Instituto Politécnico Superior “Gral. San Martin”

La arquitectura Intel x86

❖ Integrantes:

Ramiro Gatto, Luciano Duarte, Agustin Fernandez Berge, Alaia Ibarbia, Santiago Federico Agustin Mendez Ioakimidis

❖ Profesor:

Alejandro Rodríguez Costello

En este informe se exponen y detallan diferentes como fue el proceso de elaboración del trabajo de intel x86, que consiste en traducir el trabajo sobre Estructura de datos en assembly del MIPS R2000 a Intel x86.

Preámbulo

Durante este trabajo bajo ningún aspecto se pretende entender la codificación en código máquina del Intel x86, pero si se espera que se den cuenta de la enorme complejidad del mismo.

La idea de este informe es explicar en cierta medida como fue el proceso de planificación del trabajo, el funcionamiento de la funciones expresadas y una breve conclusión sobre Intel x86.

Planificación del trabajo

La metodología del trabajo fue la siguiente, primero se decidió leer el material subido sobre Intel, para poder entender el funcionamiento de los registros y la pila, para no tener ningún problema durante el trabajo.

Luego se propuso tratar de traducir el trabajo de Estructuras de datos, fijándose de qué forma se podría implementar cada función descrita en este trabajo (pasando de MIPS R2000 a Intel x86).

Al ser Intel x86 un lenguaje difícil de entender y ser este un trabajo que escala en dificultad, se optó por tratar de ir transcribiendo función a función, hasta no tenerla terminada se trató de no proceder con el resto.Al final se usó el mismo ejemplo del trabajo de Estructuras para probar el funcionamiento del programa.

Para correr el programa se utilizó la máquina virtual, la cual se encontraba adjunta con el trabajo. Además, también incluía las líneas de código necesarias para poder correrlo.

Funciones de cada Archivo

Funciones del archivo *asm_io.asm y asm_io.inc:*

El propósito de estos archivos es la de proporcionar input/output y debugging para Intel x86, las usamos a lo largo del programa para hacer testing y encontrar errores en el funcionamiento de las rutinas.

Funciones del archivo *main.c:*

El propósito del archivo *main.c* es poder realizar las impresiones tanto del menú para seleccionar una opción como para ver lo ingresado con anterioridad, además también lo usamos para el input.

La mayoría de las funciones expresadas en este archivo son solamente de impresión, **salvo** cinco de ellas, las cuales son muy importante a la hora de ejecutar el programa:

- doinlist: lo que hace es recorrer la lista, aplicando la función pasada por parámetro a cada nodo, el tercer argumento es usado como el segundo argumento de la función pasada.
- delnodeById: se usa en conjunto con la función doinlist para eliminar un nodo en caso de que su id coincida con el id pasado a doinlist.
- updateNodeId: se usa en conjunto con la funcion doinlist para actualizar los id de todos los nodos de una lista de objetos
- getString: reserva 100 bytes de memoria para ser usados como un string y coloca en ellos la cadena de caracteres almacenada en *búfer*. Nota: en buffer se guarda la última cadena de caracteres ingresada por el usuario. Finalmente, devuelve la dirección de la memoria reservada.
- getNodeId: es un atajo rápido para sacar el id de un nodo objeto en base a la rutina getNodeVal definida en *asm_main.asm*.

Al principio de este archivo se definen los prototipos de las funciones que se utilizan en el mismo y las que se usan en el *asm_main.asm*.

Función del archivo *cdecl.h*:

Este archivo define un o más "convenio" los cuales hay que seguir para que el programa entienda la sintaxis y no tire errores.

Funciones del archivo *asm_main.asm:*

aclaración: durante el el uso de las siguiente funciones se utilizan funciones de c, como free o malloc, las cuales no se van a explicar debido a que no es a fina a la materia

- ☐ **prev**: esta función lo que hace es devolver la dirección del siguiente nodo
- ☐ **next**: esta función hace lo mismo que la anterior, solo que devuelve la dirección anterior en vez de la siguiente.
- ☐ **newnode**: en pocas palabras la idea de esta función es poder agregar un nuevo nodo a una lista(ya sea de objetos o de categorías). Para poder lograrlo, la función posee cuatro secciones (una de esas es para "salir" de la función) las cuales dependiendo el caso el programa va a dirigir su flujo hacia ellas.
  - **newnode_empty:** lo que ocurre es que cuando el programa entra por primera vez (no hay ningún nodo) va a esta parte. Ingresa un nodo el cual se apunta a sí mismo dos veces (ida y vuelta).

- **newnode_unique:** el programa salta acá cuando ya hay un elemento en la lista (el programa se da cuenta porque el primer nodo es igual al último).

- **newnode_not_empty:** este sería el último caso, en el cual si hay más de dos elementos agrega un nodo al final.

- **newnode_end**: esto ocurre siempre, no importa en la parte en la que entre, lo que realiza es, actualizar el valor del nodo en memoria y recuperar el frame pointer.

- **delnode**: la función de esta rutina es poder eliminar un nodo. Al igual que la función newnode también posee diferentes secciones (una de las tres es para salir de la función).
  - **delnode_unique:** si hay un solo elemento para eliminar, se elimina y la lista queda vacía.

  - **delnode_not_null**: caso contrario si hay uno o más elementos; se elimina el nodo pedido y se enlazan el nodo previo y el siguiente. Ej:
    - prev(next(node) = prev(node)
    - next(prev(node) = next(node)

  - **delnode_end**: pasa, casi, lo mismo que el newnode_end; actualiza valores y sale

- **getNodeString:** la función de esta rutina es la de obtener la palabra del nodo en cuestión, para así realizar diferentes operaciones con dicho dato conseguido.
  - **getNodeString_end:** sirve para salir de la rutina.

- **getNodeVal:**el rol que cumple esta rutina es la de obtener el valor asociado al nodo(lista de objetos para las categorías, id para los objetos).
  - **getNodeVal_end:** esta etiqueta sirve para salir de la función.

- **getLongitud:** simplemente me devuelve la longitud de una lista.
  - **getLongitud_end**: sirve para salir de la función.

- **secure_free**: esta función lo que hace es ejecutar un free a la hora de eliminar un nodo (si el nodo es NULL, no hace el free).
  - También posee una etiqueta llamada **secure_free_end** para salir de la rutina.

Aclaración Importante: Debido a una serie de problemas que tuvimos con la función secure_free a la hora de liberar la palabra de un nodo al final optamos por no usarla en ese caso, ya que a la hora de ejecutarse la función delnode (función que elimina el nodo) y querer eliminar la palabra asociada usando secure_free (función que realiza el free) tiraba un error relacionado con la pila(esto solo ocurre con el último nodo de una lista).

Lo que concluimos es que este error se debe al uso de una máquina de 32 bits, al probarlo en una de 64 bits no tuvimos este problema.

La solución que optamos para evitar este error fue evitar el free al eliminar un string; el único inconveniente es que cada vez que se elimina un nodo se pierden 100 bytes de capacidad (pérdida prácticamente insignificante).

Prueba final del programa

Vamos a usar el programa para hacer lo siguiente

1. Crear 2 categorías: Mamíferos y Reptiles.
2. En la categoría de Mamíferos vamos a crear 3 objetos: "Perro", "Gato" y "Tigre dientes de sable".
3. En la categoría Reptiles vamos a crear 2 objetos: "Caimán" y "Tortuga".

Creacion de categorias y objetos

Para mayor comodidad, se van a reemplazar todos los mensajes del menú (salvo el primero) con "|**|".

1) Crear una nueva categoría
2) Pasar a la siguiente categoría
3) Pasar a la anterior categoría
4) Mostrar todas las categorías
5) Borrar la categoría seleccionada
6) Anexar un objeto a la categoría seleccionada
7) Borrar un objeto de la categoría seleccionada
8) Mostrar todos los objetos de la categoría seleccionada
9) Salir

Seleccione una opción por su número: 1
Ingrese una palabra : Mamíferos
Categoría en curso : Mamíferos

|**|
Seleccione una opción por su número : 1
Ingrese una palabra : Reptiles
Categoría en curso : Mamíferos

|**|
Seleccione una opción por su número : 4
*Mamíferos
Reptiles
Categoría en curso : Mamíferos

|**|
Seleccione una opción por su número : 6
Ingrese una palabra : Perro
Categoría en curso : Mamíferos

|**|
Seleccione una opción por su número : 6
Ingrese una palabra : Gatto
Categoría en curso : Mamíferos

|**|
Seleccione una opción por su número : 6
Ingrese una palabra : Tigre dientes de sable
Categoría en curso : Mamíferos

|**|

Seleccione una opción por su número : 8
Perro Gatto Tigre dientes de sable
Categoría en curso : Mamíferos

|**|
Seleccione una opción por su número : 2
Categoría en curso : Reptiles

|**|
Seleccione una opción por su número : 6
Ingrese una palabra : Tortuga
Categoría en curso : Reptiles

|**|
Seleccione una opción por su número : 6
Ingrese una palabra : Caimán
Categoría en curso : Reptiles

|**|
Seleccione una opción por su número : 8
Tortuga Caiman

Borrar categorías y objetos

Ahora vamos a probar de borrar lo que creamos.

1. En la categoría Mamíferos, vamos a borrar el 1er y 2do objeto.
2. En la categoría Reptiles, vamos a borrar directamente la categoría
3. Luego vamos a terminar el programa. Categoría en curso : Mamíferos

|**|
Seleccione una opción por su número : 7
Ingrese un numero : 1
Categoría en curso : Mamíferos

|**|
Seleccione una opción por su número : 7
Ingrese un numero : 1
Categoría en curso : Mamíferos

|**|
Seleccione una opción por su número : 8
Tigre dientes de sable
Categoría en curso : Mamíferos

|**|
Seleccione una opción por su número : 2
Categoría en curso : Reptiles

|**|
Seleccione una opción por su número : 5
Categoría en curso : Mamíferos

|**|

Seleccione una opción por su número : 4
*Mamíferos

Conclusión

Luego de realizar el trabajo y meditar un poco acerca de Intel x86, podemos decir, con mucha certeza, que no es un lenguaje fácil de entender. Esto es debido a la compleja forma en la que se programa, tanto por el uso de registros, la pila y las operaciones.

A esto hay que sumarle además el uso de no solo un archivo sino de cinco:

- uno para gestionar el input/output. (*main.c*)
- otro para las convenciones (*cdecl.h*)
- otros dos para entrada y salida en Intel X86 así como debugging(asm_io.inc y asm_io.asm, aunque estos no son usados en el programa final, sirvieron durante la creación del mismo)
- y el último, el cual es donde se escribe el código (*asm_main.asm*)

Pero, debido al conocimiento que adquirimos al hacer este trabajo (sumado con el conseguido gracias a lo visto en MIPS R2000), podemos decir que aprendimos cómo funcionan los lenguajes de bajo nivel.

Además, de las diferencias que puede tener correr un mismo programa en una máquina de 32 bits y 64 bits.