



ASSEMBLADOR

- **LLENGUATGE** DE MNEMÒNICS MOLT PROPER AL LLENGUATGE MÀQUINA
- **PROGRAMA** QUE TRADUEIX UN CODI FONT ESCRIT EN LLENGUATGE ASSEMBLADOR A LLENGUATGE MÀQUINA
 - Expandeix les macros
 - Codifica les etiquetes en adreces
 - Codifica les instruccions als “**números**” que en codi màquina representen les instruccions



FACILITATS QUE PROPORCIONA UN PROGRAMA ASSEMBLADOR

- DIRECTIVES PER DEFINIR DADES
 - `.ascii` “Els valors son:”
 - `.byte` 84, 104, 87, 23
 - `.word` 0, 10
- MACROS PER **REPETIR** CODI FÀCILMENT
 - `.macro` print_int (\$arg)
 - la \$4, 100
 - mov \$5, \$arg
 - jal 300
 - `.end_macro`
 - ...
 - print_int(\$7)



PROBLEMES DEL LLENGUATGE ASSEMBLADOR

- PROGRAMES LLARGS
 - Més difícils de mantenir
 - Més probabilitat d'error
- PROGRAMES LLIGATS A UNA MÀQUINA
 - Assemblador diferent per ix86, VAX, MIPS, ...
- PROGRAMES DÍFICILS D'ENTENDRE I SEGUIR
- NO IMPLEMENTA ESTRUCTURES DE DADES COMPLEXES



ERROR COMÚ ASSOCIAT A L'ASSEMBLADOR

- LA PROGRAMACIÓ AMB ASSEMBLADOR GENERA UN CODÍ MES COMPACTE I/O MÉS RÀPID
 - Això era cert fa temps
 - Actualment els compiladors són molt bons
 - Optimitzen per temps i/o per mida
 - Les CPUs actuals són molt difícils de controlar
 - Segmentades / Superscalar
 - Cachés
 - Memòria Virtual / Mode Sistema i Mode Usuari
 - Actualment pot tenir sentit de implementar alguna rutina molt compromesa en assemblador però no programes enters



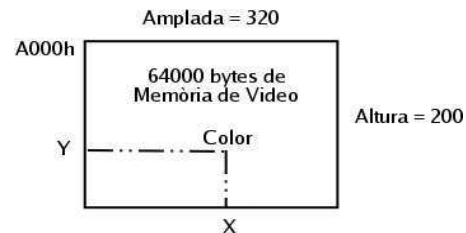
EXEMPLE D'OPTIMITZACIÓ EN L'ASSEMBLADOR

- EXEMPLE (*Turbo Pascal* amb *Assemblador 8086* incorporat)

```
Procedure posarPixel ( X, Y : Word; Color : Byte );  
Assembler;
```

Asm

```
MOV AX, Y           { AX = Y           }  
XCHG AH, AL        { AX = Y * 256     }  
MOV BX, AX         { BX = Y * 256     }  
SHR AX, 2          { AX = Y * 64      }  
ADD BX, AX         { BX = Y * 320     }  
ADD BX, X          { BX = Y * 320 + X }  
MOV AX, 0A000h     { AX = 0A000h     }  
MOV ES, AX         { ES = 0A000h     }  
MOV AL, Color      { AL = Color      }  
MOV ES:[BX], AL    { ES:[BX] = Color }  
End; { Assembler }
```



- Millor opció que multiplicar per 320 utilitzant la instrucció **MUL** que gastava molts cicles de màquina (12) en un Intel 8086 / 80286

```
MOV AX, Y           { AX = Y           }  
MOV BX, 320        { BX = 320         }  
MUL BX             { AX = Y * 320     }  
ADD AX, X          { AX = Y * 320 + X }  
MOV BX, AX         { BX = Y * 320 + X }
```



SPIM (I)

- ÉS UN SIMULADOR DE MIPS32 (R2000/R3000)
- CARREGA PROGRAMES EN ASSEMBLADOR DE MIPS32 (NO BINARIS)
- PERMET:
 - EXECUTAR PROGRAMES MIPS32
 - INSPECCIONAR MEMÒRIA
 - INSPECCIONAR / MODIFICAR ELS REGISTRES
 - INSPECCIONAR / MODIFICAR REGISTRES DE LA UNITAT DE COMA FLOTANT



SPIM (II): OBTENCIÓ

- Funciona sota plataformes Windows i X11
- FTP: [ftp.cs.wisc.edu](ftp://ftp.cs.wisc.edu) (*anonymous login*)

/pub/spim/

pcspim_src.zip	(Codi Font SPIM Windows)
spimdos.exe	(MS-DOS)
spim6.3.tar	(Unix/Linux en Codi Font)
spim6.3.tar.gz	(Unix/Linux en Codi Font comprimit)
pcspim.exe	(Windows)
spimwin.ps	(Manual d'Spim per Windows)



SPIM (III): CARACTERÍSTIQUES

- Utilitza l'*Endian* de la plataforma on s'executa
 - MIPS és ***Bi-Endian***
 - **SGI** utilitza un **MIPS** en *Big-Endian*
 - Un altra plataforma pot utilitzar un MIPS en *Little-Endian*
- Permet simular crides al sistema
- Execució pas a pas a l'estil d'un *Debugger*
- Interfície gràfica



SPIM (IV): SIMULACIÓ

- Obrir un fitxer font
 - Càrrega del programa a memòria
- Simulació (*Debugger*)
 - **Go**: Executar el programa carregat
 - **Break**: Parar l'execució del programa
 - **Single/Multiple Step**: Fer passos d'execució
 - **Breakpoint**: Parar el programa en un punt
 - **Reload**: Recàrrega i inicialització del programa



DIRECTIVES(I): ASSEMBLADOR SPIM

- TIPUS DE DADES
 - .word, .half** Enters de 32/16 Bits
 - .byte** Enters de 8 Bits
 - .ascii, .asciiz** Cadenes de caràcters
 - Expressades entre cometes (“”)
 - La segona és a l'estil C (acaba en un byte 0)
 - Permet caràcters especials a l'estil C (“\n”, “\t”, “\””)
 - .double, .float** Valors en punt flotant



DIRECTIVES(II): ASSEMBLADOR SPIM

- ALTRES DIRECTIVES

- .text** Indica que a partir d'aquest punt, tot el que es defineix es posarà en un segment de codi
- .data** Indica que a partir d'aquest punt, tot el que es defineix es posarà en un segment de dades
- .globl *symbol*** Indica que ***symbol*** es global i que es pot veure des d'altres fitxers



PROGRAMA D'EXEMPLE

```
# Programa d'exemple: Suma de dos valors

.text                                     # Secció de Codi
.globl main                               # Símbol Main: Inici => SPIM

main: la $t0, valor                       # Carregar @valor a $t0
      lw $t1, 0($t0)                      # Carregar [valor+0] a $t1
      lw $t2, 4($t0)                      # Carregar [valor+4] a $t2
      add $t3, $t1, $t2                   # Sumar i Resultat a $t3
      sw $t3, 8($t0)                      # Guardar Resultat a [valor+8]

.data                                     # Secció de Dades
valor:.word 10, 20, 0                    # Dades per Suma i el Resultat
```