Material de apoyo
Operadores de ensamblador

CONCEPTOS

El proceso de análisis de un fichero binario desensamblado puede ser algo complejo inicialmente, para ello se ha elaborado este documento que permite al alumno entender cuál es la función de cada uno de sus operadores y lograr una mejor comprensión de su funcionamiento.

OPERADORES DE ENSAMBLADOR

ADC

Instrucción que añade valores de un registro que lleve operadores contenedores.

adc “sufijo opcional” “código condicional opcional” “Registro de destino”, “Registro que contiene el primer operando”, “segundo operando”

adc r4, r0, r2

ADD

Instrucción que añade valores de un registro que lleve operadores.

add “sufijo opcional” “código condicional opcional” “Registro de destino”, “Registro que contiene el primer operando”, “segundo operando”

add r2, r1, r3

AND

Es el equivalente al operador & en Python o bash.

and “destino”, “valor”

and eax, 0x7575

BT

Selecciona el bit en una cadena de bits

bt base, offset

bt ebx, 0

CALL

Esta instrucción se usa para llamar una función

call “nombre de función”

call read_input

CDQ

Extiende el bit firmado de EAX hacia el registro EDX

mov eax, 0x5

cdq

CLD

Limpia el registro DF

cld

CMOVNZ

Instrucción condicional, mueve si el Zero Flag no ha sido establecido

cmovnz destino, origen

cmovnz cx, dx

CMP

Se utiliza para realizar comparaciones, sirve como la instrucción SUB a excepción de que no afecta los operadores.

cmp destino, origen

cmp [ebp+arg_0], 1 (compara 1 con [ebp+arg_0])

CMPSB

Compara dos valores al obtenerlos del byte que apunta el ES:DI con el byte que apunta el DS:SI, estableciendo la flag acorde a los resultados de la comparación (los operadores en sí no son alterados). Tras la comparación, SI y DI son incrementados (si la direction flag ha sido limpiada) o los reduce (si la direction flag está establecida) para preparar la comparación de la siguiente cadena.

```
cld                 (escanea la siguiente dirección)

mov cx, 100         (escanea 100 bytes, CX es usado por repe)

lea si, buffer1     (comienza la dirección del primer búfer)

lea di, buffer2     (comienza la dirección del segundo búfer)

lea cmpsb           (y los compara)

jne mismatch        (la zero flag será limpiada si hay un mismatch)
```

DEC

Reduce 1 en el registro

dec destino

dec ecx

DIV

Instrucción que realiza una división. Siempre divide el valor de los 32/64 bits en EDX:EAX por un valor. El resultado de la división se almacena en la EAX y el resto en el EDX.

```
div valor

mov edx, 0          (despeja el dividendo)

mov eax, 0x8003     (dividendo)

mov ecx, 0x100      (divisor)

div ecx             (EAX = 0x80, EDX = 0x3)
```

IDIV

Divide el contenido de un numero entero de 64 bits (EDX:EAX) por el valor especificado en el valor del operando. El coeficiente se almacena en la EAX y el resto en el EDX.

```
idiv ebx                (EDX:EAX / EBX. El coeficiente se salva en el EAX y
                        el resto en el EDX)

idiv DWORD PRT [var]    (EDX:EAX / valor de 32 bits almacenado en la
                        localización var. El coeficiente se guarda en el
                        EAX y el resto en el EDX)
```

IMUL

Multiplicación firmada de dos operadores. Si solo se provee un registro es multiplicado por la EAX

```
mov eax, 9          (eax = 9)

mov edx, 66666667h  (edx = 0x66666667)

imul edx            (edx:eax = 0x66666667 * 9= 0x39999999f, el edx = 0x3 y el
                    eax = 0x39999999f)
```

IN

Lee de un puerto (serie, impresora, teclado, ratones, sensores, etc…)

```
in destino, origen

in eax, dx
```

INC

Aumenta 1 en el registro

```
inc destino

inc edx
```

JA

Realiza un salto condicional a continuación de una comparación no firmada

ja destino, origen

JAE

Instrucción para hacer un salto condicional tras realizarse un test, realiza un salto de comparación no firmada tras un cmp si el operador de destino es mayor o igual al operando de origen

jae destino, origen

JB

Salto condicional que sigue a continuación de un test, realiza un salto de comparación no firmada si el operando de destino es menor que el operando de origen.

jb destino, origen

JZ

Salto condicional que sigue a un test, realiza el salto a la dirección especificada si la zero flag está en 1

dec ecx

jz el_contador_es_cero

JECXZ

Salto condicional que sigue a un test, hace el salto a la dirección indicada si ECX=0

jecxz localización

JG

Instrucción para hacer un salto condicional tras realizarse un test, realiza un salto de comparación firmada tras un cmp si el operador de destino es mayor al operando de origen.

jg destino, origen

JGE

Instrucción para hacer un salto condicional tras realizarse un test, realiza un salto de comparación firmada tras un cmp si el operador de destino es mayor o igual al operando de origen.

jge destino, origen

JL

Salto condicional que sigue a continuación de un test, realiza un salto de comparación firmada si el operando de destino es menor que el operando de origen.

jl destino, origen

JLE

Salto condicional que sigue a continuación de un test, realiza un salto de comparación firmada si el operando de destino es menor o igual que el operando de origen.

jle destino, origen

JNB

Salto condicional que sigue a continuación de un test, realiza un salto de comparación firmada si el carry flag es 0.

jnb localización

JNP

Salto condicional que sigue a continuación de un test, realiza un salto de comparación firmada si el parity flag es 0

jnp localización

JNZ

Salto condicional que sigue a continuación de un test, realiza un salto de comparación firmada si el zero flag es 0.

jnz localización

JO

Salto condicional que sigue a continuación de un test, realiza el salto a la dirección especificada si la instrucción previa estableció la overflow flag.

jo localización

JS

Salto condicional que sigue a continuación de un test, realiza el salto a la dirección especificada si la instrucción previa estableció la sign flag.

js localización

JZ

Salto condicional que sigue a continuación de un test, realiza el salto a la dirección especificada si la instrucción previa estableció la zero flag a 1.

jz localización

LEA

Esta instrucción se utiliza para establecer una dirección de memoria en el destino.

```
lea destino, origen
```

```
lea eax, [ebx+8]
```

LODSx

Carga un byte (B), palabra (W) o DWORD (D) del operando de origen al registro AL,AX o EAX.

```
LODS m8      (carga un Byte a AL)

LODS m16     (carga una Word a AX)

LODS m32     (carga una DWord a EAX)

LODSB        (Carga un Byte a AL)

LODSW        (carga una Word a AX)

LODSD        (carga una DWord a EAX)
```

LOOP

Reduce el ECX y hace un salto a la dirección especificada por el argumento, a no ser que al reducir el ECX el valor sea 0.

```
loop argumento
```

```
mov ecx, 5

comenzar_loop (el código que vaya a continuación se ejecutaría 5 veces)

codigo

loop comenzar_loop
```

MOV

Esta instrucción se utiliza para mover datos a registros o la RAM

```
mov destino, origen
```

MOVSB

Mueve un byte de la dirección EDS:ESI a la dirección ES:EDI

```
lea edi, [ebp+var_118]    (Destino)

esi, offset unk_43F0C9    (Origen, 47 bytes en la localización 0x43F0C9)

cld                       (Limpia la direction flag)

mov ecx, 2h               (Contador para rep, 47 bytes)

rep movsb                 (Mueve 47 bytes de la localización 0x43F0C9 a
                          var_118)
```

MUL

Esta instrucción se utiliza para realizar multiplicaciones, multiplica el EAX por un valor, el resultado es almacenado en un valor de 64 bits en el EDX (los más 32 bits más significantes) y EAX (los 32 bits menos significantes).

```
mul valor

mul 0x10 (Multiplica EAX por 0x10 y almacena los resultados en el EDX:EAX)
```

NEG

Establece un valor negativo en el destino.

```
neg registro

mov eax, 0x5 (EAX = 5)

neg eax      (EAX = -5)
```

NOP

Instrucción de “no operation”, no realiza función alguna, solo rellena el marco del buffer.

NOT

Instrucción para invertir el valor de bis de un argumento, el 0 pasará a ser 1, y el 1 pasará a ser 0.

```
mov edx, 01111110b (EDX = 01111110b)

not edx            (EDX = 10000001b)
```

OR

Esta instrucción realiza una operación lógica OR, por lo tanto se ejecutará la combinación de 1/0,0/1,1/1 cómo si fuese 1 siempre, mientras que 0/0 se ejecutará cómo 0).

or destino, origen

mov eax, 0x18 (0000000000011000 en binario, 24 en decimal)

or eax, 0x7575 (0111010101110101 en binario, 30069 en decimal)

Almacenará (0111010101111101, 30077 en decimal en la EAX)

POP

Transfiere el último valor almacenado de la pila al operando de destino, después aumenta el registro ESP en 2 bytes ya que la memoria va desde lo más alto a lo más bajo (y no del revés cómo se entiende con las instrucciones ensambladas). Lo que resulta en reducir el tamaño de la pila de memoria.

PUSH EAX (Establece cómo valor lo almacenado en la EAX)

MOV EAX, EBX (Mueve el valor almacenado de la EBX a la EAX)

POP EBX (Saca el valor almacenado al operando de destino y el ESP crece 2 bytes)

POPF

Aumenta en dos el valor del registro ESP y transfiere los bits de la parte superior de la pila de memoria al registro de **Flags**.

- **0 = CF (Carry Flag)** Indica a la CPU cuando hacer cálculos aritméticos con el punto lógico significativo. Usa un único bit por lo que 0 desactivada dicha opción, 1 activada.
- **2 = PF (Parity Flag)** Comprueba si en la última operación realizada el número de bits (1 en la secuencia) se corresponden a un número par o un número impar. 11010 la bandera de paridad sería 0 ya que existen 3 bits (1) en la secuencia, 01010 sería 1 ya que existe un numero par de bits, dos en total.
- **4 = AF (Adjust Flag)** Indica si la operación de un Carry o Borrow ocurre en los 4 bits significativos menores, se usa principalmente con códigos decimales aritméticos.
- **6 = ZF (Zero Flag)** Cuando el resultado de una operación es 0. Por ejemplo, operaciones que no se almacenan en los resultados (comparaciones, tests de bits).
- **7 = SF (Sign Flag)** Se utiliza cuando la operación resulta en un número negativo (cuando el bit más significativo estaba establecido).
- **8 = TF (Trap Flag)** Causa una interrupción del paso tras cada instrucción ejecutada.
- **9 = IF (Interrupt enable Flag)** Define cuando el procesador debe de reaccionar o no reaccionar a cada interrupción entrante.

- **10 = DF (Direction Flag)** Controla la dirección de las operaciones con cadenas. La operación se realiza desde la dirección más baja a la más alta si el bit está a 0. Y de la más alta a la más baja si está en 1.
- **11 = OF (Overflow Flag)** Suele ser uno de los complementos a la carry flag, se establece cuando el resultado de la operación es demasiado pequeña o demasiado grande para poder "caber" en la operación de destino.

Supongamos que se ha hecho un POPF con a la dirección 0x401426 que equivale a **0x206**, **0000001000000110** en binario.

Bit	15	14	13	12	11	10	**9**	8	7	6	5	4	3	**2**	**1**	0
Valor	0	0	0	0	0	0	**1**	0	0	0	0	0	0	**1**	**1**	0
Flag					OF	DF	**IF**	TF	SF	ZF		AF		**PF**		CF

PUSH

Incluye en alguno de los registros de 32 bits (EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI. Partes de la memoria) o alguno de los valores de la stack de memoria, después reduce el registro ESP en 2 bytes ya que la memoria va desde lo más alto a lo más bajo (y no del revés cómo se entiende con las instrucciones ensambladas). Lo que resulta en aumentar el tamaño de la pila de la memoria.

```
push 0xdebf  (mete el valor 0xdebf a la stack)

pop eax      (la eax es ahora 0xdebf)
```

PUSHAD

Incluye los valores almacenados en los registros EAX. ECX. EDX, EBX, ESP original, EBP, ESI y EDI en la stack. Los registros son almacenados en el orden estricto que se ha indicado.

PUSHFD

Reduce en dos el valor del registro ESP y transfiere el contenido del registro de las banderas de la pila a la dirección indicada por ESP. Las banderas quedan almacenadas en la memoria con el mismo orden usado en POPF.

RDTSC

(Read Time-Stamp Counter), se utiliza para determinar cuántos ticks realiza el procesador desde que se reinició, cargará dicho valor a los registros EDX/EAX. Se utiliza como método de defensa (técnicas anti debuggers).

```
rdtsc                       (obtiene el timestamp actual de los ticks)

xor ecx,ecx                 (ECX pasa a ser cero)

add ecx,eax                 (Salva el timestamp hecho a ECX)

rdtsc                       (Obtiene otro timestamp adicional)

sub eax,ecx                 (Resta el ECX al EAX)

cmp eax,0FFF                (Compara 0FFF con el EAX –timestamp adicional-)

jb short bintext.0041B652 (Bintext equivale a 0FFF, hace un salto condicional
                           si es menos de FFF ticks, lo que permite ejecutar
                           la aplicación –asume que no hay un debugger-)

rdtsc                       (obtiene el timestamp actual de los ticks)

push eax                    (Incluye el resultado al registro EAX).

Retn                        (Esto equivale a un else en python, a una dirección
                           de error para hacer que la aplicación crashee)
```

REPNE

Repite hasta que el ECX sea 0 o la flag ZF sea 1 (Crea un bucle).

REPNZ

Repite hasta que el ECX no sea 0 o la flag ZF no sea 1 (Crea otro tipo de bucle).

RET

Obtiene la dirección de retorno de la stack y salta a esa posición

ROL

Se rotan los bits a la izquierda

```
rol destino, cantidad

mov eax, 0xA       (0xA equivale a 00001010 en binario)

rol eax, 2         (Al rotarse dos posiciones ahora es 00101000, que es 0x28)
```

ROR

Se rotan los bits a la derecha

ror destino, cantidad

mov eax, 0x28 (0x28 equivale a 00101000 en binario)

ror eax, 2 (Al rotarse dos posiciones ahora es 00001010, que es 0xA)

SAHF

Almacena el valor AH en las EFLAGS

AH =	0	0	0	0	0	0	0	1

SBB

Resta los operandos y resta uno al resultado si la flag CF está activa.

sbb destino, origen

Destino = Destino – (origen + CF)

SCAS/SCASB/SCASW/SCASD/SCASQ

Compara una string, byte, word, doubleword o quadword que se ha especificado. Compara el contenido del valor en AL, AX o EAX contra e valor apuntado en ES:EDI. Cuando se utiliza con el prefijo REPNE escanea la cadena buscando en el primer elemento de la cadena cual es igual al valor en el acumulador.

SETLE/STNG

Establece el byt en el operando a 1 si la Zero Flag ha sido establecida o si la Sign Flag no es igual a la Overflow Flag, de no ser el caso establece el operando a 0.

SETZ

Establece el byte en el operando de destino a 1 si la Zero Flag se ha establecido, de no ser el caso el operando es 0.

SGDT

Almacena el contenido de la tabla de registros de descriptores globales (GDTR) en el operando de destino. El operando de destino especifica una dirección de memoria.

FLAG	SF	ZF		AF		PF		CF

SHL

Semejante a ROL

mov eax, 0xA (0xA equivale a 00001010 en binario)

shl eax, 2 (Al rotarse dos posiciones ahora es 00101000, que es 0x28)

SHR

Semejante a ROR

```
mov eax, 0x28      (0x28 equivale a 00101000 en binario)

ror eax, 2         (Al rotarse dos posiciones ahora es 00001010, que es 0xA)
```

SIDT

Escribe los 6 byte del registro de la tabla descriptora de interrupciones (IDT) a la región de memoria especificada.

```
sidt mem48
```

SLDT

Almacena el segmento seleccionado de la tabla descriptora de registros locales (LDTR) en el operando de destino.

SMSW

Almacena la palabra de estado de la máquina en el operando de destino.

STOS/STOSB/STOSW/STOSD

Almacena una cadena (string) del registro AL, AX o EAX a la localización apuntada por ES:EDI. Stosb almacena un byte del registro AL al operando de destino. Stosw almacena una word del registro AX al operando de destino. Stosd almacena una doubleword del registro EAX al operando de destino.

```
mov al,0           (el valor para iniciar el búfer)

lea di,buffer      (La localización inicial del búfer)

mov cx,100         (tamaño del búfer)

cld                (se mueve hacia adelante)

rep stos buffer    (Compara esa línea de ejemplo con stosb)
```

STR

Obtiene del selector de segmentos del registro de tareas, el cual apunta al segmento de estado de tarea con la tarea en ejecución.

SUB

Instrucción para substraer (restar). También modifica las flags ZF y CF

```
sub destino, valor

sub eax, 0x10 (Resta 0x10 de EAX)
```

TEST

Identico a la instrucción AND con la excepción de que no afecta a los operandos, solo afecta al estado de las flags.

test destino, origen

XCHG

Intercambia el valor de dos registros

Xchg destino, valor

Mov eax, 2 (EAX es 2)

Mov edx, 3 (EDX es 3)

xchg eax, edx (EAX es 3 y EDX es 2)

XOR

Realiza una operación lógica de “exclusive OR”

xor destino, valor

mov eax, 0x18 (0000000000**11**000 en binario)

xor eax, 0x7575 (0111010101**10**101 en binario)

RESULTADO (0111010101**01**101)

Hack by Security

We attack for your safety