Segunda tarea programada, Benchmark (Junio 2011)

Enmanuel O. Ramírez, Yendry R. Rodríguez, Alonso V. Brenes

Abstract— **El Benchmark es una técnica utilizada para medir el rendimiento de un computador o un componente del mismo.**
Estos programas no solo pueden ayudarnos en la comparación de diferentes sistemas sino que además son capaces de evaluar las prestaciones de un equipo con diferentes configuraciones de Software y Hardware. Los benchmarks son pruebas para medir el rendimiento y poder verificar que el hardware funciona de forma óptima o para comparar distintas configuraciones.
Se puede decir que existen dos grandes tipos de benchmarks, los sintéticos y los que se basan en la evaluación de aplicaciones del mundo real.
Este proyecto consiste en la creación de un Benchmark que permite contar la cantidad de operaciones de un CPU por unidad de tiempo pero por medio de un programa sintético.
El código fuente de dicho programa debe compilarse y ejecutarse en el sistema GNU/Linux con el ensamblador GAS, sintaxis AT&T. Para la medición de su tiempo se utilizará el programa de Unix time que indicará el tiempo real de la aplicación.

***Términos indexados*— Benchmark, rendimiento, cantidad de instrucciones, programa.**

I. Descripción del Problema

Los benchmarks sintéticos son programas que tratan de incorporar las instrucciones típicas que conforman las aplicaciones reales y evalúan la rapidez con que se ejecutan. Por supuesto, los benchmarks sintéticos no generan ningún producto útil más allá de un resultado, porque sólo fueron programados para evaluar performance. De alguna manera, los benchmarks sintéticos son simuladores de aplicaciones reales. Aquí es donde surgen algunos problemas, porque los resultados pueden ser apropiados para estimar el rendimiento de una computadora en aplicaciones reales.

En este caso se realizará un Benchmark que permite cuantificar la cantidad de operaciones de CPU por unidad de tiempo por medio de un programa sintético.

17 de junio, 2011.

Yendry Rojas Rodríguez, carné 201025588, estudiante de Ingeniería en Computación, TEC Costa Rica, Sede San Carlos. (email: yenrr16@gmail.com).

Enmanuel Oviedo Ramírez, carné 201041992, estudiante de Ingeniería en Computación, TEC Costa Rica, Sede San Carlos. (email: eoviedo1691@gmail.com).

Luis Alonso Vega Brenes, carné 201042592, estudiante de Ingeniería en Computación, TEC Costa Rica, Sede San Carlos. (email: lavb91@gmail.com).

Este contará la cantidad de instrucciones ejecutadas en un algoritmo ($O(n^3)$) por ello debe recibir como parámetros la cantidad de iteraciones que se ejecutarán y una bandera que indicara si se utilizara un algoritmo que produce cache misses o no (-cm|-nocm). Un cache misses se refiere a un fallido intento de leer o escribir en una hoja de datos en la caché, lo que resulta en un acceso a memoria principal con una latencia de mucho más tiempo es por ello que este programa debe de recibir como parámetro una bandera que indique o no si ocurre un fallo de este tipo. Para la medición del tiempo, se utilizará el programa de Unix time que indicará el tiempo real de la aplicación. Con ello, para cada ejecución del programa podrá calcularse la cantidad de instrucciones enteras por segundo.
El código fuente debe ser implementado en un sistema GNU/Linux con el ensamblador GAS, sintaxis AT&T.

II. Estrategia de Solución Implementada

El programa recibe dos parámetros de entrada, uno que indica una cantidad n y otro que indica si se intentará provocar que el cache falle. La función debe ser de $O(n^3)$, por lo que se optó por realizar tres ciclos anidados donde cada uno se ejecuta n veces. Por otro lado tenemos una sección de memoria de 10 Mega Bytes de tamaño, la cual se usará para ser leída.
Para provocar o no que el cache falle en tener disponible las posiciones de memoria que se intentan accesar, se realizó una función que calcula la siguiente palabra de memoria que será leída, la cual puede variar en cada iteración dependiendo de dos factores básicos:

- La bandera de entrada –cm ó –nocm
- El valor que tiene actualmente el índice de ciclo más interno “z”

La función es la siguiente:

```
Posición buscada =
Inicio del buffer + (bandera * ((z % 10) * 1 MB))
```

Efectivamente, lo que se busca es realizar “pasos” de un Megabyte. Sin embargo, cuando se ingresa –nocm como parámetro de entrada, la bandera toma un valor de cero, lo que provoca que la posición de memoria sea siempre el inicio del buffer. De otra forma, la bandera se convierte en uno y las posiciones de memoria están separadas por aproximadamente un millón de bytes en cada iteración. Una variable interna cuenta el total de iteraciones realizadas, que de todas formas es

n^3, debido a la composición de los ciclos. Al finalizar el programa (correctamente), se muestra en consola la cantidad que almacena la variable mencionada, y además un aproximado del total de instrucciones realizadas dentro del ciclo, la cual es básicamente (n^3) * 11.

Aparte de esta función, el programa está diseñado para que muestre un mensaje de error y un texto de ayuda cuando no se ejecuta correctamente (los parámetros incorrectos, etc.). Con valores demasiado grandes (números enteros que no caben en 32 bits) el programa tendrá un error al mostrar el total de instrucciones o de iteraciones, debido a que se manejan los números siempre en 32 bits. Esto implica que con números de n mayores a 1625, se superará el límite de 2^32.

Benchmark IO

Aparte del programa original, pero muy similar a él, se creó el Benchmark para IO, el cual recibe 2 parámetros, de los cuales uno es igual al anterior (n) y el segundo es el nombre de un archivo temporal en el que se realizarán lecturas y escrituras.

El funcionamiento es similar, se ejecutan n^3 iteraciones en las que se escribe y lee un byte en el archivo. Al finalizar, si no hay errores, el programa cierra y elimina el archivo, por lo que se dice que (el archivo) es temporal.

Al igual que el programa original, se muestra en la terminal la cantidad de lecturas y escrituras realizadas (n^3). Si se encuentra algún problema con los parámetros, el valor de n, o el archivo, el programa envía un mensaje de error correspondiente, muestra la ayuda y la forma correcta de llamarse, y finaliza.

III. CONJUNTO DE PRUEBAS

Los siguientes resultados son los valores obtenidos utilizando el comando *time* de Unix. Se realizaron las pruebas y se calculó el promedio por cantidad de instancias "n" ejecutadas al mismo tiempo (utilizando un script para bash), y la desviación estándar, ubicadas en la parte inferior de cada tabla.

- Resultados utilizando n instancias, 25 veces, sin provocar cache misses:

	Instancias									
Prueba	1	2	3	4	5	6	7	8	9	10
1	0,788	0,797	1,241	1,599	2,069	2,443	2,770	3,097	3,476	3,829
2	0,774	0,809	1,219	1,634	2,158	2,379	2,738	3,088	3,472	3,861
3	0,768	0,986	1,239	1,587	1,986	2,347	2,727	3,151	3,485	3,896
4	0,781	0,815	1,270	1,592	2,083	2,341	2,742	3,144	3,502	3,872
5	0,789	0,889	1,331	1,575	2,069	2,342	2,988	3,111	3,488	3,885
6	0,744	0,976	1,195	1,572	1,973	2,393	2,795	3,118	3,560	3,891
7	0,798	0,800	1,204	1,551	2,000	2,357	2,878	3,143	3,512	3,896
8	0,791	0,810	1,239	1,560	2,041	2,335	2,738	3,100	3,483	3,894
9	0,781	0,808	1,217	1,582	2,089	2,380	2,812	3,107	3,552	3,921
10	0,767	0,885	1,215	1,590	1,960	2,360	2,772	3,183	3,516	3,890
11	0,760	0,979	1,244	1,617	2,084	2,393	2,841	3,175	3,502	3,840
12	0,754	0,843	1,243	1,574	2,058	2,364	2,845	3,160	3,546	3,843
13	0,771	0,933	1,257	1,570	2,040	2,384	2,837	3,161	3,540	3,851
14	0,770	0,880	1,239	1,595	1,991	2,376	2,973	3,112	3,508	3,908
15	0,760	0,824	1,275	1,562	2,071	2,353	2,834	3,130	3,500	3,890
16	0,797	0,945	1,206	1,581	2,080	2,381	2,754	3,171	3,536	3,869
17	0,768	0,886	1,253	1,577	2,030	2,350	2,845	3,134	3,489	3,901
18	0,776	0,909	1,272	1,602	1,971	2,366	2,810	3,103	3,523	3,896
19	0,790	0,889	1,264	1,571	2,063	2,382	2,846	3,146	3,551	3,893
20	0,759	0,933	1,211	1,577	1,964	2,385	2,755	3,103	3,547	3,846
21	0,793	0,974	1,212	1,610	2,072	2,385	2,965	3,112	3,531	3,865
22	0,770	0,939	1,226	1,578	2,039	2,355	2,947	3,125	3,508	3,864
23	0,769	0,916	1,256	1,605	2,035	2,384	2,873	3,150	3,522	3,899
24	0,763	0,834	1,216	1,565	2,054	2,378	2,820	3,139	3,546	3,866
25	0,788	0,945	1,232	1,605	2,010	2,382	2,821	3,124	3,501	3,854
Promedio	0,77476333	0,88809604	1,23906211	1,585207	2,03962037	2,37176394	2,82902975	3,13152201	3,51588211	3,87674605
Desviación E.	0,01435123	0,06343888	0,02970587	0,0196048	0,04818369	0,0227944	0,07583374	0,02646791	0,02628613	0,02414756

- Resultados con n desde 1 hasta 10 instancias, realizado 25 veces, provocando cache misses:

	Instancias									
Prueba	1	2	3	4	5	6	7	8	9	10
1	0,788	0,797	1,241	1,599	2,069	2,443	2,770	3,097	3,476	3,829
2	0,774	0,809	1,219	1,634	2,158	2,379	2,738	3,088	3,472	3,861
3	0,768	0,986	1,239	1,587	1,986	2,347	2,727	3,151	3,485	3,896
4	0,781	0,815	1,270	1,592	2,083	2,341	2,742	3,144	3,502	3,872
5	0,789	0,889	1,331	1,575	2,069	2,342	2,988	3,111	3,488	3,885
6	0,744	0,976	1,195	1,572	1,973	2,393	2,795	3,118	3,560	3,891
7	0,798	0,800	1,204	1,551	2,000	2,357	2,878	3,143	3,512	3,896
8	0,791	0,810	1,239	1,560	2,041	2,335	2,738	3,100	3,483	3,894
9	0,781	0,808	1,217	1,582	2,089	2,380	2,812	3,107	3,552	3,921
10	0,767	0,885	1,215	1,590	1,960	2,360	2,772	3,183	3,516	3,890
11	0,760	0,979	1,244	1,617	2,084	2,393	2,841	3,175	3,502	3,840
12	0,754	0,843	1,243	1,574	2,058	2,364	2,845	3,160	3,546	3,843
13	0,771	0,933	1,257	1,570	2,040	2,384	2,837	3,161	3,540	3,851
14	0,770	0,880	1,239	1,595	1,991	2,376	2,973	3,112	3,508	3,908
15	0,760	0,824	1,275	1,562	2,071	2,353	2,834	3,130	3,500	3,890
16	0,797	0,945	1,206	1,581	2,080	2,381	2,754	3,171	3,536	3,869
17	0,768	0,886	1,253	1,577	2,030	2,350	2,845	3,134	3,489	3,901
18	0,776	0,909	1,272	1,602	1,971	2,366	2,810	3,103	3,523	3,896
19	0,790	0,889	1,264	1,571	2,063	2,382	2,846	3,146	3,551	3,893
20	0,759	0,933	1,211	1,577	1,964	2,385	2,755	3,103	3,547	3,846
21	0,793	0,974	1,212	1,610	2,072	2,385	2,965	3,112	3,531	3,865
22	0,770	0,939	1,226	1,578	2,039	2,355	2,947	3,125	3,508	3,864
23	0,769	0,916	1,256	1,605	2,035	2,384	2,873	3,150	3,522	3,899
24	0,763	0,834	1,216	1,565	2,054	2,378	2,820	3,139	3,546	3,866
25	0,788	0,945	1,232	1,605	2,010	2,382	2,821	3,124	3,501	3,854
Promedio	0,77476333	0,88809604	1,23906211	1,585207	2,03962037	2,37176394	2,82902975	3,13152201	3,51588211	3,87674605
Desviación E.	0,01435123	0,06343888	0,02970587	0,0196048	0,04818369	0,0227944	0,07583374	0,02646791	0,02628613	0,02414756

I. Conclusiones

- Con respecto a los Benchmark, se trabajó en un proyecto interesante en donde aplicamos y reforzamos nuestros conocimientos en el lenguaje ASM x86, además de conocer más a fondo como trabajan este tipo de programas para la medición de rendimientos de ordenadores.

- La importancia de utilizar el lenguaje ensamblador radica principalmente que se trabaja directamente con el microprocesador; por lo cual se debe de conocer el funcionamiento interno de este, además tiene la ventaja de que en él se puede realizar cualquier tipo de programas que en los lenguajes de alto nivel no se pueden realizar. Otro punto sería que los programas en ensamblador ocupan menos espacio en memoria, es por eso que el haber programado en este tipo de lenguaje nos sirvió de mucho ya que es algo de lo cual no estamos acostumbrados en los demás cursos por lo cual fue provechoso para con nuestros conocimientos.

- Hoy en día es necesario conocer el rendimiento y las prestaciones de nuestro equipo computacional, ahí la importancia de los Benchmarks, ya que así conoceremos en período de obsolescencia del mismo y cómo se comporta el sistema en distintas tareas, entre las cuales pueden estar la resolución de problemas matemáticos, el procesamiento de gráficos e incluso su rendimiento ante los video juegos de alta exigencia, tanto por parte del procesador, como de los demás componentes.

II. Referencias e información usadas en el proyecto

1. Linux System Call Table. http://bluemaster.iu.hio.no/edu/dark/linasm/syscalls.html
2. Linux System Calls for HLA Programmers. http://webster.cs.ucr.edu/Page_Linux/LinuxSysCalls.pdf

3. COPIA VERBATIM DEL CÓDIGO FUENTE

```
################################################################################
#                                                                              #
#                                                                              #
#  benchmark - Programa de benchmarking de la memoria y cache                  #
#                                                                              #
#  Disenado por:                                                               #
#       - Yendry Rojas Rodriguez  (201025588)                                  #
#       - Enmanuel Oviedo Ramírez (201041992)                                  #
#       - Alonso Vega Brenes      (201042592)                                  #
#                                                                              #
#  TEC, Santa Clara                                                            #
#  Arquitectura de Computadores (IC3101)                                       #
#  Profesor Santiago Nunez Corrales                                            #
#  ____________________________________________________________________        #
#                                                                              #
#  Uso:  ./benchmark-io n bandera                                              #
#                                                                              #
#  n:    Valor que define la cantidad de veces que seran ejecutadas las        #
#        instrucciones                                                         #
#  bandera: Indica si el algoritmo provocara cache misses o no.                #
#        -cm   : Provocar cache misses                                         #
#        -nocm : Sin provocar cache misses                                     #
#                                                                              #
#                                                                              #
################################################################################

.section .data

    # Constantes de posicion de parametros en funciones
    .equ ARG1, 8                # Posicion del argumento 1
    .equ ARG2, 12               # Posicion del argumento 2
    .equ ARG3, 16               # Posicion del argumento 3
    .equ ARG4, 20               # Posicion del argumento 4
    .equ ARG5, 24               # Posicion del argumento 5
    .equ SYSCALL, 0x80          # Codigo de llamada al sistema

    # Manejo de archivos
    .equ READ, 3                # Leer
    .equ WRITE, 4               # Escribir
    .equ OPEN, 5                # Abrir
    .equ CLOSE, 6               # Cerrar
    .equ UNLINK, 10             # Eliminar

    # File Descriptors de Consola
    .equ STDIN, 0               # Lectura de consola
    .equ STDOUT, 1              # Escritura de consola

    # Valores de tamano de memoria
    .equ MEGABYTE, 1024 * 1024

    # Mensajes de error del programa
    ERR_PARAMETROS:    .ascii    "- La cantidad de parametros no es correcta\n\0"
    ERR_BANDERA:    .ascii    "- Error en la bandera de operacion\n\0"
    ERR_ITERACION:    .ascii    "- El valor de iteraciones no es correcto\n\0"
    # Mensaje de ayuda
    TEXTO_AYUDA:    .ascii    "- Uso del programa: \n./benchmark n flag\n"
    .ascii    "\tn = cantidad de instrucciones\n"
    .ascii    "\tflag: \t-cm : Provocar cache miss\n\t\t-nocm : Sin cache miss\n\0"
    # Texto de resultado
    TEXTO_RES1:        .ascii    "Total de iteraciones ejecutadas: \0"
    TEXTO_RES2:        .ascii    ".\nAproximado de instrucciones totales: \0"
    TEXTO_RES3:        .ascii    ".\n\0"
    # Espacio para almacenar numero textual en funcion printN
    number:            .ascii "\0"        # almacena el caracter a imprimir

.section .bss
    # Buffer de 10 MB: contiene las posiciones de memoria que se visitaran
```

```
    .lcomm    BUFFER, 10 * MEGABYTE

.section .text
  .global _start

    # Argumentos de entrada del programa
    .equ    ARGC, 0               # Cuenta de argumentos
    .equ    P_N_PROGRAM, 4        # Nombre del programa
    .equ    P_ITERACION, 8        # Valor de n
    .equ    P_BANDERA, 12         # Bandera de cache miss

    # Variables locales de n y bandera
    .equ     ITERACIONES, -4      # Desplazamiento en entero
    .equ     BANDERA, -8          # Valor de la bandera

_start:
    movl    %esp, %ebp

    # Variables locales
    subl    $8, %esp              # Espacio para las variables nuevas

    # Comprobar la cantidad de parametros recibida
    cmp        $3, ARGC(%ebp)            # Comprobar cantidad de args.
    jne        start_error_parametros    # Error si ARGC != 5

    # Revisar el valor de n
    pushl    P_ITERACION(%ebp)        # Enviar despl. como parametro
    call    string_int                # Convertir de texto a numero entero
    addl    $4, %esp                  # Liberar espacio de parametro

    cmp        $0, %eax                # Si el desplazamiento es menor a cero
    jl        start_error_iteracion    # es un error
    movl    %eax, ITERACIONES(%ebp)    # Se almacena el entero

    # Revisar la bandera
    pushl    P_BANDERA(%ebp)            # Se envia la bandera como parametro
    call    funcm                       # Se comprueba si es -cm, -nocm o error
    addl    $4, %esp                    # Liberar espacio de parametro

    cmp        $-1, %eax                # Si el resultado es -1
    je        start_error_bandera       # Saltar al mensaje de error
    cmp        $2, %eax                 # Si el resultado es mayor o igual a 2
    jge        start_error_bandera      # Saltar al mensaje de error
    movl    %eax, BANDERA(%ebp)         # Almacenar resultado localmente

    # Limpiar registros de uso general
    xorl    %eax, %eax
    xorl    %ebx, %ebx
    xorl    %ecx, %ecx
    xorl    %edx, %edx
    # Mover la cantidad de iteraciones (n) a %edi
    movl    ITERACIONES(%ebp), %edi

    # Inicio del primer ciclo
start_loop1:
    cmpl    %edi, %ebx                # Verificar si ya no debe entrar mas
    je        start_ok                # Entonces saltar al final

    incl    %ebx                      # Incrementar "x"
    xorl    %ecx, %ecx                # Limpiar "y"

    # Inicio del segundo ciclo
start_loop2:
    cmpl    %edi, %ecx                # Verificar si se debe salir del 2do ciclo
    je        start_loop1             # Volver al ciclo inicial

    incl    %ecx                      # Incrementar "y"
    xorl    %edx, %edx                # Limpiar "z"

    # Inicio del tercer ciclo
start_loop3:
    cmpl    %edi, %edx                # Verificar z con respecto a n
```

```
    je        start_loop2            # Si es igual, saltar al ciclo superior
    # Respaldar los registros generales
    pushl    %eax
    pushl    %ebx
    pushl    %ecx
    pushl    %edx

    # Formula para calcular la posicion de memoria que se accesara
    # mem[buffer + (bandera  * ((z % 10) * 1 MB))]
    # De esta forma cuando se ingresa -nocm la bandera sera cero
    # y la posicion de memoria sera siempre mem[buffer]
    # Por otra parte, al utilizar -cm, se revisaran posiciones de memoria
    # separadas por 1 MB, lo que provocara cache misses
    movl    %edx, %eax           # eax = z
    movl    $10, %ebx            # ebx = 10
    xorl    %edx, %edx           # limpiar %edx
    idivl    %ebx                # edx = z % 10
    movl    %edx, %eax           # eax = edx = z % 10
    movl    $MEGABYTE, %ebx      # ebx = 1 MB
    mull    %ebx                 # eax = (z % 10) * 1 MB
    movl    BANDERA(%ebp), %ecx  # ecx = bandera
    mull    %ecx                 # eax = bandera * eax
    addl    $BUFFER, %eax        # BUFFER + anterior
    movl    (%eax), %ebx         # Traer de memoria

    # Devolver los valores anteriormente respaldados
    popl    %edx
    popl    %ecx
    popl    %ebx
    popl    %eax

    incl    %edx                 # Incrementar "z"
    # Incrementar "a" que funciona como contador de instrucciones
    incl    %eax
    jmp        start_loop3       # Ir a siguiente iteracion

start_error_parametros:
    pushl    $ERR_PARAMETROS     # Enviar texto error de parametros
    call    printf               # Imprimir ese texto
    addl    $4, %esp             # Liberar espacio de parametro
    call    ayuda                # Mostrar texto de ayuda
    movl    $1, %ebx             # Error con la cantidad de parametros
    jmp        start_fin         # Ir al fin del programa

start_error_iteracion:
    pushl    $ERR_ITERACION      # Enviar texto error de desplazamiento
    call    printf               # Imprimir ese texto
    addl    $4, %esp             # Liberar espacio de parametro
    call    ayuda                # Mostrar texto de ayuda
    movl    $4, %ebx             # Error en el desplazamiento
    jmp        start_fin         # Ir al fin del programa

start_error_bandera:
    pushl    $ERR_BANDERA        # Enviar texto error de bandera
    call    printf               # Imprimir ese texto
    addl    $4, %esp             # Liberar espacio de parametro
    call    ayuda                # Mostrar texto de ayuda
    movl    $3, %ebx             # Error en el argumento de bandera
    jmp        start_fin         # Ir al fin del programa

start_ok:
    pushl    %eax                # Respaldar total de iteraciones
    pushl    %eax

    # Imprimir texto de resultado inicial
    pushl    $TEXTO_RES1
    call    printf
    addl    $4, %esp

    # Mostrar total de iteraciones
    popl    %eax
    pushl    %eax
```

```
    call    printN
    addl    $4, %esp

    # Imprimir 2da parte del texto de resultado
    pushl    $TEXTO_RES2
    call    printf
    addl    $4, %esp

    popl    %eax
    imull    $11, %eax
    pushl    %eax
    call    printN
    addl    $4, %esp

    # Imprimir final del texto de resultado
    pushl    $TEXTO_RES3
    call    printf
    addl    $4, %esp

    movl    $0, %ebx              # No hay errores

    # Finalizar programa
start_fin:
    movl    $1, %eax              # Codigo de salida
    int        $SYSCALL           # Llamada a sistema

  # ____________________________________________________________________________
  # funcm ( cadena )
  # devuelve 0 si la bandera de entrada es igual a -cm,
  # 1 si la bandera de entrada es igual a -nocm
  # y -1 en cualquier otro caso

.type funcm, @function
funcm:
    pushl     %ebp
    movl     %esp, %ebp

    xorl     %eax, %eax               # limpia el registro eax
    xorl     %ebx, %ebx               # limpia el registro ebx
    xorl     %edx, %edx               # limpia el registro edx
    xorl     %edi, %edi               # limpia el registro edi

    movl     ARG1(%ebp), %ebx         # carga el parametro de la funcion en ebx
    movb     (%ebx, %edi, 1), %dl     # carga el primer caracter de la cadena

    cmpb     $'-' ,%dl                # compara si la cadena es correcta
    jne     funcm_error               # salto a indicador de error

    incl     %edi                     # incremento de indice
    movb     (%ebx, %edi, 1), %dl     # carga el segundo caracter de la cadena
    cmpb     $'c' ,%dl                # compara si el caracter es = 'c'
    jne     funcm_if_nocm

    incl     %edi                     # incremento de indice
    movb     (%ebx, %edi, 1), %dl     # carga el tercer caracter de la cadena
    cmpb     $'m' ,%dl                # compara si el caracter es = 'm'
    jne     funcm_error

    incl     %edi
    movb     (%ebx, %edi, 1), %dl     # carga el tercer caracter de la cadena
    cmpb     $0 ,%dl                  # si existe un valor distinto al final
    jg         funcm_error            # de la cadena, si es asi salta a error

    movl     $1, %eax                 # carga el valor de retono en eax
    jmp     funcm_final               # salto al final de la funcion

funcm_if_nocm:                        # segunda condicion
    cmpb     $'n' ,%dl                # compara si la bandera de entrada es -nocm
    jne     funcm_error               # salto a indicador de error

    incl     %edi                     # incremento de indice
    movb     (%ebx, %edi, 1), %dl     # carga el tercer caracter de la cadena
```

```
    cmpb     $'o' ,%dl                # compara si el caracter es = 'o'
    jne     funcm_error               # salto a indicador de error

    incl     %edi                     # incremento de indice
    movb     (%ebx, %edi, 1), %dl     # carga el cuarto caracter de la cadena
    cmpb     $'c' ,%dl                # compara si la bandera de entrada es -d
    jne     funcm_error               # salto a indicador de error

    incl     %edi
    movb     (%ebx, %edi, 1), %dl     # carga el quinto caracter de la cadena
    cmpb     $'m' ,%dl                # compara si el caracter es = 'm'
    jne     funcm_error               # salto a indicador de error

    incl     %edi
    movb     (%ebx, %edi, 1), %dl     # carga el sexto caracter de la cadena
    cmpb     $0 ,%dl                  # compara si el caracter final es diferente al
                                      # esperado (\0). si es asi salta a error
    jg         funcm_error

    movl    $0, %eax                  # carga el valor de retono en eax
    jmp        funcm_final            # salto al final de la funcion

funcm_error:
    movl     $-1, %eax                # carga un valor de error en eax

funcm_final:
    movl    %ebp, %esp
    popl    %ebp
    ret

  # ____________________________________________________________________________
  # printf ( cadena )
  # Imprime en consola un texto
  .type printf, @function
printf:
    pushl    %ebp
    movl    %esp, %ebp

    subl    $4, %esp                 # Obtener espacio para variable local
    movl    $0, -4(%ebp)             # Mover 0 a variable local

    xorl    %eax, %eax               # limpia el registro eax
    xorl    %ebx, %ebx               # limpia el registro ebx
    xorl    %edi, %edi               # limpia el registro edi

    movl    8(%ebp), %ebx            # direccion del texto
                                     # que se quiere imprimir
    movb    (%ebx, %edi, 1), %dl     # primer caracter de la cadena

printf_imp:
    cmpb    $0, %dl                  # fin de la cadena
    je        printf_final           # salto al final de la funcion

    movb    %dl, -4(%ebp)            # caracter a imprimir
    movl    $1, %edx                 # longitud del caracter

    movl    %ebp, %ecx               # direccion del caracter a imprimir
    subl    $4, %ecx                 # en ebp menos 4

    movl    $STDOUT, %ebx            # identificador de archivo (stdout)
    movl    $WRITE, %eax             # sys_write (=4)
    int        $0x80                 # llamada a interrupcion de software

    incl    %edi                     # incremento de edi
    movl    8(%ebp), %ebx            # carga la direccion del texto a imprimir
    movb    (%ebx, %edi, 1), %dl     # siguiente caracter de la cadena
    jmp        printf_imp            # salto a inicio del bucle

printf_final:
    movl    %ebp, %esp
    popl    %ebp
    ret
```

```
  # ____________________________________________________________________________
  # caracter_int ( caracter )
  # Transforma solo un caracter a numero
 .type caracter_int, @function
caracter_int:
    pushl    %ebp
    movl    %esp, %ebp

    xorl    %eax, %eax              # limpia el registro eax
    movl    8(%ebp), %eax           # carga el parametro de la funcion en eax

    cmp        $48, %eax            # se compara si el caracter es un número
                                    # menor a cero
    jl        caracter_int_error    # el caracter no es convertible
    cmpl    $57, %eax               # compara si el caracter es un número
                                    # mayor que nueve
    jg        caracter_int_error    # el caracter no es convertible

    subl    $48, %eax               # convierte el caracter numerico a número
    jmp        caracter_int_fin     # salto al final del método

caracter_int_error:
    movl $-1, %eax                  # indica que el caracter no es numerico

caracter_int_fin:
    movl %ebp, %esp
    popl %ebp
    ret

  # ____________________________________________________________________________
  # string_int ( cadena )
  # Transforma una cadena de caracteres a numero
  .type string_int, @function
string_int:
    pushl %ebp
    movl %esp, %ebp

    xorl    %ecx, %ecx               # limpia el registro ecx
    xorl    %edx, %edx               # limpia el registro edx
    xorl    %edi, %edi               # limpia el registro edi
    movl    8(%ebp), %ebx            # carga el parametro de la funcion en ebx
    movb    (%ebx, %edi, 1), %dl     # carga el primer caracter de la cadena

string_int_while:
    cmpb    $0, %dl                  # final de la cadena
    je        string_int_ultimo      # salto a la etiqueta "ultimo"
    pushl    %edx                    # carga el caracter que se decea convertir
    call    caracter_int             # llamada a la funcion convertir caracter
    addl    $4, %esp                 # se libera el espacio asignado en la pila

    cmpl    $-1, %eax                # si el caracter no fue convertible
    je        string_int_final       # salto al final de la funcion

    imull    $10, %ecx               # multiplica el resultado almacenado
                                     # para agregar el nuevo digito
    addl    %eax, %ecx               # se agrega el nuevo digito al total

    incl    %edi                     # incrementa del indice del while
    movb    (%ebx, %edi, 1), %dl     # se actualiza el nuevo caracter
    jmp        string_int_while      # salto al inicio del while

string_int_ultimo:
    movl    %ecx, %eax               # transferencia de resultado final a eax

string_int_final:
    movl    %ebp, %esp
    popl    %ebp
    ret

  # ____________________________________________________________________________
  # printN ( int )
```

```
  # Imprime un numero en la consola
.type printN, @function
printN:
    pushl    %ebp
    movl    %esp, %ebp

    xorl    %eax, %eax                # limpia el registro eax
    xorl    %ecx, %ecx                # limpia el registro ecx
    xorl    %edi, %edi                # limpia el registro edi

    movl    ARG1(%ebp), %eax          # carga el parametro de la funcion en eax
    movl    %ebp, %edi                # respaldo de ebp

    xorl    %ebp, %ebp                # limpia el registro ebp

printn_to_string:                     # convertir de numero a string
    movl    $10, %ecx
    xorl    %edx, %edx                # limpia el registro %edx
    cmpl    %eax, %ecx
    jg        printn_ult_d            # salta si el numero < 10
    divl    %ecx                      # divide el numero entre 10
    addl    $48, %edx                 # suma 48 al residuo para obtener
                                      # el valor en ascii
    push    %edx                      # inserta el nuevo caracter en la pila
    inc        %ebp                   # incrementa el contador de la pila
    jmp        printn_to_string       # retorna al inicio (to_string)

printn_ult_d:                         # ultimo digito
    addl    $48, %eax                 # suma 48 al ultimo digito para
                                      # convertirlo a ascii
    push    %eax                      # se inserta el caracter en la pila
    incl    %ebp                      # incrementa el contador de pila

printn_num:                           # imprimir en consola la cadena en la pila
    xorl    %ebx, %ebx                # limpia el registro %ebx
    cmpl    %ebp, %ebx                # comprueba si aun se deben
                                      # sacar elementos de la pila
    jz        printn_fin              # si ya no quedan elementos, salta a fin
    pop        %edx                   # extrae un caracter de la pila
    dec        %ebp                   # disminuye el contador de elementos en la pila

    movl    %edx, number              # envia el caracter a memoria
    movl    $WRITE, %eax              # SYS_WRITE(4)
    movl    $STDOUT, %ebx             # STD_OUT(1)
    movl    $number, %ecx             # copia el caracter a %ecx
    movl    $1, %edx                  # tamaño de caracter en %edx
    int        $SYSCALL               # llamada a interrupcion de linux

    jmp        printn_num             # salto a print_num para imprimir
                                      # los demas caracteres
printn_fin:
    movl    %edi, %esp
    popl    %ebp
    ret

  # ____________________________________________________________________________
  # ayuda ( )
  # Muestra la ayuda en consola
  .type ayuda, @function
ayuda:
    pushl %ebp
    movl %esp, %ebp

    pushl    $TEXTO_AYUDA            # Enviar texto de ayuda como parametro
    call    printf                   # Imprimir texto
    addl    $4, %esp                 # Liberar espacio de parametro

    movl    %ebp, %esp
    popl    %ebp
    ret

  # FIN DEL PROGRAMA #
```