Esqueleto básico de un programa (“hola mundo”)

Sintaxis NASM	*Sintaxis GAS*

`section .data`
`  mensaje db "Hola con NASM"`
`  mensaje_SIZE equ $ - mensaje`
 	
`section .text`
`global _start`
 	
`_start:`
`  mov ecx, mensaje`
`  mov edx, mensaje_SIZE`
`  mov eax, 4`
`  mov ebx, 1`
`  int 80h`
 	
`  mov ebx,0`
`  mov eax,1`
`  int 0x80`
	`.data`
	`mensaje: .ascii "hola con GAS\n"`
	`mensaje_SIZE = . - mensaje`
	
	`.text`
	`.globl _start`
	
	`_start:`
	`  movl $mensaje, %ecx`
	`  movl $mensaje_SIZE, %edx`
	`  movl $4, %eax`
	`  movl $1, %ebx`
	`  int $0x80`
	
	`  movl $0, %ebx`
	`  movl $1, %eax`
	`  int $0x80`

Definición de cadenas y enteros

Sintaxis NASM	Sintaxis GAS

`section .data`
`  cad db “un texto”`
`  cadSIZE equ $ - cad`
 	
`  corto dw 0`
`  entero dd 1`
`  letra db ‘A’`
	`.data`
	`  cad: .ascii “un texto”`
	`  cadSIZE = . – cad`
	
	`  corto: .hword 0`
	`  entero: .int 1`
	`  letra: .byte ‘A’`

Definición de tipos de datos en la sintaxis NASM

http://nasm.sourceforge.net/doc/nasmdoc3.html

Se usa DB, DW, DD, DQ, DT y DO para declarar datos con un valor inicial:

```
db    0x55                ; el número 0x55 representado con 8bits
db    0x55,0x56,0x57      ; sucesión de tres bytes
db    'a',0x55            ; sucesión de dos bytes (uno de ellos representado como una letra)
db    'hello',13,10,0     ; una cadena de caracteres es una sucesión de bytes (letras)
dw    0x1234              ; 0x34 0x12
dw    'a'                 ; 0x61 0x00 (la letra 'a' es el ASCII 61h)
dw    'ab'                ; 0x61 0x62 (letra 'a' seguida de la letra 'b')
dw    'abc'               ; 0x61 0x62 0x63 0x00 (una cadena)
dd    0x12345678          ; 0x78 0x56 0x34 0x12
dd    1.234567e20         ; número en coma flotante
dq    0x123456789abcdef0  ; número representado con 8 bytes
dq    1.234567e20         ; número real de doble precisión
dt    1.234567e20         ; número en coma flotante de precisión extendida
```

Para declarar datos no inicializados al cargar el programa, usaremos RESB, RESW, RESD, RESQ, REST y RESO en la sección BSS:

```
fichero:  resb    256       ;REServa 256 Bytes
caracter: resb    1         ;REServa 1 Byte (8 bits)
palabra:  resw    1         ;REServa 1 Word (palabra, 16 bits)
numero:   resd    1         ;REServa 1 DoubleWord (doble palabra, 32bits)
num_real:         resq    1 ;REServa 1 float de doble precision (64 bits)
precision:        rest    1 ;REServa 1 float de precision extendida (128 bits)
```

Definición de tipos de datos en la sintaxis GAS

La lista completa de directivas es: .byte .short .long .quad .single .double .comm .ascii .asciz

```
mensaje:       .ascii   "hola con GAS\n"
mensaje_SIZE = . - mensaje
buffer:        .ascii   "01234567890123456789012345678 9"
enteroLargo:   .int     10
enteroCorto:   . hword  2
unByte:        .byte    0x0A
nombre_fich:   .asciz      "prueba.txt"
manejador:     .int        0
```

Acceso a variables

Sintaxis NASM	*Sintaxis GAS*
`mov ecx, [var1]` `mov [var2], eax`	`movl  (var1), %ecx` `movl  %eax, (var2)`

`mov esi, 0`
`mov al, [cadena+esi]`
`mov [letra], al`
	`movl $0, %esi`
	`movb cadena(%esi), %al`
	`movb %al, (letra)`

Direccionamientos a base, con índice y desplazamiento

En NASM: `[ base + indice*escala + desplazamiento ]`

en GAS: `desplazamiento ( base , indice , escala )`

Sintaxis NASM	*Sintaxis GAS*
`mov edx, array[eax*4]`	`movl array (, %eax, 4), %edx`
`mov eax , [ebx]` `mov eax,[ebx+3]`	`movl (%ebx) , %eax` `movl 3(%ebx) , %eax`

Órdenes para compilar los programas

En NASM:

```
nasm -f elf  prog.asm
ld  -o  prog  prog.o
```

en GAS:

```
as -o  prog.o  prog.s
ld -o  prog    prog.o
```

Llamar a funciones insertando los argumentos en la pila

llamada desde el “main”	función llamada (suma)

`. . .`
 	
`push   dword [y]`
`push   dword [x]`
`call   sumar`
`add    esp, 8`
 	
`mov    [z], eax`
 	
`. . .`
	`;  int sumar(int x, int y)`
	`sumar:`
	`push   ebp`
	`mov    ebp,esp`
	
	`push   ebx`
	
	`mov    eax,[ebp+8]`
	`mov    ebx,[ebp+12]`
	`add    eax,ebx`
	
	`pop    ebx`
	
	`pop    ebp`
	`ret`