

INFORME DE PROYECTO

Leonardo Dolande

Alejandro De Vincenzo

Universidad Simón Bolívar

Organización del Computador

**Introducción**

Crear un programa a menudo es un proceso bastante complicado y normalmente lleva a cabo múltiples iteraciones de las mimas ideas hasta que finalmente se cristaliza en una solución algorítmica completa.

El objetivo de este trabajo es tratar de explicar, paso a paso, como tal solución llego a crearse y, para facilitar la lectura, se trabajará

**Diseño del Programa**

* La función Main:

Como su nombre indica, es la función principal del programa. Aunque su tamaño es pequeño, su única función es simplemente activar el resto de funciones dentro programa

* Función Impresión.

Es, quizás, la función principal del programa. A través de ella se busca imprimir por pantalla toda la información relevante del programa. Ella esta subdividida en funciones parciales. Cada una encargándose de partes diferentes de la impresión

Estas funciones parciales serían Cabecera, media y Calendario. Como su nombre indica, cada una de estas funciones se encarga de imprimir las partes de la cabecera (minuto, hora, día mes, año), el medio(HH/DD/YY), y el propio calendario.

La forma en que funcionan todas estas funciones de a través de anidamiento de funciones. Impresión llama a Cabecera, Medio y Calendario y cada una de esas funciones llama a funciones que solo imprimirán contenido.

Cabecera Hora

Min

Día

Mes

Año

Impresión Medio Formato

Calendario Imprime calendario

La mayoría del programa es construido con esta forma de pensar. Si una función es demasiado compleja, se subdividirá es funciones más pequeñas que resolverán todo de forma más simple.

Sin embargo, esta aparente simpleza trae también su propio conjunto de retos. Especialmente por el hecho de ser un programa escrito en Assembly

* Pila de “Funciones”

Dada la naturaleza anidada del programa, esto es, llamar a funciones dentro de funciones, y dado a que Assembly no posee funciones propiamente dichas, si no etiquetas para secciones de código, surge un problema.

La forma más común de hacer un “llamado a función” es través del comando **jal** and **jr.** Ambos comandos nos permiten saltar a una parte del código, guardar en un registro $RA la dirección del código de dónde provino y volver de dónde provino una vez la sección de código terminó.

Sin embargo, esta solución presenta un problema una vez se empiezan a hacer llamadas a funciones dentro de funciones. El problema es que el registro $RA se actualiza con cada llamada y eso es un problema crítico si se desea mantener el “camino de vuelta”

La solución para este problema es tan simple como crear 3 funciones auxiliares: GuardarPila, UsarPila, EliminarPila.

Estas funciones se utilizarán para guardar las direcciones de las funciones en pila de memoria, para su posterior uso. Después se eliminará la dirección de la función de ser necesaria.

**Dificultades Encontradas**

La principal dificultad del programa fue mantener orden entre las funciones. El problema simplemente era que existían muchas funciones y se tenía que tener un control de que registros usaban, si se estaban guardando o no en la pila, si se eliminó de la pila entre muchos otros problemas que surgen del manejo de tantas funciones.

También otro problema que surgió fue como hacer que las funciones set trabajan en concordancia con la función de impresión. En especial como set agregaba corchetes para señalizar que elemento del programa está a punto de ser cambiado

**Conclusión**

Sin duda alguna, trabajar en un programa de Assembly es mucho más complicado que trabajar en un lenguaje de alto nivel como Python. Más de una vez se echó en falta diferentes comandos de lenguajes de alto nivel y la inexistencia de una implementación funciones hace que manejar diferentes secciones de código se haga más complicado

Sin embargo, trabajar en Assembly da cierto nivel de control que no se obtiene en otro tipo de lenguajes. Conocer que está pasando es mucho más sencillo pues, al estar obligado a trabajar tan cerca de la máquina, la cantidad de errores se suele reducir a acceder memoria que no se puede usar, un mal nombramiento de una etiqueta o usar un registro incorrecto.

En Assembly nada está oculto. Si algo falla, no va a ser por culpa de alguna librería problemática o un comando mal implementado y eso es de apreciar