SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2198

Jezični procesor za prevođenje višeg programskog jezika u jezik procesora sustava Commodore 64

Toni Kork

Zagreb, srpanj 2011.

Zahvaljujem Prof. dr. sc. Ž. Mihajlović i Prof. dr. sc. S. Srbljiću za rješenje problema mentorstva

Zahvaljujem asistentu I. Žužaku, dipl. ing. za svu ukazanu pomoć prilikom izrade završnog rada

Zahvaljujem P. H. Sundelu na izradi emulatora CCS64, bez kojeg bi izrada ovog rada bila uvelike otežana

Zahvaljujem Commodore Buisness Machines-u na izradi računala Commodore 64 koje je zauvijek promijenilo svijet

Zahvaljujem svim obožavateljima računala Commodore 64 što su sakupili neizmjerno mnogo literature i objavili je na Internetu

Sadržaj

1 Uvod

Od ranih dana računarstva pa do danas, ljudi pokušavaju pojednostaviti izradu programa i približiti ih osobama van inženjerske domene. U tu svrhu razvijaju se jezični procesori. Jezični procesor je program koji, općenito, niz znakova jednog jezika prevodi u niz znakova drugog jezika. U domeni računarstva najčešće se podrazumijeva prevođenje iz visokog programskog jezika u strojni kôd računala.

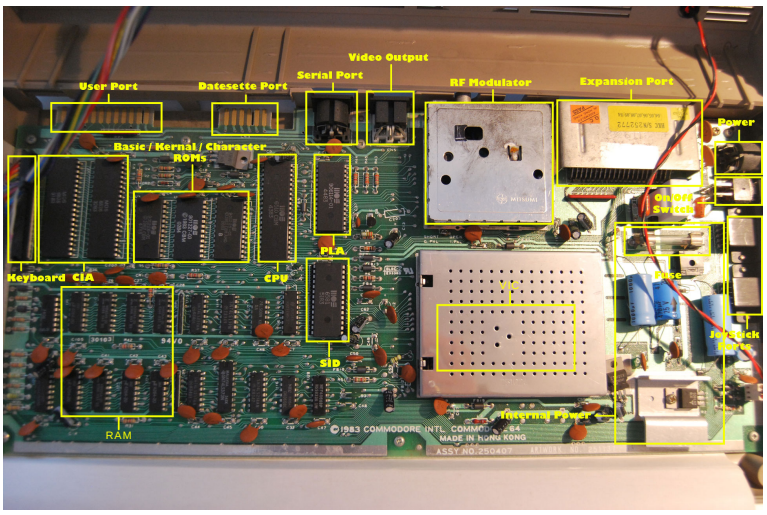
U ovom radu detaljno je obrađena izrada jezičnog procesora treće generacije, KCC64. Jezični procesor KCC64 prevodi jezik KC64 kao i mnemonički 6510 strojni jezik u izvršni kôd izvediv na računalu Commodore 64. Oba jezika osmišljena su prilikom izgradnje jezičnog procesora KCC64. Motiv ovog rada je nezadovoljstvo postojećim jezičnim procesorima za Commodore 64 platformu, te olakšanje izrade aplikacija za istu, jer današnja IBM PC-kompatibilna računala pružaju veći komfor u radu za razliku od Commodore-a 64.

U drugom poglavlju ukratko je opisano računalo Commodore 64 i njegovi najvažniji sastavni dijelovi. U trećem poglavlju je objašnjeno što je to prevođenje programskih jezika kao i osnovni mehanizmi prevođenja programskih jezika. U četvrtom poglavlju opisani su jezici jezičnog procesora KCC64, a u petom tehnički detalji njegovih programskih modula i korisničkog sučelja. Šesto poglavlje pruža osnovne upute za rad u korisničkom sučelju jezičnog procesora KCC64 kao i primjere programskih konstrukata njegovih jezika. U sedmom poglavlju izložen je zaključak rada, a u osmom popis korištene literature. Deveto i deseto poglavlje sadrže sažetke rada na hrvatskom i engleskom jeziku. U dodacima A i B nalaze se ispisi programa napisanih u programskom jeziku KC64 i mnemoničkom strojnom jeziku.

2 Računalo Commodore 64

Računalo Commodore 64 (slika 2.1) prvi puta je predstavljeno u siječnju 1982. godine na elektroničkom sajmu CES[9], u prodaju ušlo u kolovozu iste godine, a prestalo se proizvoditi u travnju 1994. godine. Može se poistovjetiti sa Fordovim modelom T, jer je sa 17 milijuna prodanih primjeraka približilo računala širokim masama.

Slika 2.1 Commodore 64 računalo iz 1983. godine, tzv. breadbox model



Slika 2.2 Matična ploča Commodore 64 računala iz 1983. godine sa označenim mikročipovima i priključcima

Karakteristike računala Commodore 64[1], [7] su: 8-bitni središnji procesor 6510, 64 KiB radne memorije, video upravljačka jedinica 6567, zvučni čip 6581, dva priključka za igrače palice, priključak za proširenje, korisnički priključak, serijski priključak, antenski i video izlaz za spajanje na TV prijemnik, priključak za kasetofon te qwerty tipkovnica.

2.1 Središnja procesorska jedinica

Središnja procesorska jedinica računala Commodore 64 je MOS Technology 6510, 8-bitni mikroprocesor iz 65xx serije, na slici 2.2 označen sa CPU. Osnovne

karakteristike mikroprocesora 6510 su: 16-bitna adresna i 8-bitna podatkovna sabirnica, trokoračni cjevovod, ulazno-izlazna sabirnica, 56 instrukcija (navedene u poglavlju 4.3) sa 13 mogućih načina adresiranja. Registri procesora 6510 su: akumulator (A) i dva indeks registra (X i Y), programsko brojilo (PC), registar stoga (S), status registar (P), registar smjera podataka (DDR), podatkovni registar (DR) koji je spojen na ulazno-izlaznu sabirnicu. Radni takt procesora ovisi o tome koji se TV standard koristi u državi te iznosi 1.023 MHz (NTSC verzija) ili 0.985 MHz (PAL verzija). 6510 koristi *little endian* poredak bajtova prilikom dohvata adresa i argumenata instrukcija.

2.2 Video podsustav

Video podsustav računala Commodore 64 temelji se na čipu MOS Technology 6567 VIC-II (Video Interface Chip II), iz popularne 65xx serije, na slici 2.2 označen sa VIC (ispod limenog poklopca sa rupicama). Podržava tekstualni režim rada sa 40x25 znakova na ekranu (slika 2.3), 16 boja, režim rada visoke rezolucije (320x200 piksela sa zajedničkom bojom pozadine i jednom od 16 boja zapisa po 8x8 području), višebojni režim rada (dvije zajedničke boje, pozadina i jednom od 16 boja zapisa po 8x8 području, ali sa rezolucijom 160x200), 8 sprajtova (slike 24x21 piksela koje su neovisne o ostalim režimima rada), koliziju među sprajtovima i koliziju sprajtova i grafike. Rasterski prekidni registar omogućava generiranje prekidnog signala kada se na ekranu iscrtava zadana ekranska linija (engl. *raster interrupt*). Za tekstualni režim rada moguće je napraviti vlastite fontove sa znakovima 8x8 piksela. VIC-II ima ugrađeni RF video modulator za prikaz slike na analognom televizoru te još razne druge, manje važne osobine.

Slika 2.3 Inicijalni sadržaj ekrana po uključenju računala

2.3 Zvučni podsustav

Temeljna komponenta zvučnog podsustava računala Commodore 64 je MOS Technology 6581 SID (Sound Interface Device), također čip iz 65xx serije, pokrenuo je glazbenu revoluciju u računalnoj industriji, na slici 2.2 označen sa SID. Podržava tri neovisna generatora zvuka koji mogu birati između: trokutastog, pilastog i pravokutnog valnog oblike, te šuma. Za pravokutni valni oblik mogu se mijenjati omjer impuls-pauza. Frekvencijski opseg svakoga od tih generatora je 20-16000 Hz, a svaki također posjeduje i modulator amplitude signala po ADSR principu (engl. *attack*, *decay*, *sustain*, *release*). Izlazni signal svakoga generatora tona moguće je propustiti kroz zajednički audio filtar koji podržava: niskopropusni, visokopropusni, i pojasnopropusni filtar sa promjenjivom rezonantnom frekvencijom i faktorom gušenja. Generatore tona moguće je međusobno sinkronizirati ili koristiti modulaciju zvonjave.

2.4 Ulazno-izlazni priključci

Računalo Commodore 64 je u usporedbi sa drugim kućnim računalima 80-tih godina prošlog stoljeća imalo bogat izbor ulazno-izlaznih priključaka. Ulazni priključci računala Commodore 64 su: dva 9-pinska D priključka za digitalnu ili analognu igraču palicu ili svjetlosnu olovku (na slici 2.2 označeni sa *Joystick port*). Ulazno-izlazni priključci su: 8-bitni paralelni korisnički priključak (na slici 2.2 označeni sa *Joystick port*), priključak za kasetofon (na slici 2.2 označen sa *Datesette port*), serijski IEEE 488 priključak sa TTL razinama (na slici 2.2 označen sa *Serial port*), 44-pinski priključak za proširenja na kojem su izvedene sve glavne sabirnice računala Commodore 64 (na slici 2.2 označen sa *Expansion port*). Izlazni priključci su: DIN-5 kompozitni video-audio priključak (na slici 2.2 označen sa *Video output*), antenski priključak (na slici 2.2 označen sa *RF Modulator*)

2.5 Uskrsnuće Commodore-a 64

Nakon bankrota tvrtke Commodore Buisness Machines (CBM, Inc.), ime i autorska prava otkupljivalo je i prodavalo nekoliko tvrtki. Trenutni vlasnik imena Commodore u svibnju 2011, na tržište je plasirao redizajnirano računalo naziva Commodore 64x. Ovaj puta 64 u nazivu ne označava 64KiB RAM-a već 64-bitni procesor. Računalo je bazirano na x86-64 platformi te je kompatibilno sa IBM PC standardom, ali dolazi sa posebnim operacijskim sustavom koje ima emulatore za sve modele računala koje je tvrtka CBM, Inc. ikad proizvela.

3 Prevođenje programskih jezika

Prevođenje programskog jezika[5] je postupak pretvorbe izvornog kôda korisničkog programa napisanog u jednom programskom jeziku, u drugi jezik, ali najčešće strojni kôd računala na kojem se program izvodi.

Programski jezici najčešće se dijele u četiri skupine prema razini apstrakcije, tzv. generacije:

Prva generacija: strojni kôd, izravno programiranje u jeziku računala. Glavni razlog nastanka jezičnih procesora. Tipični predstavnici: kratkospojnici, bušene trake, sadržaj memorije računala.

Druga generacija: mnemonički strojni jezici (asembleri). Strojni kôd računala zamijenjen je jednostavnim akronimima naredbi i njihovim argumentima. Napredniji strojni jezici posjeduju i makro naredbe koje zamjenjuju sekvence naredbi, a i neke osnovne strukture viših programskih jezika. Tipični predstavnici: MASM, C64MON

Treća generacija: imperativni programski jezici. Visoki programski jezici u kojima se detaljno opisuje tok programa i algoritmi. Računalu se zadaju naredbe koje ono izvršava u zadanom redoslijedu. Danas najraširenija skupina programskih jezika. Tipični predstavnici: BASIC, C, Java, Pascal.

Četvrta generacija: deklarativni programski jezici. Visoki programski jezici u kojima se deklariraju ulazni podaci i ono što se sa tim podacima treba napraviti. Računalo samo bira najbolji način obrade podataka. Tipični predstavnici: SQL, Prolog, Haskel.

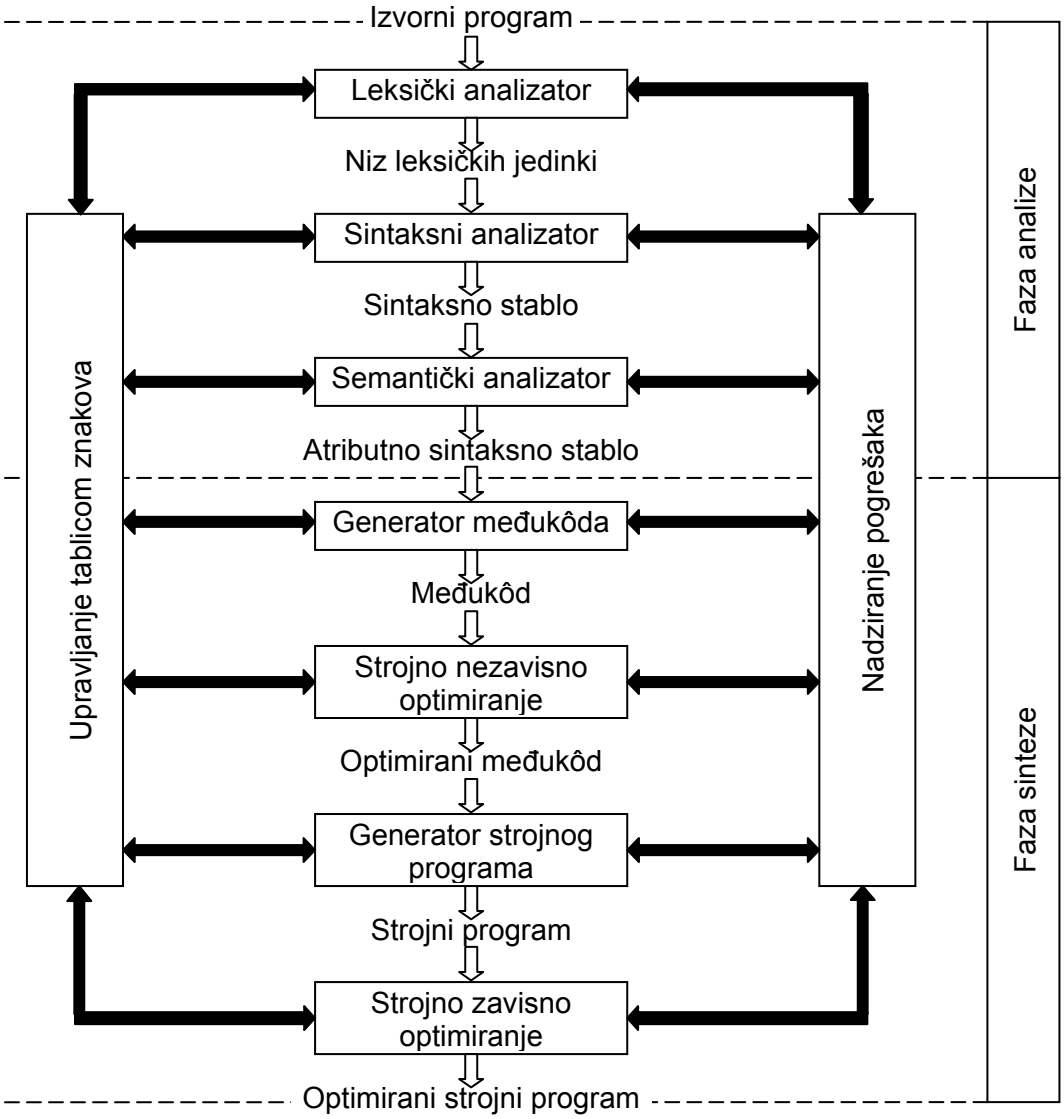
Uz ove također se često spominje i peta generacija programskih jezika kod kojih programer samo treba objasniti problem i konačni ishod, a računalo samo nalazi rješenje. Ova skupina programskih jezika još uvijek ne postoji.

3.1 Faze prevođenja programskih jezika

Prevođenje izvornog programa u ciljni program ostvaruje se u dvije osnovne faze: analiza izvornog programa i sinteza ciljnog programa, slika 3.1.

U fazi analize izvorni se program rastavlja na sastavne dijelove, provjeravaju se leksička, sintaksna i semantička pravila jezika, prijavljuju pogreške i generira se leksički zapis izvornog programa. Analiza se izvodi u više koraka: leksička analiza, sintaksna analiza i semantička analiza.

U fazi sinteze zapis izvornog programa, generiran u fazi analize, prevodi se u ciljni program. Sinteza ciljnog programa izvodi se u više zasebnih koraka: generiranje međukôda, strojno nezavisno optimiranje, generiranje strojnog programa, strojno zavisno optimiranje i priprema strojnog programa za izvođenje.



Slika 3.1 Koraci rada jezičnog procesora

Prvi korak u procesu pretvorbe izvornog programa u ciljni je leksička analiza koju obavlja leksički analizator. Leksička analiza je jedini korak rada jezičnog procesora koji izravno pristupa znakovima teksta izvornog programa spremljenog u memoriju računala. Na samom početku odbacuju se znakovi koji se ne koriste u daljnjem radu jezičnog procesora, kao što su bjeline, tabulatori, znak novog reda, znak vračanja na početak reda, itd. Sljedeći korak je grupiranje i određivanje klasa leksičkih jedinki. Postoji pet klasa leksičkih jedinki: ključne riječi, operatori, identifikatori, specijalni znakovi i konstante.

Ako u pročitanom nizu znakova nije moguće odrediti klasu, čita se sljedeći znak izvornog programa. U suprotnom se pročitani niz znakova grupira u leksičku jedinku. Nakon toga se provjeravaju zadana leksička pravila, tj. provjerava se pripada li leksička jedinka pridruženoj klasi. Ako leksička jedinka pripada klasi,

pridružuje joj se uniformni znak koji se zapisuje u tablicu uniformnih znakova. Tablica uniformnih znakova kasnije služi kao osnovna podatkovna struktura sintaksnog analizatora. Ukoliko leksička jedinka ne pripada pridruženoj klasi, prijavljuje se pogreška. Iako je nastala pogreška leksički analizator nastavlja svoj rad koristeći postupak oporavka od pogreške.

Sintaksna analiza središnji je korak rada jezičnog procesora. Osnovne funkcije sintaksnog analizatora su grupiranje uniformnih znakova u sintaksne cjeline, provjera sintaksnih pravila, stvaranje hijerarhije sintaksnih cjelina, određivanje i opis pogrešaka, oporavak od pogreške i gradnja sintaksnog stabla.

Sintaksna analiza započinje čitanjem uniformnih znakova leksičkih jedinki te njihovim grupiranjem u sintaksne cjeline. Osnovne sintaksne cjeline su izrazi, naredbe, blokovi naredbi i program. Nakon grupiranja slijedi postupak provjere zadanih sintaksnih pravila. Za opis sintaksnih pravila koristi se više različitih sustava oznaka, od regularnih izraza, formalnih gramatika, pa sve do posebno uvedenih sustava oznaka kao što su BNF i COBOL. Iako značajno otežava izgradnju sintaksnog analizatora, postupak određivanja mjesta i opisa pogreške velika je pomoć korisniku jezičnog procesora. Zahtjeva se da postupak određivanja mjesta i opis pogrešaka precizno odredi mjesto pogreške, kratko i jasno opiše pogrešku te da značajno ne uspori rad sintaksnog analizatora. Tijekom postupka oporavka od pogreške, sintaksni analizator promijeni stanje tako da omogući nastavak sintaksne analize.

Semantička analiza povezuje faze analize izvornog i sinteze ciljnog programa. U cilju određivanja značenja sintaksnih cjelina, provjere semantičkih pravila i pripreme podataka ostalim dijelovima jezičnog procesora, semantički analizator popunjava tablicu uniformnih znakova vrijednostima obilježja sintaksnih cjelina, osigurava prijenos vrijednosti obilježja po sintaksnom stablu, odrađuje početne uvjete, određuje mjesto semantičkih pogrešaka, opisuje semantičke pogreške, obrađuje makro naredbe i obrađuje naredbe koje se izvode tijekom prevođenja. Tijekom semantičke analize moguće je u potpunosti izravnati sintaksno stablo i pripremiti podatke potrebne za generiranje niza slijednih naredbi međukôda ili ciljnog programa.

Određivanje značenja i provjera semantičkih pravila središnji je zadatak semantičkog analizatora. Semantički analizator određuje značenje svim elementima jezika. Primjerice, semantički analizator određuje značenje identifikatorima. Identifikatori su imena varijabli, funkcija, polja i sl. Značenje varijabli određuje se

prema njihovom podatkovnom tipu, npr. cjelobrojne, realne, logičke, itd. Značenje polja i potprograma je složeno jer se određuje na temelju više podataka, indeksa i elemenata polja, odnosno ulaznih i izlaznih parametara.

Faza sinteze počinje sintaksnom i semantičkom analizom kada se generira međukôd najviše razine. Na njemu je moguće obaviti strojno nezavisno optimiranje kao što su optimizacija izraza i petlji. Strojno nezavisno optimiranje može se izvoditi u više koraka proizvodeći međukôd sve niže razine.

Nakon što je strojno nezavisno optimiranje obavljeno generira se međukôd najniže razine ili mnemonički strojni kôd na kojem se obavlja strojno zavisno optimiranje, kao što je zamjena kompleksnih instrukcija jednostavnijima i odbacivanje viška instrukcija. Posljednja faza je generiranje izvršivog strojnog kôda.

4 Jezici jezičnog procesora KCC64

Jezični procesor KCC64 (**K**ork-ov ***C**ompiler* za **C**ommodore **64**) može prevesti dva ulazna programska jezika, KC64 i mnemonički 6510 strojni jezik, u izvršni kôd Commodore 64 platforme. Ukoliko je ulazni program napisan u jeziku KC64 tada se uz izvršni kôd generira i ekvivalentni mnemonički strojni kôd. Tijekom prevođenja generira se i međukôd KCIL koji pomaže razlaganju višeg programskog jezika na jednostavnije konstrukte. KCIL nije moguće niti učitati niti pohraniti, program u KCIL-u postoji samo privremeno tijekom procesa prevođenja.

4.1 Programski jezik KC64

Programski jezik KC64 (**K**ork-ov **C** za Commodore **64**) po svojoj strukturi veoma je sličan programskom jeziku C. Razlog tome je jednostavnija prilagodba jeziku KC64, jer je zbog svoje rasprostranjenosti C poznat gotovo svakom programeru. Datoteke sa programima napisanim u programskom jeziku KC64 imaju ekstenziju *kc64*.

4.1.1 Leksička pravila jezika KC64

Leksičke jedinke su nizovi znakova izvornog programa, razvrstane su u više klasa. Osnovne klase leksičkih jedinki su: ključne riječi, operatori, specijalni znakovi, identifikatori i konstante. KC64 razlikuje velika i mala slova (engl. case sensitive) te nije isto ako je riječ napisana velikim ili malim slovima ili bilo kojom mješavinom velikih i malih slova. Leksička pravila jezika KC64 mogu se opisati regularnim izrazima koji odgovaraju opisu u tablici 4.1.

Tablica 4.1 Dozvoljeni znakovi leksičkih jedinki

Prvi znak leksičke jedinke		Ostali dozvoljeni znakovi
A..Z, a..z ili _	Identifikator ili ključna riječ	A..Z, a..z, 0..9 ili _
0..9	Dekadski broj	0..9, e, - ili .
$	Heksadekadski broj	0..9, A..F
"	Znakovni niz	Bilo koji znakovi do ponovnog navodnika koji označava kraj znakovnog niza
Lijevi znak operatora koji se sastoje od dva znaka		Desni znak operatora koji se sastoje od dva znaka (vidi tablicu 4.3)

4.1.1.1 Ključne riječi

Programski jezik KC64 ima četrnaest ključnih riječi prikazanih u tablici 4.2.

Tablica 4.2 Ključne riječi u programskom jeziku KC64

Oznaka	Opis
assembly	oznaka bloka naredbi koje su napisane u mnemoničkom 6510 strojnom jeziku, a ne u jeziku KC64
bool	oznaka logičkog tipa podataka
byte	oznaka cjelobrojnog nepredznačenog tipa podataka duljine 8 bita
do	oznaka početka petlje sa ispitivanjem uvjeta na kraju
else	oznaka za nastavak ako uvjet nije zadovoljen
if	oznaka uvjetne naredbe
int	oznaka cjelobrojnog predznačenog tipa podataka duljine 16 bita
long	oznaka cjelobrojnog predznačenog tipa podataka duljine 32 bita
main	oznaka početka glavne funkcije
single	oznaka brojčanog tipa podataka sa posmačnim zarezom duljine 40 bita
string	oznaka znakovnog tipa podataka
until	oznaka kraja petlje sa ispitivanjem uvjeta na kraju
void	oznaka da funkcija nema povratnu vrijednost
while	oznaka petlje sa ispitivanjem uvjeta na početku

4.1.1.2 Operatori i specijalni znakovi

Postoji ukupno četrdeset operatora i specijalnih znakova u jeziku KC64. Svi ti znakovi su ujedno i prekidni znakovi, tj. odvajaju leksičke jedinke. U tablici 4.3 dan je popis svih operatora i specijalnih znakova, zajedno sa opisom, asocijativnošću i njihovim prioritetima. Što je prioritet manji, to je prednost operatora veća.

Tablica 4.3 Operatori i specijalni znakovi u programskom jeziku KC64

Klasa	Simbol	Asocijativnost	Opis operatora	Prioritet operatora
Aritmetički i logički operatori	<<	desna	bitovni pomak ulijevo	0
	>>		bitovni pomak udesno	
	!!	desna (unarni operator)	aritmetičko NE (komplementiranje)	1
	^	desna	potenciranje (samo za *single* varijable)	2
	-	desna (unarni operator)	unarni minus (predznak)	3
	*	lijeva	množenje	4
	/		dijeljenje	
	%		ostatak dijeljenja (modulo)	
	+		zbrajanje	5
	-		oduzimanje	
	&&		aritmetičko I (AND)	6
	||		aritmetičko ILI (OR)	7
	~~		aritmetičko isključivo ILI (XOR)	
	!	desna (unarni operator)	logičko NE (NOT)	8
	==	desna ili lijeva	jednako	9
	!= <>		različito	
	<		manje od	
	>		veće od	
	<=		manje od ili jednako	
	>=		veće od ili jednako	
	&	Lijeva	logičko I (AND)	10

	|		logičko ILI (OR)		11
	~		logičko isključivo ILI (XOR)		
Operatori pridruživanja	=	desna	pridruživanje	Identifikator sa lijeve strane operatora pridruživanja	12
	+=		pribrajanje na		
	-=		oduzimanje od		
	*=		primnažanje na		
	/=		dijeljenje od		
	%=		modulo sa		
Ostali specijalni znakovi	,	N/A	nabrajanje parametara		N/A
	"		početak/kraj stringa		
	'		komentar (do kraja reda)		
	$		heksadekadska konstanta		
	@		pokazivač / vrijednost na koju pokazuje pokazivač		
	( )		izraz, parametri funkcije		
	[ ]		oznaka indeksa polja		
	{ }		početak/kraj bloka naredbi		
	;		kraj naredbe		

4.1.1.3 Imena varijabli ili identifikatora

Imena varijabli ili identifikatora mogu sadržavati slova, brojke i podvlaku (_). Prvi znak imena varijable mora biti slovo ili podvlaka kao što je navedeno u listi 4.1.

Lista 4.1 Primjeri dozvoljenih i nedozvoljenih imena varijabli i identifikatora

Primjeri dozvoljenih varijabli:	Primjeri nedozvoljenih varijabli:
`Imevarijable`	`ime varijable`
`ime_varijable`	`1imevarijable`
`ime1varijable`	
`_imevarijable`	

Važno je napomenuti sintaksno pravilo da ime varijable ne smije biti jednako niti jednoj ključnoj riječi, jer su ta imena rezervirana.

4.1.1.4 Komentari

U programskom jeziku KC64 komentari su omogućeni slično kao i u porodici programskih jezika BASIC za IBM PC kompatibilna računala. Ukoliko se želi napisati neki komentar, potrebno je na početku komentara staviti apostrof ('), a komentar se nadalje proteže do kraja tekstualne linije. Primjer komentara:

```
' Ovdje dodajte komentar.
```

4.1.1.5 Brojčane i znakovne konstante

Brojčane konstante mogu biti cjelobrojne (dekadske ili heksadekadske), brojevi sa posmačnim zarezom ili niz znakova. Dekadske cjelobrojne konstante definiraju se zapisom koji sadrži isključivo dekadske znamenke {0,1,2,3,4,5,6,7,8,9} i predznak (-), ako je riječ o negativnom broju. Heksadekadske cjelobrojne konstante moraju biti predznačene znakom $, a smiju sadržavati isključivo heksadekadske znamenke {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}. Brojčane konstante sa

posmačnim zarezom uz znakove koji se koriste u dekadskim cjelobrojnim konstantama moraju imati jednu decimalnu točku (.), a oznaka dekadskog eksponenta (e) nije obavezna. Primjeri svake od brojčanih konstanti navedeni su u listi 4.2.

Lista 4.2 Primjeri brojčanih konstanti

Cjelobrojna dekadska konstanta:	12345, -7178
Cjelobrojna heksadekadska konstanta:	$FAC0FF
Brojčana konstanta sa posmačnim zarezom:	1.234e11, -0.1234, 1.23e-7, -2e-5, 1.

Znakovne konstante definirane su svim znakovima koje se nalaze između navodnika ("), uključujući znakove prijelaza u novi red, tabulatore i ostale simbole. U znakovnoj konstanti jedino nije moguće zapisati navodnik (").

4.1.2 Sintaksna pravila jezika KC64

Strukturu programskog jezika KC64 čine slijedne naredbe, blokovi naredbi, petlje, naredbe uvjetnog grananja. Osnovu strukturu programskog jezika čini glavni program (funkcija *main*), unutar kojeg su najprije deklarirane varijable. Nakon toga slijede naredbe programa. Sve ostale funkcije moraju se navesti iza glavne funkcije.

Funkcija započinje ključnom riječi oznake tipa podatka, nakon čega slijedi lista parametara u zagradama iza koje slijedi blok naredbi. Parametri nisu neophodni i može ih biti proizvoljan broj, a mogu biti proizvoljnih tipova podataka. Početak i kraj bloka naredbi označava se vitičastim zagradama, kao i u jeziku C.

Svaka varijabla koja se u programu kani koristiti mora biti deklarirana na početku funkcije, sve varijable su globalne i statičke bez obzira gdje i kada su deklarirane. Varijable i funkcije ne smiju koristiti isti naziv kao i ključna riječ.

Prilikom deklaracije varijabli određuje se tip varijable i njezin naziv. Svaka deklaracija varijabli mora završavati oznakom točke-zareza ***;***. Također, omogućeno je ulančavanje, tj. u isto se vrijeme može deklarirati više varijabli istoga tipa tako da se varijable odvoje operatorom zarez, kao i dodjeljivanje inicijalne vrijednosti. Neinicijalizirane varijable automatski se postavljaju na nulu.

Kao i kod drugih programski jezika, u jeziku KC64 je omogućeno izračunavanje izraza. Izraze je moguće upotrijebiti u *if* naredbama, *while* ili *do* petljama ili prilikom pridruživanja vrijednosti nekoj varijabli.

Operatori <, >, <=, >=, ==, !=, <> vraćaju rezultat 1 ili 0.

Od aritmetičkih operatora koriste se uobičajeni: zbrajanje, oduzimanje, množenje, dijeljenje, modul, potenciranje te drugi navedeni u tablici 4.3.

U tablici 4.3 nalazi se pregled svih operatora i njihovih asocijativnost i prioriteta, a tablici 4.4 brzi pregled sintakse naredbi.

4.1.3 LL(1)-gramatika jezika KC64

Odlike LL(1) gramatika su: parsiranje od vrha prema dnu sa gledanjem jednog znaka unaprijed, lijeva strana produkcije sastoji se isključivo od jednog nezavršnog znaka, a desna može biti mješavina završnih i nezavršnih znakova ili prazni skup. Desna strana produkcije ne smije sadržavati lijevu rekurziju, a produkcije sa istom lijevom stranom moraju imati različiti skup generirajučih znakova na desnoj strani.

Zbog ovakvih odlika LL(1) parseri se vrlo često programski izvode u tehnici rekurzivnog spusta.

Za programski jezik KC64 izgrađena je LL(1)-gramatika, čije su produkcije:

1. <S> → VT *IDN* **(** <PAR> **) {** <DEK> <NN> **}** <S>
2. <S> → ε
3. <PAR> → <R> VT *IDN* <XPAR>
4. <PAR> → ε
5. <XPAR> → **,** <R> VT *IDN* <XPAR>
6. <XPAR> → ε
7. <R> → byval
8. <R> → ε

Produkcije 1. i 2. deklariraju 1 ili više funkcija sa 0 ili više parametara. Produkcije 3. i 4. deklariraju 0 ili više parametara funkcije. Produkcije 5. i 6. deklariraju 0 ili više dodatnih parametara funkcije. Produkcije 7. i 8. definiraju prenošenje parametara po referenci ili vrijednosti.

9. <DEK → VT *IDN* <INI> <DI> **;** <DEK>
10. <DEK → ε
11. <DI> → **,** *IDN* <INI> <DI>
12. <DI> → ε
13. <INI> → **=** *CONST*
14. <INI> → ε

Produkcije 9. i 10. deklariraju 0 ili više varijabli različitih tipova podataka. Produkcije 11. i 12. deklariraju deklaracija 0 ili više varijabli istog tipa (dodatni identifikatori). Produkcije 13. i 14. omogućuju postavljanje varijable na početnu vrijednost (inicijalizacija).

15. <NN> → <BN> <NN>

16. <NN> → ε

17. <BN> → **{** <NN> **}**

Produkcije 15. i 16. definiraju niz naredbi ili blokova naredbi, a produkcija 17. definira blok naredbi

18. <BN> → *IDN* <Z2> ASGN <IZR> **;**

19. <BN> → **@** *IDN* <Z2> ASGN <IZR> **;**

Produkcija 18. deklarira naredbu pridruživanja, a produkcija 19. deklarira naredbu pridruživanja preko pokazivača.

20. <BN> → **;**

Produkcija 20. deklarira praznu naredbu.

21. <BN> → if **(**<IZR> **)** <BN> <ELSE>

22. <ELSE> → else <BN>

23. <ELSE> → ε

Produkcija 21. deklarira naredbu uvjetnog grananja *ako* (*if*), koja može, ali i ne mora imati blok naredbi u suprotnom slučaju *inače* (*else*), produkcije 22. i 23.

24. <BN> → while **(** <IZR> **)** <BN>

Produkcija 24. definira programsku petlju *dok je...* (*while*)

25. <BN> → do <BN> until **(** <IZR> **) ;**

Produkcija 25. definira programsku petlju *ponavljaj… dok ne bude...* (*do… until…*)

26. <BN> → assembly **{** *ASMTEXT* **}**

Produkcija 26. definira dio programskog kôda napisanog u mnemoničkom strojnom jeziku, *ASMTEXT* je tekstualna konstanta (string)

27. <IZR> → **(** <IZR> **)** <dalje> – ugnježdeni aritmetičko-logički izrazi

28. <IZR> → *IDN* <ZAG> <dalje> – identifikator, element polja identifikatora ili funkcija

29. <IZR> → **-** *IDN* <ZAG> <dalje> – unarni minus

30. <IZR> → **@** *IDN* Z2 <dalje> – pokazivač ili element polja pokazivača

31. <IZR> → *CONST* <dalje> – konstanta

32. <dalje> → OP <IZR> – nastavak aritmetičko-logičkog izraza

33. <dalje> → ε

Produkcije 27. – 33. definiraju aritmetičko-logičke izraze.

34. <ZAG> → **(** <P> **)**

35. <ZAG> → **[** <IZR> <XP> **]**

36. <ZAG> → ε

Produkcije 34. – 36. definiraju da li je identifikator u produkcijama 28. i 29. funkcija, polje ili varijabla.

37. <Z2> → **[** <IZR> <XP> **]**

38. <Z2> → ε

Produkcije 37. i 38. definiraju da li je pokazivač u produkciji 30. element polja ili nije. Polje može biti višedimenzionalno, produkcije 41. i 42.

39. <P> → <IZR> <XP> – 0 ili više parametara funkcije

40. <P> → ε

41. <XP> → **,** <IZR> <XP> – dodatni parametri funkcije ili dimenzije polja

42. <XP> → ε

Produkcije 39 – 42 definiraju broj dimenzija polja ili broj parametara funkcije u produkcijama 28. i 29. ukoliko je identifikator polje ili funkcija.

43. OP → vidi tablicu aritmetičkih i logičkih operatora, tablica 4.3

44. VT → byte, int, long, single, string, bool, void

45. ASGN → vidi tablicu operatora pridruživanja, tablica 4.3

Produkcije 43. – 45. nisu prave produkcije već predstavljaju sažeti zapis završnih znakova koji se mogu pojaviti u produkcijama 1., 3., 5., 9., 18., 19., i 32.

Nezavršni znakovi obilježeni su šiljatim zagradama < >, završni znakovi koji odgovaraju elementima tablica identifikatora ili konstanti napisani su *kosim stilom*, elementarni završni znakovi su **podebljani**, crnom pozadinom označeni su završni znakovi koji se nalaze u KROS tablici.

Tablica 4.4 Brzi opis sintakse naredbi:

Tip naredbe	Sintaksa	Opis

deklaracija funkcije	`tip ime(ptip parametar,…)`
	`naredba;`
	ili
	`tip ime(ptip parametar,…){`
	`  blok naredbi;`
	`}`
	ili
	`void main(){`
	`  blok naredbi;`
	`}`
		Deklarira funkciju imena `ime`, sa parametrima `parametar` tipa `ptip`, povratne vrijednosti `tip`. Povratna vrijednost kao i svi od navedenih parametara mogu biti različitih tipova podataka. Funkcija ne mora imati parametre. Iznimka je funkcija `main`, koja nema parametre i mora biti tipa `void`.

deklaracija varijabli	`tip varijabla;`
	ili
	`tip varijabla,varijabla2,…;`
	ili
	`tip varijabla=konstanta;`
	ili
	`tip var=konst, var2=konst2, …;`
		Deklarira jednu ili više varijabli istog tipa podatka, kojima možemo odmah i pridružiti inicijalnu vrijednost.

pridruživanje	`varijabla = izraz;`	Pridružuje varijabli rezultat aritmetičko-logičkog izraza. Umjesto operatora = može biti bilo koji drugi operator pridruživanja iz tablice 4.2

programska petlja while	`while(izraz) naredba;`
	ili
	`while(izraz){`
	`  blok naredbi;`
	`}`
		Ispituje *izraz*, te ako je rezultat izraza istina ili broj različit od nule, ponavlja naredbu ili blok naredbi koje mogu biti bilo koje od ovdje opisanih.

programska petlja do..until	`do naredba; until(izraz);`
	ili
	`do{`
	`  blok naredbi;`
	`}until(izraz);`
		Obavlja naredbu ili blok naredbi te potom ispituje *izraz*. Ako je rezultat izraza istina ili broj različit od nule, ponavlja ovaj postupak.

uvjetno grananje if..else	`if(izraz) naredba;`
	ili
	`if(izraz) naredba;`
	`else naredba2;`
		Ispituje *izraz* te ako je rezultat izraza istinit ili različit od nule obavlja `naredbu`, u protivnom obavlja `naredbu2` ukoliko postoji `else` dio. Umjesto `naredbe` i/ili `naredbe2` mogu biti blokovi naredbi.

4.1.4 Semantička pravila jezika KC64

U programskom jeziku KC64 moguće je koristiti pet osnovnih tipova podataka:

1. cjelobrojni tipovi (8,16, 32 bitni)
2. brojevi sa posmačnim zarezom (40 bitni)
3. znakovni tip (proizvoljne, ali konstantne duljine)
4. logički tip (0 ili 1, zauzima 1 bajt u memoriji)
5. prazni tip (engl. void)

Prilikom deklaracije, varijable poprimaju neku od sljedećih oznaka tipa podatka: *byte*, *int*, *long*, *single*, *string*, *bool*.

Cjelobrojni tip podataka može poprimiti 8, 16 ili 32 bitni broj ovisno od podtipu, znakovni tip bilo koji broj PETSCII* znakova, a logički tip jednu od dvije mogućnosti 0 ili 1. Prazni tip koristi se kao naznaka da funkcija nema povratnu vrijednost. Cjelobrojni tipovi podataka duljine 16 i 32 bita koriste zapis dvojnog komplementa, dok je 8 bitni cjelobrojni tip podatka nepredznačen.

Svaki tip podatka određuje moguće operacije nad njim. Potenciranje je moguće samo sa brojevima sa posmačnim zarezom, a modul samo nad cijelim brojevima. Operacije dijeljenja nad cijelim brojevima daju kao rezultat najbliži manji cijeli broj u odnosu na stvarni rezultat.

* PETSCII je Commodore-ova inačica ASCII kôda [1]

Kod množenja je moguće napraviti premašenje o kojem nećete biti informirani, a dijeljenje sa nulom se ne provjerava automatski. Sve numeričke tipove podataka moguće je međusobno miješati u izrazima. Svi će biti automatski pretvoreni u najveći tip u tom izrazu.

Znakovni tipovi podataka su fiksne duljine i nad njima nije moguće obavljati nikakve operacije. Nad brojevima sa posmačnim zarezom nije moguće obavljati bitovne operacije (I, ILI, isključivo ILI, komplementiranje i bitovni pomaci).

Logički tip podatka se može koristiti i kao numerički, ali će u tom slučaju biti pretvoren u najpogodniji numerički tip podatka.

Primjer programa napisanog u KC64 programskom jeziku koji sadrži najčešće sintaksne strukture, dan je u ispisu 4.1.

```
' program u jeziku KC64
void main(){
     int a=0,tk;          'deklaracija varijabli a, tk i
     int b=3;             'b i dodjeljivanje početnih vrijednosti
     if (a==5) tk=15;     'ako je a jednako 5 tada dodjeli tk 15
     else{                'inače
          tk=a*(3+b);     'tk dodjeli a*(3+b)
     }
     b=@a+6;              'b dodjeli ono na što pokazuje a, i uvećaj za 6
     a=56-(b+37)*15-45;
     while(a<b){          'dok je a < b
          a=(a+1);        'uvećaj a za 1
     }
}
```

Ispis 4.1 Primjer programa napisanog u jeziku KC64

4.2 Međukôd KCIL

Jezik KCIL (**K**ork's **C** **I**ntermediate **L**anguage) je međukôd niže razine koji pomaže pri pretvaranju kompleksnih konstrukta jezika KC64 u jednostavnije dvoadresne i troadresne naredbe izravnatog sintaksnog stabla. Po svojoj strukturi izgleda kao napredniji strojni jezik, ali nije zamišljen za uporabu, već je isključivo strojno generiran. Struktura jezika je tablica mnemonika i njihovih parametara pohranjena u memoriji koja postoji samo za vrijeme prevođenja KC64 programa. Moguće je tražiti ispis međukôda za vrijeme prevođenja i to je jedini doticaj koji korisnik ima sa istim.

4.2.1 Opis mnemoničkog kôda

Mnemonički zapis KCIL-a sastoji se od troslovčanog akronima ili kratice punog imena naredbe i onoliko parametara koliko ta naredba ima. Uz parametre svaka naredba može imati i labelu. U listi 4.3 nalaze se svi mnemonici KCIL-a i njihovi opisi.

Lista 4.3, Mnemonici međukôda KCIL i njihovi parametri:

Mnemonik	Parametri	Opis
ADD	Res,A,B	Zbroj *A* i *B* smjesti u *Res*
SUB	Res,A,B	Razliku *A* i *B* smjesti u *Res*
MUL	Res,A,B	Umnožak *A* i *B* smjesti u *Res*
DIV	Res,A,B	Kvocijent *A* i *B* smjesti u *Res*
MOD	Res,A,B	Smjesti modulo *B* od *A* u *Res*
POW	Res,A,B	Smjesti A^B u *Res*
CVT	Res,A,Type	Pretvori *A* u tip podatka *Type* te rezultat smjesti u *Res*
MOV	Res,A	Postavi *Res* na vrijednost *A*
LTN	Res,A,B	Ako je $A<B$, tada je *Res*=1, inače *Res*=0
GTN	Res,A,B	Ako je $A>B$, tada je *Res*=1, inače *Res*=0
LEQ	Res,A,B	Ako je $A\leq B$, tada je *Res*=1, inače *Res*=0
GEQ	Res,A,B	Ako je $A\geq B$, tada je *Res*=1, inače *Res*=0
EQU	Res,A,B	Ako je $A=B$, tada je *Res*=1, inače *Res*=0
NEQ	Res,A,B	Ako je $A\neq B$, tada je *Res*=1, inače *Res*=0
AND	Res,A,B	Smjesti u *Res* logičko I nad bitovima *A* i *B*
OR	Res,A,B	Smjesti u *Res* logičko ILI nad bitovima *A* i *B*
NOT	Res,A	Smjesti u *Res* komplementirane bitove A
EOR	Res,A,B	Smjesti u *Res* logičko Isključivo ILI nad bitovima *A* i *B*
ASL	Var,numBits	Aritmetički pomak ulijevo *Var* za *numBits* bitova
ASR	Var,numBits	Aritmetički pomak udesno *Var* za *numBits* bitova
LSR	Var,numBits	Logički pomak udesno *Var* za *numBits* bitova
IF	Label	Ako je varijabla #if = 0, tada skoči na labelu *Label*
IFN	Label	Ako je varijabla #if = 1, tada skoči na labelu *Label*
WHI	Label	Ako je varijabla #if ≠ 1, tada skoči na labelu *Label*, prilagođeno dugim grananjima
JMP	Label	Skoči na labelu *Label*
ASM	TextConstant	Ubaci blok koda napisan u mnemoničkom 6510 strojnom jeziku

4.3 Mnemonički 6510 strojni jezik

Ugrađeni mnemonički 6510 asembler jezičnog procesora KCC64 razlikuje se od tipičnih asemblera i monitora strojnog kôda[1] čestih za Commodore 64 platformu. Glavne razlike su: mogućnost korištenja labela i predefiniranih konstanti, definiranje vlastitih konstanti, povezivanje kôda iz više datoteka i jednostavna povezivost za drugim Windows aplikacijama putem *Clipboard*-a. Datoteke sa programima napisanim u mnemoničkom strojnom jeziku imaju ekstenziju *c64asm*.

4.3.1 Struktura mnemoničkog strojnog jezika

Svaka instrukcija programa mora biti napisana u svom retku isključivo velikim slovima. Ako instrukcija ima labelu, onda labela mora biti napisana nekim od slijedećih znakova: malim slovima, znamenkama i podvlakom, samo znamenka ne smije biti prvi znak u imenu labele. Labela se mora nalaziti na početku retka, treba ju završiti dvotočkom te staviti jedan tabulator između nje i instrukcije. Ukoliko instrukcija nema labelu, treba ju tabulatorom odvojiti od početka retka. Kao labela može se upotrijebiti i heksadekadska konstanta sa četiri znamenke. Tako obilježena instrukcija će se smjestiti na adresu zadanu konstantom, a adrese svih instrukcija

nakon obilježene, biti će izračunate u odnosu na posljednju adresom obilježenu instrukciju. Sve konstante pišu se kao i labele samo sa velikim umjesto malim slovima. Konstanta se definira tako da se na početku retka napiše ime te doda znak jednakosti i vrijednost (koja može biti dekadska ili heksadekadska). Između imena konstante, znaka jednakost i vrijednosti ne smije biti razmaka. Postoje i predefinirane konstante za najčešće memorijske lokacije ulazno-izlaznih jedinica. Njihove oznake i vrijednosti mogu se vidjeti u datoteci *.\Libs\Default assembly consts.txt* relativno u odnosu na imenik gdje je instaliran jezični procesor KCC64. Nakon argumenata instrukcije, moguće je dodati komentar, tako da ga započnemo apostrofom. Komentari mogu biti i samostalni (samo komentar u retku), ali ih također treba započeti apostrofom.

Primjer programa napisanog u mnemoničkom 6510 strojnom jeziku koji ispisuje englesku abecedu u prvi redak ekrana dan je u ispisu 4.2:

Početak retka | Tabulator | Tabulator

```
SCREEN=$0400
        LDY #26          'inicijalizacija indeksa petlje
abc:    TYA              'kopiranje Y registra u akumulator
        STA SCREEN,Y     'ispis znaka na ekran
        DEY              'dekrement indeksa petlje
        BNE abc          'ako nije gotovo vrati se na abc
        RTS              'izlaz
```

Ispis 4.2 Primjer programa u mnemoničkom strojnom jeziku

4.3.2 Pregled naredbi ugrađenog asemblera i 6510 mnemonika

Osim mnemonika instrukcijskog seta procesora 6510, prikazanih u listi 4.4, mnemonički strojni jezik ima i naredbe za: umetanje drugog programa u strojnom jeziku na trenutno mjesto, definiranje konstante i definiranje podatkovne linije opisane u tablici 4.5, tzv. asemblerske direktive.

Tablica 4.5 Asemblerske direktive ugrađenog asemblera jezičnog procesora KCC64

Naredba	Opis
#include <*datoteka*>	Umetanje datoteke iz imenika biblioteka
#include "datoteka"	Umetanje datoteke iz istog imenika kao i trenutna datoteka
#data v_1 v_2 v_3 …	Oznaka niza bajtova u brojčanom obliku (dekadski ili heksadekadski, npr. #data 14 221 $F1 $0A …)

Lista 4.4 Mnemonici mikroprocesora 6510

ADC Pribroji memoriju akumulatoru sa pretokom
AND Logičko I memorije i akumulatora
ASL Pomakni ulijevo jedan bit (memoriju ili akumulator)

BCC Grananje ukoliko je Pretok obrisan
BCS Grananje ukoliko je Pretok postavljen
BEQ Grananje ukoliko je rezultat nula

BIT Provjeri bitove u memoriji sa akumulatorom
BMI Grananje ukoliko je rezultat negativan
BNE Grananje ukoliko rezultat nije nula
BPL Grananje ukoliko je rezultat pozitivan
BRK Prisilni prekid
BVC Grananje ukoliko je premašenje obrisano
BVS Grananje ukoliko je premašenje postavljeno

CLC Obriši zastavicu pretoka
CLD Prekini dekadski režima rada
CLI Obriši zastavicu onemogućenja prekida
CLV Obriši zastavicu premašenja
CMP Usporedi memoriju i akumulator
CPX Usporedi memoriju i indeksni registar X
CPY Usporedi memoriju i indeksni registar Y

DEC Umanji memoriju za jedan
DEX Umanji indeksni registar X za jedan
DEY Umanji indeksni registar Y za jedan

EOR Isključivo ILI memorije i akumulatora

INC Uvećaj memoriju za jedan
INX Uvećaj indeksni registar X za jedan
INY Uvećaj indeksni registar Y za jedan

JMP Skoči na novu lokaciju
JSR Skoči na novu lokacije spremajući povratnu adresu

LDA Učitaj akumulator iz memorije
LDX Učitaj indeksni registar X iz memorije
LDY Učitaj indeksni registar Y iz memorije
LSR Pomakni udesno jedan bit (memoriju ili akumulator)

NOP Prazna operacija

ORA Logičko ILI memorije i akumulatora

PHA Stavi akumulator na stog
PHP Stavi status procesora na stog
PLA Uzmi akumulator sa stoga
PLP Uzmi status procesora sa stoga

ROL Rotiraj jedan bit ulijevo (memoriju ili akumulator)
ROR Rotiraj jedan bit udesno (memoriju ili akumulator)
RTI Povratak iz prekida
RTS Povratak iz potprograma

SBC Oduzmi memoriju od akumulatora sa posudbom
SEC Postavi zastavicu pretoka
SED Postavi dekadski režim rada
SEI Postavi zastavicu onemogućenja prekida
STA Pohrani akumulator u memoriju
STX Pohrani indeksni registar X u memoriju
STY Pohrani indeksni registar Y u memoriju

TAX Premjesti akumulator u indeksni registar X
TAY Premjesti akumulator u indeksni registar Y
TSX Premjesti statusni registar u indeksni registar X

TXA Premjesti indeksni registar X u akumulator
TXS Premjesti indeksni registar X u statusni registar
TYA Premjesti indeksni registar Y u akumulator

Svi detalji u vezi programiranja u strojnom jeziku i Commodore 64 platformi nalaze se u knjigama [1], [2], [3], [4], te portalu c64.org [6]. Knjiga [1] priložena je u digitalnom obliku u instalaciji jezičnog procesora KCC64.

4.4 Commodore 64 strojni kôd

Ciljni jezik jezičnog procesora KCC64 je Commodore 64 izvršni strojni kôd. Budući da je riječ u nepreglednom nizu brojeva pohranjenih u binarnu datoteku, jedino što se može o njemu reći jest da ima zaglavlje koje ga definira kao Commodore BASIC program čija je jedina linija kôda, aktiviranje strojnog kôda u nastavku kao što se vidi u ispisu 4.3. Definiranje strojnog programa kao BASIC program napravljeno je radi lakšeg učitavanja programa.

Ispis 4.3 Primjer *.PRG datoteke i strojnog kôda (program iz ispisa 4.2, heksadekadski ispis)

```
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
01 08 0B 08 DB 07 9E 32 30 36 31 00 00 00 A0 1A 98 99 00 04 88 D0 F9 60
```

Bajtovi 01,02: Početna adresa programa, $0801 (2041)
Bajtovi 03,04: Adresa strojnog programa sa koje počinje izvršavanje, $080B, nebitno za BASIC program
Bajtovi 05,06: Redni broj BASIC linije, 2011
Bajt 07: Token BASIC naredbe SYS
Bajtovi 08-11: Memorijska adresa od koje počinje strojni program (PETSCII), 2 0 6 1
Bajtovi 12-14: Kraj BASIC programa, 00 00 00
Bajtovi 15-24: Strojni program iz ispisa 4.2, A0 1A 98 99 00 04 88 D0 F9 60

```
        LDY #26             'inicijalizacija indeksa petlje
abc:    TYA                 'kopiranje Y registra u akumulator
        STA SCREEN,Y        'ispis znaka na ekran
        DEY                 'dekrement indeksa petlje
        BNE abc             'ako nije gotovo vrati se na abc
        RTS                 'izlaz
```

5 Jezični procesor KCC64

KCC64 pripada razredu premosnih prevodioca jer se jezični procesor i ciljni jezik izvode na računalima različitih arhitektura.

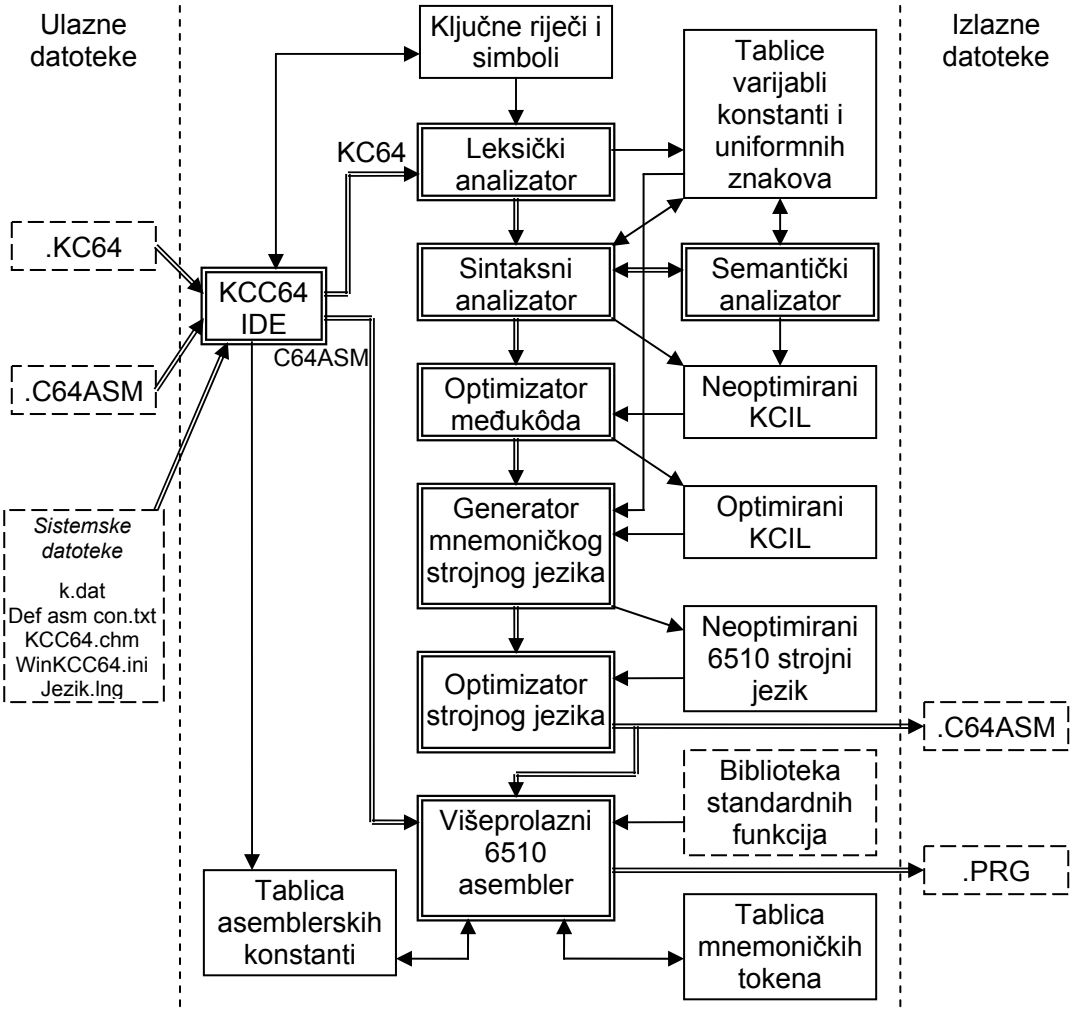
Jezik izgradnje jezičnog procesora KCC64 je programski jezik Visual Basic 6, a ciljna je platforma Windows (win32). Sintaksna pravila izvornog jezika KC64 određena su LL(1)-gramatikom, leksička, regularnim izrazima, a semantička tipom podatka i prioritetom operatora. Ciljni jezik je mnemonički strojni jezik za 6510 procesor kao i strojni kôd izvediv na Commodore 64 računalu. Alternativno kao izvorni jezik može se koristiti mnemonički strojni jezik za 6510 procesor, tada je ciljni jezik strojni kôd izvediv na Commodore 64 računalu, a optimizacije se ne provode.

Za jezični procesor KCC64 implementirani su sljedeći dijelovi: leksički analizator, sintaksni analizator, semantički analizator, generator međukôda, optimizator međukôda, generator mnemoničkog strojnog jezika za 6510 procesor i Commodore 64 platformu, optimizator strojnog programa i izlazni asembler, tj. generator izvršivog strojnog kôda.

5.1 Arhitektura

Središnji dio jezičnog procesora KCC64 jest integrirano razvojno okruženje (engl. *Integrated development enviorment*, IDE). Prilikom pokretanja ono učitava sistemske datoteke koje sadrže: tablicu ključnih riječi, operatora i simbola (KROS) jezika KC64 (k.dat), inicijalne podatke za izgled razvojnog okruženja (WinKCC64.ini), jezik izbornika (može biti Croatian.lng ili English.lng), predefinirane asemblerske konstante (Default assembly consts.txt) i datoteku sa korisničkom pomoći (KCC64.chm).

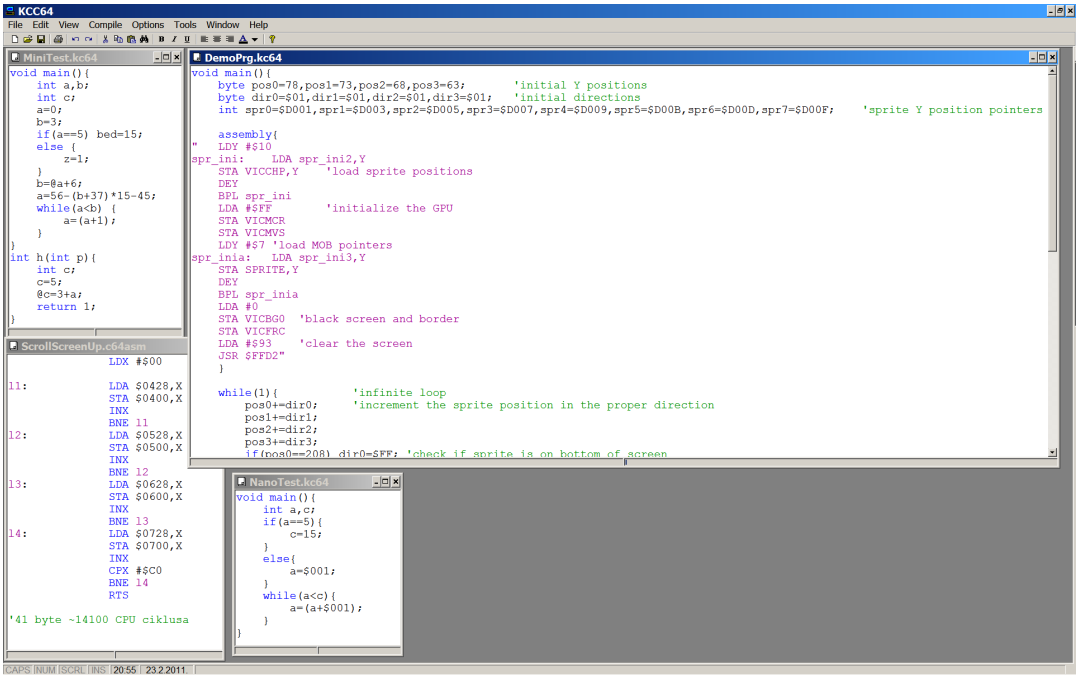
Prilikom procesa prevođenja, IDE slijedno pokreće faze obrade (slika 5.1, široka strjelica) ukoliko ona prethodna nije prijavila grešku u izvornom programu. Zavisno od ulazne datoteke (.kc64 ili .c64asm), prilikom prevođenja može se obaviti ili cjelokupan proces prevođenja od visokog programskog jezika do izvršivog kôda ili samo asembliranje u isti. Na slici 5.1 tankim okvirima obilježene su strukture podataka koje se stvaraju u memoriji računala i koriste tijekom procesa prevođenja. Dvostrukim okvirima označeni su programski moduli koji obavljaju pojedine faze prevođenja, a isprekidanim okvirima označene su datoteke. Tanke strjelice označavaju tok podataka, a široke tijek prevođenja izvornog jezika.



Slika 5.1 Blokovski dijagram jezičnog procesora KCC64

5.2 Integrirano razvojno okruženje (IDE)

Osnovni prozor jezičnog procesora KCC64 je integrirano razvojno okruženje, slika 5.2. To je višedokumentna aplikacija za uređivanje teksta, prilagođena kodiranju u KC64 jeziku i mnemoničkom 6510 strojnom jeziku. U radu sa programima napisanim u jeziku KC64, svaki dokument predstavlja jedinstveni program. Programe napisane u mnemoničkom strojnom jeziku moguće je razdijeliti u više dokumenata koristeći naredbu za umetanje datoteke. Isto je moguće i za programe napisane u mješavini strojnog jezika i jezika KC64, ali samo za dio napisan u strojnom jeziku.



Slika 5.2 Izgled integriranog razvojnog okruženja sa učitanim programima

Izbornici su svojim imenima i funkcijom jednaki su kao i u većini Windows aplikacija te korisnik ne treba mnogo vremena dok ovlada njima. Specifičnost KCC64 jest izbornik „Compile“ / „Prevođenje“ u kojem možemo odabrati potpuno ili prevođenje po fazama analize izvornog kôda. Prevedeni program moguće je i pokrenuti na priloženom emulatoru kao zadnji korak prevođenja.

5.3 Leksički analizator

Leksička pravila jezika KC64 opisana su regularnim izrazima. Ponašanje leksičkog analizatora može se opisati deterministčkim konačnim automatom.

Leksički analizator uzima znak po znak izvornog programa. Ako je prvi znak leksičke jedinke iz neke od skupina, tada leksički analizator prihvaća samo one znakove dozvoljene u toj skupini.

Ukoliko se u bilo kojoj od skupina leksičkih jedinki naiđe na znak koji joj ne pripada, taj se znak ili tretira kao prekidni te predstavlja kraj leksičke jedinke, ili je riječ o znaku koji se ne smije pojaviti na tom mjestu pa leksički analizator dojavljuje pogrešku i pomiče se na slijedeću leksičku jedinku.

5.3.1 Podatkovna struktura leksičkog analizatora

Tablica uniformnih znakova, tablica završnih znakova i izvorni program čine osnovnu podatkovnu strukturu leksičkog analizatora. Tablica završnih znakova funkcijski se razlaže na tri tablice: tablica identifikatora, tablica konstanti i tablica ključnih riječi, operatora i specijalnih znakova (KROS).

Osnovna tablica je tablica uniformnih znakova. U tu tablicu leksički analizator zapisuje uniformne znakove onim redom kojim su leksičke jedinke zadane u izvornom programu.

Tablica uniformnih znakova izgrađena je kao linearno polje. Najvažniji elementi od kojih se sastoji su: tip jedinke, redni broj unutar tablice pripadajućeg tipa jedinke, broj reda i redni broj znaka u izvornom kôdu gdje počinje dotična jedinka.

Tablica identifikatora sadrži zapise za sve identifikatore zadane u izvornom programu i njihov tip i početnu vrijednost, a tablica konstanti sadrži zapise za sve konstante i njihov tip. U tablici konstanti se na početku nalaze upisana osnovne vrijednosti nula i jedan, a u tablici identifikatora je to specijalna logička varijabla *#if*.

Tablica ključnih riječi, operatora i specijalnih znakova sadrži popis svih operatora, specijalnih znakova i ključnih riječi dozvoljenih pravilima jezika. KROS tablica učitava se prilikom pokretanja jezičnog procesora, a nalazi se u datoteci *k.dat*. KROS tablica je podijeljena na dva dijela. U gornjem dijelu se nalaze operatori i specijalni znakovi, a u donjem dijelu ključne riječi. Ovako je donekle olakšano pretraživanje tablice, jer znamo otkuda treba krenuti pretragu, a i sama tablica je sortirana prema ASCII vrijednostima sadržanih elemenata.

Ako se datoteka otvori može se vidjeti da se na početku nalaze dva broja. Prvi broj predstavlja ukupan broj elemenata u tablici, a drugi broj predstavlja broj operatora i specijalnih znakova. Osim toga, može se primijetiti da je pored svake oznake jedan broj. Taj broj predstavlja tip. Mogući tipovi dani su u tablici 5.1.

Tablica 5.1 Tipovi znakova u KROS tablici

Oznaka tipa	**Značenje oznake**
0	ključna riječ, naredba
1	prekidni znak, posebni simbol
3	ključna riječ, imena tipova podataka
100x – 112x	Prioritet aritmetičko logičkog operatora i njegova asocijativnost (ako je x=1 riječ je o desnoj asocijativnosti, a ako x ne postoji onda je lijeva asocijativnost)

Leksički analizator određuje razred leksičke jedinke, zapisuje uniforman znak u tablicu uniformnih znakova i započinje pretraživanje ostalih tablica. Ako je leksička jedinka identifikator, onda se pretražuje tablica identifikatora. Ako se ustanovi da je traženi identifikator već od prije zapisan u tablici identifikatora, onda se u tablicu uniformnih znakova zapisuje kazaljka koja pokazuje na mjesto pronađenog zapisa. Ako traženi identifikator nije zapisan u tablici identifikatora, onda se stvara novi

zapis u tablici identifikatora i u tablicu uniformnih znakova se zapisuje kazaljka koja pokazuje na novonastali zapis. Isto vrijedi i za slučaj kada je leksička jedinka konstanta. Razlika nastupa u slučaju kad leksička jedinka pripada klasi ključnih riječi ili specijalnih znakova ili operatora, jer se KROS tablica ne mijenja tijekom rada jezičnog procesora. Ako leksička jedinka pripada jednoj od KROS klasa tada se u tablicu uniformnih znakova zapisuje uniformni znak KROS i kazaljka koja pokazuje na pronađeni zapis u tablici ključnih riječi, operatora i specijalnih znakova. Ne postoji li zapis u tablici koji je jednak traženom, leksička jedinka je identifikator. Iz ovoga sa vidi da su dvije osnovne operacije nad tablicama traženje zapisa i dodavanje novog zapisa u tablicu. O tim operacijama ovisi brzina izvođenja programa.

5.3.2 Opis programskog ostvarenja

Na početku leksičkom analizatoru IDE šalje pokazivač na otvorenu datoteku u kojoj se nalazi izvorni program. Također, leksičkom analizatoru je omogućen pristup svim tablicama: tablici identifikatora (IDN), tablici konstanti (KON), tablici ključnih riječi, operatora i specijalnih znakova (KROS) i tablici uniformnih znakova.

Leksički analizator učitava znak po znak iz datoteke s izvornim programom. Ukoliko je prvi znak koji učita oznaka novog reda, povećava brojač linija izvornog koda i ponovno učitava novi znak. Ukoliko je sljedeći pročitani znak slovo, brojka ili podvlaka, dodaje ga u pomoćni niz `word`. Ako je ulazni znak oznaka praznine, tabulatora ili pak znak novog reda leksički analizator provjerava stanje pomoćnog niza. Ako je niz `word` nije prazan, onda ih šalje na analizu. Ukoliko je niz prazan, učitava se novi znak. Zadnja mogućnost je čitanje prekidnog znaka, nakon kojeg ponovno ispituje stanje niza `word`. Svaki prekidni znak se također šalje na analizu. Nakon svake analize niz `word` se prazni.

Nakon što se niz `word` pošalje na analizu, pristigli znakovi se smatraju jednom leksičkom jedinkom. Sada analizator pušta niz kroz deterministički konačni automat, koji provjerava zadovoljava li pristigli niz znakova leksička pravila jezika.

Na kraju dobivamo popunjenu tablicu uniformnih znakova, koja predstavlja osnovnu podatkovnu strukturu sintaksnog analizatora.

Leksička analiza poziva se iz glavnog programa, a ostvarena je pomoću pet osnovnih funkcija;

1.) `Public Function LexAn(text As String) As Boolean` - Poziva se unutar glavnog programa. Funkcija prima tekst izvornog programa. Vrača jednu logičku vrijednost koja označava uspješnost leksičke analize. Obavlja učitavanje znakova u niz `word` i slanje niza na analizu. Odgovara osnovnom stanju DKA. Također ispisuje neke od leksičkih grešaka.

2.) `Private Sub Analyse(word As String, BrLin As Long, CharNr As Long)` - Poziva se unutar funkcije `LexAn`. Funkcija prima leksičku jedinku i trenutni brojač broja linija koda i redni broj početnog znaka leksičke jedinke izvornog programa. Obavlja analizu leksičke jedinke i u njoj je implementiran DKA. Kada pronađe o kojoj se leksičkoj jedinci radi popunjava odgovarajuće tablice. Također ispisuje neke od leksičkih grešaka.

3.) `Public Function BinSearchKeyWord(testWord As String) As Long` - Poziva se unutar funkcije `Analyse`. Funkcija prima leksičku jedinku i pretražuje tablicu KROS u potrazi za ključnom riječi. Ukoliko ju pronađe, vraća njezin redni broj, u suprotnom vraća -1.

4.) `Public Function BinSearchOperator(testWord As String) As Long` - Poziva se unutar funkcije `Analyse`. Funkcija prima leksičku jedinku i pretražuje tablicu KROS u potrazi za operatorom ili specijalnim simbolom. Ukoliko ga pronađe, vrača njegov redni broj, u suprotnom vraća -1.

Prilikom ostvarivanja rada leksičkog analizatora korištene su četiri globalne varijable:

1.) `KW(1,n)` – Dvodimenzionalno znakovno polje koje sadrži ključne riječi, operatore i specijalne simbole kao i njihov tip i prioritet.

2.) `Identifikatori();` - Sadrži imena identifikatora, njihov tip i početnu vrijednost. Kao tip podatka koristi strukturu `Vrijednosti`.

3.) `Konstante();` - Sadrži konstante, njihov tip i ime. Kao tip podatka koristi strukturu `Vrijednosti`.

4.) `Unifrom()` - Linearno polje koje sadrži elemente tablice uniformnih znakova. Sastoji se od oznake tipa uniformnog znaka (KROS, identifikator, konstanta), njegovog rednog broja unutar odgovarajuće tablice, broj linije na kojem se jedinka nalazi u izvornom programu. Kao tip podatka koristi strukturu `UnZnakovi`.

5.4 Sintaksni analizator

Sintaksna analiza, koju obavlja sintaksni analizator, izvedena je kao zasebni prolaz nakon leksičke analize. Sintaksni analizator koristi definirane produkcije LL(1) gramatike jezika KC64 opisane u poglavlju 4.1.3 i pomoćne funkcije opisane u poglavlju 5.4.2. Svaka produkcija gramatike jezika KC64 izvedena je kao jedna funkcija i koristi tehniku rekurzivnog spusta.

5.4.1 Načelo rada sintaksnog analizatora

Sintaksni analizator kreće od tablice uniformnih znakova generiranoj u leksičkoj analizi i KROS tablice.

Općenito, sintaksni analizator kreće od početne produkcije <S> i analizira njezinu desnu stranu. Ukoliko se na desnoj strani produkcije nalazi završni znak, analizator provjerava nalazi li se on u KROS tablici. U potvrdnom slučaju pomiče se na sljedeći znak u produkciji, suprotno, ispisuje adekvatnu poruku o grešci. Ukoliko se na desnoj strani produkcije nalazi nezavršni znak, sintaksni analizator, koristeći tehniku rekurzivnog spusta, poziva funkciju produkcije tog nezavršnog znaka. Ukoliko je ta funkcija vratila grešku, na temelju dosad obrađenih ulaznih znakova određuje se mjesto pogreške, ispisuje poruka i poziva postupak oporavka od pogreške.

Ovaj postupak općeniti je za ostvarenje sintaksnog analizatora i implementiran je za svaku produkciju zasebno. Poziv semantičkog analizatora događa se na mjestima na kojima to gramatika zahtjeva (aritmetičko-logički izrazi), u svrhu omogućavanja generiranja međukôda i ciljnog programa.

5.4.2 Tehničke značajke

Sintaksni analizator poziva se iz glavnog programa te će se izvoditi samo u slučaju da je leksički analizator prethodno ispravno završio. Prilikom parsiranja aritmetičko-logičkih izraza poziva funkcije semantičkog analizatora i generira neke od naredbi međukôda KCIL. Sve funkcije produkcija nalaze se u programskom modulu SyntaxAnalyzer.bas. Svaka funkcija produkcije nazvana je `Prod_`*`imeprodukcije`* koje odgovara imenu produkcija iz poglavlja 4.1.3. Uz funkcije produkcija koriste se još i slijedeće funkcije:

1. `Private Function IsKeyWordText(KeyW As String) As Boolean` – ova funkcija provjerava da li je trenutni uniformni znak ključna riječ `KeyW`, alternativno možemo ključnu riječ provjeriti prema njezinom rednom broju funkcijom `IsKeyWordNum`
2. `Private Function IsVariableType() As Boolean` – provjerava da li je trenutni uniformni znak ključna riječ oznake tipa podatka
3. `Private Function IsSymbolText(Smb As String) As Boolean` – provjerava da li je trenutni uniformni znak operator ili drugi simbol `Smb`, alternativno možemo simbol provjeriti prema njegovom rednom broju funkcijom `IsSymbolNum`
4. `Private Function IsOperator() As Boolean` – provjerava da li je trenutni uniformni znak aritmetičko-logički operator
5. `Private Function IsAssign() As Boolean` - provjerava da li je trenutni uniformni znak operator pridruživanja
6. `Private Function IsIdentifyer() As Boolean` - provjerava da li je trenutni uniformni znak identifikator
7. `Private Function IsConstant() As Boolean` - provjerava da li je trenutni uniformni znak konstanta
8. `Private Sub DetermineVarType()` – određuje o kojem tipu podatka se radi ako je uniformni znak neka od ključnih riječi oznake tipa podatka

9. `Public Function nadjiIFmedjurez() As Long` – vraća redni broj specijalne varijable #if iz tablice

Primjerice u produkciji 44. koristi se sintaksna funkcija 8, koja određuje tip podatka identifikatora ili funkcije.

5.5 Semantički analizator

Semantički analizator izvršava se na zahtjev sintaksnog analizatora (tzv. sintaksno pogonjena semantička analiza) te provjerava kompatibilnost tipova podataka i prioritete operatora u aritmetičko-logičkim izrazima. Ukoliko su varijable i konstante kompatibilnih, ali ne i jednakih tipova podataka, semantički analizator obavlja automatsku konverziju slabijeg u jači tip podatka, u protivnom ispisuje pogrešku. Kod provjere prioriteta operatora, semantički i sintaksni analizator grade dva stoga, operandski i operatorski, tzv. semantičke stogove. Izlaz semantičkog analizatora je niz slijednih naredbi dvoadresnog i troadresnog međukôda KCIL, koje se nalaze zapisane u linearnom polju.

5.5.1 Dinamika izvođenja aritmetičko-logičkih izraza

Svaki puta kada sintaksni analizator naiđe na operator stavlja ga na operatorski stog, a varijable i konstante stavlja na operandski stog. Ta procedura traje toliko dugo dok je prednost operatora kojeg treba staviti na stog veća od onog koji se trenutno nalazi na vrhu stoga. Tada se uzima operator sa vrha stoga, generira naredba međukôda koja obavlja traženu operaciju, a sa operandskog stoga se skidaju operandi i stavlja novi međurezultat na vrh operandskog stoga. Ovaj se postupak ponavlja dokle god je operator na vrhu stoga veće prednosti od onog kojeg je sintaksni analizator želio staviti na operatorski stog.

Opisana procedura ponavlja se toliko dugo dok sintaksni analizator ne naiđe na kraj aritmetičko-logičkog izraza, te tada pokreće postupak pražnjenja semantičkih stogova, koji se sastoji od prijeopisanog postupka pražnjenja operatorskog stoga. Nakon toga se resetira brojač međurezultata te se za idući izraz koriste iste varijable međurezultata jer njihova vrijednost sada više nije bitna. Ponovo korištenje međurezultata je bitno jer ciljna platforma ima malo radne memorije.

5.5.2 Nejednoznačnosti

Semantički analizator svakoj aritmetičko-logičkoj naredbi međukôda pridružuje ulazni i izlazni tip podatka, te tako rješava problem nejednoznačnosti aritmetičko-logičkih operatora. Ostale naredbe jezika KC64 nemaju nejednoznačnosti.

5.5.3 Semantičke pogreške i postupci oporavka od pogreške

Ukoliko se dogodi neka od prijeopisanih semantičkih pogrešaka, semantički analizator ih ispisuje u listu grešaka. Nakon toga se prazne semantički stogovi.

5.5.4 Tehničke značajke

Semantička analiza poziva se iz svih funkcija produkcija gramatike tijekom sintaksne analize, a ostvarena je pomoću ovih funkcija:

1.) `Public Sub OperNaStogu()` – obrađuje operator na vrhu semantičkog stoga operatora. Nakon generirane naredbe međukôda, skida operator i operande sa respektivnih stogova i stavlja međurezultat operacije na vrh semantičkog stoga operanada.

2.) `Private Sub DvoadresnaNaredba(brAdr As Long)`, `Private Sub TroadresnaNaredba(rez As SemStog)` – generiraju dvoadresnu odnosno troadresnu naredbu međukôda zavisno o operatoru na vrhu operatorskog stoga.

3.) `Public Sub DodajOperStog(br As Long)` – dodaje operator na vrh operatorskog stoga ukoliko je većeg prioriteta od onoga na vrhu stoga, u suprotnom poziva `OperNaStogu` toliko dugo dok operator na vrhu stoga ne bude manjeg prioriteta od operatora kojeg treba staviti na stog.

4.) `Public Sub SkiniIdenStog(brPar As Long)` – skida proizvoljan broj elemenata sa vrha operandskog stoga

5.) `Public Sub SkiniOperStog(brPar As Long)` – skida proizvoljan broj elemenata sa vrha operatorskog stoga

6.) `Public Sub ZatvorenaZagrada()` – ukoliko je sintaksni analizator naišao na otvorenu zagradu, stavlja ju na vrh operatorskog stoga, svi operatori imaju veći prioritet od otvorene zagrade pa se postupa parsiranja nastavlja normalno. Kada naiđe na zatvorenu zagradu, poziva ovu funkciju koja obrađuje operatora sa vrha stoga dok ne dođe do otvorene zagrade na operatorskom stogu, te skida i nju. Nakon toga nastavlja se sa daljnjim parsiranjem aritmetičko-logičkog izraza.

7.) `Public Sub IsprazniStogove()`– ukoliko je sintaksni analizator naišao na kraj izraza, poziva ovu proceduru koja obrađuje preostale elemente na stogovima. Ukoliko se stogovi ne mogu u potpunosti isprazniti, javlja poruku o pogrešci.

8.) `Private Function CheckDataTypeOnStack() As Boolean` – provjerava da li su operandi neke aritmetičko-logičke operacije istog tipa. U protivnom provjerava da li su kompatibilni te ako jesu provodi postupak automatske konverzije tipova podataka, u protivnom prijavljuje pogrešku.

9.) `Private Function DodajMedjurez() As Long` – vraća redni broj slijedećeg praznog međurezultata iz tablice identifikatora ako on već postoji, u protivnom dodaje novi međurezultat u tablicu identifikatora.

10.) `Public Sub DodajIdenStog(`*`više parametara`*`)` – dodaje varijablu ili konstantu na vrh operandskog stoga

11.) `Public Sub dodajKCIL(`*`veoma mnogo parametara`*`)` – dodaje naredbu međukôda na kraj polja naredbi međukôda. Ovu proceduru poziva isključivo sintaksni analizator.

5.6 Generator mnemoničkog strojnog koda

Generator mnemoničkog strojnog koda je prvi korak pri sintezi ciljnog programa.

5.6.1 Načelo rada

Iz optimiranog KCIL-a generira se mnemonički strojni kôd, a nakon strojno zavisne optimizacije tog kôda, poziva se višeprolazni asembler koji izračunava adrese procesorskih instrukcija i zamjenjuje simboličke labele i konstante stvarnim vrijednostima. Također određuje o kojoj se instrukciji radi i u kojem je adresnom modu te kolika je dužina takve instrukcije. Na kraju dodaje se zaglavlje i stvara .PRG datoteka koju je moguće izvršiti u priloženom Commodore 64 emulatoru (autor, Per Håkan Sundell) ili ju je moguće posebnom procedurom prebaciti na 1541 disketnu jedinicu te izvršiti na pravom Commodore 64 računalu (taj je postupak nešto složeniji i zahtjeva prilagodni hardver)

5.6.2 Opis programskog ostvarenja

Programski modul Commodore64_Generator sadrži procedure koje iz KCIL-a generiraju mnemonički strojni kôd. Svaka instrukcija KCIL-a preslikava se u jednu ili više slijednih naredbi mnemoničkog strojnog jezika. Izbor instrukcija ovisi o tipu podatka kojima KCIL instrukcija rukuje i odabiru optimizacije, ovisno o tome da li smo odabrali brži ili manji kôd. Na kraju kôda dodaju se asemblerske biblioteke sa rutinama za zbrajanje, množenje, dijeljenje itd. kao i varijable, prostor za međurezultate i konstante. Taj se kôd zapisuje u privremenu datoteku koja se briše nakon strojno zavisne optimizacije. Optimirani program ostaje na disku u datoteci

istog imena kao i izvorni program, ali sa ekstenzijom .c64asm. Nakon finalnog asembliranja stvorena je datoteka istog imena kao i izvorni program no zapisanog samo velikim slovima, ograničenog na najviše 16 znakova i sa ekstenzijom .PRG. To je potrebno radi kompatibilnosti sa Commodore 64 platformom.

5.7 Optimizator

Jezični procesor KCC64 podržava strojno nezavisno i strojno zavisno optimiranje. Strojno nezavisno optimiranje bavi se optimiranjem međukôda, a strojno zavisno optimiranjem mnemoničkog strojnog kôda. Strojno nezavisno optimiranje zove se tako jer optimizacije primijenjene na međukôdu pomažu bilo kojem ciljnom jeziku generiranom iz tog međukôda. Strojno zavisno optimiranje uzima u obzir cjelokupnu platformu na kojoj se ciljni program izvodi, zbog toga je mnogo kompleksnije od strojno nezavisnog optimiranja.

5.7.1 Implementacija

Strojno nezavisno optimiranje provodi se na KCIL-u. Postupak parsiranja aritmetičkih izraza daje niz slijednih instrukcija KCIL-a, ali ne daje optimalan broj naredbi. Kako se nakon svake operacije njezin međurezultat stavlja na stog, događa se da postoji previše *MOV* naredbi koje se mogu izbjeći ako pogledamo od kuda se uzima i kamo stavlja međurezultat, kao što je demonstrirano u tablici 5.2:

Tablica 5.2 Usporedba neoptimiranog i optimiranog medukôda

Izvorni program	Generirani KCIL	Optimirani KCIL

`void main(){`
`  byte i=$00;`
`  int j=$000E;`
`  while(i<j){`
`  i=i+$02;`
`  }`
`}`
	main: CVT #mr1 i int
	LTN #mr0 #mr1 j
	MOV **#if** #mr0
	WHI lp0:
	ADD #mr0 i 2
	MOV **i** #mr0
	JMP main:
	lp0: RTS
		main: CVT #mr1 i int
		LTN **#if** #mr1 j
		WHI lp0:
		ADD **i** i 2
		JMP main:
		lp0: RTS

Kako varijable ***i*** i ***j*** nisu istoga tipa provodi se konverzija vrijednosti varijable ***i*** u međurezultat *#mr1*. Ako je taj međurezultat manji od ***j*** tada se stavlja 1 u *#mr0* u protivnom se stavlja 0. Premještanje iz međurezultata *#mr0* u specijalnu varijablu *#if* može se izbjeći kao što se vidi u optimiranom KCIL-u. Ako je *#if* jednak 0 skače se na labelu *lp0* u protivnom zbraja se 2 i varijabla ***i*** i rezultat smješta u *#mr0*. Kako se premještanje *#mr0* u ***i*** može izbjeći ako se odmah varijabli ***i*** pribraja 2, ta je naredba višak, te je izbačena u optimiranom KCIL-u.

Strojno zavisno optimiranje provodi slične postupke kao i strojno nezavisno, ali na mnemonicima 6510 strojnog jezika. Generirani program u strojnom jeziku, kao i međukôd iz kojega je nastao, ima nepoželjno svojstvo viška naredbi premještanja

podataka, kao što se vidi u tablici 5.3.

Tablica 5.3 Usporedba optimiranog i neoptimiranog strojnog jezika

Optimirani KCIL	Generirani asembler	Optimirani asembler
main: CVT #mr1 i int LTN #if #mr1 j WHI lp0: ADD i i 2 JMP main: lp0: RTS	main: LDA var1 STA mrz1+1 BPL 4 LDA #$FF BMI 2 LDA #$00 STA mrz1 **LDA mrz1** CMP var2 BMI 8 LDA mrz1+1 CMP var2+1 BPL 4 LDA #1 BNE 2 LDA #0 STA if **LDA if** BNE 3 JMP lp0 CLC LDA var1 ADC con4 STA var1 JMP main lp0: RTS #include <SpeedOpt.c64asm> var1: #data $0 var2: #data $0 $E mrz0: #data 0 0 0 0 0 mrz1: #data 0 0 0 0 0 con0: #data 0 con1: #data 1 con4: #data $2 if: #data 0	main: LDA var1 STA mrz1+1 BPL 4 LDA #$FF BMI 2 LDA #$00 STA mrz1 CMP var2 BMI 8 LDA mrz1+1 CMP var2+1 BPL 4 LDA #1 BNE 2 LDA #0 STA if BNE 3 JMP lp0 CLC LDA var1 ADC con4 STA var1 JMP main lp0: RTS #include <SpeedOpt.c64asm> var1: #data $0 var2: #data $0 $E mrz0: #data 0 0 0 0 0 mrz1: #data 0 0 0 0 0 con0: #data 0 con1: #data 1 con4: #data $2 if: #data 0

U neoptimiranom strojnim jeziku primjećuje se spremanje u varijablu *mrz1* i ponovo učitavanje iz iste odmah nakon. To nije potrebno te je učitavanje odbačeno. Isto se događa i u drugom označenom dijelu neoptimiranog strojnog jezika sa varijablom *if*. Na ovaj način optimirani strojni jezik je daleko od optimalnog rješenja, ali je ipak nešto bolji nego bez optimizacije.

5.7.2 Tehničke značajke

Optimizator međukôda prolazi jednom kroz cijeli kôd te prepravlja višak MOV naredbi na način da ispituje izvorište MOV naredbe i odredište neposredno prethodne naredbe, ako ga ona ima. Ukoliko je odredište naredbe neposredne MOV naredbi jednako kao i izvorište MOV naredbe, tada se MOV naredba odbacuje, a odredište prethodne naredbe postavlja se na odredište odbačene MOV naredbe. Uz MOV naredbe optimizator traži i IF naredbe te ih mijenja u IFN naredbu ukoliko se neposredno prije nalazi NOT naredba sa odredištem *#if*, koja se pritom odbacuje.

Optimizator strojnog jezika prolazi dva puta kroz kôd. U prvom prolazu optimizator strojnog jezika koristi prozorčić od šest instrukcija te traži uzorak *load-store* u raznim varijantama. Ukoliko se sprema u lokaciju iz koje je neposredno pročitano ili obrnuto, tada se druga instrukcija (ili set instrukcija, ako je riječ o čitanju ili spremanju višebajtnog podatka) odbacuje. U drugom prolazu optimizator strojnog jezika traži instrukcije koje mijenjaju sadržaje registara X i Y. Ukoliko dvije instrukcije pune istu vrijednost u registre X ili Y, a nije bilo promjene vrijednosti registra, skoka u potprogram ili labele u kôdu između te dvije instrukcije, tada se druga instrukcija odbacuje. Optimiranje je implementirano samo za najjednostavnije i najčešće boljke generiranog kôda, višak naredbi premještanja podataka.

5.8 Višeprolazni asembler

Višeprolazni asembler po svojoj kompleksnosti odgovara jednom jezičnom procesoru. On također posjeduje leksički, sintaksni i semantički analizator, samo što su oni tako usko povezani da je gotovo nemoguće raščlaniti gdje završava jedan, a počinje drugi. Leksički analizator asemblera uzima programski redak te mu određuje granice između labele, mnemonika i komentara, a obrađuje i makro naredbe kao što su **#include** (ubacivanje drugog strojnog programa u trenutni) ili **#data** (niz proizvoljne duljine bajtova podataka zapisanih kao dekadski ili heksadekadski brojevi). Sintaksni analizator asemblera provjerava da li je mnemonik ispravno zapisan, a semantički analizator asemblera provjerava da li je argument mnemonika u ispravnom rasponu.

Kod naredbe **#include** moguće je doći do beskonačne rekurzije ukoliko pozivamo program koji je već otvoren, ali nije još asembliran do kraja. Asembler može prepoznati takvu situaciju i prekinuti dalji rad uz upozorenje korisniku.

Programski redci strojnog programa moraju poštovati stroga pravila o formatiranju zapisa opisana u poglavlju 4.3.1.

Svaki ispravni mnemonik zapisuje se u polje mnemoničkih tokena (vrsta međukôda) pomoću kojih se izračunavaju stvarne adrese i veličine procesorskih instrukcija. Moguće je da se pri tom naiđe na grananje koje premašuje opseg *branch* instrukcija. Asembler ima ugrađeni ispravljač predugih grananja, no pri tom mora nanovo izračunati adrese. Zbog toga može imati nepoznati broj prolaza.

Asembler ima oko 1500 linija kôda i bez njega ne bi bilo moguće implementirati strojno zavisno optimiranje, a niti generiranje ciljnog strojnog programa, budući da je rad sa simboličkim adresama neizmjerno jednostavniji od izravnog generiranja strojnog kôda.

6 Upute za korištenje

Rad se KCC64 je veoma jednostavan. Osnovni prozor IDE-a (Integrated Development Enviorment, *hrv. integrirano razvojno okruženje*), nalik je mnoštvu poznatih aplikacija poput MS Word-a ili Adobe Photoshop-a. Riječ je o multi-dokumentnom sučelju u kojem se prikazuju sadržaji trenutno otvorenih programskih ili bilo kojih drugih tekstualnih datoteka.

Prozor dokumenta formatiran je ovisno o tome da li radimo sa KC64 ili općim tekstualnim dokumentom, ili koristimo ugrađeni asembler.

Prilikom prevođenja ili pojedinačnih analiza izvornog koda otvara se prozor sa ispisom grešaka i detaljima pojedine faza rada jezičnog procesora. Klikom na opis pogreške u listi grešaka u prozoru sa izvornim kôdom označava se linija u kojoj se greška najvjerojatnije dogodila. Ponekad može doći do odstupanja u rednom broju linije.

6.1 Opisi izbornika

Izbornici su samoizražajni (*engl. self-explanatory*) i sadrže iste elemente kao i kod drugih Windows aplikacija za uređivanje teksta: Prozor postavki malo je složeniji i omogućava niz podešenja korisničkog sučelja jezičnog procesora KCC64 opisanih detaljnije u nastavku.

6.1.1 Prozor postavki

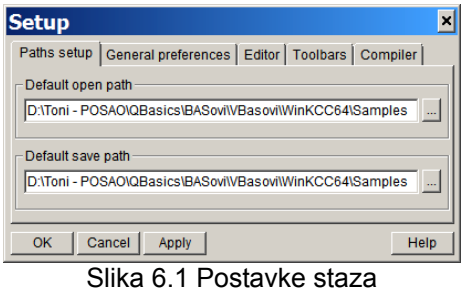
Postavke staza (slika 6.1) omogućuju postavljanje zadane staze otvaranja i snimanja koje će se ponuditi kada korisnik zatraži snimanje dokumenta pod drugim imenom (Save As… / Snimi Kao…).

U općenitim postavkama (slika 6.2) moguće je izabrati jezik izbornika, uključiti ili isključiti automatsko sigurnosno snimanje u odabranim vremenskim intervalima, postaviti maksimalnu veličinu dokumenta nakon koje više neće biti dozvoljeno pisanje, uključiti ili isključiti uvodni prozor (engl. *splash screen*) te vidjeti trenutne postavke zaslona.

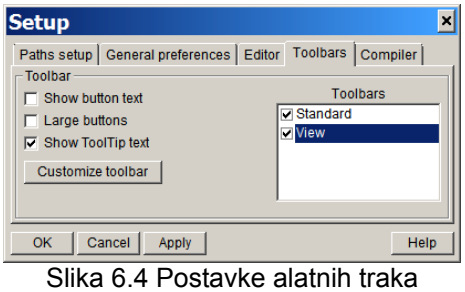
Postavke uređivača teksta (slika 6.3) određuju boje leksičkih jedinki, veličinu i font slova. Moguće je uključiti ili isključiti bojanje leksičkih jedinki.

U postavkama alatnih traka (slika 6.4) moguće je odabrati koje alatne trake će biti aktivne i koje će funkcije sadržavati, veličinu tipaka i prikaz natpisa funkcije.

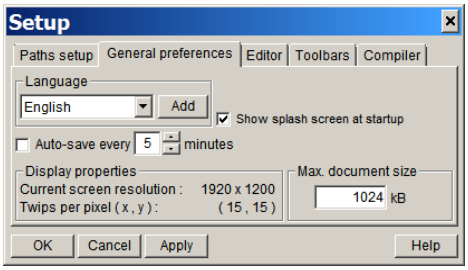
Postavke prevodioca (slika 6.5) omogućuju odabire prikaza detalja pojedinih faza prevođenja, odabir optimizacije i odredišne platforme (za sada samo Commodore 64).



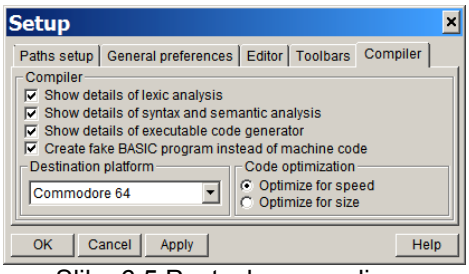
Slika 6.1 Postavke staza



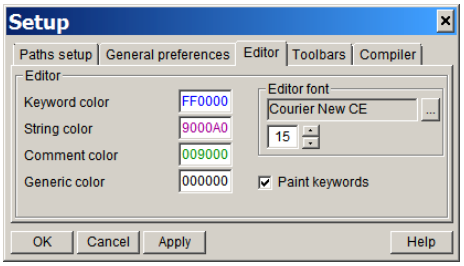
Slika 6.4 Postavke alatnih traka



Slika 6.2 Općenite postavke



Slika 6.5 Postavke prevodioca



Slika 6.3 Postavke uređivača teksta

6.2 Nepodržani elementi jezičnog procesora KCC64

Slijedeće sintaksne i semantičke cjeline nisu podržane ili u potpunosti implementirane: polja i funkcije (generiranje međukôda i niže razine sinteze), množenje i dijeljenje 32 bitnih cjelobrojnih brojeva (nije implementirano u generatoru ciljnog jezika), modulo nad 16 i 32 bitnim cjelobrojnim brojevima (nije implementirano u generatoru ciljnog jezika).

Svi tipovi podataka ne podržavaju sve operacije, detalji o tipovima podataka i operacijama nad njima nalaze se u poglavlju 4.1.4.

6.3 Primjeri programa

Slijedeći primjeri programa napisanih u KC64 i menmoničkom strojnom jeziku demonstriraju mogućnosti tih programskih jezika kao i KCC64 jezičnog procesora. Ovi i mnogi drugi primjeri priloženi su u instalaciji jezičnog procesora KCC64.

6.3.1 KC64 Demo Program

Program iz ispisa u dodatku A prikazuje velika slova TONI KORK na ekranu i pomiče ih gore-dolje u beskonačnoj petlji, slika 6.6. Program je napisan u mješavini visokog programskog jezika i mnemoničkog strojnog jezika. Inicijalizacija grafičkog procesora i slikovni podaci napisani su u mnemoničkom strojnom jeziku (C64asm), a upravljanje sprajtovima po ekranu u visokom programskom jeziku (KC64).

Demo program postoji u više inačica radi uspoređivanja performansi pojedinih izvedbi. Commodore BASIC 2.0 inačica je najsporija od svih, ona koristi ugrađeni BASIC interpretator računala Commodore 64. Brzina pomaka slova jest oko 4 piksla u sekundi. KC64 inačica iz ispisa u dodatku A ima prosječnu brzinu pomaka slova od ~34 piksela po video sličici (50 sličica u sekundi), tj. 1700 piksela u sekundi. Najbrža je inačica napisana u mnemoničkom strojnom jeziku sa prosječnom brzinom pomaka od ~140 piksela u video sličici, tj. 7000 piksela u sekundi.

Slika 6.6 Demo Program u emulatoru

6.3.2 C64asm Sound Player

Program iz ispisa u dodatku B pretvara računalo Commodore 64 u jednostavan glazbeni sintesajzer sa jednom oktavom. Pritiskom na tipke navedene u tablici 6.1 generira se zvuk odabrane note.

Tablica 6.1 Raspored tipaka virutalnog sintesajzera

Tipka	Nota
Z	C1
S	C1#
X	D
D	D#
C	E

Tipka	Nota
V	F
G	F#
B	G
H	G#
N	A

Tipka	Nota
J	A#
M	H
,	C2

7 Zaključak

Današnjim programerima nezamisliv je rad bez jezičnih procesora. Oni omogućuju brzu izradu i ispitivanje programskih projekata. Program napisan u visokom programskom jeziku mnogo je lakše prenijeti na drugu platformu ili ga preinačiti prema potrebi nego program napisan u strojnom kôdu ili mnemoničkom strojnom jeziku. Također strukturiranje i sistematiziranje programskog koda u cjeline različitih razmjera omogućuje preglednost programskog koda pri velikim projektima. To sve ne bi bilo moguće da se za nove probleme i izazove u programskom inženjerstvo ne razvijaju novi i bolji jezični procesori.

U ovom radu ostvaren je jedan jednostavan jezični procesor koji demonstrira sve aspekte kompleksnih jezičnih procesora i osnovne koncepte prevođenja. Poznavanje programskog jezika i jezičnog procesora omogućuje programerima bolje iskorištavanje dostupnih resursa i kvalitetnije prevođenje izvornog kôda. Ovaj rad upravo pokušava upoznati programera sa konceptima prevođenja i tipičnog ponašanja jezičnih procesora na jednostavnim primjerima.

Slijedeći korak u razvoju jezičnog procesora i programskog jezika KC64 bilo bi uvođenje lokalnih varijabli i struktura, kao i unaprjeđenje optimizatora kôda. Moguće je i proširiti skup ulaznih jezika i odredišnih platformi. Unatoč tome primjena ovog jezičnog procesora ostala bi i dalje u edukativne svrhe.

8 Literatura

[1] Commodore Business Machines, Inc.: Commodore 64 Programmer's Reference Guide, First edition, Fourth printing, 1983.

[2] Dragan Tanaskoski, Stevan Milinković, Vladimir Janković: Commodore za sva vremena, Mikro Knjiga, 1985.

[3] Nick Hampshire, Richard Franklin, Carl Graham: The Commodore 64 ROMs revealed

[4] Mihailo Šolajić: Commodore 64 memorijske lokacije, Kompjuter biblioteka, 1989.

[5] Siniša Srbljić: Prevođenje programskih jezika, Element, 2008.

[6] http://www.6502.org/

[7] http://www.c64.org/

[8] http://www.computerbrains.com/ccs64

[9] http://www.wikipedia.org

9 Sažetak

U radu je opisana platforma Commodore 64, pojašnjeni pojmovi jezičnih procesora i metode prevođenja programskih jezika. Programski je ostvaren jezični procesor KCC64 i osmišljeni njegovi jezici KC64, KCIL i 6510 mnemonički strojni jezik te opisana njihova arhitektura i programsko ostvarenje. Dani su primjeri programa u jezicima KC64 i mnemoničkom strojnom jeziku na kojima je demonstriran rad jezičnog procesora KCC64.

Ključne riječi: Commodore 64, 6510, jezični procesor, prevodioc, prevođenje programskih jezika

10 Abstract

This thesis describes the Commodore 64 platform, explains the work principle language processors and programming language translation. Software implementation of KCC64 language processor was made as well as the design its languages KC64, KCIL, and 6510 mnemonic assembly. Architecture of KCC64 and all of its components are in-depth described as well as software implementation of the same. Included program examples written in KC64 and assembly language are there to demonstrate the workflow of KCC64 language processor.

Keywords: Commodore 64, 6510, language processor, compiler, programming language translation

Dodatak A

Ispis "Demo Program.kc64"

```
void main(){
     byte pos0=78,pos1=73,pos2=68,pos3=63;        'inicijalne Y pozicije
     byte dir0=$01,dir1=$01,dir2=$01,dir3=$01;    'inicialni smjerovi
     int spr0=$D001, spr1=$D003, spr2=$D005, spr3=$D007, spr4=$D009,
       spr5=$D00B, spr6=$D00D, spr7=$D00F;  'pokazivači Y pozicija sprajtova

     assembly{
"    LDY #$10
spr_ini:  LDA spr_ini2,Y
     STA VICCHP,Y   'učitaj pozicije sprajtova
     DEY
     BPL spr_ini
     LDA #$FF       'inicijaliziraj GPU
     STA VICMCR
     STA VICMVS
     LDY #$7        'učitaj MOB pokazivače
spr_inia: LDA spr_ini3,Y
     STA SPRITE,Y
     DEY
     BPL spr_inia
     LDA #0
     STA VICBG0     'crni ekran i okvir
     STA VICFRC
     LDA #$93       'obriši ekran
     JSR $FFD2"
     }

     while(1){             'beskonačna petlja
          pos0+=dir0;      'promjeni pozicije sprajtova u pravom smjeru
          pos1+=dir1;
          pos2+=dir2;
          pos3+=dir3;
          if(pos0==208) dir0=$FF;  'provjeri da li je sprajt na dnu ekrana
          if(pos1==208) dir1=$FF;  'ako je, promijeni smjer na 'gore'
          if(pos2==208) dir2=$FF;
          if(pos3==208) dir3=$FF;
          if(pos0==50) dir0=$01;   'provjeri da li je sprajt na vrhu ekrana
          if(pos1==50) dir1=$01;   ' ako je, promijeni smjer na 'dolje'
          if(pos2==50) dir2=$01;
          if(pos3==50) dir3=$01;
          @spr0=pos0;         'postavi sprajtove na nove vertikalne pozicije
          @spr1=pos1;
          @spr2=pos2;
          @spr3=pos3;
          @spr4=pos0;
          @spr5=pos1;
          @spr6=pos2;
          @spr7=pos3;
     }
     assembly{      'pozicije sprajtova, MOB pokazivači, MOB slikovni podaci
"    RTS
spr_ini2: #data 45 78 77 73 109 68 141 63 205 78 237 73 13 68 45 63 $C0
spr_ini3: #data $80 $81 $82 $83 $84 $81 $85 $84
$2000:    #data $FF $FF $FF $FF $FF $FF $FF $FF $FF $C0 $3C $3 $80 $3C $1 $0
   $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0
   $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0
   $7E $0 $0 $FF $0 $0
#data $0 $7E $0 $3 $FF $C0 $7 $FF $E0 $1F $81 $F8 $1E $0 $78 $38 $0 $1C $70
   $0 $E $70 $0 $E $E0 $0 $7 $E0 $0 $7 $E0 $0 $7 $E0 $0 $7 $E0 $0 $7 $70 $0
   $E $70 $0 $E $38 $0 $1C $1E $0 $78 $1F $81 $F8 $7 $FF $E0 $3 $FF $C0 $0
   $7E $0 $0
```

```
#data $FC $0 $FF $7E $0 $7E $3F $0 $3C $3F $80 $3C $3F $C0 $3C $3F $E0 $3C
    $3F $F8 $3C $3F $FC $3C $3D $FE $3C $3C $FF $3C $3C $3F $BC $3C $1F $FC
    $3C $F $FC $3C $7 $FC $3C $3 $FC $3C $1 $FC $3C $0 $FC $3C $0 $3C $3C $0
    $1C $7E $0 $C $FF $0 $4 $0
#data $0 $FF $0 $0 $7E $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0
    $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0
    $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $3C $0 $0 $7E $0 $0 $FF $0 $0
#data $FF $0 $FF $7E $0 $7E $3C $0 $3C $3C $0 $F8 $3C $1 $E0 $3C $7 $C0 $3C
    $F $0 $3C $3E $0 $3C $78 $0 $3D $F0 $0 $3F $C0 $0 $3D $F0 $0 $3C $78 $0
    $3C $3E $0 $3C $F $0 $3C $7 $C0 $3C $1 $E0 $3C $0 $F8 $3C $0 $3C $7E $0
    $7E $FF $0 $FF $0
#data $FF $FF $C0 $7F $FF $F8 $3F $FF $FC $3C $0 $3E $3C $0 $F $3C $0 $7 $3C
    $0 $F $3C $0 $3E $3F $FF $FC $3F $FF $F8 $3F $FF $C0 $3D $F0 $0 $3C $78
    $0 $3C $3E $0 $3C $F $0 $3C $7 $C0 $3C $1 $E0 $3C $0 $F8 $3C $0 $3C $7E
    $0 $7E $FF $0 $FF $0"
      }
}
```

Dodatak B

Ispis B. "Sound Player.c64asm"

```
              SEI
              LDA #int_handler<  'preusmjeri rutinu za obradu prekida na
              STA IRQVCT         'promijenjenu, lociranu na int_handler:
              LDA #int_handler>
              STA IRQVCT+1
              LDA #$47           'inicijaliziraj SID ADSR generator
              STA SIDAD1
              LDA #$F8
              STA SIDSR1
              LDA #$0F           'inicijaliziraj SID valni oblik
              STA SIDVFT
              CLI
              RTS

int_handler:  JSR UDTIM          'odradi redovnu obradu prekida
              JSR STOP           'prekini sviranje ako je pritisnuta tipka
              BEQ stop           '"Stop"
nostop:       JSR SCNKEY
              LDA 203            'učitaj trenutno pritisnutu tipku
              LDX #$0C
nextkey:      CMP keylist,X      'provjeri da li ta tipka uzrokuje ton
              BEQ play_tune      'tipka pronađena, sviraj njezin ton
              DEX
              BPL nextkey
              LDA #$10           'tipka nije pronađena
              STA SIDV1R
              BNE end_irq
stop:         LDA #$31           'vrati obradu prekida na redovnu rutinu
              STA IRQVCT
              LDA #$EA
              STA IRQVCT+1
              LDA #0
              STA SIDVFT
end_irq:      LDA $DC0D          'okini vremenski prekidni brojač
              PLA                'vrati vrijednosti registara
              TAY
              PLA
              TAX
              PLA
              RTI

play_tune:    TXA                'pomnoži X sa 2
              ASL A
              TAX
              LDA tonetbl_c,X    'kopiraj frekvenciju tona na privremenu
              STA tmp            'lokaciju
              LDA tonetbl_c+1,X
              STA tmp+1
              LDA #7             'odredi broj dijeljenja frekvencije za
              SEC                'trenutnu oktavu
              SBC octave
              TAY
              CPX #24            'provjeri visoki C (tipka ',')
              BEQ high_c
octavelow:    DEY                'dijeli frekvenciju na privremenoj lokaciji
              BMI octave0        'sa 2 za svaku potrebnu oktavu
              LSR tmp
              ROR tmp+1
              JMP octavelow
octave0:      LDA tmp
              STA SIDV1H         'neka SID svira izračunatu frekvenciju
```

```
            LDA tmp+1
            STA SIDV1L
            LDA #$11
            STA SIDV1R
            JMP end_irq
high_c:     DEY                 'ako je sviran visoki C postavi najteži bit
            BMI octave0
            SEC
            ROR tmp
            ROR tmp+1
            JMP octavelow

octave:     #data 5
tmp:        #data 0 0
keylist:    #data 12 13 23 18 20 31 26 28 29 39 34 36 47
tonetbl_c:  #data 134 30          'frekvencija C7
tonetbl_cs: #data 142 24          'frekvencija C#7
tonetbl_d:  #data 150 139         'frekvencija D7
tonetbl_ds: #data 159 126         'frekvencija D#7
tonetbl_e:  #data 168 250         'frekvencija E7
tonetbl_f:  #data 179 6           'frekvencija F7
tonetbl_fs: #data 189 172         'frekvencija F#7
tonetbl_g:  #data 200 243         'frekvencija G7
tonetbl_gs: #data 212 230         'frekvencija G#7
tonetbl_a:  #data 225 143         'frekvencija A7
tonetbl_as: #data 238 248         'frekvencija A#7
tonetbl_h:  #data 253 46          'frekvencija H7
tonetbl_c1: #data 12 60           'frekvencija C8
```