Ejercicios 5.4, 5.5, 7.1, 7.2 y 7.3

Alumno: Edgar Moya Cáceres

Ejercicio 7.1

Symbol	.symtab entry?	Symbol type	Module where defined	Section
buf	Sí	externo	m.o	.data
bufp0	Sí	global	swap.o	.data
bufp1	Sí	global	swap.o	COMMON
swap	Sí	global	swap.o	.text
temp	No	-	-	-

- Los símbolos *buf*, *bufp0*, *bufp1* y *swap* son entradas de la tabla de símbolos, mientras que *temp* no tiene entrada en la tabla de símbolos ya que constituye una variable local.
- El símbolo *buf* está declarado como externo mientras que *bufp0*, *bufp1* y *swap* son globales.
- Las variables globales y estáticas inicializadas pertenecen a .data, por lo que *buf* y *bufp0* (variables globales inicializadas) pertenecen a esta sección.
- El símbolo *bufp1* pertenece a COMMON por ser una variable global no inicializada.
- La función *swap* pertenece a la sección .text porque es una función.

Ejercicio 7.2

A.

```
/* Module 1 */
int main()
{
}
```

```
/* Module 2 */
int main;
int p2()
{
}
```

El enlazador elige el símbolo fuerte definido en el módulo 1 sobre el símbolo débil definido en el módulo 2, se cumple la regla 2 que plantea que, dado un símbolo fuerte y múltiples símbolos débiles con el mismo nombre, se elige el símbolo fuerte.

(a) REF(main.1) → DEF(main.1)
(b) REF(main.2) → DEF(main.1)

B.

```
/* Module 1 */          /* Module 2 */
void main()             int main = 1;
{                       int p2()
}                       {
                        }
```

Error, porque cada módulo define un símbolo fuerte principal y la regla 1 plantea que no se permiten múltiples símbolos fuertes con el mismo nombre.

C.

```
/* Module 1 */          /* Module 2 */
int x;                  double x = 1.0;
void main()             int p2()
{                       {
}                       }
```

El enlazador elige el símbolo fuerte definido en el módulo 2 sobre el símbolo débil definido en el módulo 1, se cumple la regla 2 que plantea que, dado un símbolo fuerte y múltiples símbolos débiles con el mismo nombre, se elige el símbolo fuerte.

(a) REF(x.1) → DEF(x.2)
(b) REF(x.2) → DEF(x.2)

Ejercicio 7.3

A. gcc p.o libx.a

En este caso el archivo de objeto p.o depende de la biblioteca estática libx.a. Por lo tanto, para resolver las referencias de símbolos en p.o, se necesita incluir la biblioteca libx.a en el comando de enlace y así enlazar correctamente el programa.

B. gcc p.o libx.a liby.a

En este caso el archivo de objeto p.o depende de la biblioteca estática libx.a, y también depende de la biblioteca estática liby.a. Por lo tanto, para enlazar correctamente el programa, se necesita proporcionar al enlazador el archivo de objeto p.o, la biblioteca estática libx.a y la biblioteca estática liby.a.

C. gcc p.o libx.a liby.a libx.a

En este caso, se presenta que libx.a depende de liby.a y, a su vez, liby.a depende de libx.a.. Por lo tanto, necesitamos proporcionar al enlazador el archivo de objeto p.o, la biblioteca estática libx.a, la biblioteca estática liby.a y la biblioteca estática libx.a nuevamente para enlazar correctamente el programa.

Ejercicio 5.4

A. En la versión menos optimizada del código, el registro *%xmm0* se utiliza como un valor temporal en cada iteración del bucle. Esto significa que se utiliza para realizar cálculos intermedios y almacenar valores temporales necesarios para el procesamiento dentro del bucle. En cambio, en la versión optimizada, el registro anterior se utiliza para acumular el producto de los elementos del vector a lo largo de todas las iteraciones del bucle. Esto es similar a tener una variable "*acumulador*" que guarda el resultado parcial en cada paso, lo que permite evitar lecturas y escrituras innecesarias.

B. Las dos versiones de *combine3* tendrán una funcionalidad idéntica, incluso con aliasing de memoria. La optimización realizada no afectará el comportamiento esperado del código. Incluso si hay aliasing de memoria entre *dest* y el vector de datos, el producto acumulado se actualizará correctamente y se almacenará en *dest* al final de cada iteración.

C. Esta transformación puede llevarse a cabo sin alterar cómo funciona el programa. Esto se debe a que, exceptuando la primera iteración, el valor leído desde *dest* al inicio de cada iteración es el mismo valor que fue escrito en ese registro al finalizar la iteración anterior. Por lo tanto, la instrucción de combinación puede simplemente utilizar el valor que ya se encuentra en el registro *%xmm0* al inicio del bucle.

Ejercicio 5.5

A. El código proporcionado realiza dos multiplicaciones y una suma en cada iteración del bucle. Por lo tanto, el número total de adiciones es n, y el número total de multiplicaciones es 2n.

B. En el código proporcionado el cálculo repetido de la expresión *xpwr = x * xpwr* requiere una multiplicación en punto flotante (5 ciclos de reloj), y el cálculo de una iteración no puede comenzar hasta que el cálculo de la iteración anterior haya finalizado. Por otro lado, la operación de suma requerida para actualizar el resultado de la función (línea 7) solo tarda 3 ciclos de reloj en la máquina de referencia. El cálculo que requiere más tiempo y limita el rendimiento del código es la multiplicación en punto flotante realizada para calcular *xpwr*. La suma en punto flotante utilizada para actualizar el resultado requiere menos tiempo en comparación.