Procedimientos

Procedimientos

Un procedimiento es una porción reusable de programa que se almacena en memoria una vez, pero se usa tan seguido como es necesario. Esto hace ahorrar espacio de memoria y hace más fácil desarrollar programas.

La única desventaja de un procedimiento es que hacen que la computadora tome una pequeña cantidad de tiempo para encadenarse al procedimiento y regresar de el.

La instrucción **CALL** se encadena a las subrutinas, y la instrucción **RET** hace que se regrese de ella.

Procedimientos

Ejemplo:

Programa que imprime en pantalla en notación binaria el byte almacenado en AL.

```
printBin PROC
        push ax
        push cx
        mov ah,al
        mov cx,8
@@print: xor al, al ; se pone en 0 al registro AL
        shl ah,1
        adc al,'0'
        call putchar
        loop @@print

        pop cx
        pop ax
        ret
        ENDP
```

El procedimiento ***putchar*** espera en el ***registro AL*** el carácter ASCII a imprimir en pantalla.

Archivo **printBin.asm**

Ejemplo (cont.):

```
MODEL small
.STACK 100h

;----- Insert INCLUDE "filename" directives here
;----- Insert EQU and = equates here

INCLUDE procs.inc

    LOCALS

.DATA
    msg_bin  db 'AL desplegado en Binario:',0
    new_line db 13,10,0

.CODE

    ;-----   Insert program, subrutine call, etc., here

    Main PROC
        mov ax,@data    ; Inicializar DS al la direccion
        mov ds,ax       ; del segmento de Datos (.DATA)

        call clrscr
        mov dx, offset msg_bin
        call puts
        mov dx, offset new_line
        call puts
        mov al,0a7h     ; dato a imprimir en binario
        call printBin

        ; Fin del Programa ;
        mov ah,4Ch
        mov al,0
        int 21h
        ENDP
```

Archivo **printBin.asm**

Archivo completo

```
MODEL small
.STACK 100h

;----- Insert INCLUDE "filename" directives here
;----- Insert EQU and = equates here

INCLUDE procs.inc

    LOCALS

.DATA
    msg_bin  db 'AL desplegado en Binario:',0
    new_line db 13,10,0

.CODE

    ;-----   Insert program, subrutine call, etc., here

    Main PROC
        mov ax,@data    ; Inicializar DS al la direccion
        mov ds,ax       ; del segmento de Datos (.DATA)

        call clrscr
        mov dx, offset msg_bin
        call puts
        mov dx, offset new_line
        call puts
        mov al,0a7h     ; dato a imprimir en binario
        call printBin

        ; Fin del Programa ;
        mov ah,4Ch
        mov al,0
        int 21h
        ENDP

    printBin PROC
            push ax
            push cx
            mov ah,al
            mov cx,8
    @@print: xor al, al ; se pone en 0 al registro AL
            shl ah,1
            adc al,'0'
            call putchar
            loop @@print

            pop cx
            pop ax
            ret
            ENDP

END
```

Archivo **printBin.asm**

Procedimientos

La pila almacena la dirección de regreso siempre que una subrutina es llamada durante la ejecución del programa.

La instrucción **CALL** empuja en la pila la dirección de la siguiente instrucción.

La instrucción **RET** remueve una dirección de la pila así que el programa regresa a la instrucción siguiente del CALL.

Procedimientos

Tipos de Llamadas

CALL Cercano

CALL Lejano

CALL con Operando de Registro

CALL con Direccionamiento Indirecto de Memoria

Procedimientos

CALL Cercano

Es de tres bytes de largo con el primer byte conteniendo el código de operación y el segundo y tercero contienen el desplazamiento o distancia de ±32K.

Esta es similar a la forma la instrucción de salto cercano.

Cuando la **CALL cercana** se ejecuta, primero coloca en la **Pila** la dirección de la siguiente instrucción a ejecutar (coloca el valor de **IP**). Después de guardar esta dirección de regreso, entonces **suma el desplazamiento** del byte 2 y 3 a IP para transferir el control al procedimiento.

No hay instrucciones CALL cortas.

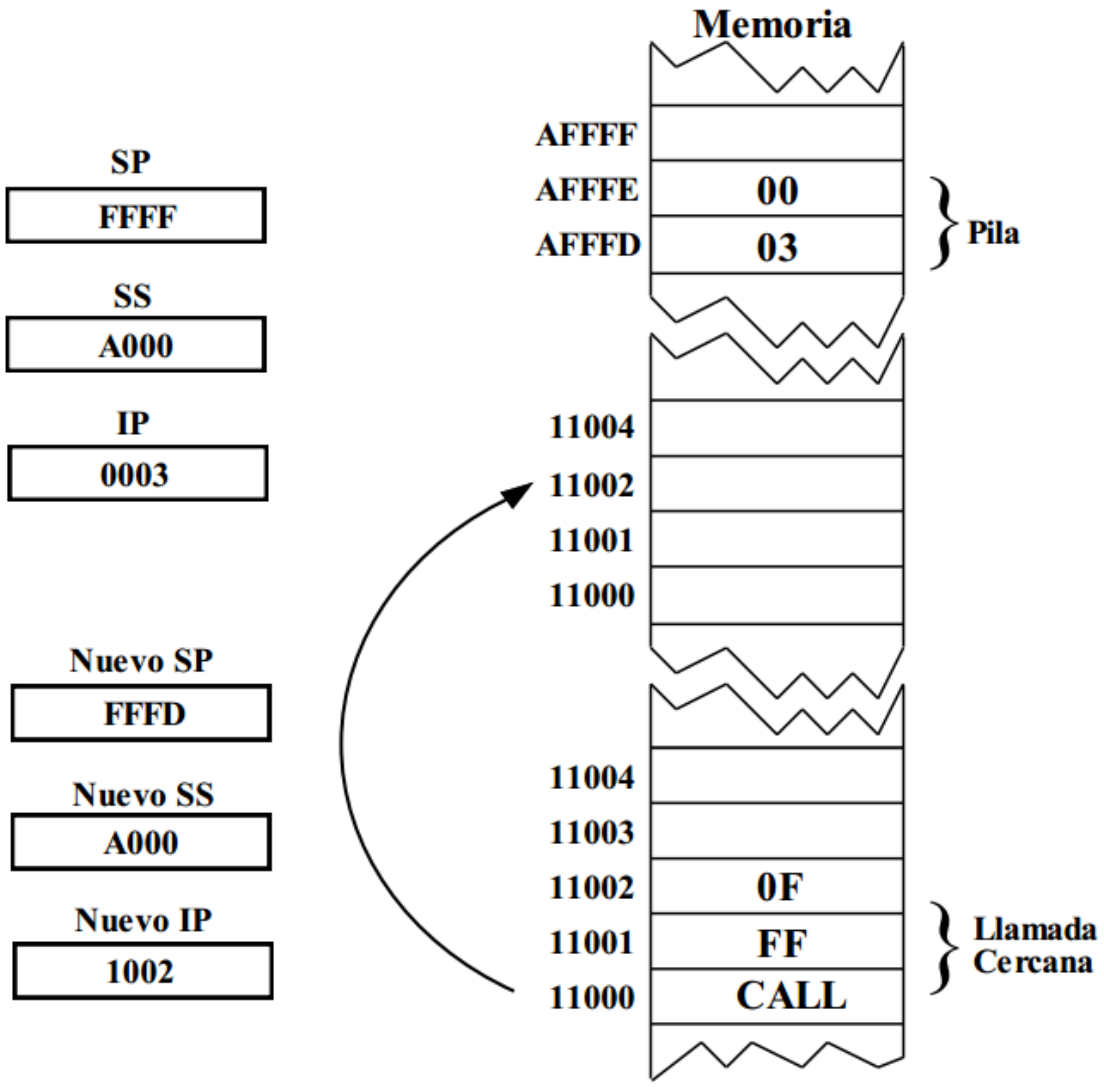
Procedimientos

¿Porque guardar IP en la pila?

El apuntador de instrucción siempre apunta a la siguiente instrucción en el programa.

Para la instrucción CALL, el contenido de IP es empujado dentro de la pila así que el control del programa pasa a la instrucción seguida de CALL después de que el procedimiento termina.

Procedimientos



Procedimientos

CALL Lejano

La instrucción CALL lejana es como el salto lejano porque esta puede llamar un procedimiento almacenado en cualquier parte de la memoria en el sistema.

La CALL lejana es una instrucción de 5 bytes que contiene un código de operación seguido por el próximo valor de IP y los bytes 4 y 5 contienen el nuevo valor para CS.

La instrucción CALL lejana coloca el contenido de **IP** y **CS** en la **Pila** antes de saltar a la dirección indicada por los bytes 2 al 5 de la instrucción.

Procedimientos

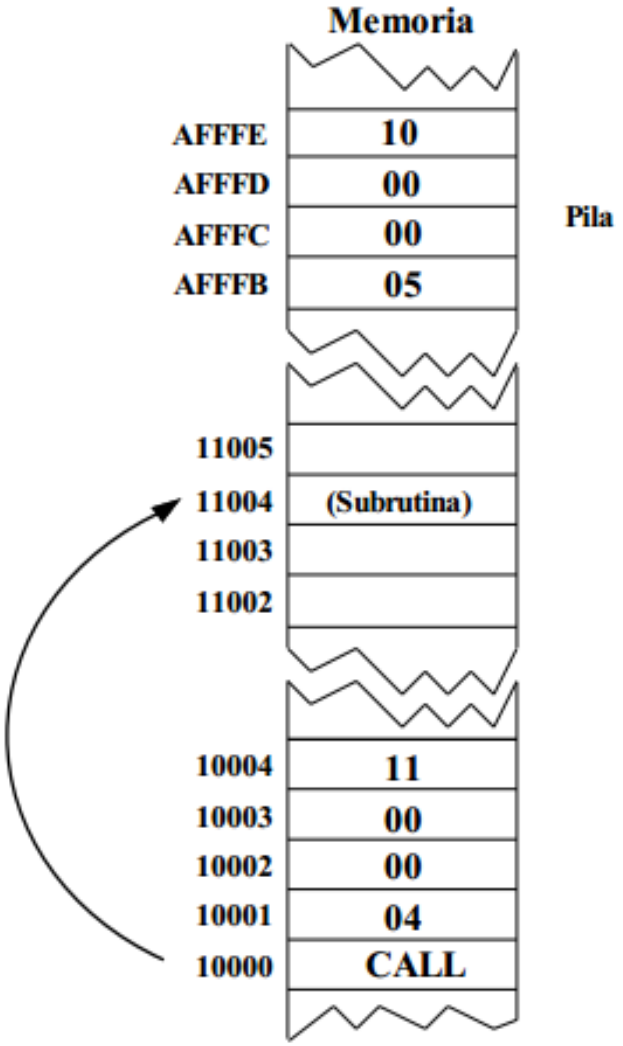


FIGURA 2. Instruccion CALL lejano

Procedimientos

CALLs con Operando de Registro

Como los saltos, las CALLs pueden contener también un registro como operando.

Un ejemplo es la instrucción **CALL BX**. Esta instrucción empuja el contenido de IP en la Pila. Luego se salta a la dirección de desplazamiento localizado en BX, en el presente segmento de código.

Este tipo de CALL siempre usa una dirección de 16 bits almacenado en cualquier registro de 16 bits excepto los registros de segmento.

Procedimientos

CALLs con Direccionamiento Indirecto de Memoria

Como los saltos, las CALLs pueden usar direccionamiento indirecto a memoria.

Ejemplo:

CALL [SI]

Procedimientos

CALL vs Saltos

La instrucción CALL difiere de las instrucciones de salto porque una CALL guarda en la pila una dirección de retorno.

La dirección de retorno permite que se pueda ejecutar la instrucción que seguía al CALL.

Procedimientos

RET

La instrucción de retorno (RET) obtiene un número de 16 bits de la pila y lo coloca en IP (retorno cercano), o un número de 32 bits y lo coloca en IP y CS (retorno lejano).

Las instrucciones de retorno cercano y lejano se definen con la directiva PROC en el procedimiento. Esta selecciona automáticamente la instrucción de retorno apropiada.

Procedimientos

Ejemplo de Procedimientos

Procedimientos

Ejemplo:

```
        call step
        mov ax,cx
        mov cx,dx
        call step
        mov dx,bx


step PROC
        add bx,dx
        mov dx,cx
        ret
ENDP
```

1

1. Se empuja en la Pila la dirección de retorno, es decir, el valor de **IP** (o si es una **CALL lejana**, se empuja **CS** y después **IP**)

1.1 Se pone el nuevo valor a IP de forma que apunte a la primera instrucción de la rutina (en este ejemplo, que apunte a *'add bx,dx'*). El nuevo valor de **IP** se obtiene ya sea **sumándole** el **desplazamiento** (**CALL cercana**) o poniendo el **valor directamente** al igual que al registro **CS** (**CALL lejana**)

Procedimientos

Ejemplo:

```
        call step
        mov ax,cx
        mov cx,dx
        call step
        mov dx,bx


step PROC
        add bx,dx
        mov dx,cx
        ret
ENDP
```

2. Se ejecutan secuencialmente las instrucciones de la rutina *'step'*, hasta llegar a RET.

2.1 La instrucción RET remueve un valor de la pila y lo coloca en IP (si es una CALL lejana remueve otro y lo coloca en CS)

Procedimientos

Ejemplo:

```
    call step
    mov ax,cx
    mov cx,dx
    call step
    mov dx,bx


step PROC
    add bx,dx
    mov dx,cx
    ret
ENDP
```

3. Causando que ahora IP apunte a *'mov ax,cx'*. Por lo que continua la ejecución secuencial de las instrucciones.

Instrucciones misceláneas

Control del Bit de Acarreo

Hay tres instrucciones que pueden controlan el contenido de la bandera de acarreo:

STC: poner en uno el acarreo

CLC: borrar el acarreo

CMC: complementar el acarreo

El bit de acarreo indica el estado de la ultima operación aritmética, sin embargo también se le pueden dar otros usos. La tarea más común es para indicar un error en el retorno de una subrutina.

Por ejemplo, en una subrutina que lee datos de un archivo, esta operación puede ser desarrollada satisfactoriamente o puede ocurrir un error tal como: archivo no encontrado. Después de retornar de esa subrutina, se puede poner CF = 1 indicando que ha ocurrido un error, y CF = 0, si no ha ocurrido error.

NOP

Una **NOP** no realiza ninguna operación.

Cuando el microprocesador 8088/86 encuentra una instrucción de no operación (NOP), toma tres períodos de reloj para ejecutarla.

Se puede utilizar para programar un delay por software.