Directive de definire de date (Data definition directives)

OllyDbg – CODE segment începe la offset-ul 00402000
OllyDbg – DATA segment începe la offset-ul 00401000

Segment data

a1 db 0,1,2,'xyz' ; 00 01 02 'x' 'y' 'z' ; offset(a1) - determinat la încărcarea lui OllyDbg) = 00401000; offset(a1) determinat la asamblare de către NASM = 0 !!!
78 79 7A - codurile ASCII

db 300, "F"+3 ; 2C 49 - 'ascii code F + 3' - Warning – byte data (300 = 1 2Ch) exceeds bounds!

a2 TIMES 3 db 44h ; 44 44 44 ; offset a2 = 00401008, însă offsetul determinat la asamblare de către NASM va fi 8 !!

a3 TIMES 11 db 5,1,3 ; 05 01 03 … de 11 ori (33 octeţi)

a4 dw a2+1, 'bc' ; offset(a2)=00401008h; a2+1=00401009h; deci se generează
09 10 'b' 'c' = 09 10 62 63 (2 cuvinte – words)
a41 dw a2+1, 'b', 'c' ; 'b' 00 'c' 00 = 09 10 62 00 63 00 (3 cuvinte – words)

a42 db a2+1 ; – syntax error – OBJ format can only handle 16 or 32 bits relocation !

09 10 (corect, DAR… această valoare particulară 10h este calculabilă doar DUPA INCARCAREA PROGRAMULUI (LOADING) !!! – deci offset-ul **începuturilor de segmente** este şi el determinabil doar la momentul încărcării programului – LOADING TIME !!!)

Offset-ul variabilelor faţă de începutul segmentelor în care apar sunt constante **(de tip pointer !, NU scalar !)** determinabile la momentul asamblării !!

a44 dw 1009h ; 09 10

a5 dd a2+1, 'bcd' ; 09 10 40 00 | 62 63 64 00
a6 TIMES 4 db '78' ; 37 38 37 38 37 38 37 38
a61 TIMES 4 db '7','8' ; 37 38 37 38 37 38 37 38 !!!
a62 TIMES 4 dw '78' ; 37 38 37 38 37 38 37 38 !!!

a7 db a2 ; syntax err. OBJ format can only handle 16- or 32- relocation (echiv. cu mov ah,a2)

a8 dw a2 ; 08 10

a9 dd a2 ; 08 10 40 00

a10 dq a2 ; 08 10 40 00 00 00 00 00

a11 db [a2] ; - expression syntax error – pt că [a2] NU este o expresie validă acceptată de către asamblor, nereprezentând "o valoare constantă determinabilă la momentul asamblării" ! Dereferenţierea implicată aici este ceea ce deranjează asamblorul !! **Conţinutul unei zone de memorie sau conţinutul unui registru NU sunt valori constante determinabile la momentul asamblării !!!! Acestea sunt accesibile şi determinabile DOAR la momentul execuţiei !! (run-time).**

a12 dw [a2] ; expression syntax error

a13 dd dword [a2] ; expression syntax error

a14 dq [a2] ; expression syntax error

a15 dd eax; expression syntax error

a16 dd [eax]; expression syntax error

mov ax, v ; Warning – 32 bit offset in 16 bit field !!!

Segment code (incepe intotdeauna la offset 00402000 - CINE decide asta ?)

Linkeditorul ia deciziile de acest tip. Adresa de baza pentru incarcarea PE-urilor, cel putin cea implicita (setata de catre linkeditorul de la Microsoft si nu numai) este 0x400000 in cazul executabilelor (respectiv 0x10000000 pentru biblioteci). Alink respecta aceasta conventie si completeaza in campul ImageBase al structurii IMAGE_OPTIONAL_HEADER din fisierul P.E. nou construit valoarea 0x400000. Cum fiecare “segment”/sectiune din program poate prevedea drepturi diferite de acces (codul este executabil, putem avea segmente read-only etc...), acestea sunt planificate sa inceapa fiecare la adresa cate unei noi pagini de memorie (4KiB, deci multiplu de 0x1000), fiecare pagina de memorie putand fi configurata cu drepturi specifice de catre incarcatorul de programe. In cazul unor programe de mici dimensiuni, implicatia este ca se va obtine urmatoarea harta a programului in memorie (la executare):

- programul este planificat a fi incarcat in memorie la exact adresa 0x400000 (insa aici vor ajunge structurile de metadate ale fisierului, nu codul sau datele programului in sine)

- primul “segment” va fi incarcat la 0x401000 (pun ghilimele deoarece nu este un segment propriu-zis ci doar o diviziune logica a programului, nu este asociat direct “segmentul” unui registru de segment – din aceasta pricina se prefera de multe ori denumirea de sectiune in loc de cea de segment)

- al doilea “segment” va fi incarcat la 0x402000 (segmentul pe care il va folosi procesorul pentru segmentare incepe la adresa 0 si are limita de 4GiB, indiferent de adresele si dimensiunile sectiunilor)

- va fi pregatit “segment” (sectiune) de importuri, “segment” de exporturi si “segment” de stack in ordinea decisa de catre likeditor (si de dimensiuni prevazute tot de catre acesta), segmente ce vor fi incarcate de la 0x403000, 0x404000 si asa mai departe (incremente de 0x1000 cat timp au dimensiune suficient de mica, in caz contrar fiind nevoie a se folosi pentru increment cel mai mic multiplu de 0x1000 care permite suficient spatiu pentru continutul intregului segment)

Conform logicii de decizie a adreselor de inceput ale sectiunilor, putem concluziona ca aici avem o sectiune (de date probabil) inaintea celei de cod, continand sub 0x1000 octeti, motiv pentru care codul porneste imediat dupa, de la 0x402000, harta programului fiind la final: metadate (antete) de la 0x400000, cod la 0x401000 si date la 0x402000 (urmat bineinteles de alte “segmente” pentru stiva, importuri si, optional, exporturi).

Segment code (starts always at offset 00402000)

Start:

```
Jmp Real_start    (2 octeţi)  - offset(instr. JMP) = 00402000
a db 17                       - offset(a) = 00402002
b dw 1234h                    - offset(b) = 00402003
c dd  12345678h               - offset(c) = 00402005
```

Real_start:

```
…….
Mov eax, c ; EAX = 00402005
Mov edx, [c]   ; mov edx, DWORD PTR DS:[00402005]
……
Mov edx, [CS:c]   ; mov edx, DWORD PTR CS:[402005]
Mov edx, [DS:c]   ; mov edx, DWORD PTR DS:[402005]
Mov edx, [SS:c]   ; mov edx, DWORD PTR SS:[402005]
Mov edx, [ES:c]   ; mov edx, DWORD PTR ES:[402005]
```

Efectul fiind in toate cele 5 cazuri EDX:=12345678h DE CE ???

Explicaţia este direct legată de modelul de memorie flat – toate segmentele descriu în realitate întreaga memorie, începând de la 0 şi până la capătul primilor 4GiB ai memoriei. Ca atare, [CS:c] sau [DS:c] sau [SS:c] sau [ES:c] vor accesa aceeaşi locaţie de memorie însă cu drepturi de acces potenţial diferite. Deşi toţi selectorii (potenţiali DIFERITI – vezi OllyDbg !!!) indică segmente IDENTICE ca adresă şi dimensiune, aceştia pot avea diferenţe în cum le sunt completate alte câmpuri de control şi de acces ale descriptorilor de segment indicaţi de către ei.

Modelul flat ne asigura ca mecanismul de segmentare este transparent pentru noi, noi nu sesizam diferente intre segmente si, ca atare, scapam complet de grija segmentarii (insa ne intereseaza impartirea logica in segmente a programului, motiv pentru care folosim sectiuni/"segmente" separate pentru cod / date). Acest lucru este valabil insa doar cat timp ne limitam la CS/DS/ES si SS! Selectorii FS si GS indica inspre segmente speciale care nu respectă întotdeauna modelul flat (rezervate interacţiunii programului cu S.O-ul), mai precis, [FS:c] **nu garantează** indicarea aceleiaşi zone de memorie ca şi [CS:c]!

De verificat:

```
Mov edx, [FS:c]   ; mov edx, DWORD PTR FS:[401005]    ?
Mov edx, [GS:c]   ; mov edx, DWORD PTR GS:[401005]    ?
(GS se pare ca merge – ES=SS=DS=GS = 002B !!)
```

FS = 0053 – Olly Dbg "descompune aiurea" aceste 2 ultime instr., deci ele trebuie verificate într-un .EXE care să fie rulat complet separat şi nu "step by step")