AAAC & TAC
Clase #7
Procesador Intel x86

Facultad de Ingeniería
Universidad de la República

Instituto de Computación
Curso 2014

Contenido

- Generalidades 80x86
- Modos de direccionamiento
- Set de instrucciones
- Assembler
- Compilando algunos ejemplos
- Rutinas recursivas
- Manejadores de dispositivos

Microprocesadores Intel 8086/8088

- Los procesadores Intel 8086 y 8088 son la base del IBM-PC y compatibles
  (8086 introducido en 1978, primer IBM-PC en 1981)
- Todos los procesadores Intel, AMD y otros están basados en el original 8086/8, y son compatibles.
  - En el arranque, Pentiums, Athlons etc se ven como un 8086: Instruction Pointer apunta a FFFF0H
- 8086 es un procesador de 16-bit
  - 16-bit data registers
  - 16 or 8 bit external data bus
- Algunas técnicas para optimizar la performance, por ejemplo la Unidad de Prefetch
- Segmentos: Offset memory model
- Formato de datos Little-Endian

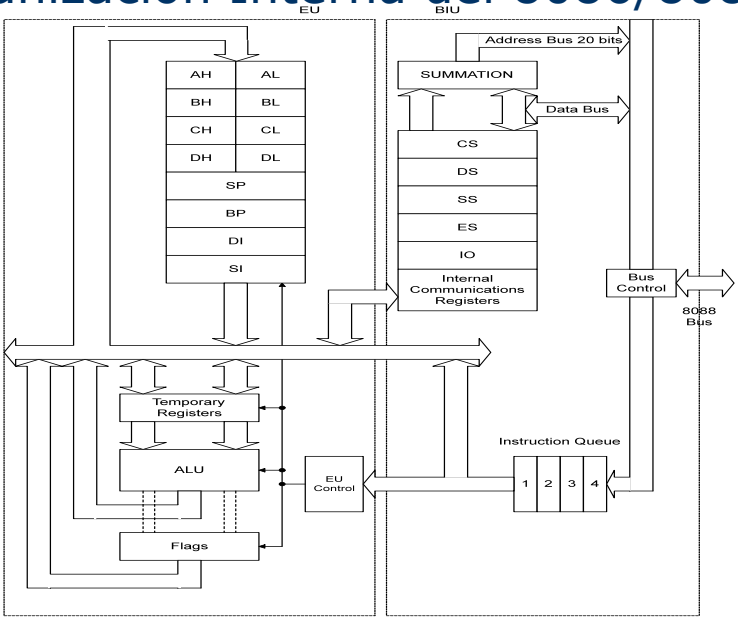
8086/8088

- Los accesos a memoria enlentecen el ciclo de ejecución
  - El 8086/8 usa una cola de instrucciones para mejorar la performance
  - Mientras el procesador decodifica y ejecuta una instrucción, la unidad de interfaz con el bus puede leer nuevas instrucciones, pues en ese momento el bus está en desuso
- El IBM PC original usa el microprocesador 8088
  - 8088 es similar al 8086 pero tiene un bus externo de 8 bits y una cola de instrucciones de solo 4 bytes
  - El bus de 8-bit reduce la performance, pero baja los costos
- La arquitectura del 8086 y el 8088 se puede considerar en conjunto
  - Algunos clones del PC usaban el 8086 para mejorar la performance

Unidades funcionales del 8086/8088

- Execution unit (EU) – ejecuta las instrucciones
- Bus interface unit (BIU) – fetch de instructiones y operandos, escritura de resultados
- Prefetch queue: 8086/6 bytes, 8088/4 bytes

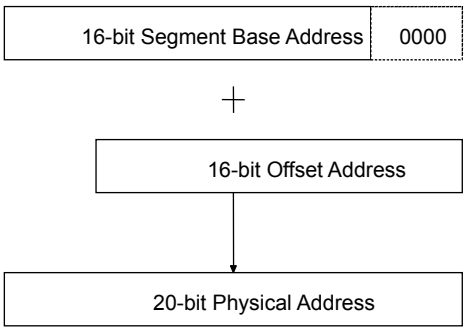
Organización Interna del 8086/8088

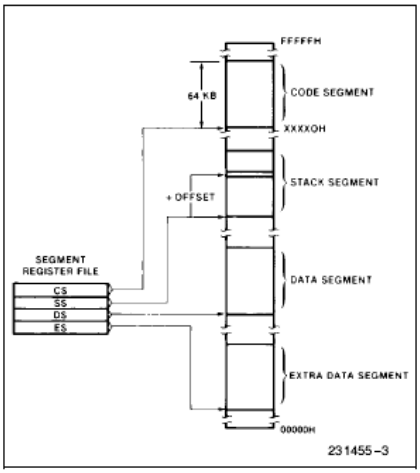


Elementos de la BIU

- Instruction Queue: la próxima instrucción u operando puede ser leído desde memoria mientras el procesador ejecuta la instrucción corriente
  - Dado que la interfaz de memoria es mucho mas lenta que el procesador, la cola de instrucciones mejora la performance global del sistema.
- Registros de Segmento:
  - CS, DS, SS y ES: registers de 16 bit
  - Usados como base para generar las direcciones de 20 bits
  - Permiten al 8086/8088 direccionar 1Mbyte de memoria
  - El programa puede utilizarlos para apuntar a diferentes segmentos durante la ejecución
- Instruction Pointer (IP) contiene el offset de la dirección de la próxima instrucción (distancia en bytes desde la dirección contenida en el registro CS)

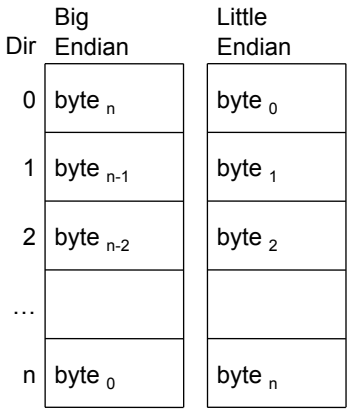
Direcciones de 20 bits en el 8086/8088





Little-Endian y Big-Endian

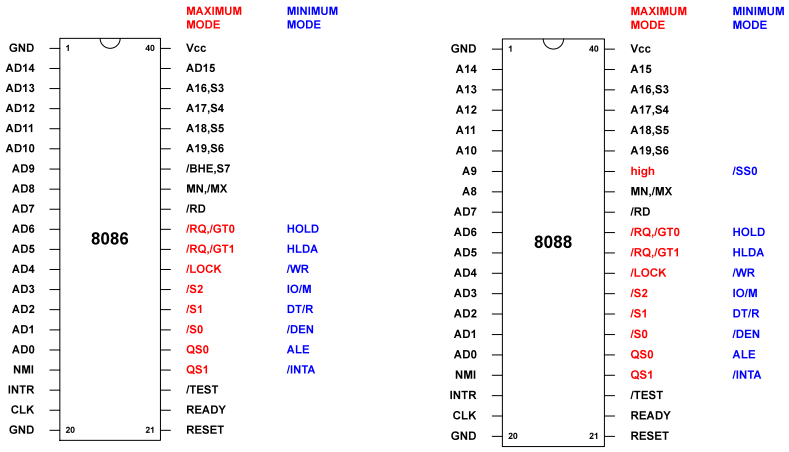
- Indican de que forma una secuencia de bytes se almacena en memoria.
- *Little endian* indica que el byte menos significativo de la secuencia de bytes será almacenado en la dirección de memoria menor.
- En la imagen se muestra como una secuencia de bytes, $byte_n$... $byte_0$, se guarda en memoria



Construyendo un sistema basado en el 8086/8

- Los microprocesadores 8086/8 necesitan circuitos extra para construir un sistema
- Los buses de datos y direcciones se multiplexan en los mismos pines del procesador
  - Se necesita lógica extra para demultiplexar direcciones y datos y poder acceder a RAMs y ROMs
- Modos de funcionamiento
  - Máximo
  - Mínimo
- El Modo Máximo el 8086/8 necesita *al menos* los siguientes circuitos extra: 8288 Bus Controller, 8284A Clock Generator, 74HC373s y 74HC245s

Modos de funcionamiento



Evolución: 80186/80188

- Set de Instrucciones aumentado
- Componentes del sistema "on-chip"
  - Clock generator
  - Controlador DMA
  - Controlador interrupciones
  - Timer
  - etc…
- No utilizado en PCs
- Popular en sistemas embebidos
  - Controladores
  - Hardware dedicado

Señal RESET#

- RESET# es activa en nivel bajo. Pone al 8086/8 en un estado definido
- Limpia los registros de flags, segmento, etc
- Pone la dirección de programa efectiva en 0FFFF0h (CS=0F000h, IP=0FFF0h)
  - Programas en el 8086/8 siempre arrancan en FFFF0H después de un Reset
  - En esta dirección deben instalarse las rutinas de inicialización del sistema: en el PC, la ROM BIOS
- Esta característica se mantiene en las últimas generaciones de procesadores

Direccionamiento:
memoria y E/S por separado (i)

- Los procesadores Intel tienen en espacio de direccionamiento de E/S separado de la memoria principal (Código y Datos)
- Decodificación de direcciones de dispositivos de E/S por separado
  - Uso de las señales IOR# and IOW#
- Se corresponden con instrucciones separadas para acceder la E/S y la memoria
  - MOV AL, [BX] ; acceso a memoria
  - IN AL, 2Ch ; acceso a E/S
- Algunos procesadores tienen un espacio de direcciones unificado. Los dispositivos de E/S son decodificados en el mapa de memoria principal
  - "E/S mapeada en memoria"

Direccionamiento:
memoria y E/S por separado (ii)

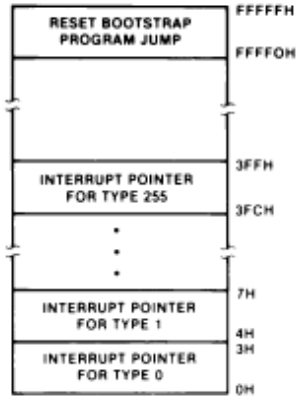
- Ventajas de la E/S mapeada en memoria
  - Dispositivos de E/S accedidos por instrucciones normales - no se necesitan instrucciones separadas
  - Se reduce el tamaño del set de instrucciones
  - No se necesitan pines especiales (IOR#, IOW#)
- Ventajas de espacios de direccionamiento separados
  - Todo el mapa de direcciones disponible para memoria
  - La E/S puede usar instrucciones más pequeñas y rápidas
  - Fácil distinguir accesos de E/S en lenguaje ensamblador
  - Menos hardware para decodificar E/S.

Interrupciones

- Tipos
  - Hardware: dispositivos de entrada salida
  - Internas: división entre cero
  - Software: llamadas al sistema
  - No enmascarables.
- Cada interrupción lleva asociado un número que identifica al servicio que se debe invocar.

Vector de Interrupciones

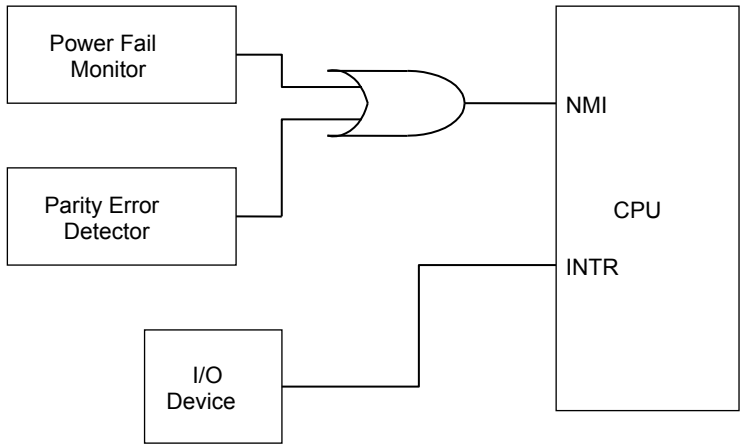
- Las localizaciones de memoria 00000H a 003FFH están reservadas para las interrupciones
- Existen 256 tipos posibles de interrupciones
  - Cada Handler o Rutina de Servicio de Interrupción está direccionada por un puntero de 4 bytes: 16 bits de segmento y 16 bits de offset
- Las rutinas y los punteros deben instalarse antes de habilitar las interrupciones
  - Servicios de la BIOS
  - Servicios del Sistema Operativo



Interrupciones enmascarables y no enmascarables

- Las interrupciones pueden enmascararse globalmente usando la Interrupt Enable Flag (IE o I)
- IE es seteada por la instrucción STI y reseteada mediante CLI
- Interrupciones no enmascarables (Non Maskable Interrupts-NMI) son prioritarias y como su nombre indica NO se pueden enmascarar
- Uso de la NMI
  - Parity Error
  - Power fail
  - Etc…

Ejemplo de NMI



Generalidades 8086/8080

- Los bytes y palabras pueden residir en cualquier lugar de la memoria (no es necesario que estén alineados)
- Puede operar con números
  - Binarios (con o sin signo de 8 ó 16 bits)
  - BCD
- Dispone de 92 instrucciones.
- Existen 7 modos de direccionamiento.
- Frecuencia típica
  - 4,77 MHz (8080)
  - 8 MHz (8086)
- Las instrucciones más rápidas se ejecutan en dos ciclos de reloj y las lentas en 206 ciclos.
- Se pueden direccionar hasta 64K puertos de E/S.

Registros (i)

- Datos o almacenamiento temporal
  - AX, acumulador.
  - BX, base.
  - CX, contador.
  - DX, dato.

Registro	Byte Superior	Byte Inferior
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL
	FEDCBA98	76543210

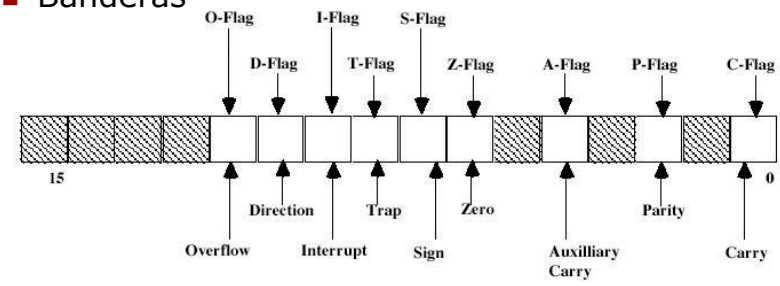
← Bit hex

Registros (ii)

- Segmento
  - CS, código
  - DS, dato.
  - SS, pila.
  - ES, extra.
- Punteros a pila
  - SP, tope de la pila.
  - BP, base a la pila.

Registros (iii)

- Registros de índice
  - SI, índice origen.
  - DI, índice destino.
- Puntero a instrucción
- Banderas



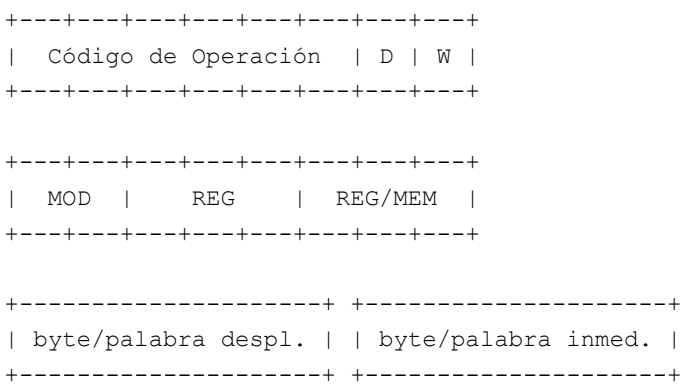
Modos de direccionamiento (i)

- Se entiende por modos de direccionamiento a las formas diferentes que pueden tomar los parámetros de las instrucciones del procesador.
- Distinguiremos fundamentalmente cuatro modos diferentes:
  - REGISTRO
    - Un parámetro que direcciona a un registro está utilizando el modo de direccionamiento REGISTRO.
    - Ej: MOV Ax,Bx
    - En este ejemplo los dos parámetros direccionan un registro.
  - VALOR o INMEDIATO
    - Utilizado cuando se hace referencia a un valor constante que se codifica junto con la instrucción. Es decir dicho parámetro representa a su valor y no a una dirección de memoria o un registro que lo contiene.
    - Ej: MOV Ax,500

Modos de direccionamiento (ii)

- DIRECTO
  - Se utiliza el modo directo cuando se referencia a una dirección de memoria y la misma está codificada junto con la instrucción.
  - Ej: MOV AL,[127]
  - En este ejemplo el offset de la dirección de memoria se codifica junto con la instrucción y el segmento se asume a DS. Si MEMORIA es un array de bytes que representa a la memoria:
  - AL := MEMORIA[DS:127]
- INDIRECTO
  - Se utiliza el modo directo cuando se referencia a una dirección de memoria a través de uno o varios registros
  - Ej: MOV AL,[Bx]
  - Aquí el offset de la dirección de memoria esta contenido en el registro Bx y al igual que el caso anterior como no se especifica el segmento se asume DS. Ejemplo:
  - AL := MEMORIA [DS:Bx]

Formato de instrucción (i)



Formato de instrucción (ii)

- El **código de operación** ocupa 6 bits.
- **D** indica que el operando destino está en el campo registro.
- **W** indica que los operandos son de tipo palabra.
- **MOD** indica el modo de direccionamiento
  - 00 sin desplazamiento.
  - 01 desplazamiento de 8 bits
  - 10 desplazamiento de 16 bits
  - 11 registro
- **REG** indica el registro involucrado en la instrucción
- **R/M**, en el caso de modo registro (MOD=11) se codifica igual que el campo REG; en caso contrario se indica la forma en que se direcciona la memoria

Set de Instrucciones

- Aritméticas
- Lógicas
- Desplazamiento
- Manejo de Flags
- Bifurcación Incondicional
- Bifurcación Condicional
- Interrupciones
- Manejo de Stack

- Ver cartilla del curso

Constantes

- Valores binarios, tiras de ceros y unos.
  - Terminan con b o B.
  - Ej: 100011 b.
- Valores octales.
  - Terminan con o, O, q o Q.
  - Ej: 664 o.
- Valores hexadecimales
  - Empiezan con 0..9.
  - Terminan con h o H.
  - Ejemplo: 0FFFh
- Valores decimales.
- Strings, secuencias de caracteres ASCII.
  - Van entre comillas simples.
  - 'Hola mundo'.

Operadores

- Operador +, -, *, /, mod, shl, shr, and, or y xor.
  - Formato: valor1 oper valor2
  - Ejemplos
    - 75+25
    - 80*25
    - 1002/123
    - 1100001 SHR 2
- Operador not
  - Formato: oper valor
  - Ejemplo: not 1100001b
- Operador offset y seg
  - Formato: oper {etiqueta|variable}

Acceso a memoria (1/2)

- Todos las direcciones se especifica como direcciones segmentadas (segmento:desplazamiento).
- El desplazamiento se define según la expresión:

```
{Bx|Bp}[+{Si|Di}][+desplazamiento]|
{Si|Di}[+desplazamiento]|
desplazamiento
```

- Ejemplos de desplazamientos:
  - BX
  - BX+DI
  - BX+SI+2
- Ejemplos de direcciones segmentadas:
  - ES:[BP+SI]
  - [BP+4]
- Las direcciones segmentadas son traducidas automáticamente por el hardware multiplicando el segmento por 16 y luego sumando el desplazamiento.

Acceso a memoria (2/2)

- Desde la perspectiva del programador
  - Las direcciones son siempre segmentadas.
  - Nunca puede especificar una dirección real de 20 bits.
- El microprocesador 8086 permite que a lo sumo uno de los operandos de la mayoría de las instrucciones esté almacenado en memoria.
- Ejemplos:
  - inc [bp]
  - mov [bx],ax
  - xor [bx],al
  - mov [bx],[bx+2] Prohibido

Instrucciones prohibidas

- No permitido
  - mul 4
  - mov ES,4
  - mov AX,[BX*4]
  - mov AX,BX*4
  - mov AX,BX+1
  - push BL
  - push 4
  - mov BL,AX

Directivas(1/5)

- La directiva EQU
  - La forma de esta directiva es:
    identificador EQU expresión
  - Ejemplo:

```
NULL EQU 0
TAM_ELEM EQU 4*8
interElem EQU CX
MASK EQU 100010 b
MASK_2 EQU MASK SHR 2
```

Directivas(2/5)

- Las directivas DB, DW, DDW y DUP
  - La forma de estas directivas es:

    etiqueta {*DB*|*DW*|*DDW*} expresión1, expresión2,...

    cantidad *dup* (valor)
  - Ejemplo:

```
iterElem DW 0
vectorChico DB 1,2,3,4,5,6
vectorGrande DB 1024 dup(0)
...
mov ax,[iterElem]
mov bl,[vectorChico+2]
```

Directivas(3/5)

- La directiva MACRO
  - La forma de esta directiva es:

```
nombreMacro MACRO [parametro[,parametro...]]
  instrucciones
ENDM
```

  - Ejemplo:

```
sqr MACRO registro
  mov AX,registro
  mul registro
  mov registro,AX
ENDM
```

Directivas(4/5)

- Las directivas byte ptr y word ptr
  - La forma de esta directiva es:
    {byte|word} ptr elemento
  - Ejemplo:
    mov byte ptr [ES:BX], 0
    mul word ptr [DI]

Directivas(5/5)

- La directiva PROC
  - La forma de esta directiva es:
    nombreProc PROC [NEAR|FAR]
- La directiva ENDP
  - La forma de esta directiva es:
    nombreProc ENDP
- Definición de un procedimiento:
    nombreProc PROC atributo
      …
    nombreProc ENDP

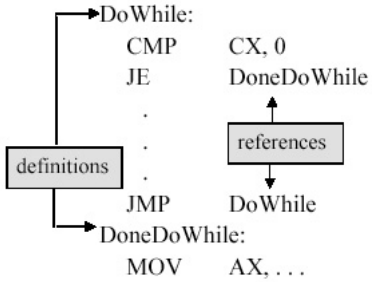
Instrucciones (1/2)

- Es una secuencia de 1 a 6 bytes.
- Formato

  [etiqueta] nombreInstruccion [operandos] [comentarios]
  - Etiqueta (i)
    - Nombre simbólico que referencia la primera posición de la instrucción.
    - Puede estar formada por
      - Letra A a Z.
      - Números 0 a 9.
      - Caracteres especiales @ - . $.

Instrucciones (2/2)

- Etiqueta (ii)

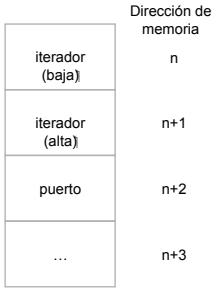


- Los comentarios comienzan con un ';'

Acceso a estructuras de datos en memoria (1/3)

- Las variables que se definen contiguas en el programa aparecen contiguas en memoria. Cada una de ellas ocupa tantos bytes como sean necesario por su tipo.
- Ejemplo:

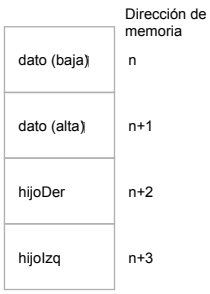
```
iterador:integer;
puerto:byte;
…
```



Acceso a estructuras de datos en memoria (2/3)

- Los campos de una variable de tipo estructurado se almacenan en posiciones contiguas de memoria en el orden en que aparecen declaradas y ocupando tantos bytes como sean necesarios para alojar al tipo del campo.
- Ejemplo:

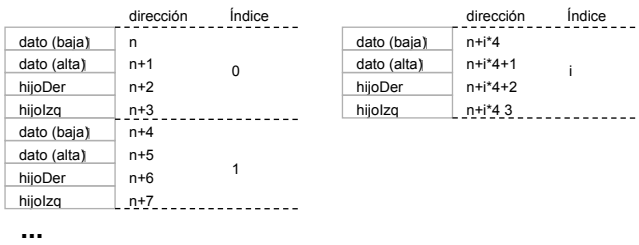
```
type nodo=
 record
   dato:integer;
   hijoDer:byte;
   hijoIzq:byte;
 end
```



Acceso a estructuras de datos en memoria (3/3)

- Los elementos de una variable de tipo array se almacenan en posiciones contiguas de memoria en el orden en que aparecen declaradas.
- Ejemplo:

```
type arbol=array[0..(MAX_NODOS-1)] of nodo;
```



Compilado de estructuras de control (1/2)

- if-then-else

Alto nivel	Asembler

`if (i<>0) then`
`  {bloque del if}`
`else`
`  {bloque del else}`
	`cmp i,0`
	`  je else`
	`  ; aquí va el bloque que`
	`  ; corresponde al then`
	`  jmp finIf`
	`else:`
	`  ; aquí va el bloque que`
	`  ; corresponde al else`
	`finIf:`

Compilado de estructuras de control (2/2)

- while

Alto nivel	Asembler

`while (i<n) do`
`  {bloque del while}`
	`while:`
	`  cmp i,n`
	`  jae finWhile`
	`  ; aquí va el bloque que`
	`  ; corresponde al cuerpo`
	`  ; del while`
	`  jmp while`
	`finWhile:`

Compilando una función(1/3)

- La función len retorna el largo de un string.
- Alto nivel:

```
String=array[0..(LARGO_MAXIMO)] of byte;

function len(str: String):integer;
  iterStr: integer;
begin
  iterStr:=0;
  while (str[iterStr]<>NULL) do
    iterStr:=iterStr+1;
  len:=iterStr;
end
```

Compilando una función(2/3)

- Asembler

```
NULL EQU 0

; el desplazamiento de str viene en bx
; el resultado se devuelve en di
len proc
  xor di,di
while:
  cmp byte ptr [bx+di],NULL
  je fin
  inc di
  jmp while
fin:
  ret
len endp
```

Compilando una función(3/3)

- Invocando a la función

```
miString db 'hola mundo'
         db NULL
…
mov bx, offset miString
call len
cmp di, …
```

```
;el string esta en la
;direccion segmentada 100:1000
mov bx,1000
mov ax,100
mov ds,ax
call len
```

```
;el string esta en la
;direccion absoluta 0x98765
mov bx,5
mov ax,9876
mov ds,ax
call len
```

Rutinas recursivas(1/10)

- Introducción
  - Salvar contexto.
  - Manejo adecuado del stack.
  - Atención con la dirección de retorno.
  - Limitadas por el tamaño del stack.

Rutinas recursivas(2/10)

- Ejemplo: Función Factorial
  - Definición

```
factorial(0)=1
factorial(n)= factorial(n-1)*n
```

  - Alto Nivel

```
function factorial(n: integer):integer;
  begin
   if (n=0) then factorial:=1
   else factorial:= factorial(n-1)*n;
  end
```

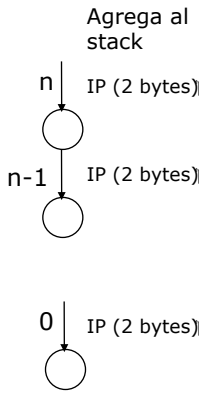
Rutinas recursivas(3/10)

Llamada	Asembler

`mov ax,n`
`call fact`
`mov resultado,bx`
	`fact proc`
	`  cmp ax,0  ; compara n con cero`
	`  je esCero`
	`  dec ax  ; ajusta parámetro para la invocación`
	`  call fact ; realiza la llamada recursiva`
	`  inc ax`
	`  mov cx,ax ; guarda ax pues mul lo modifica`
	`  mul bx  ; calcula el paso recursivo`
	`  mov bx,ax ; asigna el resultado del paso`
	`  ; recursivo`
	`  mov ax,cx ; restaura ax`
	`  jmp fin`
	`esCero:`
	`  mov bx,1  ; asigna el resultado del paso base`
	`fin:`
	`  ret`
	`fact endp`

Rutinas recursivas(4/10)

- Cálculo del consumo de stack
  - Planteo de la recurrencia
    - consumo(0)=2
    - consumo(n)=2+consumo(n-1)
  - Resolviendo la recurrencia
    - consumo(n)=2*(n+1)



Rutinas recursivas(5/10)

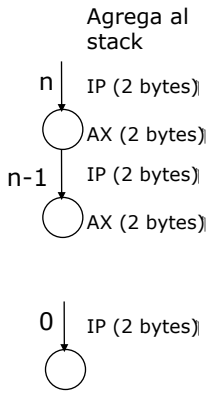
Llamada	Asembler

```
mov ax,n
call fact
mov resultado,bx
```

```
fact proc
 cmp ax,0  ; compara n con cero
 je esCero
 push ax   ; salva contexto
 dec ax    ; ajusta parámetro para la invocación
 call fact ; realiza la llamada recursiva
 pop ax    ; restaura el contexto
 mul bx    ; calcula el paso recursivo
 mov bx,ax ; asigna el resultado del paso
           ; recursivo
 jmp fin
esCero:
 mov bx,1  ; asigna el resultado del paso base
fin:
 ret
fact endp
```

Rutinas recursivas(6/10)

- Cálculo del consumo de stack
  - Planteo de la recurrencia
    - consumo(0)=2
    - consumo(n)=4+consumo(n-1)
  - Resolviendo la recurrencia
    - consumo(n)=4*n+2



Rutinas recursivas(7/10)

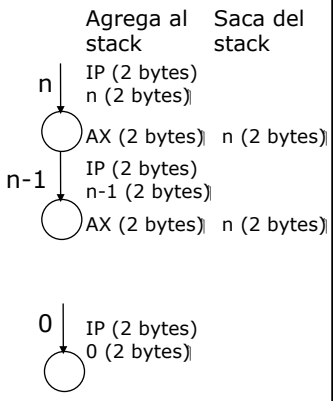
Llamada	Asembler

```
push n
call fact
pop resultado
```

```
fact proc
  pop dx ; saca dirección de retorno del stack
  pop ax ; saca parámetro del stack
  push dx ; coloca la dirección de retorno en el stack
  cmp ax,0 ; compara n con cero
  je esCero
  push ax ; guarda el contexto
  dec ax  ; ajusta parámetro para la invocación
  push ax ; coloca parámetro en el stack
  call fact ; realiza la llamada recursiva
  pop bx ; retira del stack el resultado de fact(n-1)
  pop ax ; restaura el contexto
  mul bx ; calcula el paso recursivo
  mov bx,ax ; asigna el resultado del paso recursivo
  jmp fin
esCero:
  mov bx,1  ; asigna el resultado del paso base
fin: ; acomoda el stack para el retorno
  pop dx ; saca dirección de retorno del stack
  push bx ; coloca el resultado en el stack
  push dx ; coloca la dirección de retorno en el stack
  ret
fact endp
```

Rutinas recursivas(8/10)

- Cálculo del consumo de stack
  - Planteo de la recurrencia
    - consumo(0)=4
    - consumo(n)=4+consumo(n-1)
  - Resolviendo la recurrencia
    - consumo(n)=4*n+4



Rutinas recursivas(9/10)

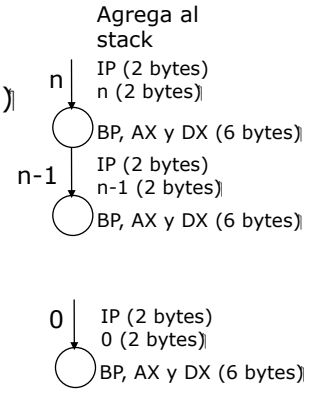
Llamada	Asembler

```
push n
call fact
pop resultado
```

```
fact proc
  push bp ; guarda el registro bp
  mov bp,sp ; apunta con bp al tope de la pila
  push ax   ; conserva registros
  push dx
  cmp word ptr [bp+4],0 ; compara n con cero
  je esCero
  mov ax,[bp+4] ; obtiene n
  dec ax ; ajusta parámetro para la invocación
  push ax ; coloca parámetro en el stack
  call fact ; realiza la llamada recursiva
  pop ax ; obtiene el resultado
  mul word ptr [bp+4] ; calcula el paso recursivo
  jmp fin
esCero:
  mov ax,1  ; asigna el resultado del paso base
fin: ; acomoda el stack para el retorno
  ;reemplaza el parámetro de entrada con el resultado
  mov [bp+4],ax
  pop dx ; restaura registros
  pop ax
  pop bp
  ret
fact endp
```

Rutinas recursivas(10/10)

- Cálculo del consumo de stack
  - Planteo de la recurrencia
    - consumo(0)=10
    - consumo(n)=10+consumo(n-1)
  - Resolviendo la recurrencia
    - consumo(n)=10*(n+1)



Interrupciones(1/5)

- Interrupciones de Hardware o Software
- Acciones tomadas por el microprocesador:
  - Salva las banderas en el stack
  - Deshabilita interrupciones
  - Salva CS:IP actual en el stack
  - Salta a la rutina de atención requerida
- Notas
  - El programador debe salvar el contexto
  - Cuidado con el stack
  - El vector de interrupciones ocupa desde la dirección de memoria absoluta 0 a la 1023.
  - Cada elemento del vector de interrupciones es una dirección segmentada que indica donde se encuentra el código del manejador de la interrupción.

Interrupciones(2/5)

- IRET
  - Restaura CS:IP desde el stack
  - Restaura el registro de banderas desde el stack.
  - ¿Quién habilita interrupciones?

Interrupciones(3/5)

- Problema
  - Se desea controlar un LED conectado al bit menos significativo del byte de ES sólo escritura ES_LED, de modo que permanezca por siempre un segundo prendido y otro segundo apagado.
  - Nota: La interrupción de TIMER es la 08h (elemento de índice 8 dentro del vector de interrupciones), y se ejecuta 18 veces por segundo.

Interrupciones(4/5)

- Alto nivel

```
var
  cantTics: integer;
  estadoLED: byte;
procedure principal();
begin
  cantTics:=0;
  estadoLED:=0;
  out(ES_LED,estadoLED);
  {deshabilitarInterrupciones}
  {instalarManejadorTimer}
  {habilitarInterrupciones}
end;
```

```
procedure TIMER();
begin
  if (cantTics=18) then
    begin
      cantTics:=0;
      estadoLED:=notb(estadoLED);
      out(ES_LED,estadoLED);
    end
  else
    cantTics:=cantTics+1;
end
```

Interrupciones(5/5)

- Asembler

```
ES_LED equ …
cantTics db 0
estadoLED db 0

principal proc
  mov byte ptr CS:[cantTics],0 ;cantTics:=0
  mov byte ptr CS:[estadoLED],0 ;estadoLED:=0
  mov dx,ES_LED
  xor al,al
  out dx,al   ; out(ES_LED,estadoLED)
  cli         ; {deshabilitoInterrupciones}
  xor ax,ax
  xor bx,bx
  mov es,ax
  ;instaloManejadorTimer sin llamar al
  ;manejador actual
  ;indico desplazamiento y segmento
  mov word ptr es:[bx+8*4],offset TIMER
  mov  word ptr es:[bx+8*4+2],cs
  sti ; {habilitoInterrupciones}
principal endp
```

```
TIEMPO proc far
  push ax
  push bx
  mov bl,cs:[cantTics]
  cmp bl,18
  jne else
  mov bl,0
  mov al, cs:[estadoLED]
  not al
  mov cs:[estadoLED],al
  mov dx,ES_LED
  out dx,al
  jmp fin
else:
  inc bl
fin:
  mov cs:[cantTics],bl
  pop bx
  pop ax
  iret
TIEMPO endp
```

Referencias

- 8088-8086/8087 Programación Ensamblador en entorno MS DOS, Miguel Angel Rodriguez-Roselló, Anaya, 1988.
- The Art of Assembly Language, Randall Hyde; Second Edition edition (March 22, 2010). ISBN-10: 1593272073, ISBN-13: 978-1593272074
  - http://www.plantation-productions.com/Webster/www.artofasm.com/index.html