# CLASE 10/05/24

A tener en cuenta para el parcial:

* Habrá ejercicios de Hamming
* En su mayoría tratará de ensamblador.

Temas vistos hoy en clase:

Assembler ARM instrucciones (PDF en Moodle).

Programas en Assembler:

Básicamente todo programa en ensamblador (y la mayoría de los programas) tiene almenos 4 secciones, hay una seccion que vamos a llamar .text (código), otra llamada .data (donde se guardan datos, constantes, espacio a reservar), otra llamada .heap que es la sección de memoria dinámica, y la ultima se llama .stack que es la de la pila. En la cursada solo nos centraremos en usar .text y .data.

Un programa y un proceso no es lo mismo. Un proceso es un programa en ejecución, y un programa es un archivo de código a ejecutar. Los programas tienen un proceso de carga de datos desde el disco duro a la memoria principal.

Las variables declaradas en un código son parte de la memoria estática, y reservan su espacio en memoria en tiempo de compilación, para usarse recién en tiempo de ejecución.

Int a = 0;

String b = “ “;

La memoria dinámica en cambio permite trabajar sin necesidad de reservar el espacio en memoria y se pueden cambiar los parámetros en tiempo de ejecución. Un ejemplo de esto es cuando trabajamos con listas de objetos en programación, las listas no tienen un tamaño fijo, se les puede agregar o restar objetos en cualquier momento.

Sin embargo no hay que confundirse, ambos tipo de memoria están en la memoria RAM.

**Directiva de control:**

/\* Comentario multilínea; nombre del programa y autor\*/

.data //Directiva de control: delimita la sección de datos

/\*---------------Definición de datos---------------\*/

Aquí se definen los datos a utilizar

/\*--------------------------------------------------------\*/

.text //Directiva de control: delimita sección de código

.global main //es visible en todo el programa

Main: //es solo una etiqueta, podría ser otra

Nop //esta instrucción no hace nada, consume 1 ciclo de reloj

Mov r7,#1 //Se carga en R7 el syscall para swi, si R7=1 swi sabe que debe salir al SO

Swi 0 //se usa para avisar al sistema operativo que el programa terminó.

Ej:

.data

/\*----------- Definición de datos: Directivas de asignación ---------------------------------------- \*/

a1: .byte 1 /\* tipo byte, se inicializa en 1 si le agrego .align rellena \*/

var2: .byte 'A' /\* tipo byte, caracter A \*/

var3: .hword 25000 /\* tipo hword 16 bits a 25000 \*/

var4: .word 0xA2345678 /\* tipo word de 32 bits \*/

b1: .ascii "hola que tal orga" /\* definición de cadena \*/

b2: .asciz "mundo cruel" /\* definicion de cadena que termina con null \*/

dato1: .zero 300 /\* 300 bytes de valor cero \*/

dato2: .space 200, 4 /\* 200 bytes del valor 4 \*/

.equ nada, 4 /\* definimos constante nada con valor 4 tambien se puede valores negativos \*/

.equ dividido, nada/2 /\* dividido=2

en el .Data podemos declarar un .byte, un .word, o lo que queramos.

En ensamblador no hay una discriminación de tipos, en .data podemos guardar lo que sea, tendremos 32 bits siempre pero nosotros administraremos como los utilizamos y cuanto espacio usamos. Por ejemplo, si queremos guardar en .word un número negativo, podemos hacerlo. Por lo tanto no debemos prestarle tanta atención a los rangos, sino a la cantidad de bits que tenemos disponible. Es la instrucción que usamos la que va a definir dato.

Además de eso, en la arquitectura en la que trabajaremos tendremos registros de 32 bits. Por lo tanto todas las instrucciones ocupan 32 bits.

En assembler no llamamos variables, sino etiquetas, la etiqueta A1 no es el dato, sino la dirección de memoria en la que está guardado el dato.

Cada cadena que creemos hay que ponerle cuanto espacio queremos reservar con un tipo de dato con un tamaño acorde al máximo tamaño que va a tener esa etiqueta. (.byte, asciz, .word, etc).

Usar siempre asciz para cadenas, por conveniencia y porque es más sencillo de utilizar (es el método de representación C).

En ensamblador, para escribir en hexadecimal hay que escribir **0xnumero**, Para binario se usa **0bnumero**, de otra manera, si no ponemos nada, assembler lo interpretará como decimal.

.zero 300 inicializa en 0 un espacio en memoria de 300 bytes.

.space 300,4 inicializa 300 bytes pero en la posición 4.

Las constantes no ocupan espacio en memoria, ocupan en el sector de código.

**Instrucciones de transferencia de datos:**

Hay instrucciones que permiten copiar datos de registro a memoria y viceversa. Un ejemplo de esto es MOV.

Mov r0, r1 ///

Mov r0, #2 /// usar esta notación para valores chicos.

Instrucción/tipo de dato/dato

Para valores grandes podemos usar la etiqueta A1.

Cuando estamos ejecutando nuestro programa, este tiene el control del procesador en ese momento

**Instrucciones aritméticas:**

La instrucción de suma usa 2 o 3 operadores

Hacer r1,#numero es igual a hacer r1 = numero.

Ejemplo:

add r1, #1 (suma 1 a r1)

adds r2, r5, r1 (suma r5 y r1, y se guarda el resultado en r2 pisando su valor, no agregándose)

Lo mismo pasa con la resta:

Sub r1, #1(resta 1 a r1)

Sub r1, r3, r4 (resta r3 a r4 y se guarda el resultado en r1 y pisa su valor)

Como vemos, no solo podemos operar con registros, sino con valores constantes (como #1)

**Raspberrypi**

Como conectarse al Raspberrypi:

Una forma es usando el comando: ssh fernandowiki.educti.ar -p9100 en Windows powershell o en la consola de Linux. (el correo de usuario y -p9100 es el puerto) luego debemos escribir la contraseña.

Datos para conectarse:

usuario: occ2gX (La X es el numero de grupo)

contraseña previsoria: plm198

**Comandos de Assembler:**

ls: muestra las carpetas disponibles en el repositorio del grupo.

nano: edita un archivo (nano nombreArchivo.asm -l (el -l muestra el numero de líneas)) para salir del nano se usa CTRL + X o CTRL + con las letras indicadas en el menú para realizar una acción.

As: es el comando para compilar (as nombreArchivo.asm -o nombreArchivo.o -g)

El terminado en .asm es el archivo donde está el código, y el terminado en .o es el archivo de salida. Esto lo que va a hacer es transformar el código en un objeto para compilarlo. Si damos enter y no dice nada, significa que no hay errores, si tira error es porque algo está mal en el código.

Gcc: genera un archivo que no tiene extensión (un archivo binario) que es el equivalente a un ejecutable en Windows (un .exe) (gcc nombreArchivo.o – o nombreArchivo)

La ultima parte de la compilación es terminar de traducir el objeto generado, y generar el linkeo.

Gdb: debuguea el ejecutable del programa (gdb nombreArchivo), una vez ingresado escribir “start” y apretar enter. Una vez hecho eso se abrirá el debugger mostrándonos las operaciones y datos del programa. Luego podemos pedir que nos muestre lo que hay en un registro usando dos letras i (de información) r (de registro) r(registro deseado a consultar). Cualquiera sea la operación realizada nos mostrará el resultado en hexadecimal, si tiene un valor raro es porque es basura. Para salir del debugger ingresamos “Quit” y podremos seguir editando nuestros archivos.

Para leer un byte hay que usar la instrucción ldr primero, y luego ldrb. Esto se hace para que solamente nos lea el byte.

Cat: copia todo el contenido de un archivo (cat archivo.asm, seleccionamos el código a copiar y apretamos enter)

./nombreArchivoBinario: ejecuta el archivo binario del programa.

Pasos para compilar y ejecutar un programa:

1. Escribir el código fuente en el archivo .asm.
2. Ejecutar el comando “as” para compilar y generar el archivo .o .
3. Ejecutar el comando “gcc” para crear el archivo binario.
4. Ejecutar el comando “./nombreArchivoBinario” para ejecutar el programa.

**Consejos:**

1. Si presionamos la flecha hacia arriba traeremos un comando que ya hicimos antes.
2. Si escribimos el nombre de un archivo y apretamos tabulación nos va a autocompletar el nombre.
3. Si apretamos un comando y apretamos ctrl+R nos va a traer el último comando de ese tipo que llamamos.

**Notas:**

1. En la wiki está la información necesaria sobre ensamblador. Además, hay un video en youtube de 40 minutos explicando lo que usamos en clase.
2. El ejercicio de algebra de Boole del parcial está disponible en el telegram de la comisión.
3. El tipo de dato hay que verlo como la cantidad de espacio que usamos.
4. Nombrar los archivos en .asm (nombreArchivo.asm).
5. No usar caracteres especiales para nombrar los archivos.
6. En ensamblador los comentarios son fundamentales para no perderse con el código. No escatimar en el uso de esta herramienta ya que es extremadamente fácil perderse y, además, que todos los integrantes del grupo entiendan los cambios que se van haciendo.