



Univerza v Ljubljani  
Fakulteta za računalništvo  
in informatiko

Univerzitetni študijski program, 3. letnik

# Sistemska programska oprema

predavatelj: doc. Tomaž Dobravec

**ZBIRNIK (2. del)**

# Prenaslavljanje

---

- ▶ Do sedaj smo v primerih uporabljali fiksen nalagalni naslov
- ▶ Programom, ki uporablja fiksen nalagalni naslov rečemo *absolutni programi*
- ▶ Absolutni programi so redki:
  - ▶ uporabljajo specializiran del pomnilnika
  - ▶ v pomnilniku je le en program (ali točno določeno vnaprej znano število programov, ki vedo eden za drugega)

# Prenaslavljanje

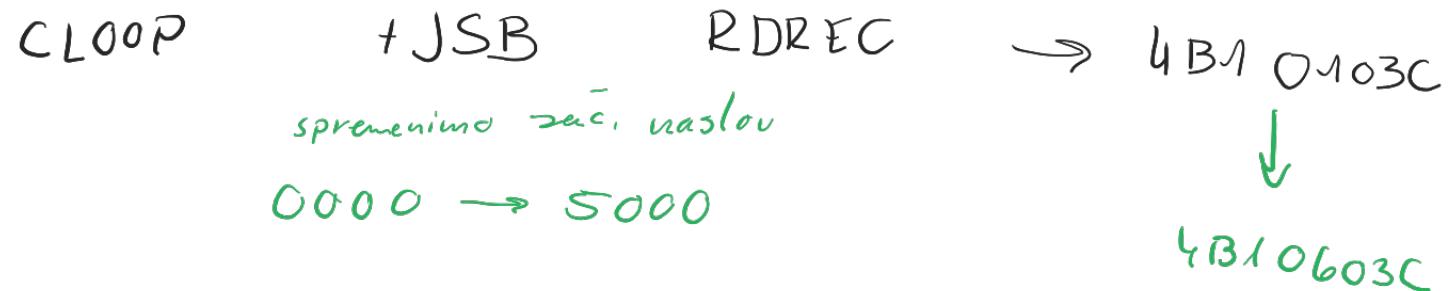
- ▶ Nekateri ukazi programa so “občutljivi na nalagalni naslov”, drugi pa niso.

SIC:  
0000                    LDA                    THREE      → 00102D      To user  
102D                THREE                WORD            3          → 000003      pravik  
pravik  
pravik  
pravik

# Prenaslavljanje

- ▶ Programi običajno “prihajajo” in “odhajajo”
- ▶ Točen (vnaprej znan) nalagalni naslov običajno ni smiseln
- ▶ Predvideti je treba možnost uporabe spremenljivih naslovov

SIC/XE :



# Prenaslavljanje

---

- ▶ **V smislu prenaslavljanja so občutljivi le neposredni (ne-relativni) naslovi!**
- ▶ Zbirnik v večini primerov ne ve, kam se bo program dejansko naložil
- ▶ Zbirnik mora neposredne naslove označiti tako, da jih bo nalagalnik lahko popravil (prištel vrednost nalagalnega naslova)
- ▶ V objektno kodo zbirnik zapiše prilagoditvene (modifikacijske) zapise

# Prenaslavljanje

## Katerih ukazov ni potrebno prenasloviti?

Prenaslavljanje ni potrebno pri ukazih,

- ▶ ki ne vsebujejo naslovov

FIX, HIO, FLOAT, ... (F1)

CLEAR r1, ADD r1,r2, ... (F2)

- ▶ pri katerih je naslov podan relativno (PC-, B- ) (F3)

# Prenaslavljjanje

## Katerih ukazov ni potrebno prenasloviti?

Primer : u programu iz slike 2.I (SIC) se ukaz

20 LDA LENGTH  
// *Ox1036*

prevede v 00.1036

Isti ukaz se v programu 2.5 (SIC/XE) prevede v 03 2 02 6  
↓  
PC-reg

# Prenaslavljanje

## Katerih ukazov ni potrebno prenasloviti?

Naloga: v programih na slikah 2.2 in 2.6 podčrtaj številke vseh vrstic, v katerih so ukazi, ki potrebujejo prenaslavljjanje.

2.2. → program v SIC, ni PC- ali B- rel uaskaljujuča, torej spremenimo vse ukaze z desnimi naslovi

2.4. → +JSVB je treba popraviti

# Prenaslavljanje

Prenaslavljanje z uporabo prilagoditvenih zapisov

**Prilagoditveni zapis (Modification – M) SIC/XE**  
objektne datoteke vsebuje podatke o tem, kateri  
naslove v programu je treba popraviti

Stolpec 1                   M

Stolpec 2-7               Lokacija (relativno glede na  
                                 začetek objektne kode)

Stolpec 8-9               Dolžina naslovnega polja  
                                 (v pol-zlogih)

# Prenaslavljanje

Vemo: v SIC/XE se neposredni naslovi uporabljajo samo v ukazih formata 4 → prenaslavljamo samo take ukaze

► Primer: program na sliki 2.6

M 000007 05

↓

Vduj bo Lc 06?

pri ukazih tipa WORD, EQU...

→ 4B 10,1036

000007

→

M 000014 05 M 000027 05

► Pozor: nekaterih ukazov formata 4 ni potrebno prenasloviti (npr. slika 2.6, ukaz 133 +LDT #4096 )

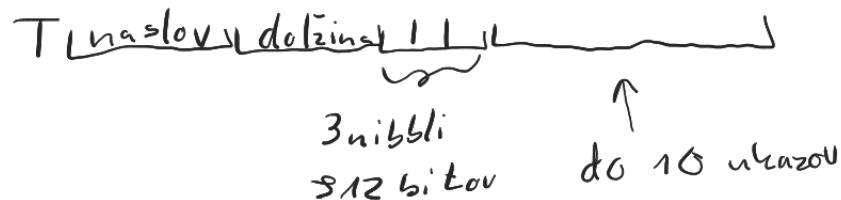
# Prenaslavljanje

Prenaslavljanje z uporabo bitne maske

Prenaslavljanje z uporabo prilagoditvenih zapisov je smiselno v primerih, ko je število prilagodljivih ukazov relativno majhno (kot v prejšnjem primeru).

Če je število prilagodljivih ukazov večje (pri ~~SCI~~ je treba prilagoditi skoraj vse ukaze), se uporablja **prenaslavljanje z bitno masko.**

Masko dodamo v T zapis:



SIC  
Programski (T) zapis  
stolpec 1:T  
stolpec 2-7: naslov  
stolpec 8-9: dolžina (byte)  
stolpec 10-12: prilagoditveni biti  
stolpec 13-72: objektna koda

# Prenaslavljanje

## Prenaslavljanje z uporabo bitne maske

### ▶ Prilagoditveni biti

- ▶ 0: sprememba ni potrebna
- ▶ 1: sprememba je potrebna

Programski (T) zapis

stolpec 1:T

stolpec 2-7: naslov

stolpec 8-9: dolžina (byte)

stolpec 10-12: prilagoditveni biti

stolpec 13-72: objektna koda

### ▶ Uporabljam 12-bitno masko v vsakem T zapisu

- ▶ vsak zapis vsebuje kvečjemu 10 ukazov ( $\rightarrow$  zadnja 2 bita vedno 0)
- ▶ če kateri od ukazov generira 1- ali 2- bajtno kodo, je treba začeti nov T zapis (poravnanost ukazov)
- ▶ biti za neobstoječe ukaze so postavljeni na 0

# Prenaslavljanje

Prenaslavljanje z uporabo bitne maske

- ▶ Primer: program (slika 2.1) in njegova objektna koda z uporabo prilagoditvene bitne maske

```
HCOPY 0000000107A
T0000001E 1400334810390000362800303000154810613C000300002A0C003900002D
T00001E15 0C00364810610800334C0000454F46000003000000
T0010391E 040030000030E0105D30103FD8105D2800303010575480392C105E38103F
T0010570A 1000364C0000F1001000
T00106119 040030E01079301064508039DC10792C00363810644C000005
E000000
```

# Možne napake pri zbiranju

---

- ▶ napačna oznaka stavka  $\rightarrow$  1. faza
- ▶ napačen ukaz  $\rightarrow$  1. faza, optable
- ▶ nedefinirano simbolično ime  $\rightarrow$  2. faza
- ▶ oznaka stavka je večkrat definirana  $\rightarrow$  1. faza
- ▶ napačno število operandov  $\rightarrow$  1. faza, optab
- ▶ simbolna tabela je polna  $\rightarrow$  1. faza
- ▶ napaka v navodilu zbirniku
- ▶ fazna napaka  $\rightarrow$  naslov predalec za PC - ali B - rel uključuje + prevelik za F3

# Dodatne možnosti zbirnika

---

Brez ujih zbirnik deluje, ampak je precej basic.

- ▶ Podpora programskim blokom
- ▶ Kontrolne sekcije
- ▶ Literali
- ▶ Simbolične konstante (podpora ukazu EQU)

# Programski bloki

---

- ▶ Programski blok je del programske kode nekega programa.
- ▶ Primer več-bločnega programa je na sliki 2.11
  - ▶ program je razdeljen na tri bloke:
    - ▶ osnovni / privzeti
    - ▶ CDATA
    - ▶ CBLCK
- ▶ Za uporabo blokov uporabimo ukaz  
USE
- ▶ Privzet (neimenovan) blok se razteza od začetka programa (ali od ukaza USE brez imena) do prvega poimenovanega bloka

# Prednosti kode s programskimi bloki

---

- ▶ Uporaba blokov lahko zelo poenostavi načine naslavljanja
  - ▶ s pravilno uporabo blokov odpade potreba po neposrednem in bazno-relativnem naslavljanju
- ▶ Bloki omogočajo lažje programiranje
  - ▶ vizualno je spremenljivka lahko deklarirana tam, kjer jo programer potrebuje
  - ▶ po prevajanju so vse spremenljivke zbrane na enem mestu
- ▶ Če zbirnik blokov ne podpira, mora “urejanje kode” opraviti programer (“lepo” programiranje)

# Naloga zbirnika pri obravnavi programskih blokov

---

Osnovna naloga zbirnika pri obravnavi programskih blokov: vso kodo posameznega bloka zbrati na enem (neprekinjenem) mestu

# Dodatna funkcionalnost zbirnika za podporo programskim blokom

---

- ▶ Zbirnik dela podobno kot v kodi brez blokov. Razlika:
  - ▶ Za vsak ukaz ve, v katerem bloku je.
  - ▶ Za vsak blok vodi svoj LS.
- ▶ Razreševanje simbolnih imen opravi zbirnik v dveh prehodih (isto kot pri kodi brez blokov!)

# Dodatna funkcionalnost zbirnika za podporo programskim blokom

I. prehod: zbirnik generira dve tabele:

- ▶ razširjena simbolna tabela

Simbolično ime	Vrednost	Blok

--

- ▶ tabela blokov

Ime bloka	Številka bloka	Naslov	Dolžina

# Dodatna funkcionalnost zbirnika za podporo programskim blokom

## Generiranje tabele blokov

Ime bloka	Številka bloka	Naslov	Dolžina

Ko zbirnik zazna ukaz za preklop v blok `ime_bloka`:

- ▶ trenutno vrednost LŠ zapiše v vrstico, ki pripada trenutno aktivnemu bloku (stolpec Dolžina)
- ▶ v tabeli blokov poišče blok z imenom `ime_bloka`
  - ▶ če ne obstaja, naredi novo vrstico,  $Dolžina=0$ ,  $LŠ=0$
  - ▶ če obstaja,  $LŠ = Dolžina$

# Dodatna funkcionalnost zbirnika za podporo programskim blokom

## Generiranje tabele blokov

Ime bloka	Številka bloka	Naslov	Dolžina

Stolpec Naslov se napolni šele po koncu 1. faze zbiranja

$$\text{Naslov}(0) = \emptyset$$

$$\text{Naslov}(i) = \text{Naslov}(i-1) + \text{Dolžina}(i-1)$$

# Dodatna funkcionalnost zbirnika za podporo programskim blokom

Primer: Ustvari obe tabele za kodo 2.11.

Ime simb.	Vrednost	Blok
FIRST	0000	0
CLOOP	0003	0
;		
;		
LENGTH	0003	1
BUFFER	0000	2
INPUT	0000	1

Ime bloka	#	Naslov	Dolžina
(default)	0	0x0	0x66
CDATA	1	0x66	0xB
CBLKs	2	0x71 0x1071	0x1000



# Dodatna funkcionalnost zbirnika za podporo programskim blokom

---

2. prehod: zbirnik uporabi obe tabeli in izračuna vrednost posameznega simboličnega imena

$$\text{LENGTH} = 0x3 + 0x66 = 0x69$$

$$\text{BUFFER} = 0x0 + 0x71 = 0x71$$

⋮

# Generiranje objektne kode v programu s programskimi bloki

- ▶ Objektno kodo izpisujemo po vrsti kot je zapisana v izvorni datoteki (ni potrebe po urejanju!)
  - ▶ T zapis vsebuje tudi nalagalni naslov
  - ▶ ni nujno, da so nalagalni naslovi urejeni po velikosti

H COPY 00000C 001071

COPY { T 000000 1E 1F2063 ... 010003  
T 00001E 09 0F2048 ... 3E203F

PROREC { T 000027 ...  
T 000044 ...

CDATA { T 00006C 01 F1

WRREC { T 00006D ...

# Generiranje objektne kode v programu s programskimi bloki

Vrstni red blokov po posamezni fazi (program 2.11):

Izvorna koda	Objektna datoteka	Po nalačanju
neimenovan(1)	neimenovan(1)	neimenovan(1-3)
CDATA(1)		CDATA (1-3)
CBLKs(1)		CBLKs(1)
neimenovan(2)	neimenovan(2)	
CDATA (2)	CDATA(2)	
neimenovan(3)	neimenovan(3)	
CPATA (3)	CDATA(3)	

# Nadzorne sekcije

---

- ▶ Osnovna ideja: program je lahko sestavljen iz ene ali več med seboj neodvisnih programskih enot – imenujemo jih nadzorne sekcije.
- ▶ Nadzorne sekcije so lahko pisane v eni ali v več datotekah

# Nadzorne sekcije

---

- ▶ V času zbiranja zbirnik ne pozna vseh nadzornih sekcij (pozna samo trenutno), zato ne pozna imen spremenljivk iz drugih sekcij
- ▶ Da ne pride do napak (nedefinirani simboli)
  - ▶ simbole, ki jih izvažamo iz sekcije označimo z EXTDEF public
  - ▶ simbole, ki jih uvažamo iz drugih sekcij, napovemo z EXTREF extern
- ▶ Imena sekcij so po definiciji zunanja imena (izvoz ni potreben)

# Nadzorne sekcije

## Sklicevanje na zunanje reference – primeri prevajanja

15 0003 CLOOP +JSUB RDREC

4B1 00000

160 0017 +STCH BUFFER, X

579 00000

190 0028 MAXLEN WORD BUFEND – BUFFER

000000

Opomba: pri zadnjem ukazu bo treba popraviti vseh 6 pol-zlogov, pri prvih dveh pa le po 5.

# Nadzorne sekcije

Za opis zunanjih referenc so objektni datoteki dodani trije novi zapisi

## I) Definicijski zapis (Definition – D)

Stolpec 1	D	<i>g)</i> LENGTH 00002D
Stolpec 2-7	Ime zunanjega simbola	
Stolpec 8-13	Relativni naslov zunanjega	
Stolpec 14-73	informacije o ostalih simbolih (stolpci 2-13)	

## 2) Referenčni zapis (Reference – R)

Stolpec 1	R	<i>7)</i> <u>RREC</u> WREC
Stolpec 2-7	Ime zunanjega simbola	
Stolpec 8-73	informacije o ostalih simbolih (stolpci 2-13)	

# Nadzorne sekcije

---

Obstoječ prilagoditveni zapis moramo razširiti z dodatnima dvema podatkoma:

## 3) Prilagoditveni zapis (Modification – M)

Stolpec 1	M
Stolpec 2-7	Začetni naslov polja v kodi
Stolpec 8-9	Dolžina polja (pol-zlogi)
Stolpec 10	Smer popravka (+ ali -)
Stolpec 11-16	Ime zunanjega simbola, katerega vrednost je treba prišteti (ali odšteti)

*M 00000465+REC*



# Nadzorne sekcije

## Primer: objektna koda po prevajanju programa 2.16

```
H COPY 000000 001033  
D BUFFER 000033 BUFEND 001033 LENGTH 00002D  
R RDREC WDREC  
T 000000 1D 172027 4B100000 ...  
...
```

M 000004 05 + RDREC

```
...
```

```
H RDREC 000000 00002B  
R BUFFER LENGTH BUFEND  
T 000000 1D B410 .... 57900000 ...  
...
```

M 000018 05 + BUFFER  
M 000028 0C + BUFEND  
M 000028 0C - BUFFER

```
E
```

```
H WRREC ...  
...
```

```
E
```

# Literali

- ▶ Literal je konstanta brez imena (anonymous constant)
- ▶ Literal uporabljamо kot konstanto neposredno v zbirniškem ukazu

WRREC	CLEAR X	
	LDT LENGTH	TD = X'05'
WLOOP	TD OUTPUT	
	JEQ WLOOP	
	LDCH BUFFER, X	↙ Moramo podati v tej obliki, zbirnik vedeli dolžino številke v bajbih
	WD OUTPUT	WD = X'05' saj rabi
	TIXR T	
	JLT WLOOP	
	RSUB	
OUTPUT	BYTE X'05'	

# Literali

---

- ▶ v prevedenem programu se uporaba literalov ne vidi
- ▶ zbirnik za literal rezervira prostor in se nanj sklicuje z enostavnim naslavljanjem
- ▶ oba programa na prejšnji prosojnici (z in brez uporabe literala) se prevedeta popolnoma enako
  - ▶ za literal X'05' zbirnik rezervira prostor tik pod ukazom RSUB
  - ▶ ukaza TD OUTPUT in TD =X'05' se prevedeta v E320II

# Literali

Primer: Napiši program LIT, in ga poženi v sic-vm

LIT	START	0
	LDA	=5
	LDB	=4
	ADDR	A, B
	STB	C
C	RESW	1

→ Tukaj bo zbirnik ustvaril zapisa za literala 5 in 4

# Literali

---

- ▶ Razlika med uporabo literalov in takojšnjim naslavljjanjem
  - ▶ pri takojšnjem naslavljanju se operand “zapeče” v strojni ukaz
  - ▶ uporaba literalov se prevede v enostavno naslavljanje.
- ▶ Zbirnik ustvari zaloge literalov *(določimo eno izmed možnosti)*
  - ▶ na koncu programa (če je program kratek)
  - ▶ po ukazu J ali RSUB
  - ▶ LTORG → programer sam pove, kje želi zaloge literalov

# Literali

## Podvojeni literali

---

### Dober zbirnik prepozna podvojene literale

- ▶ če so zapisani na enak način ali
- ▶ če so zapisani različno  
(pri `=C'EOF'` in `=X'454F46'` gre za isti literal)
  
- ▶ za enake literale rezervira le en prostor,
- ▶ za sklicevanje na enak literal uporablja isti naslov.

# Literali

## Razreševanje literalov

- ▶ Za razreševanje zbirnik uporabi tabelo literalov (LITTAB)

Koda	Literal	Dolžina	Tip	Naslov
= C 'EOF'	EOF	3	C	Lš
= X '05'	5	1	X	Lš + dolžina (0)
= h	4	3	X	
= x '00000d4'				

- ▶ implementacija: zgoščena tabela

$$naslov(0) = Lš$$

$$naslov(i) = naslov(i-1) + dolžina(i-1)$$

# Literali

Tvorjenje in uporaba tabele LITTAB

---

## I. prehod zbirnika

- ▶ za vsak najdeni literal
    - ▶ če literal v tabeli že obstaja, ga zbirnik ignorira
    - ▶ če ne obstaja, določi dolžino in tip ter ga vpiše v tabelo;  
polje Naslov pusti prazno
  - ▶ ko pride do ukaza LTORG ali do konca programa, določi vrednost stolpca Naslov v tabeli po formuli
- 

## 2. prehod zbirnika

- ▶ vsak najdeni literal zamenja z naslovom iz tabele LITTAB
- ▶ ko pride do ukaza LTORG ali do konca programa, vse literale iz tabele prepiše v objektno kodo (enako kot WORD ali BYTE)

# Simbolične konstante

---

Od uporabnika definirana imena:

✓ Oznake

<b>LOOP</b>	ADD D
<b>X</b>	WORD 5

? Simbolične konstante

<b>STO</b>	EQU 100
------------	---------



# Primer uporabe konstante

---

Namesto

+LDT                  #4096

lahko pišemo

MAXLEN	EQU	4096
	+LDT	#MAXLEN

- ▶ zbirnik v simbolno tabelo zapiše vrednost simbola MAXLEN
- ▶ drugi stavek prevede s pomočjo vrednosti iz simbolne tabele
- ▶ pozor: prevoda obenem sta popolnoma enaka!
- ▶ prednost: čitljivost kode

# Uporaba simbolov

---

Z uporabo EQU lahko simboličnemu imenu pridemo

- ▶ konstantno vrednost ali vrednost oznake,

ALFA EQU 5

- ▶ enačimo vrednosti dveh simboličnih imen ali

BETA EQU ALFA

- ▶ novemu simbolu pridemo vrednost aritmetičnega izraza (v katerem nastopajo konstante, oznake in simbolična imena).

GAMA EQU BETA - ALFA + 3



# Uporaba simbolov

---

Z ukazom EQU lahko simbolnemu imenu priredimo tudi trenutno vrednost lokacijskega števca; izraz \* v tem primeru pomeni naslov naslednje pomnilniške lokacije.

BUFFER	RESB	4096
BUFEND	EQU	*
MAXLEN	EQU	BUFEND - BUFFER



# Izrazi

V izrazih pri EQU ukazu lahko nastopajo konstante, oznake in simbolična imena (definirana z drugim EQU).

Izrazi so

- ▶ relativni (vrednost izraza je odvisna od LŠ)
  - ▶ absolutni (vrednost izraza je neodvisna od LŠ)

# Izrazi

Izraz je absoluten, če je sestavljen iz samih absolutnih komponent ali če njegove relativne komponente nastopajo obratno predznačenih parih.

zacetni  
najtgalni  
v naslov

BUFFER ...  $S + L \tilde{S}(r1)$   
BUFEND ...  $S + L \tilde{S}(r2)$

MAXLEN EQU BUFEND - BUFFER

Relativne komponente v izraz ne smejo biti povezane z operatorji za množenje in deljenje.

# Izrazi - primer

(A) MAXLEN EQU BUFFEND - BUFFER

(R) ENDP EQU ADDLP + 20

*ADDLP ...*

Izrazi      ENDP    EQ V    ADDLP + 20

---

Za **relativen izraz** običajno velja, da vse njegove komponente (**razen ene**) nastopajo v obratno predznačenih parih, edina komponenta, ki ni v paru, pa mora biti s pozitivnim predznakom in ne sme nastopati v kombinaciji z množenjem ali deljenjem.

$$x \quad EQ V \quad \underline{A - B} + \underline{C - D} + \textcircled{E}$$

► Kaj ostane?

---

Če je relativni izraz sestavljen drugače, gre verjetno za napako.

# Simbolična imena – vnaprejšnje sklicevanje

Primer:

BETA	EQU	ALFA
ALFA	RESW	1

(BETA lahko razrešim v drugem prehodu)

A EQU B  
B EQU D-G  
D RESW 1  
G EQU 1024



# Simbolična imena – vnaprejšnje sklicevanje

---

Osnovno pravilo dvoprehodnega zbirnika: po koncu prvega prehoda morajo biti znane vse vrednosti simboličnih imen.

Težavo vnaprejšnjega sklicevanja rešimo na dva načina:

- ▶ prepovemo vnaprejšnje sklicevanje
- ▶ uporabimo postopek za razreševanje vnaprejšnjega sklicevanja v enem prehodu.

# Razreševanje vnaprejšnjega sklicevanja.

- Potrebujemo tabelo vnaprejšnjih referenc

- Primer:

BETA	EQU	ALFA +5	
ALFA	EQU	10	
Simbolično ime	Odvisnost	Referenca	Odvisniki
BETA	1	ALFA +5	null
ALFA	0	10	BETA -> null

SIMTAB:

↑  
#nerazrešenih simbolov v izrazu  
izraz za izračun  
seznam simbola, ki so odvisni od mene

Cilj: vse vrednosti v stolpcu Odvisnost so enake 0

# Razreševanje vnaprejšnjega sklicevanja.

Delovanje algoritma predstavimo na spodnjem primeru:

```
HALF      EQU      LEN/2  
LEN       EQU      END-BUF  
BEF       EQU      BUF-1  
...  
0x1052    BUF      RESB    10000 4096  
END       EQU      *
```

Simbol	Odrisnost	Referenca	Odrisniki
① HALF	1	LEN/2 0x0800	null
LEN	2 2 1	END-BUF END-0x1052 0x1000	HALF → null null
② END	2 0	2 0x2052	LEN → null null
BUF	2 0	2 0x1052	LEN → null LEN → BEF → null null
BEF	1 0	BUF-1 0x1051	null
④ ⑤			

# Razreševanje vnaprejšnega sklicevanja.

Kako zbirnik obravnava vrstico SIM EQU REF?

① if not  $\text{SYM}$  in  $\text{SYMTAB}$ :

$\text{SYMTAB.insert}(\text{SYM}, ?, ?, \text{null})$

$\text{dep} = \text{number of symbols in ref}$

$\text{SYMTAB}[\text{SYM}]. \text{odvisnost} = \text{dep}$

$\text{SYMTAB}[\text{SYM}]. \text{referenca} = \text{ref}$

for each symbol in ref:

if symbol in  $\text{SYMTAB}$ :

$\text{SYMTAB}[\text{symbol}]. \text{odvisniki}. \text{add}(\text{SYM})$

else:

$\text{SYMTAB.insert}(\text{symbol}, ?, ?, \text{SYM} \rightarrow \text{null})$

end for

$\text{resolve}(\text{SYM}, \text{null})$

$\text{resolve}(\text{s}, \text{o}) :$

if ( $\text{o} \neq \text{null}$ )

$\text{SYMTAB}[\text{s}]. \text{referenca}. \text{insert}(*\text{o})$

$\text{SYMTAB}[\text{s}]. \text{odvisnost} --$

if ( $\text{SYMTAB}[\text{s}]. \text{odvisnost} == 0$ )

foreach  $d \in \text{odvisniki}(\text{s})$

$\text{resolve}(d, \text{s})$

# Razreševanje vnaprejšnjega sklicevanja.

---

Kaj stori zbirnik, ko določi vrednost levega naslova (oznaka stavka ali naslov spremenljivke)?



# Ciklične reference

## ▶ Po prebrani kodi

A EQU B

B EQU C

C EQU A

dobim tabelo:

Simbol	odv	ref	odvisnici
A	1	B	C->null
B	1	C	A->null
C	1	A	B->null

→ sprožimo

napako