Ejecutar Código Ensamblador desde el lenguaje C

Moisés García Villanueva

Mayo de 2014

1. Significado del ensamblado en línea

El ensamblado en línea es una característica de algunos compiladores que permiten escribir código de bajo nivel, como el ensamblador, el cual puede ser incrustado en un lenguaje de alto nivel como el C o Ada. Existen muchos casos donde debemos utilizar en nuestro código (en lenguaje C) un poco de ensamblador, debido a que queremos optimizar algunas lineas un poco más "a mano", o sencillamente necesitamos usar instrucciones las cuales no poseen ningún tipo de función o macro asociada en C(por ejemplo LGDT, LTR, IRET, etc).Es en estos casos en los cuales es necesario recurrir al inline assembly (o ensamblado en línea). Esta herramienta que nos brinda el Gcc nos es muy útil.
Existen al menos tres razones por las cuales es necesario incrustar código en ensamblador en un lenguaje de alto nivel:

1. Optimización
2. Acceso a instrucciones específicas del procesador
3. Llamadas al sistema

2. Formato básico del ensamblado en línea

El formato básico del ensamblado en línea se realiza en forma directa con la instrucción `asm`, es decir basta con incluir en C

```
asm("codigo en ensamblador");
```

Ejemplo 1

En el siguiente ejemplo se realizan 3 operaciones aritméticas utilizando el formato del ensamblado en línea básico. Para observar el contenido de los registros del procesador, se logra utilizando el `gdb`, dentro del cual existe la instrucción `info registers` o `info reg`.

```
// Compilar con la opcion -g: gcc -g Ejemplo01.c
// Entrar al gdb y observar los registro con la instruccion info reg o info registers

#include <stdio.h>

int main() {
```

```
    /* Sumar 10 y 20 y almacenar el resultado en el registro %eax */
    __asm__ ( "movl $10, %eax;"
                "movl $20, %ebx;"
                "addl %ebx, %eax;"
    );

    /* Restar 20 de 30 y almacenar el resultado en el registro %eax */
    __asm__ ( "movl $30, %eax;"
                    "movl $20, %ebx;"
                    "subl %ebx, %eax;"
    );

    /* Multiplicar 10 y 20, alamcenar el resultado en el registro %eax */
    __asm__ ( "movl $10, %eax;"
                    "movl $20, %ebx;"
                    "imull %ebx, %eax;"
    );

    return 0 ;
}
```

2.1. Ensamblado Extendido

En el formato de ensamblado entendido, es posible especificar los operandos. Esto nos permite especificar los registros de entrada, salida y la lista de registro reservados para escribir o almacenar información. el formato general es el siguiente:

```
asm ( "codigo en ensamblador"
           : operandos de salida               /* opcional */
           : operandos de entrada              /* opcional */
           : lista de registros reservados      /* opcional */
);
```

Aún cuando no existan operandos de salida, se deben colocar el caracter de los dos puntos en el lugar que corresponde a los operandos de salida. No es obligatorio especificar la lista de registros reservados o para sobrescribir, se puede dejar al compilador GCC el esquema de optimización necesario.

Ejemplo 2

```
int main(){
    int no = 100, val ;
        asm ("movl %1, %%ebx;"
             "movl %%ebx, %0;"
             : "=r" ( val )        /* output */
             : "r" ( no )         /* input */
             : "%ebx"         /* clobbered register */
         );
}
```

En el ejemplo, ‘‘val’’ es el operando de salida, para referirse a este operando se emplea el %0, mientras que ‘‘no’’ es el operando de entrada y se utiliza %1 para referirse a este operando. ‘‘r’’ es una restricción del operando, significa que el compilador GCC podrá utilizar cualquier registro para almacenar los operandos.

La restricción del operando puede tener un modificador de la restricción, en este caso el signo ‘‘=’’, cuyo significado es que el operando es de salida en el modo de solo escritura. Para poder distinguir entre los operandos y los registros, se emplean dos % como prefijo de los registros, mientras que los operandos contienen solamente un signo de % como prefijo.

En la parte en donde se indica la lista de registros reservados, los registro no necesitan ser especificados con un doble signo de %. Esta lista indica al GCC no utilizar los registros para almacenar algún valor, en el ejemplo se reserva el uso del registro %ebx.

Restricciones de registros

Algunas de las restricciones de registro que pueden emplearse, son la siguientes:

r	Registro(s)
a	%eax, %ax, %al
b	%ebx, %bx, %bl
c	%ecx, %cx, %cl
d	%edx, %dx, %dl
S	%esi, %si
D	%edi, %di

Ejemplo 3

```
 #include <stdio.h>

int main() {

    int arg1, arg2, add, sub, mul, quo, rem ;

    printf( "Enter two integer numbers : " );
    scanf( "%d%d", &arg1, &arg2 );

    /* Perform Addition, Subtraction, Multiplication & Division */
    __asm__ ( "addl %%ebx, %%eax;" : "=a" (add) : "a" (arg1) , "b" (arg2) );
    __asm__ ( "subl %%ebx, %%eax;" : "=a" (sub) : "a" (arg1) , "b" (arg2) );
    __asm__ ( "imull %%ebx, %%eax;" : "=a" (mul) : "a" (arg1) , "b" (arg2) );

    __asm__ ( "movl $0x0, %%edx;"
              "movl %2, %%eax;"
              "movl %3, %%ebx;"
               "idivl %%ebx;" : "=a" (quo), "=d" (rem) : "g" (arg1), "g" (arg2) );

    printf( "%d + %d = %d\n", arg1, arg2, add );
    printf( "%d - %d = %d\n", arg1, arg2, sub );
    printf( "%d * %d = %d\n", arg1, arg2, mul );
    printf( "%d / %d = %d\n", arg1, arg2, quo );
```

```
    printf( "%d %% %d = %d\n", arg1, arg2, rem );

    return 0 ;
}
```

Ejemplo 4

```
 #include <stdio.h>

float sinx( float degree ) {
    float result, two_right_angles = 180.0f ;
    /* Convert angle from degrees to radians and then calculate sin value */
    __asm__ __volatile__ ( "fld %1;"
                           "fld %2;"
                           "fldpi;"
                           "fmul;"
                           "fdiv;"
                           "fsin;"
                           "fstp %0;" : "=m" (result) :
"m"(two_right_angles), "m" (degree)
    ) ;
    return result ;
}

float cosx( float degree ) {
    float result, two_right_angles = 180.0f, radians ;
    /* Convert angle from degrees to radians and then calculate cos value */
    __asm__ __volatile__ ( "fld %1;"
                           "fld %2;"
                           "fldpi;"
                           "fmul;"
                           "fdiv;"
                           "fstp %0;" : "=m" (radians) :
"m"(two_right_angles), "m" (degree)
    ) ;
    __asm__ __volatile__ ( "fld %1;"
                           "fcos;"
                           "fstp %0;" : "=m" (result) : "m" (radians)
    ) ;
    return result ;
}

float square_root( float val ) {
    float result ;
    __asm__ __volatile__ ( "fld %1;"
                           "fsqrt;"
                           "fstp %0;" : "=m" (result) : "m" (val)
    ) ;
    return result ;
}

int main() {
```

```
    float theta ;
    printf( "Enter theta in degrees : " ) ;
    scanf( "%f", &theta ) ;

    printf( "sinx(%f) = %f\n", theta, sinx( theta ) );
    printf( "cosx(%f) = %f\n", theta, cosx( theta ) );
    printf( "square_root(%f) = %f\n", theta, square_root( theta ) ) ;

    return 0 ;
}
```