



Univerzitetni študijski program, 3. letnik

Sistemska programska oprema

predavatelj: doc. Tomaž Dobravec

ZBIRNIK (1. del)

Kazalo

- ▶ **Zbiranje – uvod**
- ▶ Delovanje zbirnika
- ▶ Objektni moduli

O zbiranju – splošno

- ▶ Vsak procesor razume le svoj **strojni jezik**
- ▶ Pisanje v strojnem jeziku je zamudna naloga → uporabljamo **zbirni jezik**.
- ▶ Zbirni jezik = linearna preslikava strojne vsebine.
- ▶ Zbirni jezik neločljivo povezan s strojno opremo.

Uporaba zbirnih jezikov

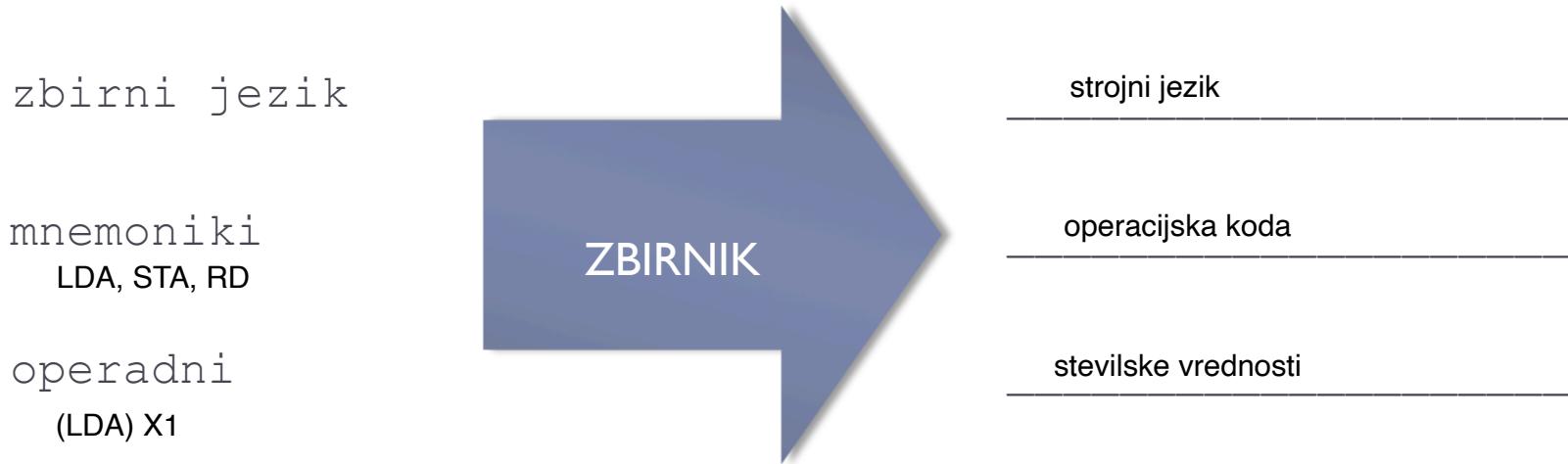
Danes zbirni jezik uporabljam

- ▶ pri sistemskem programiranju,
- ▶ za pisanje ključnih delov posameznih programov,
- ▶ za reševanje zahtevnih, časovno potratnih problemov,
- ▶ ...

Zbirni jezik je lahko ena od vmesnih stopenj prevajanja višjenivojskih programskega jezikov.

Glavna naloga zbirnika

- ▶ Zbirnik prevede programe, pisane v zbirnem jeziku, v strojni jezik



Specializirane vrste zbirnikov

- ▶ **modularni zbirnik**

- ▶ **mikro zbirnik**

za mikro mrocesorske ukaze, LDA je sestavljen iz vecih mikro ukazou

- ▶ **makro zbirnik**

omogoca macro-je razlika s funkcijo je da se ukazi vstavijo na mesto macro ukaza

- ▶ **meta zbirnik**

- ▶ **prečni zbirnik**

- ▶ **zbirnik “naloži in izvedi”**

<zbirniški stavek> ::=

[<oznaka stavka>] <locilo> <mnemonic> {<locilo> <operand>} [<locilo> <komentar>]

[] - opcijsko

{ } - nic ali vec

Primer:

ADDLP

oznaka stavka

ADDR

mnemonic

X, S

locilo, operand 2x

S = S + X

locilo komentar

```
<zbirniški stavek> ::= [<oznaka stavka>] <ločilo>  
    <mnemonik> <ločilo>{<operand> <ločilo>} [<komentar>]
```

- ▶ <mnemonik> - smiselna okrajšava za strojni ukaz
- ▶ <ločilo> - dogovorjeni znaki, izbrani tako, da je delo pregledovalnika čim bolj enostavno.

[_, \t] +

Zbirniški stavek

3/4

```
<zbirniški stavek> ::= [<oznaka stavka>] <ločilo>  
    <mnemonik> <ločilo>{<operand> <ločilo>} [<komentar>]
```

▶ <oznaka stavka> - simbolično ime, ki se lahko pojavlja na dveh mestih:

▶ na zacetku vrstice - levi naslov

▶ desno od mnemonika, kot operand - desni naslov

Zbirniški stavek

3a/4

```
<zbirniški stavek> ::= [<oznaka stavka>] <ločilo>  
    <mnemonik> <ločilo>{<operand> <ločilo>}  
    [<komentar>]
```

Primer: računanje produkta $x_1 * x_2$

leva roka ma na desni strani palec

vnaprejsnje sklicevanje - ena najtezjih nalog zbirnika



Zbirniški stavek

4 / 4

```
<zbirniški stavek> ::= [<oznaka stavka>] <ločilo>  
    <mnemonik> <ločilo>{<b>operand</b>} <ločilo>} [<komentar>]
```

▶ <operandi> so lahko:

▶ simbolicno ime (x1, endl, addlp)

▶ ime konstante (EQU)

▶ stevilska vrednost (#1, #0,..)

Zbirnik mora opraviti naslednje naloge:

- I) Pregledati in razčleniti zbirniški stavek (pregledovalnik)
- 2) Mnemonike nadomestiti z operacijskimi kodami

FIRST STL RETADR => FIRST 14 RETADR

3) Simbolična imena nadomestiti z ustreznimi številskimi ekvivalenti

1000 FIRST 14 RETADR => 1000 14 1033 (SIC)

.

.

.

1033 RETADR RESW 1

4) Razrešiti simbolične operande

BUFFER RESB 4096

BUFEND EQU *

MAXLEN WORD BUFEND-BUFFER

zbirnik to lahko izracuna takoj, lahko pa bi se zgodilo da se sklicujemo v naprej in mora zant ravnat prav tudi v tem primeru

5) Pretvorba konstant iz izvirne oblike v notranjo strojno predstavitev.

EOF BYTE ‘EOF’ --> 45 4F 46



6) Pravilno prevede in interpretira “psevdo ukaze”

- ▶ psevdo ukazi (direktive), kot so START, END, BYTE, WORD, RESW, RESB, se ne prevedejo v strojni ukaz

/ NOBENE KODA - SAMO ZA ZBIRNIK, DA VE KAKO OSTEVILCIT NASLEDNJI UKAZ **SIC object code**

1.	P1	START	0000	
2.	0000	X1	WORD	5 000005
3.	0003	X2	WORD	7 000007
4.	0006	R	RESW	1 /
procesor zacne izvijat program tuki				
5.	0009	PROG	LDA X1	00 0000
6.	000C		ADD X2	18 0003
7.	000F		STA R	0C 0006
8.	0012	HALT	J HALT	3C 0012
9.		END	PROG	/

- 7) **Tvori strojne ukaze v predpisani obliki in izpiše kodo.**
- 8) **Pripravi in izpiše listo prevajanj**
 - ▶ vhodni izvorni (zbirni) stavek,
 - ▶ objektna koda,
 - ▶ številske vrednosti, ...
- 9) **Tvori množico tabel (simbolna tabela, tabela sekcij, tabela prečnih referenc, ...).**

10) Razpozna vrsto specialnih možnosti

- ▶ parametrično voden zbirnik
- ▶ uporaba direktiv in stikal
- ▶ določi se lahko način izpisa, privzet številski sistem, format izhodne datoteke, ...

Uporaba lokacijskega števca

- ▶ Zbirnik za vsak ukaz ugotovi, koliko pomnilnika zasede.
- ▶ Trenutno zasedenost pomnilnika vodi v **LOKACIJSKEM ŠTEVCU (LŠ, angl. LOCCTR)**.
- ▶ LŠ v času zbiranja odigra podobno vlogo kot

PC

- ▶ Pregledovalnik bere vhodne vrstice in jih razgradi na posamezne enote.
- ▶ Razpoznane elemente okvalificira (oznaka / mnemonik / operand / komentar)
- ▶ Pregledovanje je časovno potratna operacija.
- ▶ Lahko je implementirano kot končni avtomat.

- ▶ Izraze, ki jih pregledovalnik opredeli kot identifikatorje, obravnava naprej:
 - ▶ če gre za desni naslov, preveri prisotnost v tabelah;
 - ▶ če gre za levi naslov, zabeleži v simbolni tabeli.
- ▶ Pregledovalnik: bralna rutina + konstruktor tabel
- ▶ Za vsak simbol pregledovalnik sintaktičnemu analizatorju sporoči tip in stanje.

Tabele zbirnika

- ▶ Za pravilen potek zbiranja potrebujemo vsaj dve tabeli:
 - ▶ ukazna tabela
 - ▶ tabela mnemonikov in navodila zbirniku
- ▶ simbolna tabela
- ▶ vsi levi in desni naslovi
- ▶ Tabeli sta po naravi in načinu uporabe zelo različni.

- ▶ Ukazna tabela – **UKTAB** (angl. operation code table; optab)
- ▶ **UKTAB** vsebuje seznam mnemonikov s pripadajočimi operacijskimi kodami, tipom ukaza in številom parametrov
- ▶ Primer: ukazna tabela za SIC (imamo na listih)

Mnemonik	Operacijska koda	Tip ukaza	Število operandov
LDA	00	3 / 4	1
FLOAT	C0	1	0
DIVR	9C	2	2
CLEAR	B4	2	1
...

▶ Poleg mnemonikov v UKTAB pišemo še

- ▶ imena registrov, direktive - zbirniske ukaze (word, byte start,...)



▶ Uporaba UKTAB:

- ▶ določanje velikosti prevedenega ukaza, da pravilno določimo lokacijski stevec

- ▶ za pretvorbo mnemonikov v strojno kodo

- ▶ UKTAB je lahko “zapečena” v sam zbirnik, lahko pa se naloži ob začetku izvajanja
- ▶ UKTAB je statična tabela, se ne spreminja

edina operacija je find
urejena staticna tabela

pri metazbirniku je to input, ni fiksna

Simbolna tabela

- ▶ Simbolna tabela – **SIMTAB** (angl. SYMTAB)
- ▶ Na začetku zbiranja je SIMTAB prazna!
- ▶ Pri zbiranju simbole vpisujemo v SIMTAB skupaj z njihovo vrednostjo (trenutna vrednost LS ali konstanta pri EQU)
find, doddaj
- ▶ Običajna implementacija: hash map

ce ne ve vrednost simbola, zapise le simbol in gleda naprej, ko se pojavi na levi strani pa zapise naslov

ime simbola		vrednost
X1		001D
ENDL		001A
X2		0020
ADDLP		000F

Simbolna tabela - primer

vrednost LŠ	ukaz	format ukaza
	LDS #0	
	LDA X1	
	COMP #0	
	JLT ENDL	
	LDX X2	
ADDLP	ADDR X, S	
	SUB #1	
	COMP #0	
	JGT ADDLP	
ENDL	RSUB	
X1	WORD 7	
X2	WORD 5	

Simbolično ime	Vrednost



Uporaba simbolne tabele

▶ Uporaba SIMTAB: za razreševanje simboličnih imen

▶ Potek razreševanja:

- ▶ _____
 - symbol desna stran - zapisi v tabelo simbol
 - symbol leva stran - zapisi v tabelo simbol po potrebi in naslov simbola
-

▶ MOŽNE TEŽAVE:

- ▶ _____
 - naslov ni definiran - uporabljen le na desni strani
 - ▶ _____
 - levi naslov se pojavi dvakrat
-

Reševanje problema “*desni naslov pred levim*”

▶ Problem “*desni naslov pred levim*” rešimo na različne načine:

▶ s pomočjo enoprehodnega zbirnika

▶ s pomočjo dvoprehodnega zbirnika - bolj pogosto

▶ Z enoprehodnim zbirnikom problem “*desni naslov pred levim*” rešimo na enega od treh načinov:

▶ omejevanje programerja

▶ turbo princip

▶ naslavljjanje s simbolno tabelo

Ka) Omejevanje uporabnika

- ▶ Zahteva: levi naslov se mora vedno pojaviti pred desnim
- ▶ Nekateri zbirniki omejujejo le vnaprejšnja sklicevanja na podatke

velik prevelika zahteva, ki posega v samo logiko programa

Kb) Turbo princip (“naloži in izvedi”, angl. *load-and-go*)

- ▶ Predpostavka: vsa orodja (zbirnik/povezovalnik/nalagalnik) in celoten preveden program so ves čas v delovnem pomnilniku

zbirnik gre cez vrstice in generira direktno v pomnilnik objektno kodo,
ko pride do konca rece jump na zacetek in program se zacne izvajat

- ▶ Ideja:
 - zakaj to resi problem: zapomni si mesta kjer rabi dodat naslov se nerazresenega naslova
ko pride do naslova takega simbola se napise naslov na prizerno mesto
 - tezava: vecji programi, moduli -> preveliki programi da bi jih sprot preverjal

Kc) Naslavljjanje s pomočjo simbolne tabele

- ▶ **Ideja:**
 - ▶ vsako ime (levo in desno) vpišemo v simbolno tabelo,
 - ▶ namesto na simbol se sklicujemo na vpis v simbolni tabeli,
 - ▶ celotno simbolno tabelo pridružimo prevedenemu programu.

- ▶ **Slabosti:** ena stopnja posrednosti vec iz neposrednega naslavljanja pridemo v posredno naslavljanje

Kc) Naslavljanje s pomočjo simbolne tabele

Primer:

LŠ	koda programa			popravljena koda programa
000000		LDA	#ENA	LDA ST_ENA
000003		J	LOOP	J @ST_LOOP
000006	ZANKA	ADD	#ENA	ADD ST_ENA
000009	LOOP	J	ZANKA	J ZNKA ST_ENA WORD 000001 ST_LOOP WORD 000009
ENA	EQU	1		

Dvoprehodni zbirnik

- ▶ Z dvoprehodnim zbirnikom problem “desni naslov pred levim” rešimo tako, da kodo beremo dvakrat:
 - ▶ pri prvem branju

 - ▶ pri drugem branju

Dvoprehodni zbirnik – 1. prehod

IZVORNA VRSTICA je lahko:

- ▶ UKAZ v zbirnem jeziku
- ▶ NAVODILO zbirniku

Rezervacija (RESB, RESW, WORD, BYTE)

Izenačitev, konstanta (EQU)

Izhodišče (ORG)

Konec (END)

Dvoprehodni zbirnik – 2. prehod

IZVORNA VRSTICA je lahko:

- ▶ **UKAZ** v zbirnem jeziku
- ▶ **NAVODILO** zbirniku

Rezervacija (RESB, RESW)

Rezervacija z inicializacijo (WORD, BYTE)

Izenačitev, konstanta (EQU)

Izhodišče (ORG)

Konec (END)

UKAZ:

iz OPTAB preberi operacijsko kodo

tip ukaza

dolzino kode

iz SIMTAB pulomi vrednost operanda (desnega naslova)
generira in izpisi strojno kodo

LS = LS + L

REZERVACIJA: (RESW, RESB)

LS = LS + dol. rez.

rezervacija + inicializacija (WORD,BYTE)
generira objektno kodo (vrednost konstante)
LS = LS + dol. rez.

EQU:

//

ORG:

LS = vrednost_operanda

END ... konec



Komunikacija med 1. in 2. prehodom

- ▶ Poznamo dva načina komunikacije med obema prehodoma:
- ▶ v obeh prehodih beremo izvorno kodo, edina komunikacija med prehodoma je simbolna tabela
- ▶ v prvem prehodu beremo originalno izvorno kodo in poleg simbolne tabele ustvarimo še vmesno datoteko, ki poleg izvirne kode vsebuje še
 - ▶ naslove posameznega ukaza v ukazni tabeli,
 - ▶ dodatne že razvozlane informacije o posameznem ukazu,
 - ▶ naslove simbolov v simbolni tabeli;
- ▶ drugem prehodu beremo le vmesno datoteko.

Dvoprehodni zbirnik - 1. prehod; psevdokoda

Pass 1:

```
begin
    read first input line
    if OPCODE = 'START' then
        begin
            save #[OPERAND] as starting address
            initialize LOCCTR to starting address
            write line to intermediate file
            read next input line
        end {if START}
    else
        initialize LOCCTR to 0
    * while OPCODE ≠ 'END' do
        begin
            if this is not a comment line then
                begin
                    if there is a symbol in the LABEL field then
                        begin
                            search SYMTAB for LABEL
                            if found then
                                set error flag (duplicate symbol)
                            else
                                insert (LABEL,LOCCTR) into SYMTAB
                        end {if symbol}
                    2   search OPTAB for OPCODE
                    if found then
                        add 3 (instruction length) to LOCCTR
                    else if OPCODE = 'WORD' then
                        add 3 to LOCCTR
                    else if OPCODE = 'RESW' then
                        add 3 * #[OPERAND] to LOCCTR
                    else if OPCODE = 'RESB' then
                        add #[OPERAND] to LOCCTR
                    else if OPCODE = 'BYTE' then
                        begin
                            find length of constant in bytes
                            add length to LOCCTR
                        end {if BYTE}
                    else
                        set error flag (invalid operation code)
                end {if not a comment}
                write line to intermediate file
                read next input line
            end {while not END}
        write last line to intermediate file
        save (LOCCTR - starting address) as program length
    end {Pass 1}
```

Dvoprehodni zbirnik – 2. prehod; psevdokoda

Pass 2:

```
begin
    read first input line (from intermediate file)
    if OPCODE = 'START' then
        begin
            write listing line
            read next input line
        end {if START}
        write Header record to object program
        initialize first Text record
    * while OPCODE ≠ 'END' do
        begin
            if this is not a comment line then
                begin
                    1   search OPTAB for OPCODE
                    if found then
                        begin
                            if there is a symbol in OPERAND field then
                                begin
                                    2   search SYMTAB for OPERAND
                                    if found then
                                        store symbol value as operand address
                                    else
                                        begin
                                            store 0 as operand address
                                            set error flag (undefined symbol)
                                        end
                                    end {if symbol}
                                else
                                    3   store 0 as operand address
                                    assemble the object code instruction
                                end {if opcode found}
                            else if OPCODE = 'BYTE' or 'WORD' then
                                convert constant to object code
                            if object code will not fit into the current Text record then
                                begin
                                    4   write Text record to object program
                                        initialize new Text record
                                end
                                add object code to Text record
                            end {if not comment}
                            write listing line
                            read next input line
                        end {while not END}
                    write last Text record to object program
                    write End record to object program
                    write last listing line
                end {Pass 2}
```

Tvorba operacijske kode in operandov

Zbirnik na podlagi zbirniškega ukaza tvori operacijsko kodo strojnega ukaza, ki ji pridruži primerno obdelane operative.

- ▶ operacijska koda
- ▶ operand

rezervirane besede:
register - OPTAB
stevilske konstante dobimo v ukazu
simbolicno imec(desni naslov)
sestavljen odoperand

- ▶ _____
- ▶ _____
- ▶ _____
- ▶ _____

Tvorba operacijske kode in operandov

Sestavljeni operandi:

Sestavljeni so iz

- ▶ _____ vec simboličnih imen in/ali konstant
- ▶ _____ racunske operacije (*, +, -, /)
- ▶ _____ logicni operatorji (&, I,)

Operand se razreši (ovrednoti) z računsko rutino.

Tvorba operacijske kode in operandov

- ▶ Tvorjena operacijska koda je včasih odvisna od načina uporabe; isti mnemonik se ob različnih okoliščinah lahko prevede v različno operacijsko kodo.
- ▶ Dejavniki, ki lahko vplivajo na operacijsko kodo:
 1. _____ nacin naslavljanja
 2. _____ tip operandov
 3. _____ velikost konstante
 4. ...



Tvorba operacijske kode in operandov

1. Sprememba operacijske kode zaradi načina naslavljanja.

Primer (Intel x86) Osnovna op. koda za ukaz RET je 0xC3; koda se spremeni, če je naslov daleč od trenutnega naslova

RET near_addr	...	0xC3
RET far_addr	...	0xC8



Tvorba operacijske kode in operandov

2. Sprememba operacijske kode zaradi tipa operandov

Primer (Intel x86): ukaz ADD se prevede v 000000sd

s ... size

- 0 ... velikost operanda je 8 bitov
- 1 ... velikost operand je 16 ali 32 bitov

d ... direction

- 0 ... prvi operand je register, drugi pomnilnik
- 1 ... prvi operand je pomnilnik, drugi register



Tvorba operacijske kode in operandov

3) Sprememba operacijske kode zaradi velikosti konstante

Primer (MC6800):

BOR EQU 128 ... LDA A, BOR	Gre za direktno naslavljanje, zato se ta koda prevede v X'9680
BOR EQU 4096 ... LDA A, BOR	Razširjeno naslavljanje, koda se prevede v X'B61000

Kako pa je s tem pri SICu?

Objektna koda

- ▶ Zbirnik program prevede v **objektno kodo** in jo zapiše v **objektno datoteko**; pri tem uporablja dogovorjeni **objektni format**
- ▶ Objektna datoteka se lahko uporabi kot
 - ▶ samostojen program ali
 - ▶ modul v večjem programu.
- ▶ Preden objektna datoteka dejansko zaživi, potrebujemo še povezovalnik in/ali nalagalnik.

SIC objektna datoteka

SIC uporablja preprost *tekstovni objektni format*, ki predvideva tri vrste zapisov:

I) Zaglavje (Header – H) vsebuje ime, začetni naslov in dolžino programa

stolpec 1: H

stolpec 2-7: ime programa

stolpec 8-13: zacetni naslov

stolpec 14-19: dolzina programa



SIC objektna datoteka

2) Programski zapis (Text – T) vsebuje prevedene ukaze
(operacijske kode skupaj s pripadajočimi naslovi)

stolpec 1: T

stolpec 2-7: zacetni naslov tega T zapisa

stolpec 8-9: dolzina T zapisa (max 30(1EJ))

stolpce 10-69: objektna koda

3) Zapis za konec (End – E) označuje konec programa
vsebuje (opcijsko) naslov prvega izvršljivega ukaza

stolpec 1: E

stolpec 2-7: naslov 1. ukaza

SIC objektna datoteka – primer 1

Program

Naslov	koda		SIC koda

0000	PRVI	START	0
0000		LDA	X 00000C
0003		SUB	Y 1C000F
0006		ADD	INCR 180012
0009		STA	Z 0C0015
000C	X	WORD	X'000009' 000009
000F	Y	WORD	X'000005' 000005
0012	INCR	WORD	X'000001' 000001
0015	Z	RESW	1

Objektna datoteka:

```
HPRVI_000000 000018
T 000000 15 00000C 1C000F 180012 0C0015 000009 000005 000001
E 000000
```



SIC objektna datoteka – primer 2

1 / 2

5	1000	COPY	START	1000	
10	1000	FIRST	STL	RETADR	141033
15	1003	CLOOP	JSUB	RDREC	482039
20	1006		LDA	LENGTH	001036
25	1009		COMP	ZERO	281030
30	100C		JEQ	ENDFIL	301015
35	100F		JSUB	WRREC	482061
40	1012		J	CLOOP	3C1003
45	1015	ENDFIL	LDA	EOF	00102A
...	manjkajo 4 ukazi
70	1024		LDL	RETADR	081033
75	1027		RSUB		4C0000
80	102A	EOF	BYTE	C' EOF'	454F46
85	102D	THREE	WORD	3	000003
90	1030	ZERO	WORD	0	000000
95	1033	RETADR	RESW	1	
100	1036	LENGTH	RESW	1	
105	1039	BUFFER	RESB	4096	
125	2039	RDREC	LDX	ZERO	041030
...			manjka 10 ukazov
180	205A		RSUB		4C0000
185	205D	INPUT	BYTE	X' F1'	F1
190	205E	MAXLEN	WORD	4096	001000
210	2061	WRREC	LDX	ZERO	041030
...			manjkata 2 ukaza
225	206A		LDCH	BUFFER, X	509039
...			manjkajo 3 ukazi
245	2076		RSUB		4C0000
250	2079	OUTPUT	BYTE	X' 05'	05
255			END	FIRST	



Objektna datoteka programa na prejšnji strani (Fig 2.1)

```
HCOPY 001000 00107A
T 001000 1E 141033 482039 .... 00102D
T 00101E 15 0C1036 ....      000000
T 002039 1E 041030 ....      38203F
T 002057 1C 101036 ....      2C1036
T 002073 07 382064 4C0000 05
E 001000
```

Posebnosti pri tvorbi kode za SIC/XE

- ▶ Tvorba objektne kode za SIC/XE se precej razlikuje od tvorbe objektne kode za SIC
- ▶ SIC zelo preprosto
 - koda je vedno 3-bajtna, en sam bit za določanje indeksnega načina naslavljanja
- ▶ SIC/XE mnogo bolj zapleteno
 - ▶ več formatov ukazov
 - koda je lahko 1-, 2-, 3- ali 4-bajtna
 - ▶ več načinov naslavljanja
 - identificirati je treba način naslavljanja in modifcirati osnovno kodo ukaza

Posebnosti tvorbe SIC/XE kode si bomo ogledali na primeru iz slike 2.5

```

5    0000 COPY      START      0
10   0000 FIRST     STL        RETADR
12   0003           LDB        #LENGTH
13           BASE      LENGTH
15   0006 CLOOP     +JSUB    RDREC
20   0009           LDA        LENGTH
...
40   0017           J         CLOOP
...
55   0020           LDA      #3
...
70   002A           J         @RETADR
...
95   0030 RETADR   RESW     1
100  0033 LENGTH   RESW     1
105  0036 BUFFER   RESB     4096
...
125  1036 RDREC   CLEAR    X
...
133  103C           +LDT    #4096
...
150  1049           COMPR   A, S
...
160  104E           STCH    BUFFER, X
...
175  1056 EXIT     STX     LENGTH

```

Del programa iz slike 2.5 (SIC/XE)

Posebnosti pri tvorbi kode za SIC/XE Registrsko-registrski ukazi

- ▶ Zbirnik mora poznati numerične vrednosti (interna reprezentacija) vseh registrov ($A=0, X=1, L=2, B=3, S=4, T=5, F=6$),
- ▶ registre v številke spremeni v 2. fazi zbiranja,
- ▶ lahko uporabi simbolno tabelo (pred začetkom zbiranja jo napolni z imeni in vrednostmi za vse registre).

Primer: prevajanje ukazov v vrsticah 125 in 150:

125	1036 RDREC	CLEAR	X	B410
150	1049 COMPR	A, S		A004

Posebnosti pri tvorbi kode za SIC/XE

Relativno naslavljjanje

- ▶ Večina registrsko-pomnilniških ukazov se prevede bodisi kot PC-relativno bodisi kot bazno-relativno naslavljjanje.
- ▶ V obeh primerih mora zbirnik izračunati *odmik*
- ▶ Odmik mora biti dovolj majhen
 - ▶ predznačeno (-2048...2047) za PC relativno
 - ▶ nepredznačeno (0...4095) za bazno-relativen način.
- ▶ Način naslavljjanja izbere zbirnik; najprej poskusи s PC-relativnim

Posebnosti pri tvorbi kode za SIC/XE

Relativno naslavljjanje

Računanje odmika pri *-relativnem naslavljjanju poteka obratno kot računanje uporabnega naslova (UN).

UN = (PC) + ODMIK

UN = (B) + ODMIK

ODMIK = UN - (B/PC)

kaksna je vrednost pc regista?

sic pozna lokacijski stevec ne pozna pc

preberemo celoten ukaz

povecamo pc

izvrsimo ukaz

ta vrstni red je pomemben, zakaj?

ce z ukazom dolocamo vrednost pc bi z drugim vrstnim redom povozili

jump ukaz in program bi skocil na lokacijo jump + 1

Posebnosti pri tvorbi kode za SIC/XE

Relativno naslavljjanje

PC-relativno naslavljjanje

Naslov se izračuna glede na vrednost PC

Primer:

10 0000 FIRST STL

RETADR

$$\begin{aligned} \text{RETARD} &\rightarrow \text{UN} = 0030 \text{ LS}=0 \\ \text{ODMIK} &= \text{UN} - ((\text{LS}) + \text{L}) \\ &= 0030 - 3 = 002D \end{aligned}$$

NXIBPE
17I202D

40 0017 J

CLOOP

$$\begin{aligned} \text{UN} &= 0006 \\ \text{ODMIK} &= \text{UN} - ((\text{LS})+\text{L}) \\ &= 0006 - 001A = \underline{\underline{\text{FEC}}} \end{aligned}$$

$$6 - 26 = -20$$

$$0x006 - 0x001A = 0xFEC$$

Posebnosti pri tvorbi kode za SIC/XE

Relativno naslavljjanje

Bazno-relativno naslavljjanje

- naslov se določi glede na vrednost baznega registra
- zbirnik vrednosti baznega registra ne pozna, uporabnik mu jo sporoči s posebnim navodilom (direktivo) BASE

CE BI BIL TA ODMIK SE VEDNO PREVELIK BI JAVIL NAPAKO IN KONCAL

Primer:

12	0003	LDB	#LENGTH
13		BASE	LENGTH
		NI LOJTRE KER JE DIREKTIVA ZA ZBIRNIK IN NI UKAZ	
...			
160	104E	STCH	BUFFER, X
			LS + L = 1051
			UN = 0036
			ODMIK = 0036 - 1051 < 1000 PC-RELATIVNO NE GRE
			ODMIK = UN - (B)
			B= 0036 - 0033 = 0003
			NIXBPEI
			57ICI003

Posebnosti pri tvorbi kode za SIC/XE

Relativno naslavljjanje

Bazno-relativno naslavljjanje

- ▶ **Opomba 1:** v primeru, da bazni register uporabljam za drug namen (recimo: začasno shranjevanje rezultata aritmetičnih operacij) moramo to zbirniku sporočiti! Uporabimo direktivo NOBASE .

- ▶ **Opomba 2:** vrstici

20	0009	LDA LENGTH
175	1056	EXIT STX LENGTH

se kljub podobnosti prevedeta različno (prva s PC-relativnim, druga z bazno-relativnim naslavljjanjem)

Posebnosti pri tvorbi kode za SIC/XE

Uporaba 4-zlogovnega razširjenega formata

- ▶ Kadar je odmik prevelik za uporabo *-relativnega naslavljanja, se uporabi 4-zlogovni zapis ukaza (20 bitov za opis naslova).
- ▶ Programer mora uporabo razširjenega formata zbirniku sporočiti z uporabo znaka + pred ukazom

Primer:

RDREC-> UN = 1036
OPCODE: 48+3
4BI1I01036

15 0006 CLOOP

+JSUB RDREC _____

Posebnosti pri tvorbi kode za SIC/XE

Takojšnje naslavljjanje

- ▶ Uporabimo znak # pred operandom
- ▶ Za zbirnik je takojšnje naslavljjanje zelo preprosto opravilo: operand pretvori v notranjo predstavitev in ga vstavi v operacijsko kodo.

Primeri:

55	0020	LDA	# 3	01 0003

133	103C	+LDT	# 4096	75 101000

12	0003	LDB	#LENGTH	69 202D

UN = LENGTH

LS + L = 0006

ODMIK = UN - 0006

0033 - 0006 = 002D

Posebnosti pri tvorbi kode za SIC/XE

Posredno naslavljjanje

- ▶ Uporabimo znak @ pred operandom

Primer:

70

002A

J

$$\begin{aligned} \text{RETADR} &\rightarrow \text{UN} = 0030 \\ \text{ODMIK} &= \text{UN} - (002A + 0003) \\ &= 0030 - 002D = 0003 \end{aligned}$$

@RETADR

^{3C+2}
3EI 2003

Naloga za vajo

- ▶ “Ročno” prevedi program in ustvari objektno datoteko.

SESTEJ	START	0
	LDS	#3
	LDT	#9
	LDX	#0
ZANKA	LDA	ALFA,X
	ADD	BETA,X
	STA	GAMA,X
	ADDR	S,X
	COMPR	X,T
	JLT	ZANKA
KONEC	J	KONEC
ALFA	WORD	1
	WORD	2
	WORD	3
BETA	WORD	4
	WORD	5
	WORD	6
GAMA	RESW	3

Naloga za vajo – preveri pravilnost rešitve

Ukaz `java -cp sictools.jar sic.Asm sestej.asm` ustvari `lst` datoteko:

01000		SESTEJ	START	4096
01000	6D0003		LDS	#3
01003	750009		LDT	#9
01006	050000		LDX	#0
01009	03A010	ZANKA	LDA	ALFA,X
0100C	1BA016		ADD	BETA,X
0100F	0FA01C		STA	GAMA,X
01012	9041		ADDR	S,X
01014	A015		COMPR	X,T
01016	3B2FF0		JLT	ZANKA
01019	3F2FFD	KONEC	J	KONEC
0101C	000001	ALFA	WORD	1
0101F	000002		WORD	2
01022	000003		WORD	3
01025	000004	BETA	WORD	4
01028	000005		WORD	5
0102B	000006		WORD	6
0102E	00....00	GAMA	RESW	3

Naloga za vajo – preveri pravilnost rešitve

Ukaz java -cp sictools.jar sic.Asm sestej.asm **ustvari** obj datoteko:

H**SESTEJ**001000**000037**

T0010001F**6D000375000905000003A0101BA0160FA01C9041A0153B2FF03F2FFD000001**

T00101**F0F0000020000030000400000500006**

E001000

