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

Índice:

* sum1toN.c:
  + análisis de los módulos
  + ejecutar
  + GDB
* sum1toN.c (modificado):
  + GDB
* sum1toN.c:
  + Ejecutar
  + GDB
* Autoevaluación:
  + Cuestiones teóricas
  + Cuestiones prácticas

sum1toN.c:

/\*

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 --save-temps -g -o sum1toN sum1toN.c

S.O: GNU/linux 4.10 ubuntu 17.04 x86-64

Librería: /usr/lib/x86\_64-linux-gnu/libc.so

CPU: Linux alvaro12 5.4.0-48-generic #52-Ubuntu SMP Thu Sep 10 10:58:49 UTC 2020 x86\_64 x86\_64 x86\_64 GNU/Linux

Compilador: gcc (Ubuntu 9.3.0-10ubuntu2) 9.3.0

Ensamblador: Ensamblador (GNU Binutils for Ubuntu) 2.34 de GNU

Linker/Loader: GNU ld (GNU Binutils for Ubuntu) 2.34

Asignatura: Estructura de Computadores

Fecha: 07/10/2020

Autor: Alvaro Larraya

\*/

void main (void)

{

int sum=0,n=5; //Declaración de variables locales

//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

*Compilar:*

gcc --save-temps -g -o sum1toN sum1toN.c

Nos proporciona sum1toN.s, sum1toN.i, sum1toN.o y sum1toN (con la Tabla de Símbolos)

## *Análisis de los módulos:*

**COMANDO**  **SALIDA**

file sum1toN.c sum1toN.c: UTF-8 Unicode text

file sum1toN.i sum1toN.i: ASCII text

file sum1toN.s sum1toN.s: assembler source, ASCII text

file sum1toN.o sum1toN.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV),

with debug\_info, not stripped

file sum1toN sum1toN: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),

dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,

BuildID[sha1]=08f218df9ddcbb205a56cfcb4505cbca23368608, for

GNU/Linux 3.2.0, with debug\_info, not stripped

## *Ejecutar:*

Con ./sum1toN ejecutamos el programa, el cual devuelve un 0 al sistema operativo, si todo se ha ejecutado sin problemas. Realizamos un echo $? justo después de ejecutar el programa, este comando nos devuelve un 0, por lo que sum1toN ha corrido correctamente.

## *GDB:*

Abrimos el debugger con el comando gdb. Ponemos a escribir en un fichero todos los comando que realicemos ejecutando (si vamos a hacer un layout split, mejor hacerlo antes de poner a guardar los pasos del gdb, ya que si lo hacemos a mitad de la recopilación da problemas):

* set trace-commands on
* set logging file bmp\_lados\_gdb\_c.txt
* set logging on

Las instrucciones que hemos utilizado son las siguientes (historial de gdb):

+shell date

+shell pwd

+shell ls

+help shell

Ejecute el resto de la línea como una orden del intérprete.

Sin argumentos, ejecute un intérprete inferior.

+file sum1toN

Reading symbols from sum1toN...

aviso: El archivo fuente es más reciente que el ejecutable.

+info sources

Source files for which symbols have been read in:

/home/ubuntu20/Escritorio/sum1toN.c

Source files for which symbols will be read in on demand:

+b main

Punto de interrupción 1 at 0x1129: file sum1toN.c, line 22.

+r

Starting program: /home/ubuntu20/Escritorio/sum1toN

Breakpoint 1, main () at sum1toN.c:22

+n

+n 5

+print sum

$1 = -5 '\373'

+r

Starting program: /home/ubuntu20/Escritorio/sum1toN

Breakpoint 1, main () at sum1toN.c:22

+c

Continuando.

[Inferior 1 (process 10989) exited normally]

+run

Starting program: /home/ubuntu20/Escritorio/sum1toN

Breakpoint 1, main () at sum1toN.c:22

Punto de interrupción 1 at 0x1129: file sum1toN.c, line 22.

+run

$1 = 5 '\005'

+p /t n

$2 = 101

+p /x n

$3 = 0x5

+ptype n

type = char

+whatis n

type = char

+p &n

$4 = 0x7fffffffdf8f "\005"

+p &eax

No symbol "eax" in current context.

+p $eax

$5 = 1431654697

+info registers

rax 0x555555555129 93824992235817

rbx 0x555555555160 93824992235872

rcx 0x555555555160 93824992235872

rdx 0x7fffffffe098 140737488347288

rsi 0x7fffffffe088 140737488347272

rdi 0x1 1

rbp 0x7fffffffdf90 0x7fffffffdf90

rsp 0x7fffffffdf90 0x7fffffffdf90

r8 0x0 0

r9 0x7ffff7fe0d50 140737354009936

r10 0x3 3

r11 0x2 2

r12 0x555555555040 93824992235584

r13 0x7fffffffe080 140737488347264

r14 0x0 0

r15 0x0 0

rip 0x555555555139 0x555555555139 <main+16>

eflags 0x246 [ PF ZF IF ]

cs 0x33 51

ss 0x2b 43

ds 0x0 0

es 0x0 0

fs 0x0 0

gs 0x0 0

+q

sum1toN.c (modificado):

Hemos cambiado el tipo de las variables de entero a carácter, la expresión sum+=n pasa a ser sum-=n y el valor de la variable n está expresado en binario:

/\*

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 --save-temps -g -o sum1toN sum1toN.c

S.O: GNU/linux 4.10 ubuntu 17.04 x86-64

Librería: /usr/lib/x86\_64-linux-gnu/libc.so

CPU: Linux alvaro12 5.4.0-48-generic #52-Ubuntu SMP Thu Sep 10 10:58:49 UTC 2020 x86\_64 x86\_64 x86\_64 GNU/Linux

Compilador: gcc (Ubuntu 9.3.0-10ubuntu2) 9.3.0

Ensamblador: Ensamblador (GNU Binutils for Ubuntu) 2.34 de GNU

Linker/Loader: GNU ld (GNU Binutils for Ubuntu) 2.34

Asignatura: Estructura de Computadores

Fecha: 07/10/2020

Autor: Alvaro Larraya

\*/

void main (void)

{

char sum=0;

char n=0b101; //Declaración de variables locales

while(n>0){

sum-=n;

n--;

}

}

//No da ningún error de compilación

## *GDB:*

+x /ldb

Se requiere un argumento (starting display address).

+b main

Punto de interrupción 1 at 0x1129: file sum1toN.c, line 22.

+run

Starting program: /home/ubuntu20/Escritorio/sum1toN

Breakpoint 1, main () at sum1toN.c:22

+x /ldb &sum

0x7fffffffdf8e: 0

+x /1db &sum

0x7fffffffdf8e: 0

+x /1tb &sum

0x7fffffffdf8e: 00000000

+x /1ob &sum

0x7fffffffdf8e: 0

+x /1xb &sum

0x7fffffffdf8e: 0x00

+help x ltb

Examine memory: x/FMT ADDRESS.

ADDRESS is an expression for the memory address to examine.

FMT is a repeat count followed by a format letter and a size letter.

Format letters are o(octal), x(hex), d(decimal), u(unsigned decimal),

t(binary), f(float), a(address), i(instruction), c(char), s(string)

and z(hex, zero padded on the left).

Size letters are b(byte), h(halfword), w(word), g(giant, 8 bytes).

The specified number of objects of the specified size are printed

according to the format. If a negative number is specified, memory is

examined backward from the address.

Defaults for format and size letters are those previously used.

Default count is 1. Default address is following last thing printed

with this command or "print".

+q

sum1toN.s:

### 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 #carga en EDX el valor de n

bucle:

add %edx,%ecx #guarda en ECX la suma EDX+ECX

sub $1,%edx #guarda en EDX la resta EDX-1

jnz bucle #mientras EDX distinto de 0 se mantiene en el bucle

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

ABI i386

## 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

## *Ejecutar:*

Compilamos sum1toN.s con gcc -m32 -g -nostartfiles -o sum1toN sum1toN.s. Con ./sum1toN ejecutamos el programa. Realizamos un echo $? justo después de correr el programa, este comando nos devuelve un 15, que es el resultado de la suma.

## *GDB:*

+shell ls -l sum1toN\_gdb\_asm.txt

+info sources

Source files for which symbols have been read in:

/home/ubuntu20/Escritorio/sum1toN.s

Source files for which symbols will be read in on demand:

+b \_start

Punto de interrupción 1 at 0x1000: file sum1toN.s, line 18.

+run

Starting program: /home/ubuntu20/Escritorio/sum1toN

Breakpoint 1, \_start () at sum1toN.s:18

+ptype n

type = <data variable, no debug info>

+p n

'n' has unknown type; cast it to its declared type

+x address

No symbol "address" in current context.

+x &n

0x56558000: 0x00000005

+x n

'n' has unknown type; cast it to its declared type

+x /1bw &n

0x56558000: 0x00000005

+x /1xw &n

0x56558000: 0x00000005

+x /4xw &n

0x56558000: 0x00000005 0x00000000 0x0000001c 0x00000002

+b bucle

Punto de interrupción 2 at 0x5655600b: file sum1toN.s, line 21.

+c

Continuando.

Breakpoint 2, bucle () at sum1toN.s:21

+start

Función «main» no definida.

Punto de interrupción temporal 3 (-qualified main) pendiente.

Starting program: /home/ubuntu20/Escritorio/sum1toN

Breakpoint 1, \_start () at sum1toN.s:18

+c

Continuando.

Breakpoint 2, bucle () at sum1toN.s:21

+n

+p $ecx

$1 = 5

+p $edx

$2 = 5

+until

+p $ecx

$3 = 5

+p $edx

$4 = 4

+info registers

eax 0x1c 28

ecx 0x5 5

edx 0x4 4

ebx 0xf7ffd000 -134230016

esp 0xffffd200 0xffffd200

ebp 0x0 0x0

esi 0xffffd20c -11764

edi 0x56556000 1448435712

eip 0x56556010 0x56556010 <bucle+5>

eflags 0x202 [ IF ]

cs 0x23 35

ss 0x2b 43

ds 0x2b 43

es 0x2b 43

fs 0x0 0

gs 0x63 99

+exit

orden indefinida: «exit». Intente con «help»

+q

Autoevaluación:

# *Cuestiones teóricas*:

1. **Cuál es la principal diferencia entre las arquitecturas de los dos procesadores utilizados en prácticas:** la principal diferencia entre las arquitecturas i386 y AMD64 es que poseen distintas ISAs o repertorio de instrucciones máquina, lo cual no quiere decir que no vayan a funcionar con el mismo lenguaje ensamblador.
2. **Qué fases comprende el toolchain:** primero se precompila el codigo fuente, luego se compila, lo cual nos devuelve el modulo ASM, lo ensamblamos, obteniendo así el módulo objeto binario reubicable. Una vez hecho esos pasos con todos los ficheros que queremos utilizar, le pasamos al linker todos los módulos objeto binario reubicables que vayamos a utilizar, para tener por último el módulo objeto ejecutable.
3. **Lista las herramientas de desarrollo a utilizar durante la realización de las prácticas mediante los dos procesadores utilizados:** un editor de texto lo más simple posible, un sistema operativo con una terminal de linux y en ella tener instaladas las librerias necesarias para que gcc, as y ld sean operativas en la arquitectura i386
4. **Libro: Programming from the Ground-Up:**
   1. **Qué es GNU/Linux:** sistemas operativos de tipo Unix, que suelen ser de código abierto, multiplataforma, multiusuario y multitarea.[1](https://es.wikipedia.org/wiki/GNU/Linux#cite_note-1)​ Estos sistemas operativos están formados mediante la combinación de varios proyectos, entre los cuales destaca el entorno GNU y el núcleo de sistema operativo conocido como «Linux»,
   2. **Qué es GNU:** sistema operativo de tipo Unix, así como una gran colección de programas informáticos que componen al sistema. Está formado en su totalidad por [software libre](https://es.wikipedia.org/wiki/Software_libre), GNU es el acrónimo recursivo de *"*GNU's Not Unix*"*
   3. **Qué es gcc:** es un compilador integrado del proyecto GNU para C, C++, Objective C y Fortran; es capaz de recibir un programa fuente en cualquiera de estos lenguajes y generar un programa ejecutable binario en el lenguaje de la máquina donde ha de correr.
   4. **Qué gestiona el kernel:** decide la cantidad de memoria que administra a cada elemento; que procesos pueden usar la CPU, cuando y durante cuanto tiempo; actúa como interprete entre el hardware y los procesos; y recibe solicitudes de servicio por parte de los procesos.
   5. **¿Se puede acceder simultáneamente a instrucciones y datos?¿Por qué?:** no, ya que como comparten el bus de acceso a memoria, pero se encuentran en dos secciones de memoria diferentes el bus debe cargar un tipo cada vez.
   6. **Cuál es la función del registro PC:** llevar la cuenta de cuantas operaciones se han realizado hasta el momento.
   7. **Cuáles son los dos tipos de registros de la CPU:** los GPR que son accesibles por el programador y los MBR a los cuales no tiene acceso.
   8. **Qué significa Word Size:** es el espacio de memoria que se reserva para un dato o dos instrucciones, los cuales ocupan 40 bits.
   9. **Qué es una varible puntero:** es un tipo de dato cuyo valor es una dirección de memoria.
   10. **Lista cuatro modos diferentes de direccionar un operando:** indirecto,inmediato,indexado y registro

# *Cuestiones prácticas*:

1. **Comando de compilación del programa fuente ensamblador mediante el front-end gcc que incluya la tabla de símbolos para el depurador:** gcc -m32 -g -nostartfiles -o sum1toN sum1toN.s
2. **Comando de enlace (linker) del módulo objeto reubicable:** ld -melf\_i386 -o sum1toN sum1toN.o
3. **Declaración en lenguaje C de la variable n tipo entero con signo de un byte:** sbyte n;
4. **Instrucción en lenguaje ensamblador del programa sum1toN.s que realiza una suma:** add %edx,%ecx
5. **Comandos del depurador gdb para la impresión del contenido de la variable n:** print n, p n
6. **Ejecutar sum1toN, compilado de sum1toN.s, paso a paso mediante el depurador GDB ejecutando los comandos necesarios para:**
   1. **imprimir el contenido de la variable n y su dirección en memoria principal:** p n, p &n
   2. **imprimir la dirección de la etiqueta bucle:** p bucle
   3. **imprimir el contenido del registro ECX al salir del bucle:** p $ecx
7. **Cambiar el tamaño de los operandos de la suma a 2 bytes:** cambiar la instrucción add%edx,%ecx por la instrucción addw%dx,%ec
8. **Cambiar el tamaño del operando n dos bytes:** n: .word 5
9. **En GDB qué comando hay que utilizar para ejecutar todas las iteraciones del bucle del programa de forma continuada:** continue, c
10. **Comparando las versiones en lenguajes C y ASM de los módulos fuente, por qué la instrucción until del depurador GDB en el caso del módulo fuente en lenguaje C se ejecuta durante la sentencia while:** porque en el punto c ejecuta una iteración entera, mientras que en el .s actúa como un next.