Guía final

Shifts

El bit que se pierde se guarda en CF

Logical

Mueve bits *n* lugares a la derecha/izquierda y rellena con cero

- SHR == * 2^n
- SHL == / 2^n

Arithmetic

Mueve los bits *n* lugares a la derecha/izquiera y copia el MSB (bit del signo) en la nueva posición

- SAL == SHL (llena de ceros)
- SAR

Shift Double

Muevo los bits *n* lugares izquierda/derecha y relleno con los *n* MSB/LSB de la fuente

- SHLD
- SHRD

Rotate

Muevo los bits *n* lugares a la izquierda/derecha de manera cíclica

No pierdo bits! (El MSB/LSB se guarda en CF y se mueve al LSB/MSB)

- ROL
- ROR

Rotate Carry

Utilizo CF como extensión de mi operando

- RCL
  - CF $\rightarrow$ LSB
  - MSB $\rightarrow$ CF
- RCR
  - CF $\rightarrow$ MSB
  - LSB $\rightarrow$ CF

NOTA: CLC == Clear CF, STC == Set CF

Extended Precision Operations

- **Addition**: Sumo + + CF

  ADC ,

- **Substraction:** - - CF

  SBB

FPU

- signo + mantissa (número) + exponente (e.g. -38.75×10^5)
- para obtener exponente real le restamos 127

Presición	# bits	signo	exponente	mantissa
Simple	32	1	8	23
Doble	64	1	11	52
Doble Extendido	80	1	16	63

- 8 registros (R0-R7) de 80 bits

Comparación

- FCOM == ST(0) vs ST(1)
- FCOM (ST(0) vs)
- No puedo usar macro directivas!

```
FNSTSW AX              ; mover banderas de FPU a AX
SAHF                   ; mover AH a las banderas de estado
```

- FCOMI ST(0), ST(i)
  - Activa las banderas de Zero, Parity y Carry directamente

Operaciones

- FABS (valor absoluto)
- FCHS (cambiar signo)
- FSQRT (raíz cuadrada)
- FLDZ (ST(0) == 0)
- FILD/ FIST (guardar/convertir a entero)

Saltos condicionales

- CMP == -

Basados en una bandera

Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

No signados

Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* >= *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* <= *rightOp*)
JNA	Jump if not above (same as JBE)

Signados

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* >= *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* <= *rightOp*)
JNG	Jump if not greater (same as JLE)

Macro directivas

Directive	Description
.BREAK	Generates code to terminate a .WHILE or .REPEAT block
.CONTINUE	Generates code to jump to the top of a .WHILE or .REPEAT block
.ELSE	Begins block of statements to execute when the .IF condition is false
.ELSEIF condition	Generates code that tests condition and executes statements that follow, until an .ENDIF directive or another .ELSEIF directive is found
.ENDIF	Terminates a block of statements following an .IF, .ELSE, or .ELSEIF directive
.ENDW	Terminates a block of statements following a .WHILE directive
.IF	Generates code that executes the block of statements if condition is true.
.REPEAT	Generates code that repeats execution of the block of statements until condition becomes true
.UNTIL	Generates code that repeats the block of statements between .REPEAT and .UNTIL until condition becomes true
.WHILE	Generates code that executes the block of statements between .WHILE and .ENDW as long as condition is true

Condiciones

Operator	Description
expr1 == *expr2*	Returns true when *expression1* is equal to *expr2*.
expr1 != *expr2*	Returns true when *expr1* is not equal to *expr2*.
expr1 > *expr2*	Returns true when *expr1* is greater than *expr2*.
expr1 >= *expr2*	Returns true when *expr1* is greater than or equal to *expr2*.
expr1 < *expr2*	Returns true when *expr1* is less than *expr2*.
expr1 <= *expr2*	Returns true when *expr1* is less than or equal to *expr2*.
! *expr*	Returns true when *expr* is false.
expr1 && *expr2*	Performs logical AND between *expr1* and *expr2*.
expr1 || *expr2*	Performs logical OR between *expr1* and *expr2*.
expr1 & *expr2*	Performs bitwise AND between *expr1* and *expr2*.
CARRY?	Returns true if the Carry flag is set.
OVERFLOW?	Returns true if the Overflow flag is set.
PARITY?	Returns true if the Parity flag is set.
SIGN?	Returns true if the Sign flag is set.
ZERO?	Returns true if the Zero flag is set.

Direccionamiento Indirecto

- Operando Indirecto: Es mejor para procedimientos!

```
.DATA
array SDWORD 2,3,5,6

.CODE
MOV ESI, OFFSET array

MOV EAX, [ESI]
ADD ESI, TYPE array
```

- Operando Indexado
  - Normal

```
.DATA
array SDWORD 2,3,5,6

.CODE
MOV ESI, 0                    ; indice = 0, 4, 8, ...

MOV EAX array[ESI]            ; opcion 1
ADD ESI, TYPE array

MOV EAX [array + ESI]         ; opcion 2
```

- Escalado

```
.DATA
array SDWORD 2,3,5,6

.CODE


MOV ESI, OFFSET array           ; dirección del arreglo
MOV EBX, 0                      ; indice = 0, 1, 2, 3, ...

MOV EAX [ESI+EBX*TYPE array]

INC EBX                         ; aumento indice
```

- Desplazamiento

```
.DATA
array SDWORD 2,3,5,6

.CODE

MOV ESI, OFFSET array           ; dirección del arreglo
MOV EBX, 0                      ; indice = 0, 4, 8, ...

MOV EAX [ESI+EBX]

ADD EBX, TYPE array
```

Instrucciones Irvine

Instrucción	Descripción	Params	Return
DumpRegs	Despliega registros y banderas	-	-
DumpMem	Despliega rango de memoria	**ESI:** Dirección inicial **ECX:** # elementos **EBX:** tamaño de elementos (1B, 2B)	-
WriteInt	Escribe entero signado de 32 bits	**EAX:** Número a escribir	-
WriteHex	Escribe entero no signado en hexa	**EAX:** Número	-
WriteHexB	Escribe entero no signado de *n* bits en hexa	**EAX:** Número **EBX:** # bytes a escribir (1 == AL, 2 == AX, 4 == EAX)	
ReadHex	Lee entero sin checar validez	-	**EAX:** Número leído
WriteString	Escribe cadena, terminada en 0	**EDX:** Offset de la cadena	-
ReadString	Lee cadena hasta el sig. enter	**EDX:** Offset donde se va a guardar **ECX:** max. # de caracteres (+1)	**EAX:** # de caracteres
Gotoxy	Coloca el cursor en la posición x,y	**DH:** x (renglón) **DL:** y (columna)	-
WriteChar	Escribe el caracter	**AL:** caracter a escribir	-

Notas

- **LENGTHOF** = # de elementos
- **SIZEOF** = # bytes
- **MOVZX** = zero extend
- **MOVSX** = sign extend

Esqueleto básico

```
TITLE Program Template          (OpArrArg.asm)
```

```
; Irvine Library procedures and functions
INCLUDE \masm32\Irvine\Irvine32.inc
INCLUDELIB \masm32\Irvine\Irvine32.lib
INCLUDELIB \masm32\Irvine\User32.lib
INCLUDELIB \masm32\Irvine\Kernel32.lib
; End Irvine

;SIMBOLOS
mcr=0dh
mlf=0ah
mnul=0h

.DATA
; PROC main


; PROC sycArrdw, variables locales


.CODE
main PROC

  EXIT
main ENDP

sycArrdw PROC

  RET
sycArrdw ENDP

END main
```

Presentaciones

- Leer de archivos → CA
- Shifts múltiples DWORDS → CBb (*MultiShf.asm*)