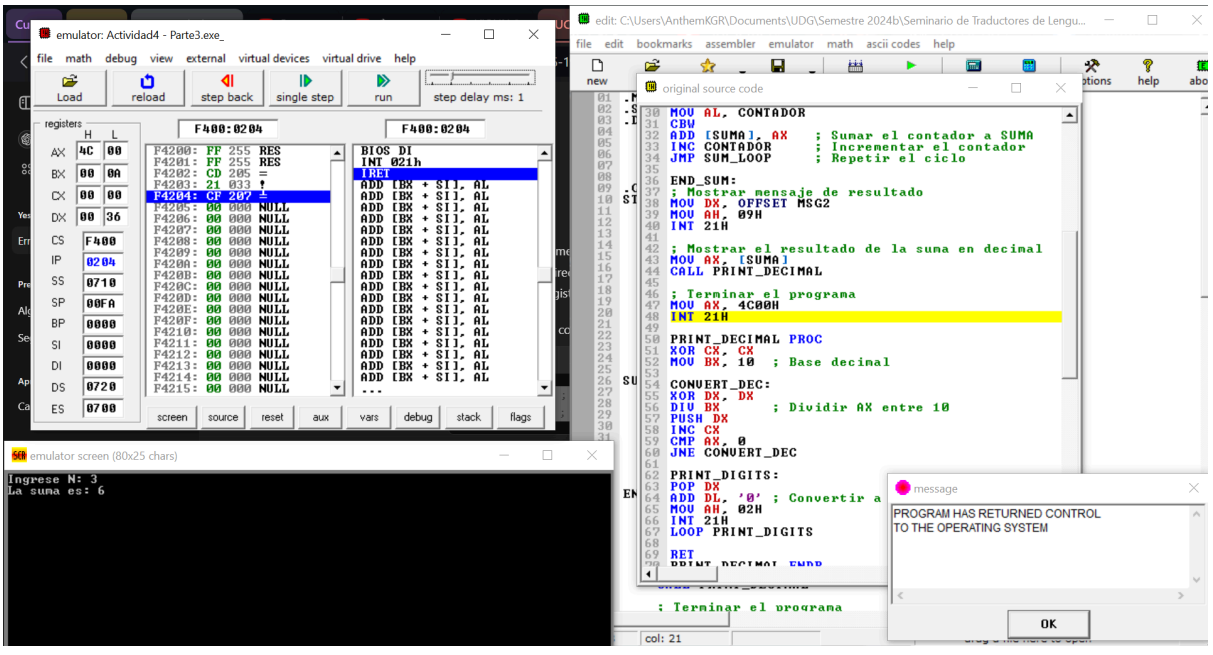
LOOPNZ:
Esta función decrementa el valor del registro contador CX, si el valor contenido en CX es cero y ZF es diferente de cero ejecuta la siguiente instrucción, en caso contrario transfiere el control a la ubicación definida por la etiqueta utilizada al momento de declarar el ciclo, esta es la operación contraria a loopz.

Ejemplo:
Ciclo: Etiqueta que se utilizará como referencia para el ciclo loopnz.
int 21h: Instrucción contenida dentro del ciclo.
loopnz: Ciclo loopnz que transferirá el control a la línea de la etiqueta ciclo en caso de que CX no sea cero y ZF sea igual a cero.

Realice un programa que calcule la suma de los primeros N(Que en mi caso serán los primeros 4) números enteros.

```
Unset
.MODEL SMALL
.STACK 100H
.DATA
    N DB ?        ; Numero N
    CONTADOR DB 1 ; Contador inicial
    SUMA DW 0     ; Almacena la suma total
    MSG1 DB 'Ingrese N: $'
    MSG2 DB 10,13,'La suma es: $'
.CODE
START:
    MOV AX, @DATA
    MOV DS, AX

    ; Pedir al usuario que ingrese N
    MOV DX, OFFSET MSG1
    MOV AH, 09H
    INT 21H


    MOV AH, 01H
    INT 21H
    SUB AL, 30H
    MOV N, AL

    ; Comenzar la suma
SUM_LOOP:
    MOV AL, N        ; Mover N a AL
    CMP CONTADOR, AL
    JA END_SUM
    MOV AL, CONTADOR
    CBW
    ADD [SUMA], AX   ; Sumar el contador a SUMA
    INC CONTADOR     ; Incrementar el contador
    JMP SUM_LOOP     ; Repetir el ciclo

END_SUM:
    ; Mostrar mensaje de resultado
    MOV DX, OFFSET MSG2
    MOV AH, 09H
    INT 21H

    ; Mostrar el resultado de la suma en decimal
    MOV AX, [SUMA]
    CALL PRINT_DECIMAL
```

```
    ; Terminar el programa
    MOV AX, 4C00H
    INT 21H

PRINT_DECIMAL PROC
    XOR CX, CX
    MOV BX, 10  ; Base decimal

CONVERT_DEC:
    XOR DX, DX
    DIV BX      ; Dividir AX entre 10
    PUSH DX
    INC CX
    CMP AX, 0
    JNE CONVERT_DEC

PRINT_DIGITS:
    POP DX
    ADD DL, '0' ; Convertir a ASCII
    MOV AH, 02H
    INT 21H
    LOOP PRINT_DIGITS

    RET
PRINT_DECIMAL ENDP

END START
```



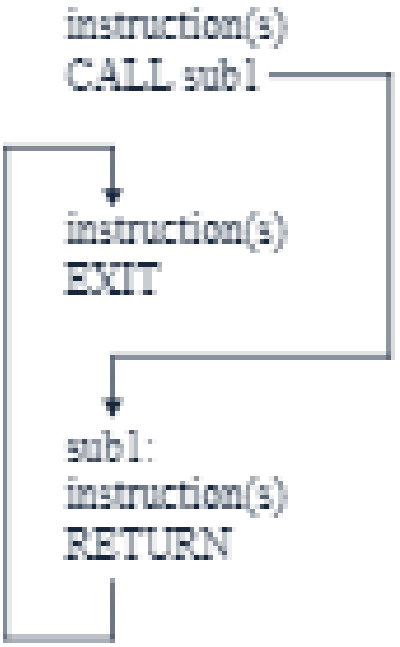
Actividad 4: Parte IV

Llamadas a procedimientos

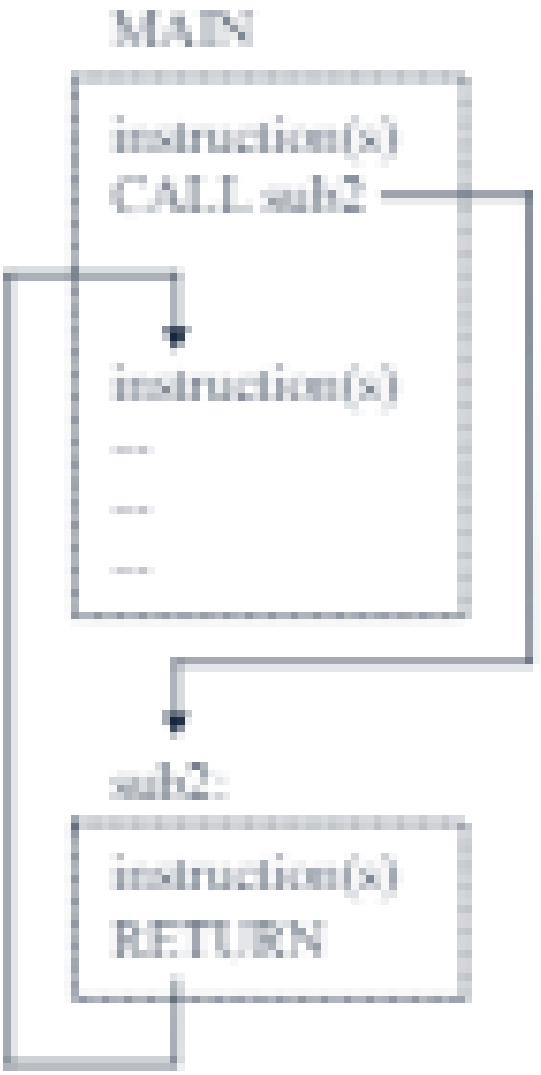
Investigue el funcionamiento de las instrucciones CALL y RET.
Realice un programa que calcule un factorial de un número.

La instrucción CALL interrumpe el flujo de un programa pasando el control a una subrutina interna o externa. Una subrutina interna forma parte del programa de llamada. Una subrutina externa es otro programa. La instrucción RETURN devuelve el control de una subrutina al programa de llamada y, opcionalmente, devuelve un valor.

Al llamar a una subrutina interna, CALL pasa el control a una etiqueta especificada después de la palabra clave CALL. Cuando la subrutina finaliza con la instrucción RETURN, se procesan las instrucciones que siguen a CALL.



Al llamar a una subrutina externa, CALL pasa el control al nombre de programa que se especifica después de la palabra clave CALL. Cuando se complete la subrutina externa, puede utilizar la instrucción RETURN para volver a donde la dejó en el programa de llamada.



Instrucción de llamada a procedimiento CALL y RET

La instrucción CALL se usa para realizar una llamada a un procedimiento y la instrucción RET se usa para volver de un procedimiento. Cuando se realiza una llamada a procedimiento con CALL, se guardan en la pila el valor de IP en caso de un salto corto, y de CS e IP en caso de un salto lejano.

Cuando se ejecuta la instrucción RET se recuperan de la pila los valores de IP o de CS e IP dependiendo del caso.

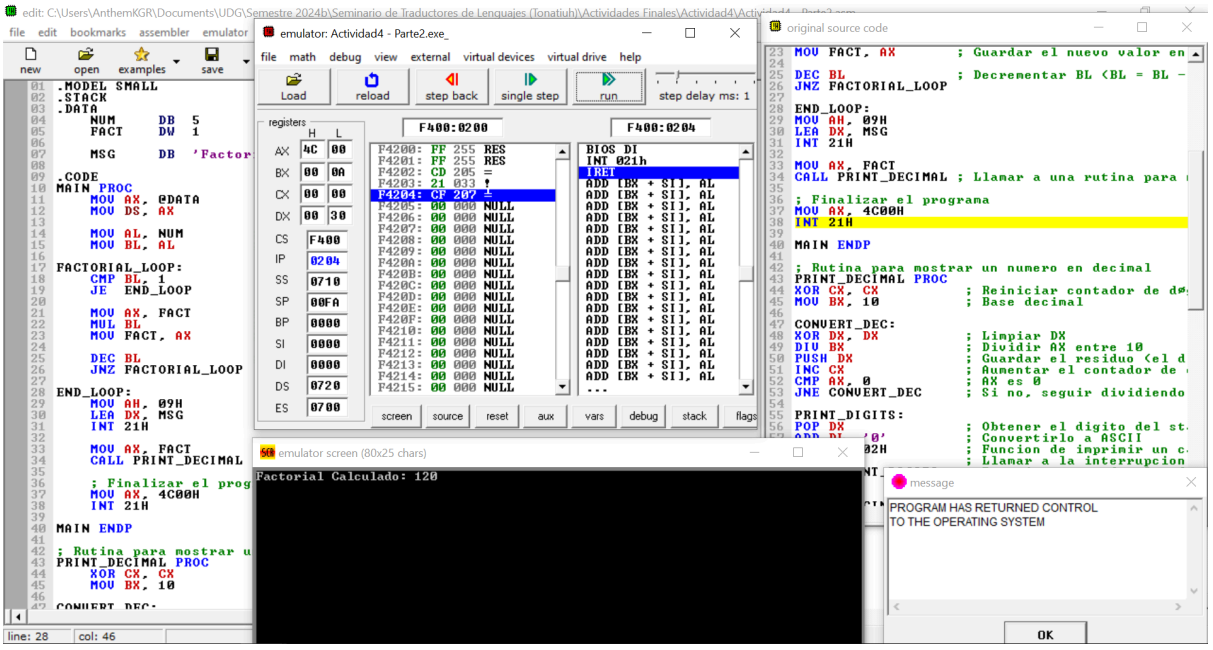
Al salir de un procedimiento es necesario dejar la pila como estaba; para ello podemos utilizar la instrucción pop, o bien ejecutar la instrucción RET en donde no es el número de posiciones que deben descartarse de la pila.

Programa para calcular el factorial de un número:

```
Unset
.MODEL SMALL
.STACK
```

```
.DATA
    NUM     DB  5       ; Numero para calcular el factorial
    FACT    DW  1       ; Variable para almacenar el factorial

    MSG     DB  'Factorial Calculado: $'

.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX

    MOV AL, NUM         ; Mover el numero al registro AL
    MOV BL, AL          ; Guardar el valor en BL para el ciclo

FACTORIAL_LOOP:
    CMP BL, 1
    JE  END_LOOP

    MOV AX, FACT        ; Cargar el valor actual de FACT en AX
    MUL BL              ; Multiplicar AX por BL (AX = AX * BL)
    MOV FACT, AX        ; Guardar el nuevo valor en FACT

    DEC BL              ; Decrementar BL (BL = BL - 1)
    JNZ FACTORIAL_LOOP

END_LOOP:
    MOV AH, 09H
    LEA DX, MSG
    INT 21H

    MOV AX, FACT
    CALL PRINT_DECIMAL ; Llamar a una rutina para mostrar el valor en
decimal

    ; Finalizar el programa
    MOV AX, 4C00H
    INT 21H

MAIN ENDP

; Rutina para mostrar un numero en decimal
PRINT_DECIMAL PROC
    XOR CX, CX          ; Reiniciar contador de dígitos
    MOV BX, 10          ; Base decimal

CONVERT_DEC:
    XOR DX, DX          ; Limpiar DX
    DIV BX              ; Dividir AX entre 10
```

```
    PUSH DX             ; Guardar el residuo (el digito actual)
    INC CX              ; Aumentar el contador de digitos
    CMP AX, 0           ; AX es 0
    JNE CONVERT_DEC     ; Si no, seguir dividiendo

PRINT_DIGITS:
    POP DX              ; Obtener el digito del stack
    ADD DL, '0'         ; Convertirlo a ASCII
    MOV AH, 02H         ; Funcion de imprimir un caracter
    INT 21H             ; Llamar a la interrupcion para mostrar el digito
    LOOP PRINT_DIGITS   ; Repetir para todos los digitos

    RET
PRINT_DECIMAL ENDP
END MAIN
```



Reflexión

Vaya actividad más llena de información, esta sin duda es la actividad más extensa, demasiada información para procesar, lo bueno que son conceptos que ya conozco por la programación en Python y en C++, de no ser por eso probablemente estaría frito.

Además de que vuelvo a confirmar que no se me da la programación estructurada, creo que debería hacer uno de esos cursos de udemy para mejorar mi lógica de programación porque realmente soy terrible, tuve que ver varios tutoriales, checar repositorios y leer PDFs en inglés. Para mi buena suerte hay muchos repositorios de ensamblador en github, así que como el Dr Frankestein, agarré lo que me servía, espero sea correcto.

Realmente me gustó, pero por alguna razón se me dificulta y eso que tuve tiempo para desarrollar holgadamente esta actividad.
Profe, si lee esto, no suba más la dificultad, para mí fue horrible programar esto.

Bibliografía:

Gurugio. (n.d.). How to use MOV. Book Assembly 8086. GitBook. https://gurugio.gitbooks.io/book_assembly_8086/content/usemov.html

Microcontrollerslab.com. https://microcontrollerslab.com/8086-addressing-modes-explained-with-assembly-language-examples/

8051 Set de instrucciones: JMP @. (n.d.). http://www.sc.ehu.es/sbweb/webcentro/automatica/web_8051/Contenido/set_8051/Instrucciones/51jmp.htm

Esquivel, B. (2020, April 28). Unidad 2. Programación básica. NANONBLOGS. https://brandon22esquivel.wixsite.com/misitio/post/unidad-2-programaci%C3%B3n-b%C3%A1sica

CICS Transaction Server for z/OS 5.6. (2024, January 4). https://www.ibm.com/docs/es/cics-ts/5.6?topic=functions-subroutines#subrou

CICS Transaction Server for z/OS 5.6. (2024, January 4). https://www.ibm.com/docs/es/cics-ts/5.6?topic=instructions-call-return

ChecheSwap. (n.d.). GitHub - ChecheSwap/assembly8086-Factorial: Factorial de un número en ensamblador 8086 (Asm 86). GitHub. https://github.com/ChecheSwap/assembly8086-Factorial

https://ocw.uc3m.es/pluginfile.php/1918/mod_page/content/13/ejercicio3_resuelto.pdf

http://www.dacya.ucm.es/hidalgo/estructura/ensamblador.pdf

Paszniuk, R. (n.d.). Ejercicios resueltos en Ensamblador 8086 – Programación. https://www.programacion.com.py/escritorio/ensamblador/ejercicios-resueltos-en-ensamblador-8086

https://www.cs.buap.mx/~mtovar/doc/ejCiclosProp.pdf