Programación Ensamblador AT&T x86

HISTORIAL DE REVISIONES			
NÚMERO	FECHA	MODIFICACIONES	NOMBRE
v1.0.0	2018 September 19		CA

Índice

1. Programa básico x86-32

- Módulo fuente: *sum1toN.s*

```
### Programa: sum1toN.s
### Descripción: realiza la suma de la serie 1,2,3,...N
### 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
        .section .data
n:      .int 5
        .global _start
        ##  Comienzo del código
        .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
        .end
```

- Compilación: `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`
- Directivas del traductor ensamblador: .section, .data, .text, .byte, .end, etc... empiezan con un punto como prefijo
- Sintaxis instrucción asm: *etiqueta: mnemónico Operando_fuente, Operand_Destino #comentarios*
- Las etiquetas llevan el sufijo **:**
- La etiqueta **_start**: es el punto de entrada al programa. Obligatoria. La utiliza el linker.
- Sufijos de los mnemónicos
  - **b** → byte → 1Byte →Ej: movb
  - **w** → word → 2Bytes →Ej: movw . En este contexto word son 2 bytes por razones históricas.
  - **l** → long → 4Bytes →Ej: movl . Valor por defecto.
  - **q** → quad → 8Bytes →Ej: movq
- Direccionamientos de los operandos:
  - En la misma instrucción los operandos fuente y destino no pueden hacer ambos referencia a la memoria Principal.
  - inmediato: prefijo del operando **$**
  - registro: prefijo del registro **%**
  - directo: el operando es una etiqueta que apunta a la memoria principal
  - indirecto: el operando es una etiqueta o un registro: utiliza paréntesis. (etiqueta) ó (%registro). Ver indexado.
    - La etiqueta referencia una posición de memoria que contiene a su vez una dirección de la memoria principal que apunta al operando.
    - El registro contiene la dirección de la memoria principal que apunta al operando.

- indexado
  - dirección efectiva: *base* + *index*scale* + *disp* → la sintaxis es: disp(base,índice,escala)
  - foo(%ebp, %esi,4) → dirección efectiva= *EBP + 4*ESI + foo*
  - (%edi) → dirección efectiva= *EDI* → direccionamiento indirecto
  - -4(%ebp) → dirección efectiva= *EBP - 4*
  - foo(, %eax,4) → dirección efectiva= *4*EAX + foo*
  - foo(,1) → dirección efectiva= foo
- Cualquier instrucción que tiene una referencia a un operando en la memoria principal y no tiene una referencia a registro, debe especificar el tamaño del operando (byte, word, long, or quadruple) con una instrucción que lleve el sufijo (‘b’, ‘w’, ‘l’ or ‘q’, respectivamente).

2. Directivas Assembler AS

- Manual
  - https://sourceware.org/binutils/docs/as/

Cuadro 1: Directivas básicas

.global o .globl : variables globales
.section .data : sección de las variables locales estáticas inicializadas
.section .text : sección de las instrucciones
.section .bss : sección de las variables sin inicializar
.section .rodata : sección de las variables de sólo lectura
.type name , type description : tipo de variable, p.ej @function
.end : fin del ensamblaje
.common 100 : reserva 100 bytes sin inicializar y puede ser referenciado globalmente
.lcomm bucle, 100 : reserva 100bytes referenciados con el símbolo local bucle. Sin inicializar.
.space 100 : reserva 100 bytes inicializados a cero
.space 100, 3 : reserva 100 bytes inicializados a 3
.string "Hola" : añade el byte 0 al final de la cadena
.asciz "Hola" : añade el byte 0 al final de la cadena
.ascii "Hola" : no añade le caracter NULL de final de cadena
.byte 3,7,-10,0b1010,0xFF,0777 : tamaño 1Byte y formatos decimal,decimal,decimal,binario,hexadecimal,octal
.2byte 3,7,-10,0b1010,0xFF,0777 : tamaño 2Bytes
.word 3,7,-10,0b1010,0xFF,0777 : tamaño 2Bytes
.short 3,7,-10,0b1010,0xFF,0777 : tamaño 2B
.4byte 3,7,-10,0b1010,0xFF,0777 : tamaño 4B
.long 3,7,-10,0b1010,0xFF,0777 : tamaño 4B
.int 3,7,-10,0b1010,0xFF,0777 : tamaño 4B
.8byte 3,7,-10,0b1010,0xFF,0777 : tamaño 8B
.quad 3,7,-10,0b1010,0xFF,0777 : tamaño 8B
.octa 3,7,-10,0b1010,0xFF,0777 : formato octal
.double 3.14159, 2 E-6 → precisión doble
.float 2E-6, 3.14159 → precisión simple
.single 2E-6 → precisión simple
.include "file" : incluye el fichero . Obligatorias las comillas.
.macro macname macargs : define el comienzo de una macro de nombre macname y argumentos macargs
.endmacro : define el final de una macro
.align n : las instrucciones o datos posteriores empezarán en una dirección multiplo de n bytes.

- Alineamiento **Little Endian**: El byte de menor peso, LSB, se almacena en la posición de memoria más baja.
  - .int OxAABBCCDD → 0xDD se almacena primero en la dirección más baja, el resto de bytes se almacenan en sentido ascendente en el orden 0xCC,0xBB,0xAA

3. Repertorio de Instrucciones Ensamblador

- Lenguaje Ensamblador AT&T

3.1. TRANSFERENCIA

Nombre	Comentario	Código	Operación	O D I T S Z A P C
MOV	Mover (copiar)	MOV Fuente Dest	Dest=:Fuente	

```
Nombre Comentario                    Código           Operación  ↩
                                       O D I T S Z A P C
MOV    Mover (copiar)                MOV Fuente,Dest   Dest:=Fuente
XCHG   Intercambiar                  XCHG Op1,Op2      Op1:=Op2 , Op2:=Op1
STC    Set the carry (Carry = 1)     STC               CF:=1  ↩
                                                           1
CLC    Clear Carry (Carry = 0)       CLC               CF:=0  ↩
                                                           0
CMC    Complementar Carry            CMC               CF:=Ø  ↩
                                                           \ensuremath{\pm}
STD    Setear dirección              STD               DF:=1(interpreta strings de arriba  ↩
    hacia abajo)  1
CLD    Limpiar dirección             CLD               DF:=0(interpreta strings de abajo  ↩
    hacia arriba)  0
STI    Flag de Interrupción en 1     STI               IF:=1  ↩
                                                     1
CLI    Flag de Interrupción en 0     CLI               IF:=0  ↩
                                                     0
PUSH   Apilar en la pila             PUSH Fuente       DEC SP, [SP]:=Fuente
PUSHF  Apila los flags               PUSHF             O, D, I, T, S, Z, A, P, C 286+:  ↩
    También NT,IOPL
PUSHA  Apila los registros generales PUSHA             AX, CX, DX, BX, SP, BP, SI, DI
POP    Desapila de la pila           POP Dest          Destino:=[SP], INC SP
POPF   Desapila a los flags          POPF              O,D,I,T,S,Z,A,P,C 286+: También NT, ↩
    IOPL        \ensuremath{\pm} \ensuremath{\pm} \ensuremath{\pm} \ensuremath{\pm} \ ↩
    ensuremath{\pm} \ensuremath{\pm} \ensuremath{\pm} \ensuremath{\pm} \ensuremath{\pm}
POPA   Desapila a los reg. general.  POPA              DI, SI, BP, SP, BX, DX, CX, AX
CBW    Convertir Byte a Word         CBW               AX:=AL (con signo)
CWD    Convertir Word a Doble        CWD               DX:AX:=AX (con signo)
CWDE   Conv. Word a Doble Exten.     CWDE 386          EAX:=AX (con signo)
IN_i   Entrada                       IN Dest,Puerto    AL/AX/EAX := byte/word/double del  ↩
    puerto especifi.
OUT_i  Salida                        OUT Puerto,Fuente Byte/word/double del puerto  ↩
    especifi. := AL/AX/EAX
```

- **i**: para más información ver especificaciones de la intrucción, Flags: ± =Afectado por esta instrucción, **?** =Indefinido luego de esta instrucción

3.2. ARITMÉTICOS

```
Nombre  Comentario                        Código           Operación  ↩
                                           O D I T S Z A P C
ADD     Suma                             ADD Fuente,Dest  Dest:=Dest+ Fuente  ↩
                                \ensuremath{\pm}       \ensuremath{\pm} \ensuremath{\pm} \ ↩
    ensuremath{\pm} \ensuremath{\pm} \ensuremath{\pm}
ADC     Suma con acarreo                 ADC Fuente,Dest  Dest:=Dest+ Fuente +CF  ↩
                              \ensuremath{\pm}       \ensuremath{\pm} \ensuremath{\pm} \ ↩
    ensuremath{\pm} \ensuremath{\pm} \ensuremath{\pm}
SUB     Resta                            SUB Fuente,Dest  Dest:=Dest- Fuente  ↩
                                \ensuremath{\pm}       \ensuremath{\pm} \ensuremath{\pm} \ ↩
    ensuremath{\pm} \ensuremath{\pm} \ensuremath{\pm}
SBB     Resta con acarreo                SBB Fuente,Dest  Dest:=Dest-(Fuente +CF)  ↩
                            \ensuremath{\pm}       \ensuremath{\pm} \ensuremath{\pm} \ ↩
    ensuremath{\pm} \ensuremath{\pm} \ensuremath{\pm}
DIV     División (sin signo)             DIV Op           Op=byte: AL:=AX / Op AH:=Resto  ↩
                      ?       ? ? ? ? ?
DIV     División (sin signo)             DIV Op           Op=word:         AX:=DX:AX / Op DX ↩
    :=Resto       ?       ? ? ? ? ?
DIV     386 División (sin signo)         DIV Op           Op=doublew.:     EAX:=EDX:EAX / Op ↩
     EDX:=Resto   ?      ? ? ? ? ?
IDIV    División entera con signo        IDIV Op          Op=byte:        AL:=AX / Op AH:= ↩
    Resto           ?      ? ? ? ? ?
IDIV    División entera con signo        IDIV Op          Op=word: AX:=DX:AX / Op DX:=Resto  ↩
                  ?      ? ? ? ? ?
IDIV    386 División entera con signo    IDIV Op          Op=doublew.: EAX:=EDX:EAX / Op EDX ↩
    :=Resto       ?      ? ? ? ? ?
MUL     Multiplicación (sin signo)       MUL Op           Op=byte: AX:=AL*Op si AH=0 #  ↩
                        \ensuremath{\pm}     ? ? ? ? \ensuremath{\pm}
MUL     Multiplicación (sin signo)       MUL Op           Op=word: DX:AX:=AX*Op si DX=0 #  ↩
                    \ensuremath{\pm}     ? ? ? ? \ensuremath{\pm}
MUL     386 Multiplicación (sin signo)   MUL Op           Op=double: EDX:EAX:=EAX*Op si EDX ↩
    =0 #           \ensuremath{\pm}     ? ? ? ? \ensuremath{\pm}
IMUL    i Multiplic. entera con signo    IMUL Op          Op=byte: AX:=AL*Op si AL es  ↩
    suficiente #        \ensuremath{\pm}      ? ? ? ? \ensuremath{\pm}
IMUL    Multiplic. entera con signo      IMUL Op          Op=word: DX:AX:=AX*Op si AX es  ↩
    suficiente #     \ensuremath{\pm}      ? ? ? ? \ensuremath{\pm}
IMUL    386 Multiplic. entera con signo IMUL Op          Op=double: EDX:EAX:=EAX*Op si EAX  ↩
    es sufi. #    \ensuremath{\pm}      ? ? ? ? \ensuremath{\pm}
INC     Incrementar                      INC Op           Op:=Op+1 (El Carry no resulta  ↩
    afectado !)       \ensuremath{\pm}      \ensuremath{\pm} \ensuremath{\pm} \ensuremath{\ ↩
    pm} \ensuremath{\pm}
DEC     Decrementar                      DEC Op           Op:=Op-1 (El Carry no resulta  ↩
    afectado !)       \ensuremath{\pm}       \ensuremath{\pm} \ensuremath{\pm} \ensuremath{\ ↩
    pm} \ensuremath{\pm}
CMP     Comparar                         CMP              Op1,Op2  Op1-Op2  ↩
                                  \ensuremath{\pm}       \ensuremath{\pm} \ensuremath{\pm}  ↩
    \ensuremath{\pm} \ensuremath{\pm} \ensuremath{\pm}
SAL     Desplazam. aritm. a la izq.      SAL              Op,Cantidad  ↩
                                         i       \ensuremath{\pm} \ensuremath{\pm} ? \ ↩
    ensuremath{\pm} \ensuremath{\pm}
SAR     Desplazam. aritm. a la der.      SAR              Op,Cantidad  ↩
                                         i       \ensuremath{\pm} \ensuremath{\pm} ? \ ↩
    ensuremath{\pm} \ensuremath{\pm}
RCL     Rotar a la izq. c/acarreo        RCL Op,Cantidad  ↩
                                                     i               \ensuremath{\pm}
RCR     Rotar a la derecha c/acarreo     RCR Op,Cantidad  ↩
                                                     i               \ensuremath{\pm}
ROL     Rotar a la izquierda             ROL Op,Cantidad  ↩
                                                     i               \ensuremath{\pm}
```

```
ROR     Rotar a la derecha              ROR Op,Cantidad  ↩
                                                  i               \ensuremath{\pm}
```

- **i**:para más información ver especificaciones de la intrucción,
- **#**:entonces CF:=0, OF:=0 sino CF:=1, OF:=1

3.3. LÓGICOS

```
Nombre Comentario                    Código             Operación  ↩
                                          O D I T S Z A P C
NEG    Negación (complemento a 2)    NEG Op             Op:=0-Op si Op=0 entonces CF:=0  ↩
    sino CF:=1      \ensuremath{\pm}       \ensuremath{\pm} \ensuremath{\pm} \ensuremath{\pm} \ensuremath{\pm} ↩
    } \ensuremath{\pm} \ensuremath{\pm}
NOT    Invertir cada bit             NOT Op             Op:=Ø~Op (invierte cada bit)
AND    'Y' (And) lógico              AND Fuente,Dest    Dest:=Dest ^ Fuente  ↩
                                0       \ensuremath{\pm} \ensuremath{\pm} ? \ensuremath{\pm} ↩
     0
OR     'O' (Or) lógico               OR  Fuente,Dest    Dest:=Dest v Fuente  ↩
                                0       \ensuremath{\pm} \ensuremath{\pm} ? \ensuremath{\pm} ↩
     0
XOR    'O' (Or) exclusivo            XOR Fuente,Dest    Dest:=Dest (xor) Fuente  ↩
                            0       \ensuremath{\pm} \ensuremath{\pm} ? \ensuremath{\pm} 0
SHL    Desplazam. lógico a la izq.   SHL Op,Cantidad  ↩
                                                        i       \ensuremath{\pm} \ensuremath ↩
    {\pm} ? \ensuremath{\pm} \ensuremath{\pm}
SHR    Desplazam. lógico a la der.   SHR Op,Cantidad  ↩
                                                        i       \ensuremath{\pm} \ensuremath ↩
    {\pm} ? \ensuremath{\pm} \ensuremath{\pm}
```

3.4. MISCELÁNEOS

```
Nombre  Comentario                  Código           Operación  ↩
                                              O D I T S Z A P C
NOP     Hacer nada                  NOP              No hace operación algun
LEA     Cargar direciión Efectiva   LEA Fuente,Dest  Dest := dirección fuente
INT     Interrupción                INT Num          Interrumpe el proceso actual y  ↩
                              0 0
                                                     salta a la subrutina con vector de  ↩
                                                         interrupción Num
```

3.5. SALTOS (generales)

```
Nombre  Comentario                    Código          Operación
CALL    Llamado a subrutina           CALL Proc
JMP     Saltar                        JMP Dest
JE      Saltar si es igual            JE Dest         (= JZ)
JZ      Saltar si es cero             JZ Dest         (= JE)
JCXZ    Saltar si CX es cero          JCXZ Dest
JP      Saltar si hay paridad         JP Dest         (= JPE)
JPE     Saltar si hay paridad par     JPE Dest        (= JP)
JPO     Saltar si hay paridad impar   JPO Dest        (= JNP)
JNE     Saltar si no es igual         JNE Dest        (= JNZ)
JNZ     Saltar si no es cero          JNZ Dest        (= JNE)
JECXZ   Saltar si ECX es cero         JECXZ Dest 386
JNP     Saltar si no hay paridad      JNP Dest        (= JPO)
RET     Retorno de subrutina          RET
```

3.6. SALTOS Sin Signo (Cardinal) SALTOS Con Signo (Integer)

```
Nombre  Comentario                    Código          Operación
JA      Saltar si es superior         JA Dest         (= JNBE)
JAE     Saltar si es superior o igual JAE Dest        (= JNB = JNC)
```

```
JB      Saltar si es inferior           JB Dest         (= JNAE = JC)
JBE     Saltar si es inferior o igual   JBE Dest        (= JNA)
JNA     Saltar si no es superior        JNA Dest        (= JBE)
JNAE    Saltar si no es super. o igual  JNAE Dest       (= JB = JC)
JNB     Saltar si no es inferior        JNB Dest        (= JAE = JNC)
JNBE    Saltar si no es infer. o igual  JNBE Dest       (= JA)
JC      Saltar si hay carry JC Dest     JO  Dest        Saltar si hay Overflow
JNC     Saltar si no hay carry          JNC Dest
JNO     Saltar si no hay Overflow       JNO Dest
JS      Saltar si hay signo (=negativo) JS Dest
JG      Saltar si es mayor              JG Dest         (= JNLE)
JGE     Saltar si es mayor o igual      JGE Dest        (= JNL)
JL      Saltar si es menor              JL Dest         (= JNGE)
JLE     Saltar si es menor o igual      JLE Dest        (= JNG)
JNG     Saltar si no es mayor           JNG Dest        (= JLE)
JNGE    Saltar si no es mayor o igual   JNGE Dest       (= JL)
JNL     Saltar si no es inferior        JNL Dest        (= JGE)
JNLE    Saltar si no es menor o igual   JNLE Dest       (= JG)
```

3.7. FLAGS (ODITSZAPC)

```
O: Overflow resultado de operac. sin signo es muy grande o pequeño.
D: Dirección
I: Interrupción Indica si pueden ocurrir interrupciones o no.
T: Trampa Paso, por paso para debugging
S: Signo Signo del resultado. Razonable sólo para enteros. 1=neg. 0=pos.
Z: Cero Resultado de la operación es cero. 1=Cero
A: Carru Aux. Similar al Carry, pero restringido para el nibble bajo únicamente
P: Paridad 1=el resultado tiene cantidad par de bits en uno
C: Carry resultado de operac. sin signo es muy grande o inferior a cero
```

4. Registros

Registros amd64	Registros Flags

- Los registros de propósito general RPG son:
  - ‘%eax’ (el acumulador), ‘%ebx’, ‘%ecx’, ‘%edx’, ‘%edi’, ‘%esi’, ‘%ebp’ (puntero frame), and ‘%esp’ (puntero stack).

Registros RPG

gdb 1

gdb 2

5. GDB

- Comandos básicos

```
gdb
shell date
shell pwd
shell ls
shTAB
shell daTAB
C-x a
C-x o
histórico comandos: navegar con las flechas
set trace-commands on
set logging file gdb_salida.txt
set logging on
shell ls -l gdb_salida.txt
file modulo_bin
info sources
break main
run
next ,n ,n 5
step ,s
RETURN
continue, c
until, RETURN, RETURN ...
ptype varible
whatis variable
print variable, p variable, p /t variable, p /x n
p &n
x dirección
x &variable, x /1bw &variable, +x /1xw &variable, x /4xw &variable
layout split
next instruction, ni, RET, RET, RET, RET, until, RET,..hasta salir del bucle
step ,s
si
```