23 de Noviembre del 2016

Programación en Lenguaje Ensamblador

Organización de Computadoras

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación

Diaz Figueroa, Rodrigo Martin
Figliuolo, Nestor Orlando

Definiciones y especificación de requerimientos

La finalidad del sistema es replicar parte de la funcionalidad del programa hexdump (*http://linuxcommand.org/man_pages/hexdump1.html*) mostrando por pantalla el contenido de un archivo en representación hexadecimal y como caracteres ascii.

Este sistema esta apuntado a personas que necesitan una herramienta para visualizar este tipo de datos, estos usuarios para poder usar el sistema deben tener conocimientos de como ejecutar un programa por consola y como ingresar los parámetros que sean adecuados.

El sistema cumple con los siguientes requerimientos:

El programa debe tomar el contenido del archivo de entrada y mostrarlo por pantalla, organizado de la siguiente forma:

`[Dirección base] [Contenido hexadecimal] [Contenido ASCII]`

La salida debe organizarse en filas de a 16 bytes. La primera columna muestra la dirección base de los siguientes 16 bytes, expresada en hexadecimal. Luego siguen 16 columnas que muestran el valor de los siguientes 16 bytes del archivo a partir de la dirección base, expresados en hexadecimal. La última columna (delimitada por caracteres '|') de cada fila muestra el valor de los mismos 16 bytes, pero expresados en formato ASCII, mostrando sólo los caracteres imprimibles, e indicando la presencia de caracteres no imprimibles con '.').

El sistema es un desarrollo origina y fue desarrollado en parte con el editor de texto GNU nano (*https://www.nano-editor.org/*), el ensamblador YASM (http://yasm.tortall.net/) y el enlazador GNU Linker (http://sourceware.org/binutils/docs-2.21/ld/)

Para poder ejecutar el programa primero se debe descomprimir el archivo comprimido que contiene el código fuente del sistema, cuyo archivo principal es "volcar.asm". Alternativamente se puede clonar el repositorio ubicado en el siguiente enlace: *https://github.com/Makaan/ProyectoASM*.

El programa debe ser compilado sobre un sistema operativo GNU/Linux con arquitectura x86 para garantizar su funcionamiento. Para realizar el ensamblado se debe ejecutar el siguiente comando sobre la carpeta donde se descomprimió el programa:

"yasm -f elf volcar.asm"

"ld -o volcar volcar.o"

Una vez finalizado el ensamblador del programa, puede ser ejecutado con la siguiente sintaxis:

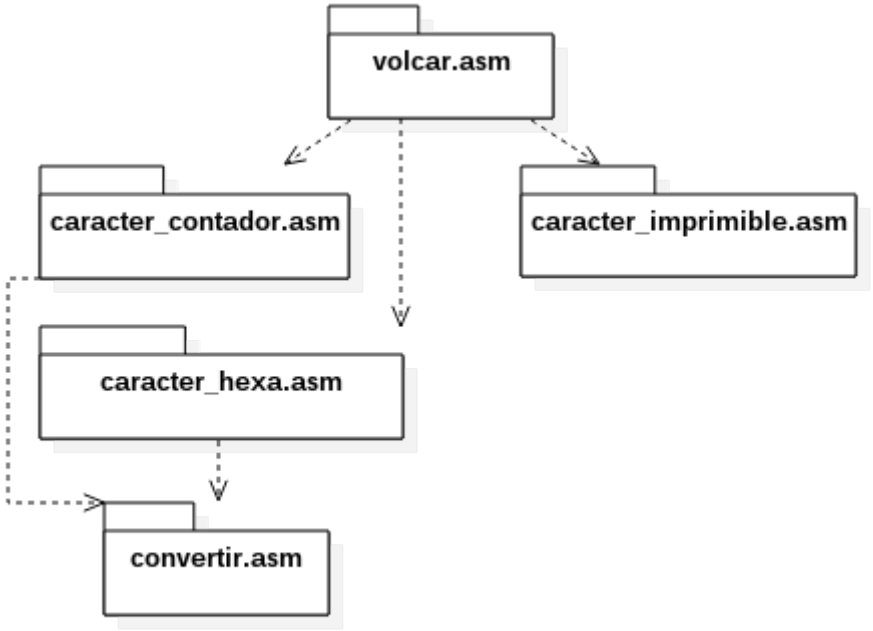
“./volcar [-h] <archivo>”

Siendo -h un parámetro opcional que solo mostrara por pantalla una ayuda de como usar el sistema.

Arquitectura del sistema

El sistema esta conformado por tres módulos separados en base a su funcionalidad:

- volcar.asm
- caracter_contador.asm
- caracter_hexa.asm
- caracter_imprimible.asm
- convertir.asm



Cada modulo tiene la siguiente descripción:

- volcar.asm: aquí esta empaquetado el programa principal, y por ende en el se encuentra toda la implementación concerniente al manejo de los datos de entrada/salida y a la forma como se van a representar los datos en pantalla. Este modulo depende de los otros cuatro para funcionar, así también como de las funciones que provee el sistema operativo.
- caracter_contador.asm: escribe el contador de bytes leídos en forma hexadecimal en las primeros 6 caracteres del renglón.
- carácter_hexa.asm: recibe un carácter y retorna su representación en hexadecimal. Ej: ‘A’=41 en hexadecimal.
- caracter_imprimible.asm: recibe un carácter y retorna, ese mismo carácter si es imprimible o un ‘.’ (punto) en caso contrario.

- convertir.asm: convierte un digito de 4 bits hexadecimal en su representacion como carácter ascii.

Todo el sistema esta programado usando el lenguaje de programación Ensablador y ensamblado bajo el sistema operativo GNU/Linux.

Diseño del modelo de datos

El sistema esta diagramado de la siguiente forma:

volcar.asm usa a los otros módulos como operaciones auxiliares para hacer el procesamiento de caracteres a sus representaciones en hexadecimal o como caracteres imprimirles.

El sistema usa como dato de entrada el archivo pasado como parámetro, internamente varios buffers de memoria donde se almacenaran el contador, uno donde se representa un renglon de la salida esperada y dos donde se representan las posiciones donde se insertaran los caracteres y su representación en hexadecimal.

Al invocar el programa se espera que el usuario pase como parámetro la ruta al archivo que se quiere representar, si se invoca con el parámetro -h se mostrara una ayuda de como usar el sistema. Este archivo sera leído y se almacenara su contenido hasta un tamaño máximo de 1 megabyte de información, descartando el resto.

Luego se aplicara el siguiente algoritmo:

- Mientras queden caracteres para leer:
  - Mientras la cantidad de caracteres no sea múltiplo de 16
    - Leo un carácter, lo transformo en carácter imprimible y lo inserto en la linea en la posición correspondiente
    - Ese mismo carácter lo convierto en hexadecimal y lo inserto en la linea en la posición correspondiente
    - Incremento la posición donde insertar el caracteres
    - Incremento la posición donde insertar el hexadecimal en 3
    - Incremento el contador de caracteres
  - Imprimo el renglón por pantalla

    - Restablezco los valores de las posiciones donde insertar a su valor original.
- Relleno la ultima linea (si existe) con espacios en blanco en las posiciones donde debería haber caracteres para imprimir.
- Imprimo el ultimo renglón
- Imprimo el contador con la cantidad de caracteres leídos.

Especificación técnica:

volcar.asm:

Se miran los argumentos pasados como parámetro:

- Si no hay argumentos se sale con error 1
- Si el primer argumento es “-h” se imprime la ayuda

Se abre el archivo pasado como parámetro, si ocurre un error a la hora de abrir el archivo (es decir el descriptor es -1) se sale con error 2.

Con el descriptor del archivo se lee hasta un máximo de 1 MB de datos del archivo, ignorando el resto.

Mientras haya caracteres para leer:

Leo uno y lo transformo en su representación como carácter imprimible y hexadecimal y lo inserto en la posicion que corresponda en la linea es decir:

```
      | posición hexa                                        | posición char
      v                                                     v
cont  hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh  |cccccccccccccccc|
```

Se hace eso hasta que se copian 16 caracteres y luego se imprime la linea.

Eso tantas veces como lineas haya para imprimir, con excepción de la ultima en la cual se reemplazan los espacios que quedaron libres para copiar caracteres con espacios en blanco.

Luego se imprime el contador final y se cierra el archivo, finalizando la ejecución exitosamente del programa.

Convertir.asm:

Recibe en el registro DL un numero mayor o igual a 0 y menor o igual a 15.

Retorna el mismo numero en hexadecimal ascii (con letras mayúsculas)

Compara al numero en el registro DL con 9.

Si es mayor, se le suma 55, para transformarlo en una letra mayuscula.

Si es menor, se le suma 48, para transformarlo en un numero ascii.

`caracter_hexa.asm:`

Recibe en el registro CL el carácter a transformar a hexadecimal ascii.

Retorna en el registro CL el primer carácter hexadecimal y en CH el segundo carácter hexadecimal. Notar que se almacenan en orden inverso.

Para convertir el carácter de 8 bits a hexadecimal se necesitan 2 caracteres de 8 bits.

Primero se aíslan los últimos 4 bits del carácter haciendo un AND con 00001111 en binario y se almacena el resultado en el registro DL para poder utilizar la rutina convertir.

Luego de la rutina convertir, en el registro DL queda almacenado el segundo carácter hexadecimal que es copiado a CH.

Luego se aíslan los primeros 4 bits del carácter original haciendo un shift right de 4 y se almacena el resultado en el registro DL para poder utilizar nuevamete la rutina convertir. El resultado de convertir que se encuentra en DL y es el primer carácter hexadecimal se almacena en CL.

`caracter_imprimible:`

Recibe en el registro CL un carácter.
Retorna en el registro CL el mismo carácter si este es imprimible, sino retorna el carácter “.”.

Compara al carácter en CL con 31.

Si es mayor:

- Compara a CL con 127
- Si es igual: Asigna a CL 46 (que es el carácter “.”)
- Si no es igual termina

Si es menor o igual:

- Asigna a CL 46 (que es el carácter “.”)

`caracter_contador.asm:`

Recibe en el registro EAX un número (contador) menor o igual a 2^{20} (1MB)

Recibe en el registro EBX la direccion de memoria de un buffer

Retorna en las 6 primeras posiciones del buffer el numero EAX en representación hexadecimal ascii.

Primero se suma 5 al registro EBX para empezar desde la posición menos significativa.

Se asigna un 16 al registro ECX para utilizar como divisor.

Repetir hasta que el cociente (el contenido de EAX) sea 0:

Asignar un 0 a EDX, para usar la instrucción IDIV.

Dividir por 16 .

El resto queda almacenado en EDX.

Llamar a rutina convertir.

Copiar en el buffer el contenido de DL (que es el resto de la division en hexadecimal ascii).

Decrementar EBX en 1 (Para obtener la direccion el siguiente digito menos significativo).

Códigos de retorno del programa:

EBX	***Significado***
0	Terminación Normal.
1	Terminacion Anormal
2	Terminacion Anormal por error en el archivo de entrada.

Conclusiones

Asumimos que si no se ingresan parámetros a la hora de ejecutar el programa se sale con error 1.
Tambien que si se quiere usar un archivo vacio es correcto y sale sin error.
Este proyecto fue mas completo y tal vez mas difícil que el del año pasado, pero mucho mejor explicado y mas contundente.

Codigo Fuente:

volcar.asm

```
1. %include "convertir.asm"
2. %include "caracter_hexa.asm"
3. %include "caracter_imprimible.asm"
4. %include "caracter_contador.asm"
5.
6. %define hex_offset 8
7. %define char_offset 58
8. %define linea_max 16
9.
10. %define SYS_EXIT 1
11.
12. %define SYS_READ 3
13.
14. %define SYS_WRITE 4
15. %define STDOUT 1
16.
17. %define SYS_OPEN 5
18. %define RDONLY 0
19.
20. %define SYS_CLOSE 6
21.
22. section .data
23.
24.   ;Ayuda que se imprimira por pantalla
25.   ayuda db "El programa hace un volcado del contendido del archivo pasado como
       parametro y lo expresa en hexadecimal y en ascii junto con su direccion de memoria
       relativa en porciones de a 16 bytes. El programa saldrá con error (2) si no se
       encuentra el archivo pasado como parametro."
26.   ayudal equ $ - ayuda
27.   ;Prototipo de la linea que se imprimira por pantalla
28.   linea db "000000  hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh
       |................|"
29.   lineal equ $ - linea
30.   buffer0s db "000000"            ;Buffer para escribir la ultima vez la cantidad
       de elementos leidos
31.   buffer0sl equ $ - buffer0s
32.
33.   char_max dd 0                   ;cantidad de caracteres leidos del archivo
34.   contador dd 0                   ;contador de lineas
35.   hex_pos dd hex_offset                 ;offset a la posicion de la linea para
       insertar la representacion hexadecim
```

```
36.   char_pos dd char_offset          ;offset a la posicion de la linea donde insertar
       el char
37.
38.
39.   salto db 10                     ;"\n"
40.   espacio db 0x20                 ;" "
41.   barra db 7ch                    ;"|"
42.   resto dd 0                      ;Diferencia entre el contador y la cantidad de
       lineas
43.
44.
45. section .bss
46.
47.   buffer: resb 1048576       ;Buffer para leer de archivo
48.
49. section .text
50.
51. global _start
52.
53. imprimir_salto:
54.
55. ; imprimo un salto de linea por pantalla
56. mov EAX,SYS_WRITE
57. mov EBX,STDOUT
58. mov ECX,salto
59. mov EDX,1
60. int 80h
61. ret
62.
63. _start:
64.
65. ;Miro la cantidad de parametros
66.
67. pop EAX                       ;saco la cantidad de parametros
68. cmp EAX, 2                    ;argc == 2?
69. jne salir_error               ;si no es 2 salgo con error
70.
71.
72. ;Miro los argumentos
73.
74. pop EAX                       ;Descarto el nombre del programa
75. pop EAX                       ;Guardo el puntero al segundo parametro
76. mov EBX,EAX                   ;Hago una copia del puntero
77.
78.
```

```
79. ;Compruebo que el segundo parametro sea "-h"
80. cmp BYTE [EAX], 2Dh          ;Comparo el primer caracter con "-"
81. jne abrir_archivo            ;Si no es "-" procedo a abrir el archivo
82. inc EAX                      ;incremento el puntero
83. cmp BYTE [EAX], 68h          ;Comparo el segundo caracter con "h"
84. jne salir_error              ;Si no es "h", procedo a abrir el archivo
85. inc EAX                      ;Incremento el puntero
86. cmp BYTE [EAX], 0h           ;Comparo con el caracter nulo
87. jne salir_error              ;Si no es caracter nulo salgo con error
88. jmp imprimir_ayuda           ;Si el argv es "-h" imprimo la ayuda
89.
90. abrir_archivo:
91.
92. ;Abro el archivo que tiene el texto a imprimir, la ruta al archivo se encuentra en
      EBX
93.  mov EAX,SYS_OPEN           ;Pongo el numero de llamada al sistema para abrir el
      archivo
94.  mov ECX,0                  ;No pongo ningun flag para el archivo a abrir
95.  mov EDX,RDONLY             ;Voy a abrir el archivo en solo lectura
96.  int 80h
97.
98. add EAX,2
99. cmp EAX, 0                  ;Si hubo error el descriptor del archivo sera -1 y salgo
      con error 2
100. je salir_error_archivo
101. sub EAX,2
102.
103. push EAX             ;Guardo el descriptor del archivo para cerrarlo despues
104.
105. mov EBX,EAX                ;Pongo el descriptor del archivo en EBX
106. mov EAX,SYS_READ           ;LLamada al sistema para leer
107. mov ECX,buffer             ;Buffer donde va a quedar el archivo
108. mov EDX,1048576                  ;Tamaño maximo del buffer
109. int 80h
110.
111. cmp EAX, 0                 ;Si el tamaño del archivo es 0
112. je  salir_sin_error        ;Salgo sin error
113.
114. mov [char_max],EAX         ;Guardo la cantidad de caracteres leido
115.
116. leer_linea:
117.
118. ;Cargo el caracter del archivo
119. mov EBX,buffer             ;Muevo la direccion inicial del buffer
120. add EBX,[contador]         ;Le sumo el contador donde tengo que char leer
```

```
121. mov CL,[EBX]               ;Copio el caracter almacenado en la posicion
       buffer+contador
122.
123. push ECX             ;Guardo el caracter
124.
125. ;Escribo en la posicion correspondiente de la linea el caracter que leo del buffer
126. mov EAX,linea              ;Muevo la direccion inicial de la linea
127. add EAX,[char_pos]         ;Le summo el offset
128. call caracter_imprimible   ;Convierte el caracter leido en un caracter imprimible
129. mov [EAX],CL               ;Copio el caracter que lei en linea+char_pos
130.
131. pop ECX                    ;Saco el caracter que lei de la pila
132.
133. mov EAX,linea              ;Muevo la direccion inicial de la linea
134. add EAX,[hex_pos]          ;Le sumo el offset
135. call caracter_hexa         ;Convierto el caracter en hexadecimal
136. mov [EAX],CX               ;Lo escribo en la linea
137.
138.
139. inc DWORD [char_pos]       ;Incremento la posicion donde escribir caracteres en la
       linea
140. add [hex_pos],DWORD 3      ;Incremento la posicion donde escribir el hexa en la
       linea
141.
142. ;Incremento el contador, si es igual a la cantidad de caracteres que tiene el
       archivo dejo de leer
143. inc DWORD [contador]       ;incremento el contador
144. mov EAX,[contador]         ;lo muevo a EAX para comparar
145. cmp [char_max],EAX         ;Si es igual que la cantidad de caracteres leidos
146. je fin_archivo             ;salto a fin_archivo
147.
148. ;Veo si el contador es multiplo de 16, para cambiar de linea
149. mov EAX,[contador]         ;Muevo el contador al registro EAX como dividendo
150. mov EBX,linea_max          ;Muevo un 16 como divisor
151. mov EDX,0                  ;Reseteo EDX con 0
152. idiv EBX             ;Uso division entera
153. cmp EDX,0                  ;Si el resto es 0 salto a resetear la linea para leer
       una nueva
154. je reset
155.
156. jmp leer_linea             ;Vuelvo a leer un caracter
157.
158. ;Reestablezco la linea para seguir leyendo caracteres
159. reset:
160.
```

```
161. ;Imprimo la linea por pantalla
162. mov EAX,SYS_WRITE
163. mov EBX,STDOUT
164. mov ECX,linea
165. mov EDX,lineal
166. int 80h
167.
168. ;Reseteo las posiciones donde voy a escribir los caracteres
169. mov [char_pos],DWORD char_offset         ;char_pos=57
170. mov [hex_pos],DWORD hex_offset                ;hex_pos=8
171.
172.
173. ;Escribo en la linea el contador, exceptuando la primera que ya esta en 000000
174. mov EAX,[contador]                       ;Cargo el contador para imprimir la
       cantidad actual en la linea
175. mov EBX,linea
176. call caracter_contador                          ;Llamo a la funcion que me escribe
       el contador en la linea
177.
178. call imprimir_salto
179.
180. jmp leer_linea                           ;Vuelvo a imprimir una linea
181.
182. fin_archivo:
183.
184. ;Si no hay nada mas para imprimir salgo a imprimir el contador
185. mov EAX,[contador]          ;Muevo el contador
186. mov EDX,0                   ;Reseteo EDX en 0
187. mov EBX,linea_max           ;Muevo a EBX 16
188. idiv EBX             ;Divido contador/16
189. cmp EDX,0                  ;Comparo el resto con 0
190. je imprimir_contador       ;Si es 0 no hay nada mas para copiar
191.
192. ;Guardo el resto
193. sub EBX,EDX                ;16-resto
194. mov [resto], EBX           ;Guardo el resto para saber cuantos caracteres tengo que
       reemplazar
195.
196. ;Agrego una sola vez una barra vertical al final de los caracteres
197. mov EAX,linea              ;"|"
198. add EAX,[char_pos]
199. mov BL,BYTE [barra]
200. mov [EAX], BL
201. inc BYTE [char_pos]
202.
```

```
203. reemplazar:
204. ;Reemplazo todos los caracteres restantes con espacios
205.
206.   ;Reemplazo el caracter en la linea por un espacio
207.   mov EAX,linea
208.   add EAX,[char_pos]        ;Quiero agregarlo en linea+char_pos que es donde deberia
      seguir escribiendo
209.   mov BL,[espacio]
210.   mov [EAX],BL
211.   inc BYTE [char_pos]       ;Incremento char_pos para no sobreescribir lo que acabo
      de escribir
212.
213.   ;Reemplazo los dos hexadecimales por dos espacios
214.   mov EAX,linea
215.   add EAX,[hex_pos]
216.   mov BL,[espacio]
217.   mov BH,[espacio]
218.   mov [EAX],BX
219.   add BYTE [hex_pos],3
220.
221.   dec BYTE [resto]
222.   cmp [resto], WORD 0
223.   je imprimir_faltante
224.
225.
226.   jmp reemplazar
227.
228. imprimir_faltante:
229. ;Imprimo la linea que falta
230. mov EAX,SYS_WRITE
231. mov EBX,STDOUT
232. mov ECX,linea
233. mov EDX,lineal
234. int 80h
235.
236. call imprimir_salto
237.
238. ;Imprimo el contador con el valor final de caracteres leidos
239. imprimir_contador:
240.
241. mov EAX,[contador]
242. mov EBX,buffer0s            ;Buffer especial que contiene "000000"
243. call caracter_contador
244.
245. mov EAX,SYS_WRITE
```

```
246. mov EBX,STDOUT
247. mov ECX,buffer0s
248. mov EDX,buffer0sl
249. int 80h
250.
251. call imprimir_salto
252.
253. ;Cierro el archivo
254. pop EBX
255. mov EAX,SYS_CLOSE
256. int 80h
257.
258. salir_sin_error:
259.
260. ; Salgo sin error
261. mov EAX,SYS_EXIT
262. mov EBX,0
263. int 80h
264.
265. imprimir_ayuda:
266.
267. ;Imprimo el texto de ayuda
268. mov EAX,SYS_WRITE
269. mov EBX,STDOUT
270. mov ECX,ayuda
271. mov EDX,ayudal
272. int 80h
273.
274. call imprimir_salto         ;Imprimo un salto de linea
275.
276. salir_error:
277.
278. ;Salgo con error 1
279. mov EAX,SYS_EXIT
280. mov EBX,1
281. int 80h
282.
283. ;Si hubo problema cuando se quiso abrir el archivo
284. salir_error_archivo:
285.
286. ;Salgo con error 2
287. mov EAX,SYS_EXIT
288. mov EBX,2
289. int 80ha
```

caracter_contador.asm

```
1. section .text
2.
3. ; Funcion caracter_contador: convierte un numero (menor a 2^20) en EAX a ascii hexa y
lo guarda en
4. ; los primeros 5 lugares del buffer de EBX
5. ; PARAMETROS:
6. ;   EAX - Numero
7. ;   EBX - Buffer
8. ; RETORNO:
9. ;   EBX - Con sus primeros 6 lugares representando al numero en hexa ascii
10.
11. caracter_contador:
12.
13.   add EBX, 5          ;Sumo 4 a buffer, empiezo desde la posicion menos
significativa
14.   mov ECX,16          ;ECX=16 divisor
15.
16.
17. bucle_contador:
18.
19.   mov EDX,0           ;Preparo EDX=0 para usar idiv
20.   idiv ECX            ;Divido por 16, EDX=resto, EAX=cociente
21.
22.   call convertir      ;Convierto 0<resto<16 en ascii hexa
23.   mov [EBX], DL       ;Muevo ascii hexa a buffer
24.   dec EBX       ;Decremento buffer. Muevo una posicion a la izquierda
25.
26.   cmp EAX,0           ;Comparo Cociente con 0.
27.   jne bucle_contador  ;Si (Cociente!=0): sigo dividiendo
28.
29.   ret                 ;Si (Cociente=0): Fin
```

caracter_hexa.asm

```
1. section .text
2.
3. ; Funcion caracter_hexa: Convierte el caracter en CL a hexa ascii. Los caracteres hexa
4. ; se almacenan en CH y CL EN ORDEN INVERTIDO.
5. ; Para que los caracteres se impriman en el orden correcto se debe usar CX.
6. ; PARAMETROS:
7. ;  CL - Caracter
8. ; RETORNO:
9. ;  CL - Primer hexa en ascii.
10. ;  CH - Segundo hexa en ascii.
11.
```

```
12. caracter_hexa:
13.
14.   mov DL,CL           ;Hago copia de caracter
15.   and DL,00001111b    ;Obtengo ultimos 4 bits
16.
17.   call convertir      ;Convierto 4 bits a hexa ascii (0..9A..F)
18.
19.   mov CH,DL           ;Copio segundo hexa en CH. (ORDEN INVERTIDO)
20.   mov DL,CL           ;Hago copia de caracter
21.   shr DL,4            ;Obtengo primeros 4 bits
22.
23.   call convertir      ;Convierto 4 bits a hexa ascii (0..9A..F)
24.
25.   mov CL,DL           ;Copio primer hexa en CL. (ORDEN INVERTIDO)
26.   ret                 ;Fin
```

caracter_imprimible.asm

```
1. section .text
2.
3. ; Funcion caracter_imprimible: Dado un caracter en CL, si no es imprimible lo
convierte a "."
4. ; PARAMETROS:
5. ;   CL - Caracter
6. ; RETORNO:
7. ;   CL - Caracter original si este es imprimible, "." si no era imprimible
8.
9. caracter_imprimible:
10.
11.   cmp CL, 31          ;Comparo Caracter con 31
12.   jg imprimible       ;Si (Caracter>31): Voy a imprimible
13.
14. no_imprimible:        ;Sino (Caracter<=31): Caracter no es imprimble
15.
16.   mov CL, 46          ;Entonces Caracter="."
17.   jmp fin_car_imprimible;Ir a fin
18.
19. imprimible:
20.
21.   cmp CL, 127         ;Comparo Caracter con 127 ( 127 en ascii es DEL )
22.   je no_imprimible    ;Si (Caracter=127): Ir a no_imprimible
23.
24. fin_car_imprimible:
25.
26.   ret                 ;Fin
```

convertir.asm

```
1. section .text
2.
3. ; Funcion convertir: Convierte el caracter en DL a hexa en ascii (0..9A..F)
4. ; PARAMETROS:
5. ;   DL - Caracter
6. ; RETORNO:
7. ;   DL - Caracter en hexa ascii (0..9A..F)
8.
9. convertir:
10.
11.   cmp DL,9            ;Comparo Caracter con 9
12.   jg esletra          ;Si Caracter>9: Ir a esletra
13.
14. esnumero:       ;(Caracter<=9)
15.
16.   add DL,48           ;Caracter es numero. Sumo 48 para convertir a ascii numero
(0..9)
17.   jmp fin_convertir   ;Ir a fin
18.
19. esletra:        ;(Caracter>9)
20.
21.   add DL,55           ;Caracter es letra. Sumo 55 para convertir a ascii letra
(A..F)
22.
23. fin_convertir:
24.
25.   ret                ;Fin
```