

② Када шамо повезивање у време извршавања није директно унапред експлицитно дефинисано неке ф-је користимо из неких DLL-а. Нето, када нам у програму зашреба неки ф-ја из неких DLL-а, позваћемо ф-ју LoadLibrary и затим ћемо позвати ф-ју GetProcAddress да добијемо адресу изражене ф-је и њој ћемо приписати преко указивања на ф-ју (ф-ја GetProcAddress враћа указивач на изражену ф-ју):

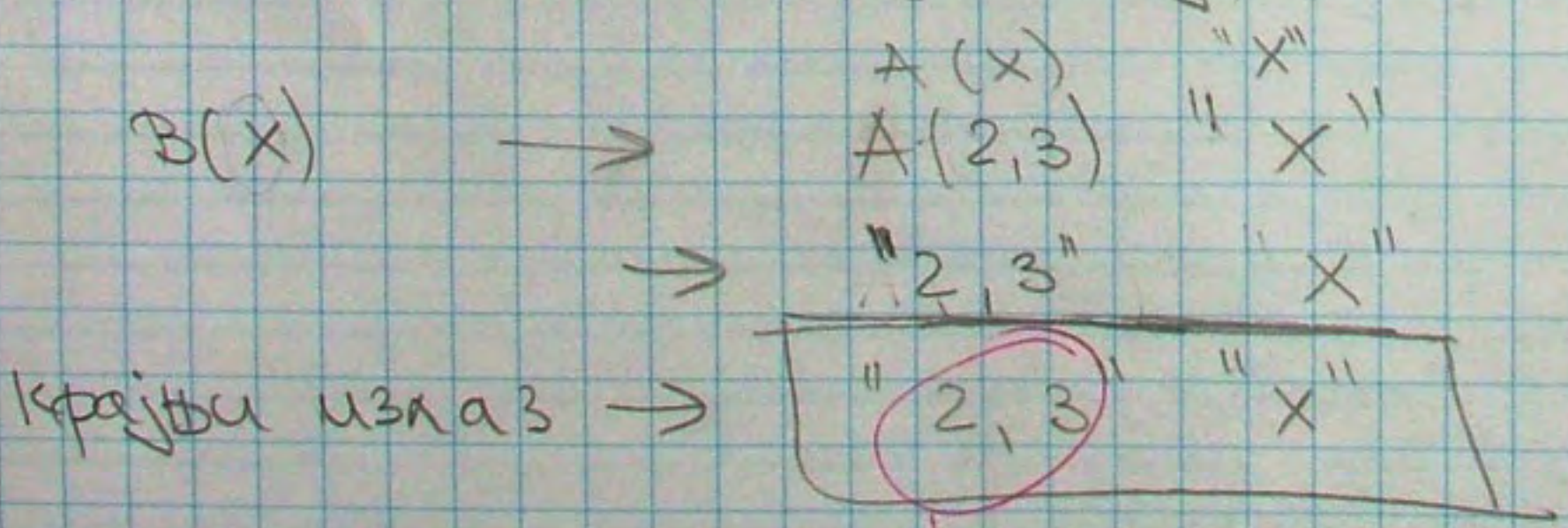
```
hinstance = LoadLibrary("myutils.dll")  
procAddr = GetProcAddress(hinstance, "funcName");
```

Уколико изражени DLL не постоји приликом покретања програма, програм ће се ипак извршити, иако можда у коду програма обрађивати погрешак од прејке ако изражени DLL не постоји.

free lib

1. nestor PROC NEAR ; bliski poziv dr-je
 push BP ; sacuvamo BP na steku
 mov BP, SP;
 mov AX, [BP+4] ; smatramo u AX zadatu vred dr-je
 cmp AX, #0 ; uporedimo sa 0
 je dalje ; ako je 0 idemo dalje
 inc AX; ; povecemo AX
 dalje: pop BP ; restituiramo BP sa steka
 ret ; restituiramo PC
 nestor ENDP ✓

4. #define X 2,3
 #define A(x) #x
 #define B(y) A(y) A(X)



UMESTO OVOGA PRIJAVI
 GREŠKE: "POGREŠAN BROJ PARAMETARA"
 STVARNIH

У првом пролазу ћемо додати В и X у табелу симбола, и до краја првог пролаза за симбол В ћемо израчунавати њену релокацијону вредност и евидентно да је дефинисан у .txt сегменту (што ће вероватно бити први сегмент), а за симбол X ће имаати value и seg бити 0, и назначити да неће дефинисан. Овако ће изгледати табела симбола (за оба 2 симбола):

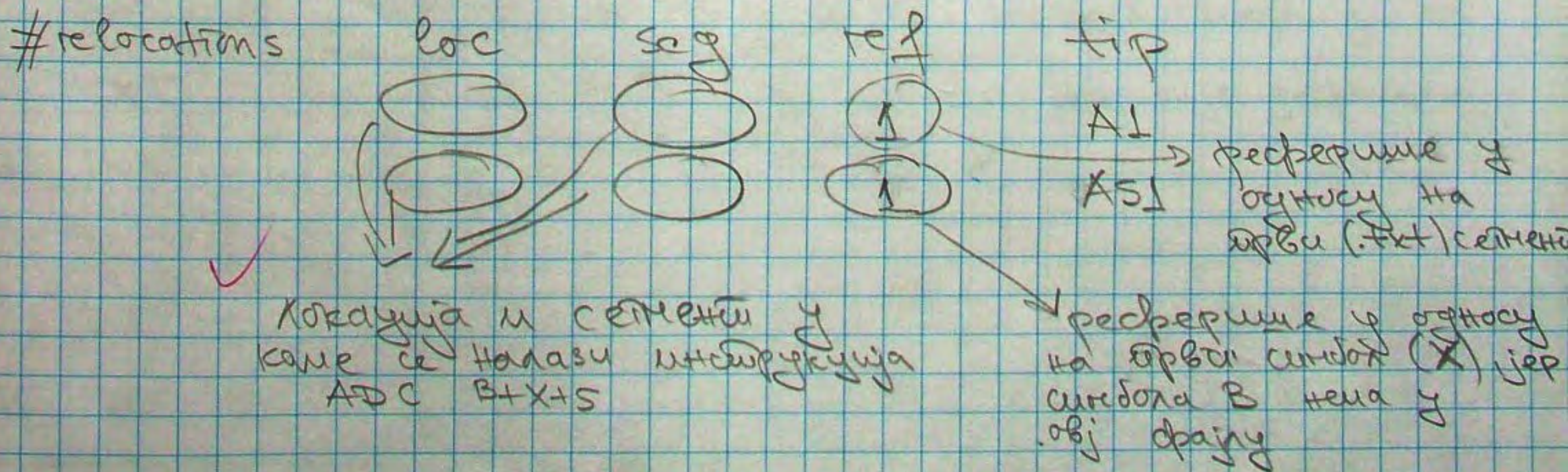
#	symbol	(name	value	seg	def)
	В	В		1	D
	X	X	0	0	U

симбол В не остављан у табели симбола, јер је он локални симбол (приређан нам је за други пролаз, али та у овј фајл не уписујемо)

симбол X ће бити у овј фајлу

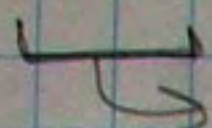
овде ће бити вредности В израчунала.

У другом пролазу ћемо дефинисати релокације на овј локацији. За симбол В ће релокација бити бити А1, јер је он релокабилан у односу на .txt сегмент. За симбол X ћемо имати А51 релокацију јер се његова вредност налази у неким другом модулу.

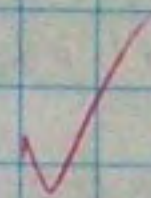


и на крају у #data секцији ће као
машински код за ову инструкцију бити
генерисано

23h



вредности израза B-5



Приликом повезивања овог модула и модула у
коме је дефинисан симбол X, у новом обј
екту неће извршити релокацију за X јер је
већа вредности апсолутна, па не зависи од неких
учињавања модула.

6

0D → INA

18 → HLT

∞ →

Како је вредности симбола А 00 и дефинисан је у првом сегменту, следи да прва инструкција изгледа овако (пошто 0D је за инструкцију INA):

A INA

Ово је једнобајтна инструкција, па зато сада прелазимо на следећу.

18h је пошто за инструкцију HLT и на како није симбол нема вредности која је једнака овој локацији следи да друга инструкција изгледа овако:

HLT

И сада прелазимо на .data сегмент. На локацији 2 у овом сегменту је дефинисан симбол B, а на наредној локацији се присутан симболу C који није дефинисан у овом модулу. Следи да прва инструкција изгледа овако:

B DC C

На крају следи да асемблерски код изгледа овако:

BEG
DEF A, B
USE C
TXT

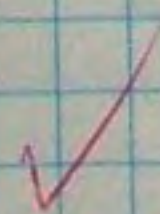
A INA

HLT

DAT

B DC C

—→ нису у табели симбола значи да их
морамо где извозимо
—→ овај симбол увозимо, јер није дефинисан у
нашем модулу



③ Позивanje микротрама код ARM процесора се врши инструкцијом VL. Када уђемо у микротрам директно је на стеку сачувати регистре pc и lr да би у повратку знали где да се вратимо. И при том ће се вредности регистра sp променити за 2, због обих регистра који су стављени на стек. И ако се у току микротрама користи неки регистар као локалне променљиве и њих треба сачувати на стеку пре коришћења. На изласку из микротрама директно је ресетујемо прво све регистре опште намене који су сачувани на стеку, а затим и регистре pc и lr , и тиме се контрола враћа.

Ако се из једног микротрама ^{на стек} позива други микротрам ^{на стек} прво мувају директне вредности регистра pc и lr и затим неке од регистра опште намене ако се користе. При повратку из овог позваног микротрама прво се ресетујемо регистри опште намене који су коришћени у њему и затим регистри pc и lr и тиме се контрола микротрама враћа у први микротрам.

