The Italian CRaCKiNG Encyclopedia vol. 6

Assembly for CRaCKeRS

Scritto da

---==: ALoR :==---
---==> Proud member of NEURO ZONE 2 <==---

Gennaio 1999

Target = *,**

(*=Lamer, **=Novizio, ***=Apprendista, ****=Esperto, *****=CrackMaster)

==========

Table of context :

1.0 Disclaimer.
2.0 I tools necessari.
3.0 Intro.
4.0 L'architettura x86.
5.0 Istruzioni Assembly

6.0 Conclusioni

==========

1.0 DISCLAIMER

Every reference to facts, things or persons (virtual, real or esoteric) are purely casual and involuntary. Any trademark nominated here is registered or copyright of their respective owners. The author of this manual does not assume any responsability on the content or the use of informations retriven from here; he makes no guarantee of correctness, accuracy, reliability, safety or performance. This manual is provided "AS IS" without warranty of any kind. You alone are fully responsible for determining if this page is safe for use in your environment and for everything you are doing with it!

Questa volta non servirebbe.... ma non si sa mai... e poi ormai ci sono affezionato :)

2.0 I TOOLS NECESSARI

Solo la voglia di imparare...e un po' di curiosita' (non fa mai male).
Ma soprattutto un ingrediente fondamentale e':

* Un cervello FUNZIONANTE :))

3.0 INTRO

Probabilmente puo' scoraggiare dove imparare l'assembly ad un programmatore ad alto livello. Ma non ci si deve perdere d'animo e bisogna innanzitutto partire senza pregiudizi. L'assembly ci permette di fare cose che altrimenti potremmo solo sognare usando solo linguaggi ad alto livello.
La CPU controlla tutto cio' che e' collegato ad essa (RAM, video, audio, ecc), una buona conoscenza della sua architettura e programmazione ci permettera' di

controllare appieno anche queste periferiche.

Sfortunatamente per noi la CPU comunica con le periferiche solo attraverso una sequenza di UNO e ZERO (01000001010011000110111101010010). Risulterebbe assai difficile una programmazione di questo tipo. Qui entra in gioco l'assembly. L'assembly e' il linguaggio mnemonico, cioe' usa dei nomi al posto dei codici binari della CPU. In questo modo la programmazione e' piu' "Uman like". Il compilatore e il linker si occuperanno di codificarle nel vero linguaggio macchina binario (1 e 0).

In questo modo l'assembly ci permette di programmare la CPU in ogni sua parte. Ogni sorgente, indipendentemente dal linguaggio, e' prima convertito in assembly poi in linguaggio macchina. Per questo motivo conoscendo l'assembly sapremo che cosa sta facendo un programma passato per esempio attraverso un debugger :)

4.0 L'ARCHITETTURA x86

*** SISTEMA DECIMALE, BINARIO e ESADECIMALE ***

L'uomo e' abituato a ragionare in base 10, anche se poi e' andato ad inventare i secondi che sono 60 in un minuto che si ripete 60 volte in un ora... ecc... Il computer ragiona SOLO in base 2 (vi ricordate 1 e 0...).
Una via di mezzo tra uomo e macchina puo' essere il sistema esadecimale. Esso utilizza 16 simboli: 0 1 2 3 4 5 6 7 8 9 A B C D E F ed e' come vedremo molto piu' compatto dei quello binario.

Decimale. Base 10.

```
   0 =                   10^0*0
   1 =                   10^0*1
  53 =          10^1*5 + 10^0*3
 106 = 10^2*1 + 10^1*0 + 10^0*6
```

Binario. Base 2.

```
   0 =                                                  2^0*0 =       0
   1 =                                                  2^0*1 =       1
  53 =         2^5*1 + 2^4*1 + 2^3*0 + 2^2*1 + 2^1*0 + 2^0*1 =  110101
 106 = 2^6*1 + 2^5*1 + 2^4*0 + 2^3*1 + 2^2*0 + 2^1*1 + 2^0*1 = 1101010
```

Esadecimale. Base 16.

```
   0 =          16^0*0 =  0
   1 =          16^0*1 =  1
  53 = 16^1*3 + 16^0*5 = 35
 106 = 16^1*6 + 16^0*A = 6A
```

CONVERSIONE DI BASE

Decimale --> binario, esadecimale

Dividere il numero da convertire per la base in cui lo si vuole convertire.
Moltiplicare il resto per la base, continuare fino ad arrivare allo 0.

Es:

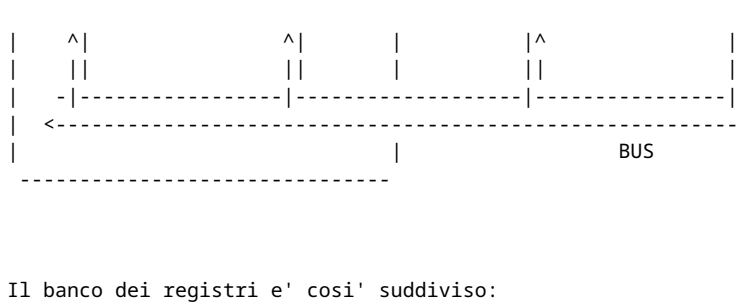
Esadecimale

```
106/16 =  6,625       0.625*16 = A   ( 10 = A in Hex)
  6/16 =  0,375       0.375*16 = 6

   107(dec) = 6B(hex)
```

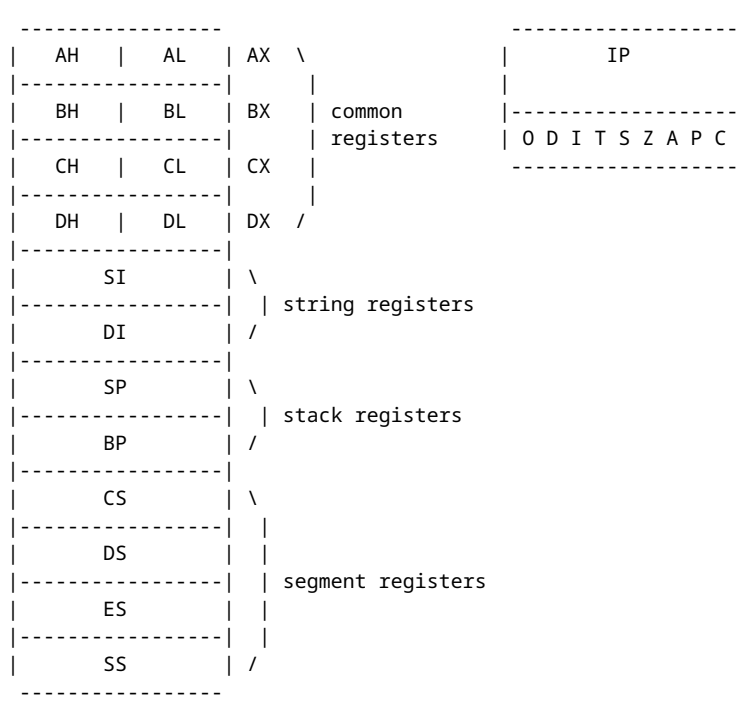
Binario

```
 106/2 = 53                   0
  53/2 = 26.5         0.5*2 = 1
  26/2 = 13                   0
  13/2 = 6.5          0.5*2 = 1
   6/2 = 3                    0
```

```
    3/2 = 1.5          0.5*2 = 1
    1/2 = 0.5          0.5*2 = 1

    106(dec) = 1101010(bin)


Esadecimale --> binario

Spezzare il numero in parti da 4 bit. Convertire le parti secondo la seguente
tabella:

   0000 = 0    0100 = 4    1000 = 8    1100 = C
   0001 = 1    0101 = 5    1001 = 9    1101 = D
   0010 = 2    0110 = 6    1010 = A    1110 = E
   0011 = 3    0111 = 7    1011 = B    1111 = F


Es:

  1000000000111101(bin)

  1000 0000 0011 1101(bin)
  8    0    3    D   (hex)

  803D(hex)



*** BIT, BYTE, WORD e DWORD ***

Un bit e' la piu' piccola porzione di informazione che possiamo ottenere. Il
bit puo' assumere valore 1 o 0. Per la CPU questi valori sono rappresentati
da tensioni diverse: 0 = da 0 a 2.5V    1 = da 2.5 a 5V

8 bit formano un byte. Un byte puo' assumere fino a 2^8 = 256 valori diversi.

00000000 = 2^7*0 + 2^6*0 + 2^5*0 + 2^4*0 + 2^3*0 + 2^2*0 + 2^1*0 + 2^0*0 =   0
00100101 = 2^7*0 + 2^6*0 + 2^5*1 + 2^4*0 + 2^3*0 + 2^2*1 + 2^1*0 + 2^0*0 =  37
11111111 = 2^7*1 + 2^6*1 + 2^5*1 + 2^4*1 + 2^3*1 + 2^2*1 + 2^1*1 + 2^0*1 = 255


16 bit formano una word. Una word puo' assumere 2^16 = 65.536 valori diversi.
Le word di solito sono rappresentate da 4 cifre esadecimali.

0000 = 16^3*0 + 16^2*0 + 16^1*0 + 16^0*0 =      0
1A5D = 16^3*1 + 16^2*A + 16^1*5 + 16^0*D =  6.749
FFFF = 16^3*F + 16^2*F + 16^1*F + 16^0*F = 65.536

32 bit formano una dword. Una dword puo' assumere 2^32 = 4.294.967.296 valori.
Le dword di solito sono rappresentate da 8 cifre esadecimali.

00000000 = 16^7*0 + 16^6*0 + ... + 16^1*0 + 16^0*0 =             0
FFFFFFFF = 16^7*F + 16^6*F + ... + 16^1*F + 16^0*F = 4.294.967.296



*** THE INTEL PROCESSOR ***


Tutta la famiglia dei processori Intel (x86) si basa ed e' compatibile con il
primo processore di questo tipo: l'8086.
L'8086 e' un processore a 16 bit quindi i suoi registri potranno contenere al
massimo 16 bit di informazione. I processore piu' moderni quali 80486, Pentium
e Pentium II sono processori a 32 bit. Intel rilascera' a meta del 2000 anche
il Merced a 64 bit, ma fino allora tratteremo solo fino a registri a 32 bit.
Quindi per poter conoscere come funziona un processore Pentium e' necessario
partire dal suo bis-bis-nonno: 8086.
La struttura interna dell'8086 si presenta grossomodo cosi:



           PROCESSOR                          RAM
 --------------------------------     --------------------------------
|                                |   |              |               |
|  -------         ----------    |   |   Segmento   |   Segmento    |
| |       |       |   Banco  |   |   |              |               |
| |  ALU  | <---- |    dei   |   |   |  Istruzioni  |     Dati      |
| |       | ----> | Registri |   |   |              |               |
|  -------         ----------    |    --------------------------------
```



Il banco dei registri e' cosi' suddiviso:



AX, BX, CX, DZ sono registri a 16 bit; AL e AH, ecc sono registri a 8 bit.
Nei processori successivi al 80386 ci sono anche registri a 32 bit. La loro
denominazione e' semplicissima: basta preporre una E sul registro desiderato.
es: EAX, EBX. EBP, ESP, EIP, ecc.

Vediamo ora a cosa servono i registri della cpu.

AX...DX sono i registri comuni. Sono usati come accumulatori temporanei.

SI e DI sono i registri per le stringhe. Essi sono usati per confrontare,
copiare, spostare le stringhe.

SP e BP sono i registri per lo stack. Lo stack e' un pila (LIFO) di dati.
LIFO vuol dire Last In First Out. Per spiegarci meglio la coda che fate al
supermercato e' FIFO (First In First Out) cioe' il primo che si mette in fila
sara' il primo ad uscire. Lo stack e' il contrario: il primo che entra sara'
l'ultimo ad uscire. Un po' come un cestino per la carta... quando lo svuoti le
prime cose che escono sono le ultime che hai buttato.
Il registro SP (Stack Poiter) punta all'ultimo elemento dello stack (quindi
puntera' al pezzo di carta che hai buttato per ultimo). Per recuperare dati
che si trovano in mezzo allo stack, bastera' spostare questo puntatore.
Il registro BP punta alla base dello stack ed e' usato per i record di
attivazione delle funzioni chiamate.

CS...SS sono i registri di segmento. Vediamo che cosa e' un segmento.

La memoria gestita da un 8086 e' divisa in blocchi da 64 Kb, poiche' puo'
indirizzare solo 16 bit (2^16 = 65536 = 64 K). Il segmento rappresenta
l'indirizzo del blocco utilizzato in quel momento. Per conoscere la posizione
all'interno del segmento, si usa l'offset, rappresentato dal registro IP.
Nei processori a 32 bit i valori cambiano... provate a calcolare quanto si
puo' indirizzare.... cmq le cose si complica un po' e non e' questo l'intento
del tutorial.

Ecco in dettaglio i registri di segmento:

CS   Code segment. Contiene il codice da eseguire.
DS   Data segment. Contiene i dati da trattare: stringhe, variabili, ecc.
ES   come DS ma usato per confrontare le stringhe.

SS   Stack Segment.

IP e' l'instruction pointer. Esso punta alla prossima istruzione che dovra'
essere eseguita.

Le Flags possono assumere valore 1 o 0 (on or off).

C = Carry flag
P = Parity flag
A = Auxilary carry flag
Z = Zero flag
S = Sign flag (+ or -)
O = Overflow flag
I = Interrupt enable
D = Direction flag
T = Trap flag

La piu' importante e' sicuramente la ZERO flag. Essa e' settata dopo ogni
confronto. Assume il valore 0 oppure 1 a seconda del tipo di istruzione usata
per effettuare il confronto.

 5.0 ISTRUZIONI ASSEMBLY

All'inizio di questo tutorial abbiamo parlato delle istruzioni assembly.
Vediamone ora il significato di alcune di esse. (non ne posso piu'... che
palle di manuale... per fortuna siamo quasi alla fine)

 ** CALL **

Utilizzo: CALL destinazione

Es.:   CALL 004DF856   ---> esegue la sub 004DF856

            subroutine:

            004DF856 ...
            004DF857 ...
              ....
            004DFA21 RET   ---> ritorna all'istruzione successiva a CALL

Esegue una subroutine alla "destinazione" e ritorna quando incontra RET.

 ** CMP **

Utilizzo : CMP registro1, registro2
            CMP memoria, registro
            CMP registro, memoria

Es.:   CMP eax, ebx

Confronta due registri e assegna il valore 1 al flag ZF (Zero Flag) se i
registri sono uguali, altrimenti lascia 0.

Varianti: CMPSB   Sorgente e' messo in DS:SI e destinazione in ES:DI.
            CMPSW   Come sopra solo che compara una word al posto di un byte.

 ** DEC **

Utilizzo : DEC registro

Es.:   DEC eax

Diminuisce di 1 il valore del "registro". (vedi anche INC).

 ** DIV **

Utilizzo : DIV registro1, registro2

Es.:   DIV eax, 014

Divide registro1 per registro2. Il risultato viene messo in registro1.

** INC **

Utilizzo : INC registro

Es.:   INC eax

Aumenta di 1 il valore del "registro". (vedi anche DEC).

** INT **

Utilizzo : INT numero_dell_interrupt

Es.:   INT 21

Effettua una chiamata all' Interrupt desiderato.

** Jump **

Utilizzo: Jxx indirizzo

Es.:   Jxx 004056DE

Esistono molti tipi di Jump e, a seconda del tipo le "xx" di Jxx cambieranno.

Opcode	Mnemonic	Meaning	Jump Condition
77	JA	Jump if Above	CF=0 and ZF=0
73	JAE	Jump if Above or Equal	CF=0
72	JB	Jump if Below	CF=1
76	JBE	Jump if Below or Equal	CF=1 or ZF=1
72	JC	Jump if Carry	CF=1
E3	JCXZ	Jump if CX Zero	CX=0
74	JE	Jump if Equal	ZF=1
7F	JG	Jump if Greater (signed)	ZF=0 and SF=OF
7D	JGE	Jump if Greater or Equal (signed)	SF=OF
7C	JL	Jump if Less (signed)	SF != OF
7E	JLE	Jump if Less or Equal (signed)	ZF=1 or SF!=OF
	JMP	Unconditional Jump	unconditional
76	JNA	Jump if Not Above	CF=1 or ZF=1
72	JNAE	Jump if Not Above or Equal	CF=1
73	JNB	Jump if Not Below	CF=0
77	JNBE	Jump if Not Below or Equal	CF=0 and ZF=0
73	JNC	Jump if Not Carry	CF=0
75	JNE	Jump if Not Equal	ZF=0
7E	JNG	Jump if Not Greater (signed)	ZF=1 or SF!=OF
7C	JNGE	Jump if Not Greater or Equal (signed)	SF != OF
7D	JNL	Jump if Not Less (signed)	SF=OF
7F	JNLE	Jump if Not Less or Equal (signed)	ZF=0 and SF=OF
71	JNO	Jump if Not Overflow (signed)	OF=0
7B	JNP	Jump if No Parity	PF=0
79	JNS	Jump if Not Signed (signed)	SF=0
75	JNZ	Jump if Not Zero	ZF=0
70	JO	Jump if Overflow (signed)	OF=1
7A	JP	Jump if Parity	PF=1
7A	JPE	Jump if Parity Even	PF=1
7B	JPO	Jump if Parity Odd	PF=0
78	JS	Jump if Signed (signed)	SF=1
74	JZ	Jump if Zero	ZF=1

** MOV **

Utilizzo: MOV destinazione, sorgente

Es.:   MOV eax, ebx

Copia il valore di [sorgente] in [destinazione].

Varianti: MOVSB   Sorgente e' messo in DS:SI e destinazione in ES:DI.
            MOVSW   Come sopra solo che copia una word al posto di un byte.

** MUL **

Utilizzo : MUL registro1, registro2

Es.:   MUL eax, 014

Moltiplica registro1 per registro2. Il risultato viene messo in registro1.

 ** NOP **

Utilizzo: NOP

Es.:   NOP

Non esegue alcuna istruzione. (e allora a che serve??? :-)
Serve serve... :))) se devi eliminare una call, che fai ?? la cancelli ?? no !
Usi una NOP...

 ** POP **

Utilizzo : POP destinazione

Es.:   POP eax

Toglie dallo stack l'ultimo elemento e lo mette in "destinazione".

 ** PUSH **

Utilizzo : PUSH sorgente

Es.:   PUSH eax
            PUSH 004056FD

Mette in cima allo stack "sorgente".

 ** REP **

Utilizzo : REP

Es.:   REPZ
            CMPSW   <--- istruzione ripetuta

Ripete l'esecuzione di una istruzione su stringhe "while" CX != 0. Ad ogni
passo CX e' decrementato di 1.

Varianti   REPE   Es.: REPE CMPSB compara ogni byte in una stringa "while"
                         sono uguali.
            REPZ   Ripete "while" zero flag e' 0 (off)

"while" ha lo stesso significato del C.
quindi in italiano "while" CX != 0 si potrebbe tradurre "continua finche' CX
diventa 0"

 ** RET **

Utilizzo : RET

Es.:   RET

Ritorna da una procedura. (vedi CALL)

 ** TEST **

Utilizzo : TEST destinazione, sorgente

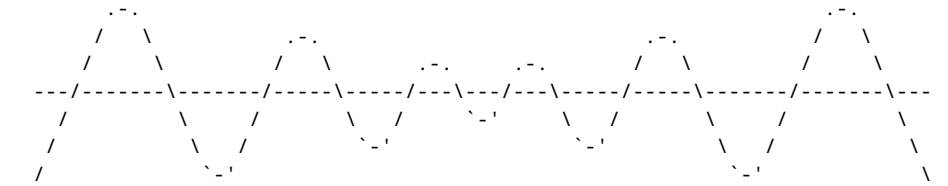
Es.:   TEST eax, eax

Esegue un AND logico tra destinazione e sorgente. Aggiorna i flag ma non salva
il risultato.

 6.0 CONCLUSIONI

Probabilmente il vol 7 sara' dedicato al cracking under Linux. Pero' non e' ancora sicuro. Se avete suggerimenti per il vol 7 contattatemi pure via mail ALoR@thepentagon.com
Per avere eventuali future release di questo manuale scrivete a: nz2@usa.net

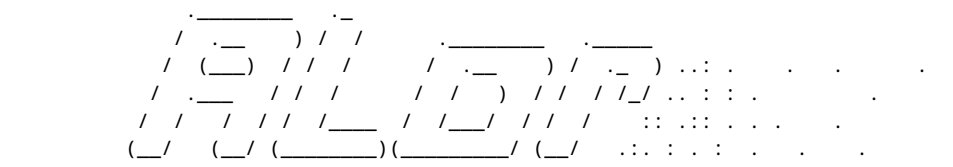
"If you give a man a crack he'll be hungry again
tomorrow, but if you teach him how to crack, he'll
never be hungry again"
+ORC



Greetings to: LordKasKo (my cracking teacher...)

The other NEURO ZONE 2 members (10t8or, DK2DEnd, LordKasKo, MaPHas, Ob1, XXXX, ZenGa)

All the Cracker on the NET from whom I have learned something.



--==> ALoR <==---------- - - -
ICQ: 10666678 In IRC: WhiteFly

e-mail: ALoR@thepentagon.com www: http://ALoR.cjb.net