Introducción a la Programación en Lenguaje Ensamblador AT&T x86-32: sum1toN

HISTORIAL DE REVISIONES

NÚMERO	FECHA	MODIFICACIONES	NOMBRE
v1.0.0	2017 Septiembre 24		CA

Índice

1. Introducción

1.1. Objetivos

- Introducción a la **programación de bajo nivel** mediante los lenguajes *C* y *ensamblador AT&T* para la arquitectura **x86 de 32 bits** de Intel.
- Utilización de herramientas de desarrollo de sw de bajo nivel como el toolchain (compilador, ensamblador, linker) y el depurador GDB, libres de la fundación GNU.
- Desarrollo de una workstation mediante la instalación de herramientas de desarrollo en un entorno GNU/linux/x86 en una computadora personal.
- Comprender el funcionamiento de la computadora desde el punto de vista de un programador de bajo nivel.
- Relacionar las propiedades de un lenguaje de alto nivel con un lenguaje de bajo nivel.
- El lenguaje ensamblador puede ser utilizado como una herramienta para analizar la arquitectura y el funcionamiento de la computadora. No es el objetivo central de esta asignatura ser un experto en lenguajes de programación de bajo nivel ni en el desarrollo de algoritmos, aunque sí un nivel muy básico.

1.2. Requisitos

1.2.1. Teóricos

- Conocimientos muy básicos de una arquitectura ISA: Arquitectura modelo Von Neumann(Microarquitectura CPU, arquitectura Memoria Principal, ciclo de instrucción), representación de datos y operaciones aritméticas, formato de instrucciones y programación básica en lenguaje ensamblador y lenguaje máquina. Por ejemplo la máquina IAS de John von Neumann.
- Haber desarrollado un progama sencillo en lenguaje ensamblador : Por ejemplo de la máquina IAS de John von Neumann.

1.2.2. Prácticos

- Haber emulado o ejecutador
- Tener configurada la "Plataforma de Desarrollo" GNU/linux(AMD64)/x86_64(Intel ó AMD) con las herramientas apropiadas.
- Conocimientos de programación imperativa en lenguaje C y del "Lenguaje de Transferencia entre Registros" RTL, manejo básico del entorno GNU/linux y una herramienta de edición.
- Estar dado de alta en el sitio de la asignatura en servidor miaulario.

2. LEEME

- Lectura del guión de prácticas y de los capítulos 1 y 2 del Libro Programming from the Ground-Up.
- Apuntes y Libro de Texto
- Documentación Memoria: Contenido y Formato de la Memoria
- Evaluación: sistema de evaluación
- Programación : metodología

3. Estación de Trabajo

- Anotar las características de la Plataforma de Desarrollo en el Documento Memoria.

4. Programación sum1toN.c

4.1. Algoritmo

- Desarrollar un programa en lenguaje C que realice la suma $\sum_{i=1}^{N} i$ cuyo resultado es $N(N+1)/2$ con la arquitectura *i386* utilizando el método de programación de descripción inicial en lenguaje pseudocódigo y organigrama.

4.2. Módulo fuente

- Cabecera con información complementaria:

```
/*
 Programa: sum1toN.c
 Descripción: realiza la suma de la serie 1,2,3,...N
 Es el programa en lenguaje C equivalente a sum1toN.ias de la máquina IAS de von Neumann
Lenguaje:       C99
Descripción:    Suma de los primeros 5 números naturales
Entrada:        Definida en una variable
Salida:         Sin salida
Compilación:    gcc -m32 -g -o sum1toN sum1toN.c
S.O:            GNU/linux 3.2.0 ubuntu 12.04 x86-64
Librería:       /usr/lib/x86_64-linux-gnu/libc.so
CPU:            Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz
Compilador:     gcc version 4.6.4 (Ubuntu/Linaro 4.6.4-1ubuntu1~12.04)
Ensamblador:    GNU assembler version 2.22
Linker/Loader:  GNU ld (GNU Binutils for Ubuntu) 2.22
Asignatura:     Estructura de Computadores
Fecha:          25/09/2018
Autor:          Cándido Aramburu
*/
```

- Programa:

```
// Módulo Principal
void  main (void) {
  //Declaración de variables locales e inicialización de los parámetros del bucle
  int sum=0,n=5;
  //Bucle que genera los sumandos y realiza la suma
  while(n>0){   //Condición de salida del bucle cuando el sumando es negativo
    sum+=n;
    n--;        //Actualización del sumando
  }
}               //Si las instrucciones se han ejecutado sin interrupción ni fallo main()  ↩
    devuelve el valor cero al sistema operativo.
```

4.3. Desarrollo del programa

4.3.1. Edición del módulo fuente

- Editar el programa descargando el módulo fuente "sum1toN.c de miaulario y añadiendo los comentarios apropiados.

4.3.2. Compilación

- Compilación
  - `gcc -m32 -o sum1toN sum1toN.c`

  - *m32* : 32 bits architecture machine
  - *sum1toN.c* : módulo fuente en lenguaje C
  - *-o* : output
  - *sum1toN* sin extensión: módulo objeto ejecutable aunque sería más preciso decir cargable en la memoria principal.
- carga en memoria principal
  - la hace automáticamente el S.O. al llamar al programa ejecutable desde un terminal o escritorio.
- `gcc -m32 -g -o sum1toN sum1toN.c`
  - *-g*: especifica que se genere la tabla de símbolos del programa fuente *sum1toN.c* para el debuger GDB y se inserte en el módulo ejecutable *sum1toN* . De está manera se asocian el código binario, por ejemplo de una etiqueta, a su símbolo (lenguaje texto).

Fases de compilación

```
* Fases de la compilación:
** Parar la compilación en la 1ª fase: preprocesamiento: +gcc -E sum1toN.c -o sum1toN.i+
*** '*.i': Salida del preprocesador: elimina la información que no es código (comentarios, ↩
    etc)
** Parar la compilación en la 2ª fase: traducir C a ensamblador: +gcc -S sum1toN.c -o  ↩
    sum1toN.s+
*** '*.s': módulo en lenguaje fuente ensamblador '*.s'
** Parar la compilación en la 3ª fase: Generar el módulo objeto reubicable: +gcc -c sum1toN ↩
    .c -o sum1toN.o+
*** '*.o': módulo objeto reubicable : código binario antes de ser enlazado mediante el  ↩
    linker con otros módulos objeto del sistema operativo, de la librería de C 'libc' u  ↩
    otros módulos del programador.
** Realizar las 4 fases : Generar el módulo objeto ejecutable: +gcc -c sum1toN.c -o sum1toN ↩
    +
*** fichero sin extensión: módulo objeto ejecutable: módulo binario configurado para ser  ↩
    cargado en la memoria principal y ejecutado por la CPU.
* +gcc -m32 --save-temps -o sum1toN sum1toN.c+
** '--save-temps': gcc genera (save) los 3 ficheros parciales (temps) del proceso total de  ↩
    compilación '*.i','*.s','*.o' .
** Comprobar que en total disponemos de 5 ficheros: '*.c','*.i','*.s','*.o' y el ejecutable ↩
     sin extensión.

Toolchain
```

- Cómo alternativa a realizar la compilación mediante un único comando con el driver **gcc** que ejecuta las distintas fases de compilación el proceso de compilación de puede realizar mediante el encadenamiento de herramientas que realizan cada una de ellas una de las distintas fases.
- Herramientas del toolchain:
  - Traducción de C a Ensamblador: no tiene una herramienta propia: `gcc -S sum1toN.c -o sum1toN.s`
  - **as**: Herramienta de Ensamblaje o ensamblador: `as -32 -gstabs -o sum1toN.o sum1toN.s`
    - *-32*: arquitectura de 32 bits
    - *-gstabs*: genera la tabla de símbolos
    - *-o* : fichero de salida : módulo binario reubicable **.o*
  - **ld**: Herramienta de Enlazado ó Lincado: `ld -melf_i386 -o sum1toN sum1toN.o`
    - *melf_i386* : arquitectura 32 bits
    - *-o*: fichero de salida : módulo binario ejecutable

4.4. Análisis

- Análisis para comprobar los distintos módulos:
  - `file sum1toN.c`
  - `file sum1toN.i`
  - `file sum1toN.s`
  - `file sum1toN.o`
  - `file sum1toN`

4.5. Ejecución

- `./sum1toN` : llamada al módulo binario ejecutable
- `echo $?` : visualización del valor devuelto por el programa *sum1toN* al sistema operativo linux.
  - El valor cero se utiliza como indicador de que el programa se ha ejecutado sin ningún tipo de contratiempo.

4.6. Depuración

4.6.1. Inicio

- La ejecución del programa paso a paso, instrucción a instrucción, permite un análisis minucioso de bajo nivel de la ejecución del programa pudiendo detener el programa y volcar el valor de las variables en la memoria principal, estado de los registros de la cpu, etc.
- Depurador GDB: GNU DeBugger.
- `gcc -m32 -g -o sum1toN sum1toN.c` : Inserta la "Tabla de Símbolos" en el módulo binario ejecutable.
- En la línea de comandos emplear el TABULADOR TAB para Completar los nombres: `gcc -m32 -g -o sum1TAB suTAB`
- Abrir el depurador: `gdb`
  - Ventana: línea de comandos propios del debugger.
  - Logging: Entradas → `set trace-commands on`, Salidas → `set logging file sum1toN_gdb_c.txt` , `set logging on`

4.6.2. Comandos linux

- `shell ls -l sum1toN_gdb_c.txt`
  - `shell date`
  - `shell pwd`
  - `shell ls`
  - `shTAB` : Emplear el TABULADOR TAB para Completar los nombres.
  - `shell daTAB`

4.6.3. Ventanas

- Layout: `C-x a` → por defecto dos ventanas: módulo fuente y línea de comandos.
- Navegador ventanas: `C-x o`
- Navegar por el histórico de comandos
  - Logging histórico de comandos: Activar la ventana de comandos del GDB. Navegar con las teclas flecha arriba/abajo.

4.6.4. Ayuda

- `help shell` ó `h shell`

4.6.5. Cargar módulo objeto ejecutable

- Cargar el módulo objeto binario que contiene la Tabla de Símbolos: `file sum1toN`
- módulo fuente con los símbolos asociados a la Tabla de Símbolos: `info sources`

atención
Observar que el depurador confirma la existencia de la tabla de símbolos, imprescindible para la depuración.

4.6.6. Ejecución paso a paso

- punto ruptura en la entrada al programa: `break main`
- Ejecución: `run` hasta el punto de ruptura cuya línea NO se ejecuta → Aparece el código fuente.
- Next source line: `next`, `n`
- Next 5 source lines: `n 5`, `print sum`, `p sum`
- Comenzar desde el principio nuevamente: `run` ó `start`
- Continuar hasta el próximo punto de ruptura: `continue`, `c`
- `run`, `n`, RETURN, RET, RET, `p sum`

4.6.7. Bucle

- ¿cómo salir de un bucle de cientos o miles de iteracciones hasta la siguiente instrucción fuera del bucle?
- `run`, `until`, RET,RET,RET..hasta salir del bucle.., `p sum`, `c`

4.6.8. Análisis de la memoria

- imprimir el contenido de variables y sus direcciones en la memoria principal
- `print n`, `p n`, `p /t n`, `p /x n`, `ptype n`, `whatis n`,`p &n`
- `print symbol` : symbol es el nombre de la variable, no su dirección.
- `p $eax`
- `p $ebx`
- `p $ecx`
- `info registers`

4.6.9. Desensamblar

- Desensamblar: ingeniería inversa . Convierte el código binario en código ensamblador
  - `layout split`
  - Next machine instruction: `ni`, RET, RET, RET, RET, `until`, RET,..hasta salir del bucle
    - Ejecuta instrucciones máquina (observar ventana con el código ensamblador)

4.6.10. Salir

- `exit`

atención
Comprobar que el contenido del fichero *sum1toN_gdb_C.txt* es correcto.

4.7. Documento Memoria

- En la consola abrimos el fichero `gedit sum1toN_gdb_C.txt` que contiene todos los comandos utilizados con sus volcados.
- Guardar el contenido de *sum1toN_gdb_C.txt* en el Documento Memoria añadiendo los comentarios necesarios.
- Cambios en el Módulo fuente en lenguaje C.
  - Cambiar el tamaño de los datos con alguno de los siguientes tipos:
    - char,short,int,long
  - Cambiar el formato de los números con alguno de los siguientes bases:
    - decimal, hexadecimal, octal, binario → prefijos 0x, 0, 0b → 0x5, 05, 0b5
  - Compilar y ejecutarlos. Indicar en la memoria si da o no algún error
- GDB
  - Cambiar en el módulo fuente el tamaño de las variables a *char* y la sentencia sum+=n por la sentencia sum-=n .
  - Compilar el módulo fuente con la opción de inserción de la tabla de símbolos
  - Abrir el depurador y cargar el módulo binario
  - Ejecutar en modo paso a paso observando las sumas parciales con lo siguientes comandos:
    - `x /1db &sum`, `x /1tb &sum`, `x /1ob &sum`, `x /1xb &sum`
    - indicar para comando el resultado
    - Con la ayuda de `help x` explica el significado de `1db`, `1tb`, `1ob`, `1xb`.

5. Programación sum1toN.s

5.1. Algoritmo

- Desarrollar un programa en lenguaje ensamblador AT&T con la arquitectura *i386* que realice la suma $\sum_{i=1}^{N} i$ cuyo resultado es $N(N+1)/2$ utilizando el método de programación de descripción inicial en lenguaje pseudocódigo y organigrama.

5.2. Módulo fuente: sum1toN.s

- x86 es la arquitectura de Intel de 32 bits
- i386 significa en linux: arquitectura x86-32
- Lenguaje ensamblador AT&T de GNU para la arquitectura i386 → lenguaje GNU as → lenguaje gas

```
### Programa: sum1toN.s
### Descripción: realiza la suma de la serie 1,2,3,...N. La entrada se define en el propio ↩
     programa y la salida se pasa al S.O.
### Lenguaje: Lenguaje ensamblador de GNU para la arquitectura i386 -> GNU as -> gas -> AT ↩
    &T
### Es el programa en lenguaje AT&T i386 equivalente a sum.ias de la máquina IAS de von  ↩
    Neumann
### gcc -m32 -g -nostartfiles -o sum1toN  sum1toN.s
### Ensamblaje as --32 --gstabs sum1toN.s -o sum1toN.o
### linker -> ld -melf_i386   -o sum1toN sum1toN.o

        ##  Declaración de variables
        ##  SECCION DE DATOS
        .section .data

n:      .int 5

        .global _start

        ##  Comienzo del código
        ##  SECCION DE INSTRUCCIONES
        .section .text
_start:
        mov $0,%ecx # ECX implementa la variable suma
        mov n,%edx
bucle:
        add %edx,%ecx
        sub $1,%edx
        jnz bucle

        mov %ecx, %ebx # el argumento de salida al S.O. a través de EBX según convenio

        ## salida
        mov $1, %eax  # código de la llamada al sistema operativo: subrutina exit
        int $0x80     # llamada al sistema operativo para que ejecute la subrutina según  ↩
            el valor de EAX


        .end
```

5.3. Análisis del módulo Fuente

- Sección de datos
  - Directiva `.section .data`
  - Etiqueta n:
  - Directiva `.int`
  - Literal 5
- Sección de instrucciones
  - Directiva `.section .text`
  - *_start:* Punto de entrada al programa.
  - Directiva .global :
    - La etiqueta *_start* tiene que ser "visible" fuera del programa sum1toN para que el linker la enlace con el sistema operativo linux como punto de entrada.

- Instrucciones: sintaxis : etiqueta: operación operando_fuente,operando_destino #comentario
- Mnemónicos de las operaciones: mov,add,sub,jnz(jump non zero),int(interruption call)
- Direccionamientos
  - Inmediato: prefijo **$** : $0
  - Registro: prefijo **%** : %ecx
  - Registros eax,ebx,ecx,edx,edi,esi: 32 bits

- Directiva .end : fin del ensamblaje

5.4. Compilación

- Comentar el programa fuente de manera abstracta funcional/operativa y no literal RTL
- Toolchain manual:
  - `as --32 -gstabs -o sum1toN.o sum1toN.s` : ensamblaje
    - **.s* : módulo fuente en lenguaje asm
    - **.o* : módulo objeto reubicalble
    - *gstabs*: generación de la tabla de símbolos e inserción en el módulo ejecutable.
    - *32* : módulos fuente y objeto para la ISA de 32 bits
  - `ld -melf_i386 -o sum1toN sum1toN.o`
    - *melf_i386*: módulos objeto para la ISA de 32 bits
- Toolchain automático
  - `gcc -m32 -nostartfiles -g -o sum1toN sum1toN.s`
    - *m32*: módulos fuente y objeto para la arquitectura i386.
    - *nostartfiles* : especifica que el punto de entrada no es main sino *_start*.

---

nota
Si el punto de entrada es **main** entonces el compilador gcc no necesita ninguna especificación pero si utilizamos el linker ld si necesitamos especificarlo mediante la opción -entry `ld -melf_i386 -e main -o sum1toN sum1toN.o`

- *g*: especifica que se genere la tabla de símbolos del programa fuente *sum1toN.s* para el debuger GDB y se se inserte en el módulo ejecutable *sum1toN*

5.5. Depuración

- Depurador GDB: GNU DeBugger.
- `gcc -m32 -g -o sum1toN sum1toN.s`
- `gdb`
- `C-x a`
- `C-x o`
- `set trace-commands on`
- `set logging file sum1toN_gdb_asm.txt`
- `set logging on`

- `shell ls -l sum1toN_gdb_asm.txt`
- `file sum1toN`
- `info sources`
- `b _start`
- `run`
- `ptype n`
- `p n`
  - `x address` : examine main memory address . Devuelve el contenido de la dirección de memoria address → *&symbol* donde symbol es el nombre de la variable.
- `x &n`, `x n`, `x /1bw &n`, `x /1xw &n`, `x /4xw &n`
- `b bucle`
- `c`
- `start`
- `c`
- `n`
- `p $ecx`
- `p $edx`
- `until`
- `p $ecx`
- `p $edx`
- `info registers`
- `layout split`
- `start`
- `c`
- `exit`
- En la consola abrimos el fichero `gedit sum1toN_gdb_asm.txt` que contiene todos los comandos utilizados con sus volcados.
- Guardar el contenido de *sum1toN_gdb_asm.txt* en el Documento Memoria añadiendo los comentarios necesarios.