

1. OBJETIVOS DE LA PRÁCTICA

- Aprender a utilizar los compiladores de ensamblador para arquitecturas Intel bajo el sistema operativo Linux (compiladores “NASM” y “GAS”).
- Saber integrar código ensamblador dentro de archivos fuente escritos en C o C++ (compiladores “GCC” e “ICC”).
- Conocer las instrucciones orientadas al procesamiento multimedia que Intel ha ido agregando a su arquitectura en sucesivas entregas (MMX, SSE, SSE2, SSE3, SSE4...).
- Valorar y comparar las mejoras que supone la paralelización de operaciones aritméticas (tanto en punto fijo como en punto flotante) presentes en algoritmos de procesamiento multimedia, frente a la implementación de esos mismos algoritmos con técnicas estrictamente secuenciales.

2. HARDWARE Y SOFTWARE NECESARIO

2.1 - HARDWARE NECESARIO

- Máquinas con CPU's Intel Pentium 4 o superiores (Intel Core Duo, Intel Core 2 Duo, Intel Quad Core, etc).

2.2 - SOFTWARE NECESARIO

- Sistema Operativo: Linux (distribuciones probadas en el laboratorio: Ubuntu y Fedora.
- Compiladores ICC (/opt/intel/Compiler/<versión>/bin/ia32/icc -para código C- y /opt/intel/Compiler/<versión>/bin/ia32/icpc -para código C++). Son gratuitos; pueden descargarse desde <http://software.intel.com/en-us/articles/non-commercial-software-download/#compilers>.

2. HARDWARE Y SOFTWARE NECESARIO

SOFTWARE NECESARIO (continuación)

- Compiladores de ensamblador Intel bajo Linux:
 - GAS (GNU Assembler): Es “/usr/bin/as” y, al igual que GCC, suele formar parte de cualquier distribución de Linux.
 - NASM (Netwide Assembler): Es “/usr/bin/nasm”. Si no está preinstalado **(1)**, instalar el correspondiente paquete **(2)**.

(1)

```
[root@localhost ~]$ rpm -q nasm  
nasm-0.98.39-5.fc7
```

(2)

```
[root@localhost ~]$ yum install nasm
```

3. PRIMERA TOMA DE CONTACTO CON NASM

3.1 - CARACTERÍSTICAS GENERALES DE NASM

- Utiliza una sintaxis similar a la del “Macro Assembler” (MASM) del DOS.
- Estructura del código fuente de un programa: Sección “.data” (constantes), sección “.bss” (variables) y sección “.text” (código).
- Operaciones de E/S: Realizables a través de llamadas al sistema operativo, con la interrupción software 80h (equivalente a la 21h del DOS). *En esta práctica no será necesario su uso* (si hay operaciones de E/S, éstas se implementarán en código C y no en ensamblador, que quedará reservado sólo para el uso de las instrucciones que formen parte de las extensiones multimedia que sea preciso utilizar).
- Edición del código fuente: Con el “vi” o con cualquier otro editor de texto.

3. PRIMERA TOMA DE CONTACTO CON NASM

CARACTERÍSTICAS GENERALES DE NASM (continuación)

- Archivos intervinientes: <Fuente>.asm, <Objeto>.o y <Ejecutable>
- Compilación y lincaje:

- Compilación (generación del fichero objeto):

```
nasm -f elf <nombre>.asm
```

- Lincaje (generación del fichero ejecutable):

```
ld -o <nombre> <nombre>.o
```

- Más información: Archivo “[asm-linux.pdf](#)”, en <http://webs.um.es/einiesta/miwiki/doku.php?id=docencia>

3. PRIMERA TOMA DE CONTACTO CON NASM

3.2 - PROGRAMA DE EJEMPLO EN NASM

- Código fuente (“hola.asm”):

```
section .data
    mensaje db "hola mundo",0xA
    longitud equ $ - mensaje

section .text
    global _start      ; definimos el punto de entrada.
_start:
    mov edx,longitud ; EDX=longitud de la cadena.
    mov ecx,mensaje  ; ECX=cadena a imprimir.
    mov ebx,1        ; EBX=manejador del fichero (STDOUT)
    mov eax,4        ; EAX=función sys_write() del kernel.
    int 0x80         ; interrupción software 80h (llamada al kernel).
    mov ebx,0        ; EBX=código de salida al SO ($?).
    mov eax,1        ; EAX=función sys_exit() del kernel.
    int 0x80         ; interrupción software 80h (llamada al kernel).
```

3. PRIMERA TOMA DE CONTACTO CON NASM

PROGRAMA DE EJEMPLO EN NASM (continuación)

- Compilación, lincaje y ejecución:

```
[einiesta@localhost holamundo]$ ls -l
total 16
-rw-rw-r-- 1 einiesta einiesta 529 mar  4 20:47 hola.asm
[einiesta@localhost holamundo]$ nasm -f elf hola.asm
[einiesta@localhost holamundo]$ ls -l
total 24
-rw-rw-r-- 1 einiesta einiesta 529 mar  4 20:47 hola.asm
-rw-rw-r-- 1 einiesta einiesta 736 mar  8 20:38 hola.o
[einiesta@localhost holamundo]$ ld -o hola hola.o
[einiesta@localhost holamundo]$ ls -l
total 32
-rwxrwxr-x 1 einiesta einiesta 440 mar  8 20:38 hola
-rw-rw-r-- 1 einiesta einiesta 529 mar  4 20:47 hola.asm
-rw-rw-r-- 1 einiesta einiesta 736 mar  8 20:38 hola.o
[einiesta@localhost holamundo]$ ./hola
hola mundo
```

4. PRIMERA TOMA DE CONTACTO CON GAS

4.1 - CARACTERÍSTICAS GENERALES DE GAS

- Utiliza una sintaxis propia, diferente de la utilizada por NASM (linux) y MASM (dos).
- Es el ensamblador **estándar** de Linux, y **el que utilizaremos en las prácticas** cuando incluyamos llamadas a extensiones multimedia en nuestro código C o C++.
- Estructura del código fuente de un programa: Similar a NASM, es decir, con sección “.data” (constantes), sección “.bss” (variables) y sección “.text” (código).
- Operaciones de E/S: Como en el caso de NASM, realizables a través de llamadas al sistema, con la interrupción software 80h.
- Edición del código fuente: Con el “vi” o con cualquier otro editor de texto.

4. PRIMERA TOMA DE CONTACTO CON GAS

CARACTERÍSTICAS GENERALES DE GAS (continuación)

- Archivos intervinientes: <Fuente>.s, <Objeto>.o y <Ejecutable>

- Compilación y lincaje:

· Compilación (generación del fichero objeto):

```
as -o <nombre>.o <nombre>.s
```

· Lincaje (generación del fichero ejecutable):

```
ld -o <nombre> <nombre>.o
```

- Más información: Archivo “[asm-linux.pdf](#)”, en <http://webs.um.es/einiesta/miwiki/doku.php?id=docencia>

4. PRIMERA TOMA DE CONTACTO CON GAS

4.2 - PRINCIPALES DIFERENCIAS ENTRE LA SINTAXIS GAS Y LA SINTAXIS NASM

- A los nombres de los registros se les añade el prefijo “%” (ejemplos: `%eax`, `%ebx`, `%ecx`, `%edx`, `%edi`, `%esi`, etc).
- El operando destino se coloca a la derecha, y el operando fuente a la izquierda (ejemplo: lo que en NASM es un `mov ebx, eax`, en GAS es un `movl %eax, %ebx`).
- El tamaño del resultado se establece explícitamente en las instrucciones “mov”, utilizando los sufijos “b”, “w” o “l” para, respectivamente, los tamaños “byte”, “word” o “long word” (ejemplos: `movw %bx, %ax`, `movl (%ebx), %eax`).
- Los valores inmediatos han de ir precedidos por el carácter “\$” (ejemplo: `movl $0xf02, %ebx`).
- Otros matices (“`[.section]`” por “`section`”, “`.globl`” por “`global`”, “`ascii`” por “`db`”, etc).

4. PRIMERA TOMA DE CONTACTO CON GAS

4.3 - PROGRAMA DE EJEMPLO EN GAS

- Código fuente (“hola.s”):

```
.section .data
    mensaje: .ascii "hola mundo \n"
    longitud = . - mensaje

.section .text
    .globl _start
_start: movl $longitud,%edx    # EDX=longitud de la cadena.
    movl $mensaje,%ecx       # ECX=cadena a imprimir.
    movl $1,%ebx             # EBX=manejador del fichero (STDOUT).
    movl $4,%eax             # EAX=función sys_write() del kernel.
    int $0x80                # Interrupción software 80h (llamada al kernel).
    movl $0,%ebx             # EBX=código de salida al SO ($?).
    movl $1,%eax             # EAX=función sys_exit() del kernel.
    int $0x80                # Interrupción software 80h (llamada al kernel).
```

4. PRIMERA TOMA DE CONTACTO CON GAS

PROGRAMA DE EJEMPLO EN GAS (continuación)

- Compilación, lincaje y ejecución:

```
[einiesta@localhost holamundo]$ ls -l
total 8
-rw-rw-r-- 1 einiesta einiesta 522 mar  8 22:17 hola.s
[einiesta@localhost holamundo]$ as -o hola.o hola.s
[einiesta@localhost holamundo]$ ls -l
total 16
-rw-rw-r-- 1 einiesta einiesta 616 mar  8 22:18 hola.o
-rw-rw-r-- 1 einiesta einiesta 522 mar  8 22:17 hola.s
[einiesta@localhost holamundo]$ ld -o hola hola.o
[einiesta@localhost holamundo]$ ls -l
total 24
-rwxrwxr-x 1 einiesta einiesta 637 mar  8 22:18 hola
-rw-rw-r-- 1 einiesta einiesta 616 mar  8 22:18 hola.o
-rw-rw-r-- 1 einiesta einiesta 522 mar  8 22:17 hola.s
[einiesta@localhost holamundo]$ ./hola
hola mundo
```

5. PRIMERA TOMA DE CONTACTO CON ICC

5.1 - CARACTERÍSTICAS GENERALES DE ICC

- ICC, o “Intel C++”, es un grupo de dos compiladores creados por Intel, el “icc” y el “icpc”, que permiten, respectivamente, la implementación de aplicaciones utilizando los lenguajes C y C++.
- Es de libre distribución, y dispone de versiones para Windows y Linux, y también para diversas variantes de la arquitectura Intel (CPU's de 32 y 64 bits).
- La sintaxis de los dos compiladores (línea de comandos) es muy similar a la de los respectivos compiladores homólogos de GCC (“gcc” y “g++”).
- Permiten la integración de código de GAS (“ensamblador en línea”).
- Posibilitan el uso de extensiones multimedia (mediante ensamblador GAS en línea, mediante librerías en C o a través de clases de C++).

5. PRIMERA TOMA DE CONTACTO CON ICC

5.2 - INSTALACIÓN Y PUESTA EN MARCHA

- La descarga de la instalación se lleva a cabo desde <http://software.intel.com/en-us/articles/non-commercial-software-download/#compilers>. Hay que tener especial cuidado en elegir la versión apropiada del ICC. A fecha de Marzo de 2010, esta versión sería la siguiente: “*Intel C++ Compiler Professional Edition 11.1 for Linux*”; en cuanto a la arquitectura, si no se está seguro, elegir la versión de 32 bits, que es compatible con todo tipo de máquinas (aunque del Intel Core 2 Duo en adelante se podría elegir también la versión de 64 bits).
- Para instalar ICC, síganse las instrucciones correspondientes que se incluyen en el paquete de instalación. Resultado de la instalación (respetando las opciones propuestas por el asistente de instalación):
 - Directorio base de instalación: `/opt/intel`.
 - Ubicación de los compiladores: `/opt/intel/Compiler/<versión>/bin/ia32/icc` (compilador de C), y `/opt/intel/Compiler/<versión>/bin/ia32/icpc` (compilador de C++).
 - Ubicación del depurador: `/opt/intel/Compiler/<versión>/bin/ia32/idb`

5. PRIMERA TOMA DE CONTACTO CON ICC

INSTALACIÓN Y PUESTA EN MARCHA (continuación)

- Una vez terminada la instalación, hay que proceder a realizar una “postinstalación”:
 - Asegurarse de que al inicio de cada sesión se ejecuta el siguiente script. En caso necesario, actualizar “.bashrc” **(3)**.

```
/opt/intel/Compiler/<version>/bin/ia32/iccvars_ia32.sh
```

- Comprobar que la variable de entorno “LD_LIBRARY_PATH” está instanciada con la ruta de las librerías que utiliza ICC (/opt/intel/Compiler/<versión>/lib/ia32).

(3)

```
source /opt/intel/Compiler/<version>/bin/ia32/iccvars_ia32.sh
```

5. PRIMERA TOMA DE CONTACTO CON ICC

5.3 - COMPILACIÓN CON ICC

- Archivos intervinientes: <Fuente>.c y <Ejecutable>.
- Compilación: `icc <nombre>.c -o nombre`

IMPORTANTE:

Si algún fichero fuente incluye extensiones multimedia, puede ser necesario (aunque no siempre) obligar al compilador a que adapte el código máquina generado a la arquitectura Intel concreta en la que dicho código va a ejecutarse. Para ello, hay que utilizar el argumento “-ax” seguido de una letra que identifica la arquitectura. Por ejemplo, para compilar un programa en C con extensiones multimedia que vaya a ejecutarse sobre un Intel Core 2 Duo, habría que escribir:

```
icc -axT <nombre>.c -o nombre.
```


5. PRIMERA TOMA DE CONTACTO CON ICC

COMPILACIÓN CON ICC (continuación)

Valores posibles de “-ax<Arquitectura>”:

- -axT: Intel Core 2 Duo, Intel Core 2 Quad.
- -axP: Intel Core Duo.
- -axW: Intel Pentium IV.
- -axK: Intel Pentium III.

- Más información:

- Archivo “[resumen_icc.pdf](#)”, en <http://webs.um.es/einiesta/miwiki/doku.php?id=docencia>
- Páginas del manual: `man icc, o konqueror man:icc.`

5. PRIMERA TOMA DE CONTACTO CON ICC

5.4 - COMPILACIÓN CON ICPC

- Archivos intervinientes: <Fuente>.cpp y <Ejecutable>.
- Compilación: `icpc <nombre>.cpp -o nombre`

IMPORTANTE:

Si algún fichero fuente incluye extensiones multimedia, puede ser necesario (aunque no siempre) obligar al compilador a que adapte el código máquina generado a la arquitectura Intel concreta en la que dicho código va a ejecutarse. Para ello, hay que utilizar el argumento “-ax” seguido de una letra que identifica la arquitectura. Por ejemplo, para compilar un programa en C++ con extensiones multimedia que vaya a ejecutarse sobre un Intel Core 2 Duo, habría que escribir:

```
icpc -axT <nombre>.c -o nombre.
```

5. PRIMERA TOMA DE CONTACTO CON ICC

COMPILACIÓN CON ICPC (continuación)

Valores posibles de “-ax<Arquitectura>”:

- -axT: Intel Core 2 Duo, Intel Core 2 Quad.
- -axP: Intel Core Duo.
- -axW: Intel Pentium IV.
- -axK: Intel Pentium III.

- Más información:

- Archivo “[resumen_icpc.pdf](#)”, en <http://webs.um.es/einiesta/miwiki/doku.php?id=docencia>
- Páginas del manual: `man icpc`, `0 konqueror man:icpc`.

5. PRIMERA TOMA DE CONTACTO CON ICC

5.5 - DEPURACIÓN DEL CÓDIGO

- Se emplea el programa depurador “idb” (/opt/intel/Compiler/<versión>/bin/ia32/idb), adjuntándole como argumento el fichero ejecutable a depurar. Condición previa para la depuración: Utilizar el argumento “-g”, tanto en “icc” como en “icpc”. Ejemplo:

```
[einiesta@localhost ejicc]$ ls -l
total 8
-rw-rw-r-- 1 einiesta einiesta 406 mar  9 12:47 suma.c
[einiesta@localhost ejicc]$ icc -g suma.c -o suma
[einiesta@localhost ejicc]$ ls -l
total 20
-rwxrwxr-x 1 einiesta einiesta 6720 mar  9 13:03 suma
-rw-rw-r-- 1 einiesta einiesta  406 mar  9 12:47 suma.c
[einiesta@localhost ejicc]$ idb ./suma
Intel(R) Debugger for applications running on IA-32, Version 10.0-29 , Build 20070405
-----
object file name: ./suma
Reading symbols from /home/einiesta/am_0708/prac_0708/ejicc/suma...done.
(idb)
```

5. PRIMERA TOMA DE CONTACTO CON ICC

DEPURACIÓN DEL CÓDIGO (continuación)

- Comandos interactivos de “idb”:
 - r [<argumentos>]: Inicia la ejecución del programa.
 - b [<fich>]:<nlinea>: Detiene la ejecución en la línea <nlinea> (punto de ruptura).
 - b <función>: Detiene la ejecución en la primera línea de <función> (punto de ruptura).
 - info b: Muestra todos los puntos de ruptura.
 - delete b: Elimina todos los puntos de ruptura.
 - delete b <npr>: Elimina el punto de ruptura número <npr>.
 - c: Continúa una ejecución previamente detenida hasta el siguiente punto de ruptura, si lo hay, o hasta el final del programa, si no lo hay.
 - p <variable>: Muestra el valor de la variable <variable>.
 - p \$<registro>: Muestra el valor del registro <registro> (ejemplo: `print $xmm0`).
 - n: Ejecuta la línea siguiente, y si esa línea corresponde a una función, ejecuta la función completa.
 - s: Ejecuta la línea siguiente, y si esa línea corresponde a una función, entra dentro de la función y se detiene en la primera de sus líneas.
 - backtrace: Muestra la pila de llamadas a funciones.

5. PRIMERA TOMA DE CONTACTO CON ICC

5.6 - EJEMPLO DE DEPURACIÓN

```
1  #include <stdio.h>
2
3  int calcular_suma(int sumando_1,int sumando_2);
4
5  int main()
6  {
7      int a,b,s;
8      printf("Introduzca sumando 1: ");scanf("%d",&a);
9      printf("Introduzca sumando 2: ");scanf("%d",&b);
10     s=calcular_suma(a,b);
11     printf("Resultado de la suma: %d\n",s);
12     return(0);
13 }
14
15 int calcular_suma(int sumando_1,int sumando_2)
16 {
17     int resultado;
18     resultado=sumando_1+sumando_2;
19     return resultado;
20 }
```

5. PRIMERA TOMA DE CONTACTO CON ICC

EJEMPLO DE DEPURACIÓN (continuación)

```
[einiesta@localhost ejicc]$ idb ./suma
Intel(R) Debugger for applications running on IA-32, Version 10.0-29 , Build 20070405
-----
object file name: ./suma
Reading symbols from /home/einiesta/am_0708/prac_0708/ejicc/suma...done.
(idb) b 9
Breakpoint 1 at 0x804843c: file suma.c, line 9.
(idb) b calcular_suma
Breakpoint 2 at 0x80484a6: file suma.c, line 18.
(idb) r
Starting program: /home/einiesta/am_0708/prac_0708/ejicc/suma
Introduzca sumando 1: 5
Breakpoint 1, main () at suma.c:9
9      printf("Introduzca sumando 2: ");scanf("%d",&b);
(idb) p a
$1 = 5
(idb) p b
$2 = 13672436
(idb) n
```

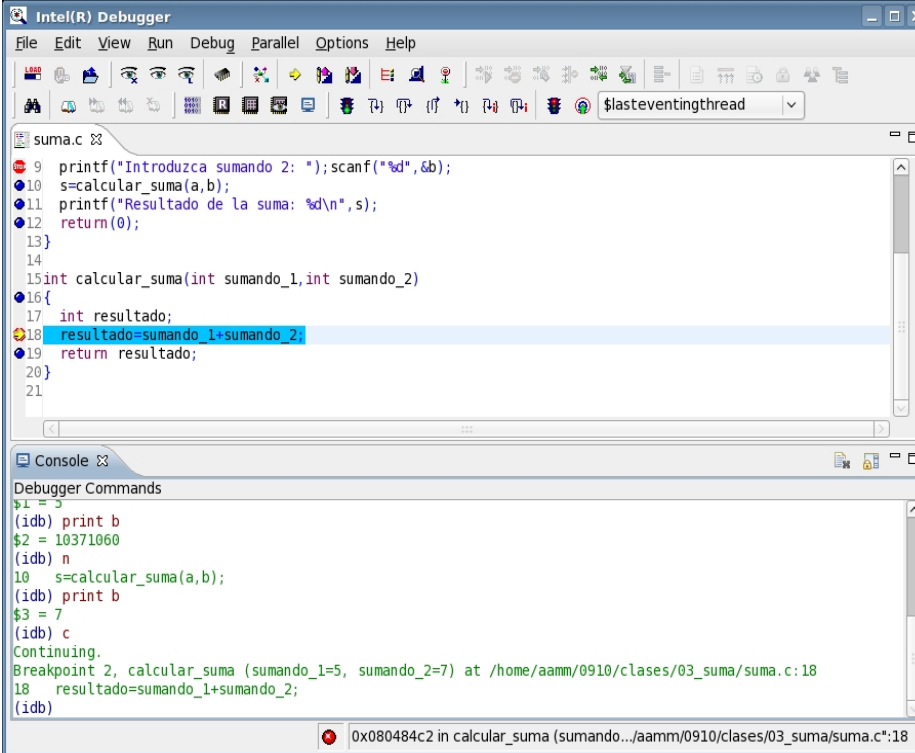
5. PRIMERA TOMA DE CONTACTO CON ICC

EJEMPLO DE DEPURACIÓN (continuación)

```
Introduzca sumando 2: 7
10      s=calcular_suma(a,b);
(idb) print b
$3 = 7
(idb) c
Continuing.
Breakpoint 2, calcular_suma (sumando_1=5, sumando_2=7) at suma.c:18
18      resultado=sumando_1+sumando_2;
(idb) print resultado
$4 = 2198688
(idb) n
19      return resultado;
(idb) print resultado
$5 = 12
(idb) backtrace
#0  0x080484af in calcular_suma (sumando_1=5, sumando_2=7) at suma.c:19
#1  0x08048478 in main () at suma.c:10
(idb) c
Continuing.
Resultado de la suma: 12
Program exited normally.
```


5. PRIMERA TOMA DE CONTACTO CON ICC

EJEMPLO DE DEPURACIÓN (interfaz gráfica)



```
Intel(R) Debugger
File Edit View Run Debug Parallel Options Help
suma.c
9 printf("Introduzca sumando 2: "); scanf("%d",&b);
10 s=calcular_suma(a,b);
11 printf("Resultado de la suma: %d\n",s);
12 return(0);
13 }
14
15 int calcular_suma(int sumando_1,int sumando_2)
16 {
17     int resultado;
18     resultado=sumando_1+sumando_2;
19     return resultado;
20 }
21

Console
Debugger Commands
$1 = 5
(idb) print b
$2 = 10371060
(idb) n
10 s=calcular_suma(a,b);
(idb) print b
$3 = 7
(idb) c
Continuing.
Breakpoint 2, calcular_suma (sumando_1=5, sumando_2=7) at /home/aamm/0910/clases/03_suma/suma.c:18
18 resultado=sumando_1+sumando_2;
(idb)
0x080484c2 in calcular_suma (sumando.../aamm/0910/clases/03_suma/suma.c*:18
```

5. PRIMERA TOMA DE CONTACTO CON ICC

5.7 - DEPURACIÓN “POSTMORTEM”

- Cuando en un programa se produzca una excepción de origen “desconocido”, podemos, con “`idb`”, resituar la ejecución justo en el instante anterior a la excepción, lo que nos permitirá, entre otras cosas, consultar el valor de las variables y determinar el contexto exacto bajo el que se ha producido el fallo. Pasos:
 - Forzar en las sesiones actuales la posibilidad de generar ficheros core de tamaño ilimitado: `ulimit -c unlimited` (recomendación: incluir esta orden en `$HOME/.bashrc`).
 - Compilar los fuentes con la opción “-g” (opción de depuración convencional).
 - Después del error, ejecutar “`idb`” pasándole como argumento el fichero core recién generado: `idb <programa> --core=<fichero_core>`
- La compilación definitiva del programa, una vez depurado, no debe llevarse a cabo con “-g” (ejecución mucho menos eficiente).

5. PRIMERA TOMA DE CONTACTO CON ICC

5.8 - ANÁLISIS DE LA EFICIENCIA

- Con “gprof” (/usr/bin/gprof) es posible conocer el desglose de tiempos consumidos por cada una de las funciones (o métodos) de un programa. Para ello, hay que seguir los pasos siguientes:
 - Compilar con la opción “-pg”: `icc -pg -o <nombre_ejecutable> <nombre_fuente>.c`
 - Ejecutar el programa (se generará un fichero llamado “gmon.out”).
 - Ejecutar `gprof ./<nombre_ejecutable>`
 - De la salida que produce la orden anterior, prestar especial atención a los campos “name” y “%time” (nombre de función y porcentaje de tiempo consumido).
- La compilación definitiva del programa, una vez depurado, no debe llevarse a cabo con “-pg” (aunque nos sirva para obtener tiempos relativos de funciones, resulta algo menos eficiente).

6. ENSAMBLADOR EN LÍNEA DENTRO DE CÓDIGO C/C++

6.1 - INTEGRACIÓN DE CÓDIGO ENSAMBLADOR GAS EN PROGRAMAS DE C/C++

- El código en ensamblador debe ir dispuesto del siguiente modo:

```
int main() {  
    ...  
    __asm__ volatile (  
        <CÓDIGO GAS>  
        <SECCIÓN E/S>  
    );  
    ...  
}
```

- <CÓDIGO GAS>: Cada línea debe corresponderse con una instrucción del ensamblador GAS. Además, la línea debe ir entrecomillada y ha de terminar en “\n\t”. Ejemplo:

```
"addl %%ebx, %%eax\n\t"
```

6. ENSAMBLADOR EN LÍNEA DENTRO DE CÓDIGO C/C++

INTEGRACIÓN DE CÓDIGO ENSAMBLADOR GAS EN PROGRAMAS DE C/C++ (continuación)

- <SECCION E/S>: Se compone de tres partes,

- :<salidas> Variables del fuente C/C++ cuyo valor va a ser alterado por el código ensamblador. Cada variable irá encerrada por “()”, y precedida de “=<origen>”, donde <origen> será (entre otras posibilidades) una “m” si la variable procede de memoria, y una “g” si procede de memoria, de un registro de la CPU o es un literal. Si hay más de una variable de salida, hay que emplear una “,” para separar cada bloque de variable. Ejemplo (“resultado” es una variable declarada en C, fuera del bloque GAS):
: "=m" (resultado)
- :<entradas> Variables del fuente C/C++ cuyo valor va a ser leído dentro del código ensamblador. Cada variable irá encerrada por “()”, y precedida de “<origen>”, donde <origen> será (entre otras posibilidades) una “m” si la variable procede de memoria, y una “g” si procede de memoria, de un registro de la CPU o es un literal. Si hay más de una variable de entrada, hay que emplear una “,” para separar cada bloque de variable. Ejemplo (“sumando_1” y “sumando_2” son variables declarada en C):
: "m" (sumando_1) , "m" (sumando_2)

6. ENSAMBLADOR EN LÍNEA DENTRO DE CÓDIGO C/C++

INTEGRACIÓN DE CÓDIGO ENSAMBLADOR GAS EN PROGRAMAS DE C/C++ (continuación)

- `<rec_pres>` Conjunto de recursos preservados (memoria, registros...) que, después del bloque GAS, deben conservar el mismo valor que tenían antes de dicho bloque. Notación: “memory” (memoria), “<registro>” (registro de la CPU), “cc” (registro de estado). **IMPORTANTE:** Las variables de salida son “inmunes”, es decir, NUNCA se preservan (de lo contrario, nunca obtendríamos una salida).

Ejemplo (forzar a que la memoria y los registros “eax” y “ebx” recuperen su valor anterior después de un bloque GAS):

```
: "memory", "%eax", "%ebx"
```

6. ENSAMBLADOR EN LÍNEA DENTRO DE CÓDIGO C/C++

INTEGRACIÓN DE CÓDIGO ENSAMBLADOR GAS EN PROGRAMAS DE C/C++ (continuación)

- Otras consideraciones:
 - Los registros siempre van precedidos de “%%”. Ejemplo: %%eax, %%edi, %%xmm0, etc.
 - Si una variable de salida, en lugar de ir con “=”, va con “+”, se convierte en variable de entrada/salida.
 - Las variables de entrada y las variables de salida son referenciadas dentro del bloque GAS mediante %0, %1, %2..., donde %0 es la primera variable de salida, %1 es la segunda variable de salida, y así hasta llegar a la última variable de salida, %n. A partir de ahí, %n+1 será la primera variable de entrada, %n+2 la segunda, etc.
- Más información: Archivo “asm-gcc-linea.pdf”, en <http://webs.um.es/einiesta/miwiki/doku.php?id=docencia>

6. ENSAMBLADOR EN LÍNEA DENTRO DE CÓDIGO C/C++

6.2 - EJEMPLO: PROGRAMA “suma inline” (código fuente)

```
#include <stdio.h>
int calcular_suma(int sumando_1,int sumando_2);

int main()
{
    int a,b,s;
    printf("Introduzca sumando 1: ");scanf("%d",&a);
    printf("Introduzca sumando 2: ");scanf("%d",&b);
    s=calcular_suma(a,b);
    printf("Resultado de la suma: %d\n",s);
    return(0);
}
int calcular_suma(int sumando_1,int sumando_2)
{
    int resultado;
    __asm__ volatile (
        "movl %1, %%eax\n\t"
        "movl %2, %%ebx\n\t"
        "addl %%ebx, %%eax\n\t"
        "movl %%eax,%0\n\t"
        : "=m" (resultado)
        : "m" (sumando_1), "m" (sumando_2)
        : "memory", "%eax", "%ebx"
    );
    return resultado;
}
```


6. ENSAMBLADOR EN LÍNEA DENTRO DE CÓDIGO C/C++

EJEMPLO: PROGRAMA “suma inline” (compilación, ejecución y depuración)

```
[einiesta@localhost prog_suma]$ gcc -g suma_inline.c -o suma_inline
[einiesta@localhost prog_suma]$ ./suma_inline
Introduzca sumando 1: 7
Introduzca sumando 2: 4
Resultado de la suma: 11
[einiesta@localhost prog_suma]$ gdb ./suma_inline
Intel(R) Debugger for applications running on IA-32, Version 10.0-29 , Build 20070405
object file name: ./suma_inline
Reading symbols from /home/einiesta/am_0708/prac_0708/ejinline/prog_suma/suma_inline...done.
(gdb) b calcular_suma
Breakpoint 1 at 0x80484a6: file suma_inline.c, line 19.
(gdb) r
Starting program: /home/einiesta/am_0708/prac_0708/ejinline/prog_suma/suma_inline
Introduzca sumando 1: 7
Introduzca sumando 2: 4
Breakpoint 1, calcular_suma (sumando_1=7, sumando_2=4) at suma_inline.c:19
19     __asm__ volatile (
(gdb) print $ebx
$1 = 13672436
(gdb) n
30     return resultado;
(gdb) print $ebx
$2 = 4
(gdb) c
Continuing.
Resultado de la suma: 11
```

7. EJEMPLO BÁSICO DE USO DE EXTENSIONES MULTIMEDIA

7.1 - CÓDIGO FUENTE DE “sumasse” (SUMA DE DOS VECTORES CON SSE)

```
/* Ejemplo de suma de dos vectores de 4 floats cada uno (1 float = 32 bits,
IEEE 754 de simple precisión), utilizando extensiones SSE. */

#include <stdio.h>
#include <xmmintrin.h>

// Suma de dos vectores utilizando SSE.
// - Parámetros de entrada: 2 vectores de 4 floats, en formato empaquetado (128 bits).
// - Parámetros de salida: El vector de los 4 floats resultado, en formato empaquetado (128 bits).
__m128 sumar_vectores(__m128 v01, __m128 v02);

// Función principal.

int main(void)
{
// Declaramos punteros a las áreas de memoria en donde se alojarán los vectores
// operandos (v01 y v02) y el vector resultado (v03).
float *v01, *v02, *v03;
// Buffers para guardar los operandos y el resultado en formato empaquetado.
__m128 buf_v01, buf_v02, buf_v03;
```

7. EJEMPLO BÁSICO DE USO DE EXTENSIONES MULTIMEDIA

CÓDIGO FUENTE DE “sumasee” (continuación)

```
// Reserva de la memoria necesaria para las áreas v01, v02 y v03.
// Importante: El valor "16" con el que es instanciado el segundo parámetro
// de "_mm_malloc" nos asegura que cada área de memoria reservada empieza en
// una dirección que sea múltiplo de 16 bytes. Ello optimiza el acceso
// a la memoria, ya que los accesos a memoria realizados por instrucciones
// SSE son accesos alineados a 16 bytes.
v01 = (float *) _mm_malloc(4*sizeof(float),16);
v02 = (float *) _mm_malloc(4*sizeof(float),16);
v03 = (float *) _mm_malloc(4*sizeof(float),16);

// Inicialización de sumandos.
v01[0]=1;v01[1]=2;v01[2]=3;v01[3]=4;
v02[0]=5;v02[1]=6;v02[2]=7;v02[3]=8;

// Introducción de los sumandos en los buffers (la instrucción SSE que
// utilizaremos a continuación, "addps", exige que los operandos estén
// en estructuras "__m128").
buf_v01=_mm_set_ps(v01[3],v01[2],v01[1],v01[0]);
buf_v02=_mm_set_ps(v02[3],v02[2],v02[1],v02[0]);
```

7. EJEMPLO BÁSICO DE USO DE EXTENSIONES MULTIMEDIA

CÓDIGO FUENTE DE “sumasee” (continuación)

```
// Suma de los vectores.
buf_v03=sumar_vectores(buf_v01,buf_v02);

// Desempaquetamiento del resultado.
_mm_store_ps(v03,buf_v03);

// Exhibición de resultados.
printf("%f %f %f %f\n",v03[0],v03[1],v03[2],v03[3]);

// Liberación de la memoria previamente reservada.
_mm_free(v01);
_mm_free(v02);
_mm_free(v03);
return 0;
}
```

7. EJEMPLO BÁSICO DE USO DE EXTENSIONES MULTIMEDIA

CÓDIGO FUENTE DE “sumasee” (continuación)

```
// Suma de dos vectores utilizando SSE.
// - Parámetros de entrada: 2 vectores de 4 floats,
// en formato empaquetado (128 bits).
// - Parámetros de salida: El vector de los 4 floats
// resultado, en formato empaquetado (128 bits).

__m128 sumar_vectores(__m128 v01, __m128 v02)
{
    __m128 resultado=_mm_set_ps(0,0,0,0); // Inicializamos a 0 el resultado.
    // Código inline: Suma vectorial mediante la instrucción SSE "addps"
    __asm__ volatile (
        "movaps %1,%%xmm0\n\t" // xmm0 = v01
        "movaps %2,%%xmm1\n\t" // xmm1 = v02
        "addps %%xmm1,%%xmm0\n\t" // xmm0 = xmm0 + xmm1
        "movaps %%xmm0,%0\n\t" // resultado = xmm0
        : "=m" (resultado)
        : "m" (v01), "m" (v02)
        : "memory"
    );
    return resultado;
}
```

7. EJEMPLO BÁSICO DE USO DE EXTENSIONES MULTIMEDIA

7.2 - EJECUCIÓN DE “sumasee”

```
[aamm@localhost 07_sumasse]$ ls -l
total 4
-rw-rw-r-- 1 aamm aamm 2907 mar 14 14:58 sumasse.c
[aamm@localhost 07_sumasse]$ icc -o sumasse sumasse.c
[aamm@localhost 07_sumasse]$ ls -l
total 28
-rwxrwxr-x 1 aamm aamm 20885 mar 14 14:58 sumasse
-rw-rw-r-- 1 aamm aamm 2907 mar 14 14:58 sumasse.c
[aamm@localhost 07_sumasse]$ ./sumasse
6.000000 8.000000 10.000000 12.000000
```

8. USO DE LAS EXTENSIONES MULTIMEDIA DESDE GCC

8.1 – COMPILACIÓN DE FUENTES CON GCC

- El conjunto de compiladores GCC (“Global Compiler Collection”) constituyen un estándar de desarrollo dentro de la programación bajo Linux (de hecho, suelen venir incorporados en todas las distribuciones).
- De entre todos los compiladores de GCC (“gcc” para C, “g++” para C++, “gcj” para Java, etc) dos de ellos, el “gcc” y el “g++”, pueden representar una alternativa al “Intel icc” y al “Intel icpc”, respectivamente (SI BIEN EN ESTAS PRÁCTICAS SE USARÁ, EN TODO MOMENTO, INTEL C++).
- Los ficheros de cabecera necesarios para compilar con “gcc” o “g++” fuentes que hagan uso de extensiones multimedia (mmintrin.h, xmmmintrin.h, emmintrin.h, etc) se denominan igual que sus homólogos de Intel C++. En el caso de GCC, dichos ficheros de cabecera están (para un Fedora 8) en /usr/lib/gcc/i386-redhat-linux/4.1.2/include.
- Los mismos fuentes que hayan sido compilados con Intel C++ pueden ser recompilados, sin alterarlos, con GCC (comprobado con la versión 4.1.2 de gcc).

8. USO DE LAS EXTENSIONES MULTIMEDIA DESDE GCC

COMPILACIÓN DE FUENTES CON GCC (continuación)

- La compilación con “gcc” (C) o “g++” (C++), cuando el código fuente incluya extensiones multimedia, requiere la participación de un parámetro adicional que establezca cuáles son exactamente las extensiones que se quieren utilizar:
 - -mmmx: Extensiones MMX.
 - -msse: Extensiones SSE.
 - -msse2: Extensiones SSE2.
 - -msse3: Extensiones SSE3.
 - Etc (ver hoja del manual de “gcc” o “g++”).

8. USO DE LAS EXTENSIONES MULTIMEDIA DESDE GCC

8.2 – EJEMPLO DE COMPILACIÓN CON GCC (código fuente `sumasse.c`, ver apartado anterior)

```
[aamm@localhost 08_sumasse_gcc]$ ls ../07_sumasse/*.c
../07_sumasse/sumasse.c
[aamm@localhost 08_sumasse_gcc]$ gcc -msse -o sumasse_gcc ../07_sumasse/sumasse.c
[aamm@localhost 08_sumasse_gcc]$ ls
sumasse_gcc
[aamm@localhost 08_sumasse_gcc]$ ./sumasse_gcc
6.000000 8.000000 10.000000 12.000000
[aamm@localhost 08_sumasse_gcc]$ gcc --version
gcc (GCC) 4.1.2 20070925 (Red Hat 4.1.2-33)
Copyright (C) 2006 Free Software Foundation, Inc.
Esto es software libre; vea el código para las condiciones de copia. NO hay
garantía; ni siquiera para MERCANTIBILIDAD o IDONEIDAD PARA UN PROPÓSITO EN
PARTICULAR
```