Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Remec

Emulacija računalnika ZX Spectrum z vpogledom v stanje sistema

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: izr. prof. dr. Jurij Mihelič

Somentor:

Ljubljana, 2026

To delo je ponujeno pod licenco *Creative Commons Priznanje avtorstva-Deljenje pod enakimi pogoji 2.5 Slovenija* (ali novejšo različico). To pomeni, da se tako besedilo, slike, grafi in druge sestavine dela kot tudi rezultati diplomskega dela lahko prosto distribuirajo, reproducirajo, uporabljajo, priobčujejo javnosti in predelujejo, pod pogojem, da se jasno in vidno navede avtorja in naslov tega dela in da se v primeru spremembe, preoblikovanja ali uporabe tega dela v svojem delu, lahko distribuira predelava le pod licenco, ki je enaka tej. Podrobnosti licence so dostopne na spletni strani creativecommons.si ali na Inštitutu za intelektualno lastnino, Streliška 1, 1000 Ljubljana.

Izvorna koda diplomskega dela, njeni rezultati in v ta namen razvita programska oprema je ponujena pod licenco GNU General Public License, različica 3 (ali novejša). To pomeni, da se lahko prosto distribuira in/ali predeluje pod njenimi pogoji. Podrobnosti licence so dostopne na spletni strani `http://www.gnu.org/licenses/`.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Kandidat: Klemen Remec
Naslov: Emulacija računalnika ZX Spectrum z vpogledom v stanje sistema
Vrsta naloge: Diplomska naloga na univerzitetnem programu prve stopnje Računalništvo in informatika
Mentor: izr. prof. dr. Jurij Mihelič
Somentor:

Opis:
Računalnik ZX Spectrum 48K je podjetje Sinclair predstavilo leta 1982. Bil je eden prvih uporabnih in širše dostopnih računalnikov po svetu in tudi v Sloveniji. V diplomskem delu izdelajte program za njegovo emulacijo, ki naj poleg nalaganja in izvajanja programov omogoča tudi vpogled v notranje stanje emuliranega sistema in funkcionalnosti razhroščevanja. Predstavite računalnik in njegove komponente. Opišite zasnovo in izvedbo emulatorja ter primerno ovrednotite njegovo uporabnost in pravilnost delovanja.

Title: Emulation of the ZX Spectrum computer with system state introspection

Description:
The ZX Spectrum 48K computer was introduced by Sinclair in 1982. It was one of the first useful and widely available computers in the world and also in Slovenia. In your diploma thesis, create a program for its emulation, which, in addition to loading and executing programs, should also provide inspection of the internal state of the emulated system and debugging functionalities. Introduce the computer and its components. Describe the design and implementation of the emulator, and appropriately evaluate its usability and correct operation.

Zahvaljujem se mentorju izr. prof. dr. Juriju Miheliču za strokovno vodstvo pri izdelavi diplomske naloge. Zahvala gre tudi družini, dekletu in prijateljem za podporo med študijem.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 ZX Spectrum **3**

2.1 Zgodovinski pregled 3

2.2 Pregled strojne opreme 6

3 Pregled področja **11**

3.1 Razvoj emulatorjev 11

3.2 Obstoječi emulatorji za ZX Spectrum 12

4 Zasnova emulatorja **15**

4.1 Uporabljene tehnologije 15

4.2 Splošna arhitektura 16

4.3 Izvajalna zanka 16

4.4 Tok podatkov med emulacijo in vmesnikom 17

5 Pomnilnik **19**

5.1 Območja pomnilnika 19

5.2 ROM 20

5.3 Sistemske spremenljivke 21

5.4 Implementacija v emulatorju 22

6 Centralna procesna enota Z80 **27**
6.1 Registri in zastavice 28
6.2 Ukazi 33
6.3 Prekinitve 38
6.4 Implementacija v emulatorju 41

7 Vezje ULA **47**
7.1 Zaslon 48
7.2 Tipkovnica 55
7.3 Kasete 62
7.4 Zvok 65

8 Evalvacija emulatorja **67**
8.1 Obseg izvorne kode 67
8.2 Pravilnost emulacije 69
8.3 Notranji vpogled in razhroščevanje 71
8.4 Časovno delovanje in omejitve 73

9 Zaključek **75**

Literatura **79**

Povzetek

Naslov: Emulacija računalnika ZX Spectrum z vpogledom v stanje sistema

Avtor: Klemen Remec

Diplomsko delo obravnava emulacijo računalnika Sinclair ZX Spectrum 48K z dodanim vpogledom v notranje stanje sistema. Namen dela je bil razviti emulator, ki ne omogoča le zagona programov za ZX Spectrum, temveč tudi pregled in spreminjanje vrednosti registrov, zastavic, pomnilnika ter razstavljenih ukazov. V delu so predstavljeni zgodovinski pomen računalnika ZX Spectrum, njegova strojna zgradba, pomnilniška organizacija, procesor Z80, vezje ULA, zaslon, tipkovnica, kasetni podsistem in upravljanje zvoka. Emulator je implementiran kot namizna aplikacija, ki podpira emulacijo sistema in razhroščevalni uporabniški vmesnik. Delovanje je ovrednoteno z zagonom izbranih programov in iger ter s testnimi programi delovanja procesorja Z80. Rezultat dela je delujoč emulator, uporaben za poganjanje stare programske opreme, razumevanje delovanja sistema in digitalno ohranjanje računalniške dediščine.

Ključne besede: emulacija, Sinclair ZX Spectrum, digitalno ohranjanje.

Abstract

Title: Emulation of the ZX Spectrum computer with system state introspection

Author: Klemen Remec

This thesis addresses the emulation of the Sinclair ZX Spectrum 48K computer with insight into the internal state of the system. The goal was to develop an emulator that not only runs programs for the ZX Spectrum, but also allows inspection and modification of register values, flags, memory and disassembled instructions. The thesis presents the historical importance of the ZX Spectrum, its hardware architecture, memory organization, Z80 processor, ULA chip, display, keyboard, tape subsystem and sound control. The emulator is implemented as a desktop application that supports system emulation and a debugging user interface. Its operation is evaluated by running selected programs and games, and with test programs for the Z80 processor. The result is a working emulator, useful for running legacy software, understanding the operation of the system and digitally preserving computing heritage.

Keywords: emulation, Sinclair ZX Spectrum, digital preservation.

Seznam uporabljenih kratic

kratica	angleško	slovensko
ASCII	American Standard Code for Information Interchange	ameriški standardni nabor za izmenjavo informacij
BASIC	Beginners' All-purpose Symbolic Instruction Code	programski jezik BASIC
CPE	central processing unit	centralna procesna enota
CRC	cyclic redundancy check	ciklično preverjanje redundance
DC	direct current	enosmerni tok
EAR	earphone	vhod za signal iz kasetofona
IFF	interrupt flip-flop	prekinitveni preklopnik
KMP	Kotlin Multiplatform	večplatformska tehnologija Kotlin Multiplatform
LIFO	last in, first out	oblika delovanja sklada po principu zadnji noter, prvi ven
MIC	microphone	izhod za signal proti kasetofonu
PC	program counter	programski števec
PROM	programmable read-only memory	programirljiv bralni pomnilnik
PSP	interrupt handler	prekinitveno-servisni program
RAM	random access memory	bralno-pisalni pomnilnik
ROM	read-only memory	bralni pomnilnik
SP	stack pointer	kazalec sklada
ULA	uncommitted logic array	logično vezje ULA
V/I	input/output	vhod/izhod

Poglavje 1

Uvod

ZX Spectrum, viden na sliki 1.1, je 8-bitni domači računalnik podjetja Sinclair Research, ki je v osemdesetih letih pomembno zaznamoval evropski trg mikroračunalnikov in postal dostopen širokemu krogu domačih uporabnikov.

Slika 1.1: Računalnik ZX Spectrum 48K [26].

Namen diplomskega dela je razviti delujoč emulator in razhroščevalnik sistema ZX Spectrum 48K. Emulator mora podpirati poganjanje programov, napisanih za Spectrum, in razhroščevanje z vpogledom v notranje stanje sistema. Zaradi poudarka na razhroščevanju mora pri tem omogočati pregled in spreminjanje vrednosti registrov ter pomnilnika, pregled in razlago pomena naloženih ukazov in delov pomnilnika. Tak vpogled ni pomemben le za boljše razumevanje sistema, temveč tudi za ohranjanje znanja o njegovem delovanju. Delovanje emulatorja se ovrednoti z zagonom nekaj izbranih programov in iger, s čimer potrdimo pravilno delovanje emulacije.

V nadaljnjih poglavjih je najprej predstavljen računalnik ZX Spectrum, njegov vpliv na slovensko in svetovno okolje ter pomen ohranjanja takega znanja. Sledi pregled sorodnih del in primerjava razvitega emulatorja z obstoječimi emulatorji, nato pa še poglavje o zasnovi in splošni zgradbi emulatorja. Zatem so obravnavani posamezni deli sistema, to so pomnilnik, procesor Z80 in ULA, ki obravnava časovnik, zaslon, tipkovnico, kasete in zvok. Na koncu je emulator ovrednoten z vidika obsega izvorne kode, pravilnosti emulacije, notranjega vpogleda in časovnih omejitev.

Poglavje 2

ZX Spectrum

2.1 Zgodovinski pregled

Sinclair Research je pred ZX Spectrumom že uveljavil nizkocenovne domače računalnike, kot sta ZX80 in ZX81, ki sta poleg računalnika ZX Spectruma prikazana na sliki 2.1. Bila sta bila namenjena predvsem učenju programiranja in prvi uporabi računalnika doma. ZX Spectrum je bil predstavljen aprila 1982 kot zmogljivejši naslednik ZX81 in je na trg prinesel barvno grafiko, po kateri je tudi dobil ime, višjo ločljivost zaslonske slike, zvok in bolj razvito podporo okolju BASIC [4].

Slika 2.1: Računalniki Sinclair ZX80, ZX81 in ZX Spectrum [5].

Prvotni Spectrum je bil na voljo v različicah s 16 KB in 48 KB RAMa, obe pa sta uporabljali procesor Zilog Z80 in 16 KB ROMa. Računalnik je bilo mogoče priključiti na navaden televizor, programe pa je praviloma nalagal s kasete, kar je pomembno vplivalo na način uporabe in razširjanja programske opreme. Zaradi nizke cene in velike programske knjižnice je Spectrum postal eden najprepoznavnejših evropskih domačih računalnikov osemdesetih let [4, 24].

Družina ZX Spectrum se je pozneje širila z različnimi modeli. Spectrum+ je obdržal tehnično osnovo modela 48K, vendar jo je prestavil v večje ohišje z izboljšano tipkovnico, poznejši Spectrum 128 pa je razširil pomnilnik na 128 KB, dogradil implementacijo okolja BASIC, izboljšal zvočni čip in dodal možnosti za priklop zunanjih naprav. Ker je model 48K predstavljal osrednjo in najbolj razširjeno različico Spectruma ter osnovo za velik del Spectrumove programske opreme, se ravno zato nadaljnja obravnava in implementacija emulatorja osredotočata na ta model [4, 15].

2.1.1 Vpliv na slovenski prostor

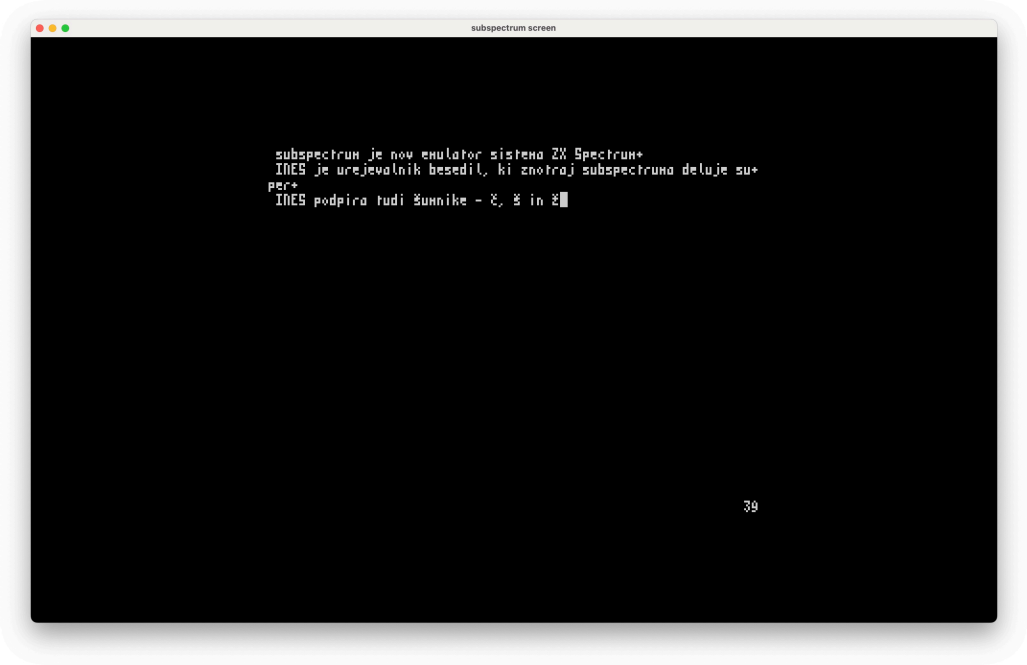
V slovenskem prostoru je bil ZX Spectrum pomemben predvsem kot cenovno dostopen domači računalnik. Prvi Spectrumi so se pri nas pojavili sredi leta 1982, kmalu zatem pa naj bi jih bilo samo v Ljubljani že nekaj tisoč. Njegova priljubljenost je bila povezana z nizko ceno in velikim številom iger, vendar se uporaba ni ustavila le pri zabavi [12].

Takšno okolje je pomembno za razumevanje pomena programa INES Primoža Jakopina. Program ni bil le uporabniški pripomoček, temveč primer doma razvite programske opreme, ki je iz Spectruma, takrat večinoma uporabljanega za igranje iger, naredila delovno orodje. INES je bil januarja 1985 objavljen kot kasetni program, ki je omogočal obdelavo besedil spremenljive dolžine (primer delovanja je viden na sliki 2.2), slik ter manjših podatkovnih zbirk. Posebej pomembna za domačo rabo je bila podpora šumnikom in jugoslovanskim črkam [8, 12, 20].

Tehnično je z INES pokazal, kje so bile postavljene meje učinkovitosti BASICa za uporabo v zahtevnejših programih – iskanje po besedilu je bilo v BASICu bistveno počasnejše glede na implementacijo v strojnem jeziku. Ta premik od tolmačenega BASICa proti programiranju v strojnem jeziku se je poznal skozi vso razvojno pot programov od TESS, BESS do INES, ter kasneje programov STEVE za Atari ST in EVA za modernejše osebne računalnike [9, 10, 11, 12].

2.1.2 Pomen ohranjanja računalniških sistemov in programske opreme prek emulacije

Ohranjanje celotne programske dediščine ZX Spectruma in drugih sistemov ni vezano le na shranjeni kasetni zapis programov, temveč tudi na konkretno izvajalno okolje: procesor, pomnilnik, ROM, način nalaganja, tipkovnico, prikaz na zaslonu in druge lastnosti sistema. Digitalno ohranjanje zato zajema tudi računalniške sisteme in programsko opremo kot digitalne artefakte, katerih dostopnost je odvisna od hitro spreminjajočih se nosilcev podatkov,



Slika 2.2: Prikaz delovanja urejanja besedila v programu INES znotraj razvitega emulatorja.

formatov, programov in strojne opreme [17].

Računalniška emulacija pomeni programsko posnemanje drugega računalniškega sistema. Za ohranjanje programov je pomembno posnemanje delovanja notranjih komponent sistema in vhodno-izhodnih naprav, možnost uporabe sistema preko nalaganja programov, vnosa podatkov in prikaza rezultatov. Emulacija tako ohranja digitalni artefakt skupaj z zadostnim delom njegovega izvajalnega in uporabniškega okolja [17].

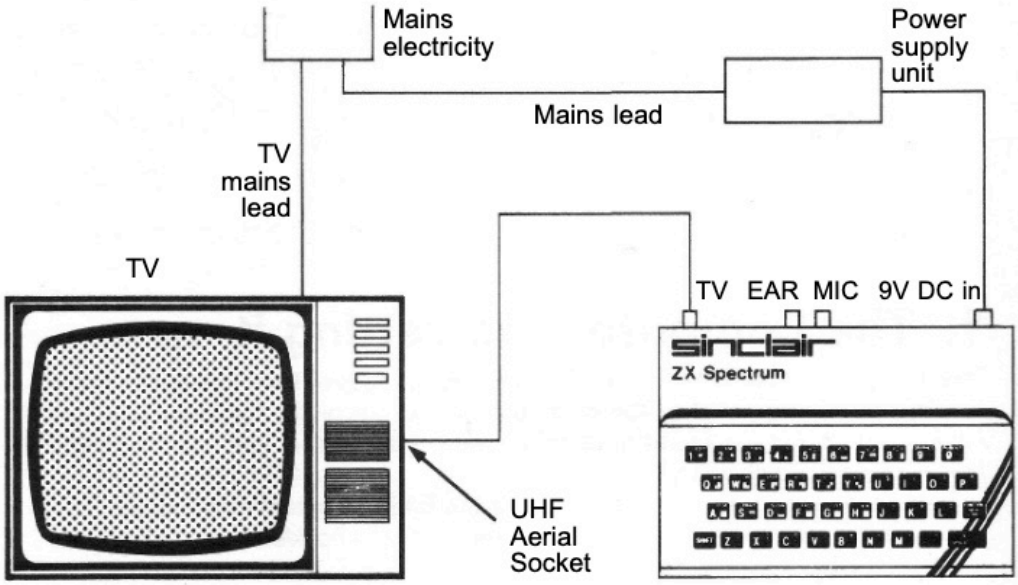
2.2 Pregled strojne opreme

2.2.1 Računalnik ZX Spectrum

Računalnik ZX Spectrum ima na zadnji strani več vtičnic in priključkov: vtičnico 9V DC IN, ki se prek napajalne enote priključi v električno omrežje,

3,5 milimetrsko vtičnico EAR se lahko poveže z izhodno vtičnico za slušalke ali zvočnik na kasetofonu, 3,5 milimetrsko vtičnico MIC se poveže z vhodno vtičnico za mikrofon na kasetofonu, vtičnico za televizijo in robni priključek za povezavo z dodatno opremo. Povezave med osnovnimi deli sistema so vidne na sliki 2.3 [24].

Ko računalnik ugasnemo, se vse v njem shranjene informacije izgubijo, če jih pred tem ne prenesemo na zunanji medij, kot je na primer kaseta. Če se kadar koli zapletemo pri uporabi računalnika, ga lahko le izključimo iz električnega omrežja in vklopimo nazaj ter ga s tem ponastavimo brez skrbi za poškodbo računalnika [24].



Slika 2.3: Skica povezav komponent sistema [24].

CPE Z80 predstavlja možgane računalnika, ki nadzira vse operacije, izvaja aritmetične račune, sprejema odločitve in na splošno upravlja, kaj naj računalnik naredi. Razporeditev glavnih notranjih komponent računalnika je vidna na sliki 2.4 [24].

ROM vsebuje 16384 zlogov (16K) navodil v strojnem kodu Z80, ki sestavljajo osnovni program računalnika, in procesorju narekuje, kaj naj stori v

vseh predvidljivih okoliščinah. Ta program je edinstven za ZX Spectrum [24].

RAM obsega osem čipov za shranjevanje podatkov, ki jih procesor želi obdržati, kot so programi v BASICu, spremenljivke, slika za televizijski zaslon in različni elementi, ki spremljajo stanje računalnika [24].

Vezje ULA deluje kot komunikacijski center sistema, ki poskrbi, da se vse, kar zahteva procesor, tudi dejansko izvede, hkrati pa bere pomnilnik za sestavo televizijske slike in pošilja ustrezne signale televizijskemu vmesniku [24].

PAL kodirnik analognega televizijskega barvnega sistema je skupina komponent, ki pretvarja televizijski izhod iz logičnega čipa v obliko, primerno za barvne televizije [24].

Regulator napetosti pretvarja rahlo nestabilno napetost napajanja v konstantnih pet voltov za zanesljivo delovanje sistema [24].

UHF/VHF modulator prenaša sliko na televizijski zaslon [24].

Zvočnik predstavlja zvočni izhod za zvok [24].

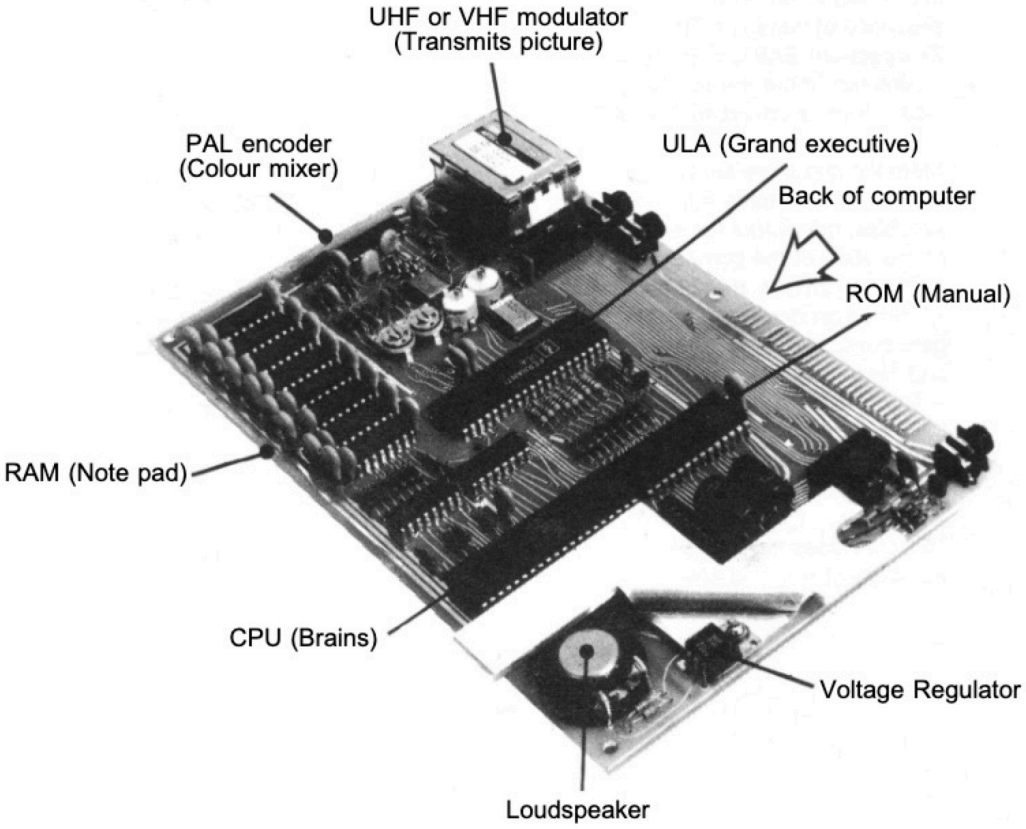
2.2.2 Dodatna oprema

Obstaja še druga oprema, ki se lahko uporabi s Spectrumom [23].

ZX Microdrive je naprava za hitro množično shranjevanje in je veliko bolj prilagodljiva pri načinu uporabe kot kasetni snemalnik [23].

Omrežni priključek se uporablja za povezovanje več računalnikov ZX Spectrum, da lahko med seboj komunicirajo [23].

Vmesnik RS232 omogoča standardno povezavo računalnika s tipkovnicami, tiskalniki, računalniki in drugimi zunanjimi napravami, tudi če ti niso bili zasnovani posebej za Spectrum [23].



Slika 2.4: Skica notranjih komponent računalnika ZX Spectrum [24].

Poglavje 3

Pregled področja

3.1 Razvoj emulatorjev

Pri razvoju emulatorjev je treba izbrati, kako bo gostiteljski sistem izvajal programsko kodo emuliranega sistema. Pri tolmačenju ukazov emulator ukaze gosta bere, dekodira in izvaja zaporedno, običajno tako, da po vsakem ukazu posodobi stanje procesorja, pomnilnika in povezanih naprav. Implementacijo takega pristopa je praviloma lažje povezati z razhroščevalnikom, saj ima program po vsakem izvedenem ukazu jasen vpogled v registre, pomnilnik in druge dele stanja [2, 22].

Druga možnost je dinamično prevajanje, kjer se deli kode gosta med izvajanjem prevedejo v kodo gostiteljskega sistema. Tak pristop je lahko hitrejši, vendar je zahtevnejši za implementacijo, oteži natančno ustavljanje po posameznih ukazih in je smiseln predvsem pri sistemih, kjer bi bilo tolmačenje ukazov prepočasno. Podobni kompromisi med hitrostjo, prenosljivostjo in zahtevnostjo implementacije so značilni tudi za splošnejše virtualne stroje in dinamične prevajalnike [2, 22].

Za emulator ZX Spectruma 48K je primeren pristop tolmačenja ukazov. Izvorni procesor Z80 deluje pri frekvenci približno 3,5 MHz, zato je mogoče ukaze na sodobnem računalniku izvajati dovolj hitro tudi brez dinamičnega prevajanja. Hkrati je cilj tega diplomskega dela tudi vpogled v notranje stanje

sistema, zato je izvajanje po korakih pomembnejše od največje možne hitrosti. Tak način omogoča preprostejšo povezavo procesorja, pomnilnika, V/I vrat, časovnika in uporabniškega vmesnika razhroščevalnika.

3.2 Obstoječi emulatorji za ZX Spectrum

Obstaja veliko število emulatorjev računalnika ZX Spectrum, ki se razlikujejo po namenu. Nekateri so namenjeni predvsem udobnemu poganjanju iger in programov, drugi čim natančnejši emulaciji različnih modelov in zunanjih naprav, tretji pa vgradnji emulacije v spletne strani ali razvojnemu delu. Tako je, poleg zmožnosti zagona poljubnega programa, pomembno tudi razlikovanje v podpori različnih modelov računalnika ZX Spectrum, podpora različnim datotečnim formatom, natančnost emulacije ter možnosti vpogleda v notranje stanje sistema.

Med najbolj uveljavljenimi emulatorji sta Fuse in Spectaculator. Fuse je odprtokodni namizni emulator, namenjen splošni uporabi in široki združljivosti, saj podpira vse glavne modele računalnika ZX Spectrum. Spectaculator je komercialni zaprtokodni emulator za platformo Windows in mobilne naprave, pri katerem je večji poudarek na uporabniški izkušnji: udobnem nalaganju programov, virtualnem kasetofonu, shranjevanju stanja in pripravljenosti za igranje. Posebno skupino predstavljajo spletni emulatorji, na primer JSSpeccy 3. Ta je odprtokoden emulator, razvit v okolju WebAssembly, ki se izvaja neposredno v brskalniku, zato je primeren za enostavnejše predstavitve programov brez namestitve samostojne aplikacije [6, 18, 25].

Ti obstoječi zreli emulatorji so večinoma namenjeni široki združljivosti, udobni uporabi ter podpori številnim modelom, formatom in dodatnim napravam. Odprtokodni emulator, razvit v okviru tega diplomskega dela, po obsegu ni namenjen neposredni zamenjavi teh rešitev. Omejen je na najpopularnejši model ZX Spectrum 48K in na pravilno delovanje delov sistema, ki so potrebni za poganjanje tipičnih programov: procesor Z80, pomnilnik, ULA, zaslon, tipkovnico, zvok ter nalaganje kasetnih datotek. Njegova po-

sebnost je pregleden razhroščevalni pregled notranjega stanja sistema, ki ponuja vpogled v vrednosti registrov, pomnilnika in v pomnilnik naloženih razstavljenih ukazov, lažji pregled namembnosti posameznih delov pomnilnika, ROMa in posameznih ukazov znotraj ROMa, izvajanje naloženega programa po korakih, prekinitvene točke in druge pripomočke za lažje razumevanje delovanja originalnega sistema.

Poglavje 4

Zasnova emulatorja

Emulator je razvit kot aplikacija, ki poleg izvajanja programov omogoča tudi vpogled in posege v notranje stanje emuliranega računalnika. Posamezni deli sistema zato niso skriti za emulacijsko zanko, ampak so implementirani kot ločeni modeli komponent delujočega sistema: registri, pomnilnik, V/I vrata, časovnik ter ostale naprave in funkcionalnosti računalnika. Procesor med izvajanjem ukazov spreminja stanje teh modelov, uporabniški vmesnik pa jih opazuje in prikaže v razhroščevalniku.

4.1 Uporabljene tehnologije

Projekt uporablja ogrodje Kotlin Multiplatform, ki se uporablja za ustvarjanje aplikacij s skupno kodo za različna izvajalna okolja, na primer namizne, spletne in mobilne aplikacije. V emulatorju se ta zmožnost uporablja predvsem tako, da sta skupna emulacijska logika in večina uporabniškega vmesnika definirana v skupnem modulu, platformno odvisni deli, kot sta izbira datotek in zvočni izhod, pa v ločenih modulih.

Uporabniški vmesnik je narejen s knjižnico Compose Multiplatform. Z njo so implementirana podokna za registre, pomnilnik in razstavljene ukaze, orodna vrstica in ločeno okno zaslona. Daljši tek procesorja se izvaja asinhrono v ozadju, uporabniški vmesnik pa se posodobi le ob spremembah stanja.

Gradnjo, izvajanje testov in pripravo namiznih paketov se izvaja preko orodja Gradle.

4.2 Splošna arhitektura

Osrednji povezovalni del je objekt `SpectrumMachine`. Ob ponastavitvi ustavi procesor, ponastavi registre, pomnilnik, V/I vrata, časovnik, tipkovnico in kasetni podsistem, ob nalaganju ROMa pa vsebino izbranega ROMa zapiše na začetek pomnilniškega naslovnega prostora. Objekt `Processor` izvaja procesorski cikel in pri tem uporablja stanje, ki ga hranijo objekti notranjih registrov `Registers`, pomnilniškega prostora `Memory` in V/I vrat `IO`.

Časovni del emulacije vodi `ULATiming`, ki beleži urine periode in procesorju posreduje periodične prekinitve. Deli sistema, ki so pri pravem računalniku povezani z vezjem ULA in perifernimi učinki, so razdeljeni v ločene objekte za lažje razumevanje sestavnih delov sistema. `ULAKeyboard` skrbi za matriko tipkovnice in prenos podatkov o pritiskih tipk, `ULAScreen` skrbi za zaslonski del pomnilnika, `ULATapeDeck` skrbi za nalaganje kasetnih datotek in `ULABeeper` skrbi za zvočni izhod. Uporabniški vmesnik za namen prikaza stanja sistema in funkcionalnosti razhroščevalnika uporablja isto stanje, ki ga uporablja emulator.

4.3 Izvajalna zanka

Ob zagonu emulator najprej ponastavi stanje sistema in v pomnilnik naloži izbrani ROM. Osrednji del izvajanja je metoda `Processor.step`. Ta najprej preveri, ali je programski števec na prekinitveni točki. Če je procesor v neprekinjenem teku, se izvajanje ustavi in uporabniški vmesnik lahko prikaže trenutno stanje sistema.

Nato procesor obravnava morebitno nemaskirno in običajno prekinitev. Pred dekodiranjem ukaza se preveri še možnost hitrega nalaganja kasete prek standardnih ROMovih rutin. Če kasetni podsistem prepozna klic ustrezne

rutine, lahko učinek nalaganja izvede neposredno, brez emulacije celotnega zvočnega signala kasete.

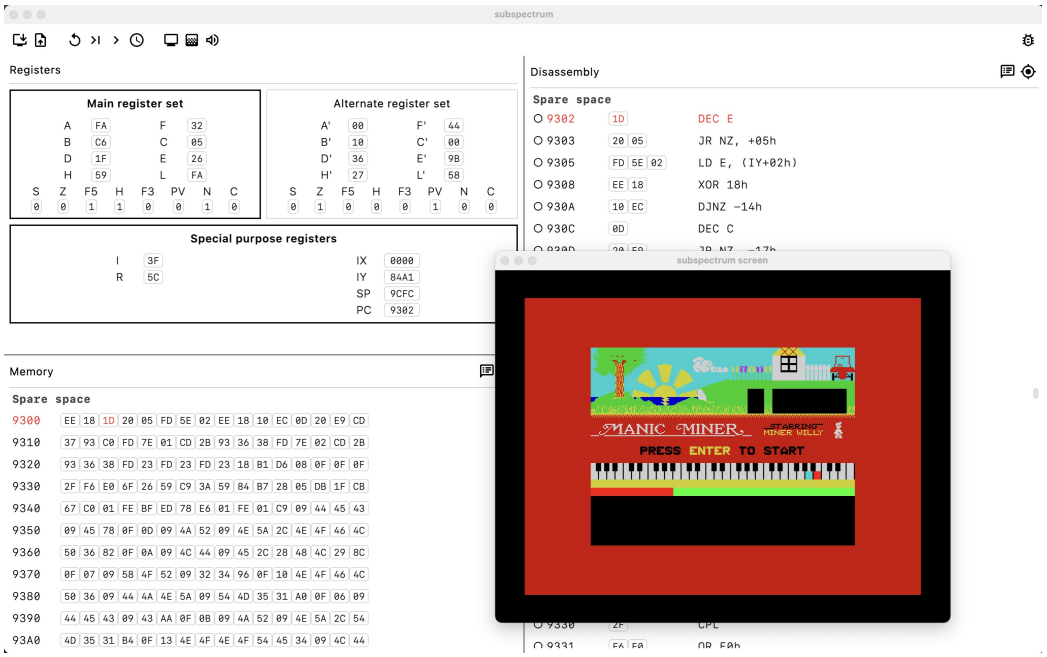
V običajnem primeru `Processor.step` prebere ukaz na naslovu programskega števca, ga dekodira, poveča register `R`, premakne programski števec za dolžino ukaza, izvede učinek ukaza in poveča števec izvedenih ukazov. Na koncu posodobi število urinih period glede na dolžino ukaza in preveri, ali je na koncu ukaza treba sprožiti prekinitev. S tem je emulacija organizirana okoli posameznega procesorskega ukaza, pri čemer se časovni model posodablja z vsoto urinih period izvedenih ukazov.

4.4 Tok podatkov med emulacijo in vmesnikom

Na sliki 4.1 vidimo primer vmesnika emulatorja. Podokno za registre prikazuje vrednosti splošnih in posebnih registrov ter zastavic, podokno za pomnilnik prikazuje vrednosti pomnilniških lokacij po naslovih, podokno z razstavljenimi ukazi pa prikazuje ukaze, dekodirane iz trenutne vsebine pomnilnika. Grafični izhod emuliranega računalnika je prikazan v ločenem oknu zaslona. Ko procesor ne teče, lahko uporabnik neposredno spremeni vrednosti registrov, pomnilniških celic in ukaznih zlogov.

Iz orodne vrstice lahko uporabnik ponovno naloži ROM ali kasetni program, ponastavi stanje sistema, izvede posamezen procesorski korak, ki izvede trenutni ukaz, zažene in ustavi tek procesorja, v katerem se ti koraki ponavljajo do zaustavitve ali prekinitvene točke, preklopi hitrost izvajanja emulacije, odpre okno zaslona ter nastavi način tipkovnice in omogoči zvok.

Ob spremembah stanja sistema, bodisi zaradi izvajanja ukazov bodisi neposrednega spreminjanja stanja sistema preko uporabniškega vmesnika razhroščevalnika, so te vidne v istem trenutku, kar pripomore k lažji uporabi emulatorja in njegovih funkcionalnosti.



Slika 4.1: Posnetek zaslona ob uporabi razvitega emulatorja.

Poglavje 5

Pomnilnik

5.1 Območja pomnilnika

Pomnilnik hrani podatke v obliki zlogov (vrednosti med 0 in 255), ki se nahajajo na dvozložnih pomnilniških naslovih med `0x0000` in `0xFFFF`. [23]

V računalniku ZX Spectrum je pomnilnik razdeljen na ROM, ki se nanaša na naslovni prostor med `0x0000` in `0x3FFF`, ter RAM, ki se nanaša na naslovni prostor med `0x4000` in `0xFFFF` [23].

Branje vrednosti iz pomnilnika deluje pri delu s podatki v ROMu in RAMu, pisanje vrednosti v pomnilnik pa deluje le v RAMu (ukaz pisanja v ROM se bo izvedel, a brez učinka) [14].

RAM je nadalje razdeljen na območja z različnimi nameni uporabe, pomembnejša območja pa so sledeča [14, 23]:

- zaslonska datoteka med naslovoma `0x4000` in `0x57FF` vsebuje podatke za prikaz vsebine na zaslon;
- atributi položajev znakov med naslovoma `0x5800` in `0x5AFF` hranijo informacije o prikazu zaslona;
- medpomnilnik tiskalnika med naslovoma `0x5B00` in `0x5BFF` hrani znake, ki jih mora tiskalnik natisniti, kadar pa tiskalnika ne uporabljamo, ostaja ta prostor neizkoriščen;

- sistemske spremenljivke med naslovoma `0x5C00` in `0x5CB5` vsebujejo razne informacije o stanju računalnika;
- sistem BASIC med naslovoma `0x5CCB` in `RAMTOP` vsebuje uporabniški program v BASICu, njegove spremenljivke in druge podatke ter sklad za aritmetični kalkulator;
- prosti pomnilnik med naslovoma `RAMTOP+1` in `0xFF57` je zavarovan pred vplivi programa v BASICu in pred posegi nadzornega programa;
- strojni sklad, ki se začenja 2 zloga pod vrhom prostega pomnilnika, uporablja procesor Z80 in zaseda spremenljivo območje pod naslovom `SP`;
- računski sklad med naslovoma `0x5C92` in `0x5CAF` predstavlja 30 zlogov velik prostor z začetkom na naslovu, shranjenem v sistemski spremenljivki MEMBOT, ki se uporablja za računske operacije v plavajoči vejici;
- definicije uporabniško definiranih grafičnih znakov med naslovoma `0xFF58` in `0xFFFF`.

Pri vnosu več podatkov v dano območje, kot je njegova privzeta velikost, se ostala območja premaknejo navzgor, obratno pa velja pri brisanju podatkov [23].

5.2 ROM

V ROMu shranjen nadzorni program poleg pravilnega zagona opravlja tudi mnoge druge naloge osnovnega delovanja sistema, kot so izrisovanje slike na zaslon, spremljanje pritiskov tipk, tolmačenje pognanega programa v BASICu in posredovanje napak o njegovem izvajanju, delo s plavajočo vejico itd [14, 24].

Procesor ne pozna sintakse in pravil izvajanja BASICa. Programe v BASICu zato berejo in izvajajo ROMovi podprogrami v strojnem jeziku,

strojni programi, tako nadzorni program ROMa kot uporabniški strojni programi, pa se izvajajo neposredno na procesorju [14].

5.3 Sistemske spremenljivke

Zlogi na pomnilniških naslovih med `0x5C00` in `0x5CB5` se uporabljajo za hrambo in spreminjanje informacij o stanju oziroma delovanju sistema [23].

Bolj pomembne sistemske spremenljivke so naslednje [14, 23]:

- VARS na naslovu `0x5C4B`, ki vsebuje naslov začetka spremenljivk programa BASIC;
- PROG na naslovu `0x5C53`, ki vsebuje naslov začetka programa v BASICu;
- E_LINE na naslovu `0x5C59`, ki vsebuje naslov konca programa v BASICu;
- RAMTOP na naslovu `0x5CB2`, ki vsebuje naslov zadnjega zloga sistema BASIC;
- P-RAMT na naslovu `0x5CB4`, ki vsebuje naslov zadnjega zloga fizičnega pomnilnika;
- BORDCR na naslovu `0x5C48`, kjer prvi trije biti definirajo barvo obrobe zaslona;
- FRAMES na naslovih `0x5C78-0x5C7A`, ki vsebuje 3-zložni števec prikazanih slikovnih okvirov, ki se pri evropskem video standardu poveča vsakih 20 ms oziroma vsako 1/50 sekunde. Števec je bolj natančen kot naivna programska implementacija, vendar se začasno ustavi med spuščanjem zvoka oziroma operacijami s kasetofonom, tiskalnikom in drugimi zunanjimi, zaradi česar izgublja na natančnosti;
- ATTR_P na naslovu `0x5C8D`, ki vsebuje definicije uporabljanih barv papirja in papirja ter drugih atributov stalnega prikaza znakov;

- CHARS na naslovu `0x5C36`, ki vsebuje naslov začetka definicije nabora znakov.

5.4 Implementacija v emulatorju

Ker model ZX Spectrum 48K ne uporablja preklapljanja pomnilniških bank, je pomnilnik v emulatorju predstavljen kot en sam 64 KB velik naslovni prostor. Pomnilniški naslov je implementiran kot 16-bitna vrednost `UShort`, vsebina pomnilnika pa kot polje zlogov. Polje je ovito v pomožni razred `DataByteArray`, ki poleg dostopa do posameznih zlogov omogoča tudi kopiranje obsegov in oblikovan heksadecimalni izpis.

Osrednji razred za delo s pomnilnikom je `MemorySet`. Ta vsebuje metode za branje in pisanje ene celice, branje in pisanje zaporednega bloka celic ter branje in nastavljanje posameznega bita. Nad njim je definiran objekt `Memory`, ki vsebuje skupno instanco razreda `MemorySet`. Zato procesor, ULA, nalaganje kaset in uporabniški vmesnik ne hranijo ločenih kopij pomnilnika, ampak vsi dostopajo do istega stanja emuliranega računalnika. Procesor iz njega bere ukazne zloge, operande in podatke, ULA pa iz območij `0x4000-0x57FF` in `0x5800-0x5AFF` bere podatke zaslonske datoteke in atributov.

Meja med ROMom in RAMom je v emulatorju določena s konstanto `ROM_END_ADDRESS`, katere vrednost je `0x3FFF`. Metodi `setMemoryCell` in `setMemoryCells` zato pri običajnem izvajanju ne spremenita celic v območju ROMa. Pri zapisu enega zloga se tak zapis preprosto prezre, pri zapisu večjega bloka pa metoda zapiše samo del bloka, ki pade v območje RAMa. S tem se ohrani obnašanje pravega računalnika, kjer procesor lahko naslovi ROM, zapis vanj pa nima učinka.

Blokovni zapis preveri še dve napaki: ali je zapisani blok večji od celotnega pomnilnika in ali bi se zapis raztezal čez naslov `0xFFFF`. To je pomembno pri nalaganju ROMa, kasetnih blokov in testnih programov, saj se napačen naslov ali napačna dolžina podatkov odkrije že ob poskusu zapisa. Branje obsega deluje prek metode `getMemoryCells`, ki vrne kopijo zahtevanega dela

pomnilnika, zato klicatelj ne more mimo razreda `MemorySet` spreminjati notranjega polja.

Ob ponastavitvi emulator najprej ustavi procesor, ponastavi registre, V/I vrata, časovno stanje ULA in stanje kasetnega pogona, nato pa razred `MemorySet` celoten pomnilnik napolni z vrednostjo `0x00`. Metoda `loadRom` po ponastavitvi prebere datoteko ROMa za model 48K in jo zapiše od naslova `0x0000` naprej. Isti mehanizem blokovnega zapisa se uporablja tudi pri hitrem nalaganju vsebine izbrane datoteke posnetka kasete ob ukazu `LOAD`, blokovno branje pomnilnika pri ukazu `VERIFY` pa enak obseg iz pomnilnika prebere in ga primerja z vsebino kasetnega bloka.

Podokno razhroščevalnika za pregled pomnilnika prikazuje začetni naslov vrstice, vrednosti zlogov in trenutni položaj programskega števca. Podokno za razstavljene ukaze iz istega pomnilnika sproti dekodira ukaze procesorja Z80 in prikaže naslov, pripadajoče zloge, razstavljeno obliko ukaza, trenutni položaj programskega števca in prekinitvene točke.

Vsak zapis v pomnilnik sproži obvestilo o spremembi vsebine prek toka `SharedFlow`. Obe podokni ta obvestila zbirata in prikaz osvežita ob naslednjem izrisu. Ker se zaporedna obvestila združujejo, hitro izvajanje programa ne povzroči ločenega preračunavanja uporabniškega vmesnika ob vsaki spremembi vsebine poljubne pomnilniške celice.

Poleg običajnega heksadecimalnega prikaza (viden na sliki 5.1) razhroščevalnik podpira še anotiran pogled (viden na sliki 5.2). Stalna območja pomnilnika, kot so ROM, zaslonska datoteka, atributi, medpomnilnik tiskalnika in sistemske spremenljivke, so zapisana v registru območij, meje dinamičnih območij pa se izračunajo iz vrednosti določenih sistemskih spremenljivk ter skladovnega kazalca `SP`. Zato lahko vmesnik predstavitev teh območij prilagodi trenutnemu programu v BASICu, njegovim spremenljivkam, delovnemu prostoru, računskemu skladu, prostemu pomnilniku, strojnemu skladu in uporabniško definiranim grafičnim znakom.

Imena in opisi označenih naslovov temeljijo na anotirani razčlembi stalnih območij RAMa, podprogramov v ROMu in sistemskih spremenljivk. Iz teh

virov so povzeti začetni naslovi, meje območij in kratki opisi njihove vloge v sistemu [16, 21].

Ko je procesor zaustavljen, ima uporabnik v teh podoknih možnost neposredno urejati vsebino pomnilniških celic, kar velja tudi za območje pomnilnika z ROMom, na katerega običajno izvajanje uporabniškega programa ne vpliva.

Memory

15E0	15	D8	28	FA	CF	07	D9	E5	2A	51	5C	23	23	18	08	1E
15F0	30	83	D9	E5	2A	51	5C	5E	23	56	EB	CD	2C	16	E1	D9
1600	C9	87	C6	16	6F	26	5C	5E	23	56	7A	B3	20	02	CF	17
1610	1B	2A	4F	5C	19	22	51	5C	FD	CB	30	A6	23	23	23	23
1620	4E	21	2D	16	CD	DC	16	D0	16	00	5E	19	E9	4B	06	53
1630	12	50	1B	00	FD	CB	02	C6	FD	CB	01	AE	FD	CB	30	E6
1640	18	04	FD	CB	02	86	FD	CB	01	8E	C3	4D	0D	FD	CB	01
1650	CE	C9	01	01	00	E5	CD	05	1F	E1	CD	64	16	2A	65	5C
1660	EB	ED	B8	C9	F5	E5	21	4B	5C	3E	0E	5E	23	56	E3	A7
1670	ED	52	19	E3	30	09	D5	EB	09	EB	72	2B	73	23	D1	23
1680	3D	20	E8	EB	D1	F1	A7	ED	52	44	4D	03	19	EB	C9	00
1690	00	EB	11	8F	16	7E	E6	C0	20	F7	56	23	5E	C9	2A	63
16A0	5C	2B	CD	55	16	23	23	C1	ED	43	61	5C	C1	EB	23	C9
16B0	2A	59	5C	36	0D	22	5B	5C	23	36	80	23	22	61	5C	2A
16C0	61	5C	22	63	5C	2A	63	5C	22	65	5C	E5	21	92	5C	22
16D0	68	5C	E1	C9	ED	5B	59	5C	C3	E5	19	23	7E	A7	C8	B9
16E0	23	20	F8	37	C9	CD	1E	17	CD	01	17	01	00	00	11	E2
16F0	A3	EB	19	38	07	01	D4	15	09	4E	23	46	EB	71	23	70
1700	C9	E5	2A	4F	5C	09	23	23	23	4E	EB	21	16	17	CD	DC
1710	16	4E	06	00	09	E9	4B	05	53	03	50	01	E1	C9	CD	94
1720	1E	FE	10	38	02	CF	17	C6	03	07	21	10	5C	4F	06	00
1730	09	4E	23	46	2B	C9	EF	01	38	CD	1E	17	78	B1	28	16
1740	EB	2A	4F	5C	09	23	23	23	7E	EB	FE	4B	28	08	FE	53

Slika 5.1: Posnetek zaslona emulatorja z običajnim vpogledom v stanje pomnilnika.

Memory

ROM

THE 'SAVE, LOAD, VERIFY and MERGE' COMMAND ROUTINES

15D4 FD CB 02 6E 20 04 FD CB 02 DE CD E6

15E0 15 D8 28 FA CF 07

THE 'INPUT-AD' SUBROUTINE

15E6 D9 E5 2A 51 5C 23 23 18 08

THE 'MAIN PRINTING' SUBROUTINE

This subroutine is called with the A register holding a valid stream number normally +FD to +03. Then depending on the stream data a particular channel will be made the current channel.

56 EB CD 2C 16 E1 D9

THE 'CHAN-OPEN' SUBROUTINE

1601 87 C6 16 6F 26 5C 5E 23 56 7A B3 20 02 CF 17

1610 1B 2A 4F 5C 19

THE 'CHAN-FLAG' SUBROUTINE

1615 22 51 5C FD CB 30 A6 23 23 23 23

1620 4E 21 2D 16 CD DC 16 D0 16 00 5E 19 E9

THE 'CHANNEL CODE LOOK-UP' TABLE

162D 4B 06 53

1630 12 50 1B 00

THE 'CHANNEL 'K' FLAG' SUBROUTINE

1634 FD CB 02 C6 FD CB 01 AE FD CB 30 E6

1640 18 04

THE 'CHANNEL 'S' FLAG' SUBROUTINE

1642 FD CB 02 86 FD CB 01 8E C3 4D 0D

Slika 5.2: Posnetek zaslona emulatorja z anotiranim vpogledom v stanje pomnilnika z označenimi območji in anotiranimi deli območij pomnilnika.

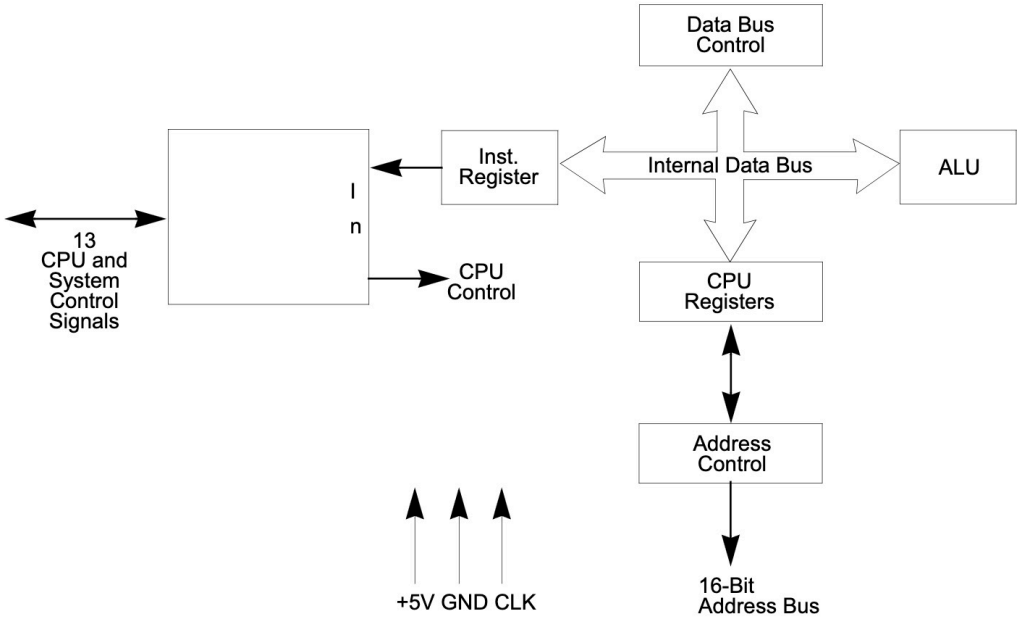
Poglavje 6

Centralna procesna enota Z80

Procesor Z80 najdemo v mnogih računalnikih – ZX Spectrum, ZX81, Galaksija, Gorenje Dialog 20, Amstrad CPC 454 in celo večjih računalnikih, kot je IskraDelta Partner [14].

Družina komponent procesorjev Z80 podjetja Zilog je četrta generacija mikroprocesorjev, ki dosega hitrosti od 6 do 20 MHz. Velika inspiracija arhitekturi procesorja Z80 je bil procesor Intel 8080, saj si je Zilog zadal cilj izdelave procesorja, ki bi poleg kompatibilnosti s tedaj popularnim 8080 prinesel večje zmogljivosti procesorja, cenejšo izdelavo preostalega sistema ter enostavnejše programiranje. Čeprav ima Z80 mnogo 8-bitnih in nekaj 16-bitnih registrov, velja za 8-bitni procesor in večino časa obdeluje 8-bitne podatke v 8-bitnih registrih in 8-bitnih pomnilniških celicah [14, 30, 31].

Glavni notranji elementi procesorja so vidni na sliki 6.1. Vsak sistem s procesorjem Z80 mora vsebovati določene zunanje strojne elemente, kot so vir napajanja +5V in oscilator, ki procesorju podajata vir napajanja in urin signal CLK. Procesor prek ukaznega registra, notranjih registrov, ALU in notranjega podatkovnega vodila izvaja ukaze, prek naslovnega in podatkovnega vodila pa dostopa do pomnilniških naprav (RAM, ROM ali PROM) ter vhodno-izhodnih vezij. Njihovi signali so časovno usklajeni s procesorjem za pravilno delovanje pomnilnika ali perifernih vezij [31].



Slika 6.1: Diagram notranje arhitekture in glavnih elementov procesorja Z80 [31].

6.1 Registri in zastavice

6.1.1 Podatkovni tipi procesorja Z80

Večina operacij Z80 je 8-bitnih, torej dela z vrednostmi med `0x00` in `0xFF`, kar lahko predstavlja bodisi nepredznačen obseg vrednosti od 0 do 255 bodisi predznačen obseg vrednosti od –128 do 127 [14].

Nekatere operacije pa so 16-bitne, torej CPE dela z vrednostmi med `0x0000` in `0xFFFF`, kar lahko predstavlja bodisi nepredznačen obseg vrednosti od 0 do 65535 bodisi predznačen obseg vrednosti od –32768 do 32767. 16-bitna števila največkrat potrebujemo za naslavljanje pomnilniških celic [14].

Posebnost so vrednosti števil v obliki plavajoče vejice $x = m \times 2^n$, ki zavzemajo kar 5 zlogov (prvi zlog definira eksponent in vsebuje $128 + n$, ostali 4 zlogi pa vsebujejo mantiso z vrednostjo $[\frac{1}{2}, 1)$). Ker 5-zložnih vrednosti ne moremo shraniti v posamezen register ali registrski par, za njihovo shranje-

vanje uporabljamo bodisi registre A, E, D, C in B bodisi računski sklad v pomnilniku. Vendar pa tak tip podatka ni specifičen za procesor Z80, temveč za podprograme ROMa računalnika ZX Spectrum [14].

6.1.2 Razporeditev registrov, njihove velikosti in pomen

Procesor Z80 vsebuje 208 bitov bralno-pisalnega pomnilnika, implementiranega kot statični RAM, ki predstavlja osemnajst 8-bitnih registrov in štiri 16-bitne registre [31].

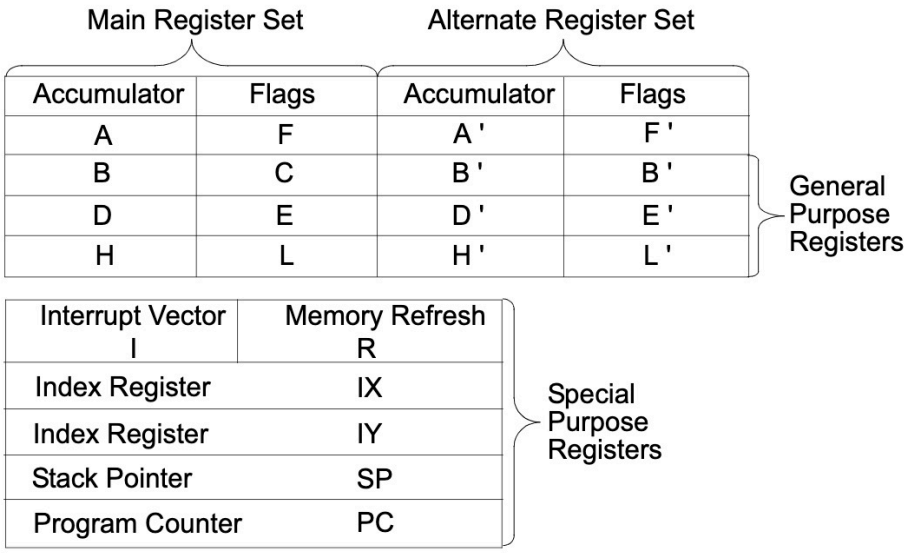
Ti notranji registri se delijo na dva registrska nabora, od katerih vsak vsebuje 6 splošnonamenskih registrov (registri B, C, D, E, H in L), akumulator (register A) in register zastavic (register F) ter 6 posebnonamenskih registrov, ki so 16-bitni programski števec (register PC), 16-bitni kazalec sklada (register SP), 16-bitna indeksna registra IX in IY, osvežitveni register R in register naslova strani prekinitev I. Razporeditev notranjih registrov je vidna na sliki 6.2 [14, 31].

Pri tem lahko splošnonamenske 8-bitne registre uporabljamo posamično ali pa pare 8-bitnih registrov združimo v 16-bitne registrske pare BC, DE in HL [14, 31].

Med uporabo osnovnega in zamenljivega registrskega nabora (katerega imena registrov označujemo z apostrofom ob imenih) se lahko preklaplja. Ta značilnost procesorja Z80 je posebej potrebna v sistemih, ki zahtevajo hiter odziv na prekinitve. Ob začetku obdelave prekinitev ali izvajanja podprograma je treba star izvajalni kontekst (vrednosti registrov) shraniti do obnovitve pred vrnitvijo iz podprograma. Zamenljiva registrska nabora tako omogočata mnogo hitrejšo menjavo konteksta napram hrambi in obnavljanju vrednosti vseh registrov v pomnilniku [14, 31].

Akumulator oz. register A velja za edini register, kjer procesor lahko opravlja 8-bitne aritmetično-logične operacije. Pri njihovem izvajanju register zastavic F v splošnem hrani posebne pogoje izvedenih operacij ali značilnosti dobljenega rezultata [14, 31].

Register naslova strani prekinitev (I) hrani višji zlog naslova PSP, ki se



Slika 6.2: Diagram notranjih registrov procesorja Z80 [31].

dopolni z nižjim zlogom naslova, zagotovljenega s strani zunanje naprave, ki je sprožila prekinitev. Procesor Z80 namreč deluje v načinu, v katerem je mogoče v odziv na prekinitev izvesti posredni klic na PSP na poljubnem mestu v pomnilniku [31].

Register osveževanja pomnilnika (R) hrani 7-bitni števec osveževanja pomnilnika, ki se samodejno poveča po vsakem pridobljenem ukazu. Ob prelivu vrednosti registra procesor pomnilniku pošlje signal za osvežitev vsebine pomnilnih celic. To omogoča enostavno uporabo dinamičnih in statičnih pomnilnikov brez upočasnjevanja delovanja CPE (osveževanje se dogaja med dekodiranjem prej pridobljenega ukaza) [31].

Za 16-bitne registre in registrske pare velja, da je višji zlog, shranjen v registru/registrskem paru, prvi, nižji pa drugi. Pri tem je treba opozoriti, da je tak vrstni red ravno obraten kot pri hranjenju 16-bitne vrednosti v pomnilniku, kjer prva pomnilniška celica vsebuje nižji del vrednosti, druga pa višjega [14, 31].

Indeksna registra IX in IY hranita 16-bitni naslov, ki se uporablja v

indeksnem načinu naslavljanja kot bazni kazalec, iz katerega po prištetju odmika (dodaten zlog, predstavljen kot število v dvojiškem komplementu), dobimo končni pomnilniški naslov operanda [31].

Programski števec PC hrani naslov trenutnega ukaza, ki se pridobiva iz pomnilnika in se samodejno poveča, ko se njegova vsebina prenese na naslovno vodilo. PC se uporablja tudi za klice in vračanje iz podprogramov, pri čemer se povratni naslov hrani na skladu v pomnilniku [14, 31].

Kazalec sklada SP hrani naslov vrhnje vrednosti na skladu. Sklad, organiziran v obliki LIFO, omogoča preprosto izvajanje večstopenjskih prekinitev, teoretično neomejeno gnezdenje podprogramov in poenostavitev mnogih vrst manipulacije podatkov [14, 31].

6.1.3 Zastavice

Zastavice, hranjene v registrih F in F', predstavljajo informacijo o stanju bodisi vsebine registra A bodisi o ravnokar opravljeni računski operaciji. Razporeditev bitov v registru zastavic je prikazana v tabeli 6.1. Kljub temu da ima register F možnost shranjevanja 8 bitov, za dokumentirane zastavice uporablja le 6 bitov (bita 3 in 5 se ne uporabljata) [14, 31]:

Bit v registru F	7	6	5	4	3	2	1	0
Zastavica	S	Z	YF	H	XF	P/V	N	C

Tabela 6.1: Razporeditev bitov v registru zastavic F procesorja Z80.

- zastavica znaka S (angl. Sign flag) je dvignjena, ko je rezultat operacije negativen;
- ničelna zastavica Z (angl. Zero flag) je dvignjena, ko je rezultat operacije enak nič;
- zastavica polprenosa H (angl. Half carry flag) je dvignjena, ko se je v zadnji operaciji spremenil 5. bit registra;

- zastavica parnosti/prepolnjenja P/V (angl. Parity/oVerflow flag) na bitu 2 je dvignjena, ko je v rezultatu operacije parno število bitov ali ko se je spremenil najvišji bit registra (kar pomeni možnost preliva);
- zastavica odštevanja N (angl. subtract flag) je dvignjena, ko je zadnja operacija bila odštevanje;
- zastavica prenosa C (angl. Carry flag) je dvignjena, ko se je v zadnji operaciji zgodil prenos.

Zastavice omogočajo vejanje programa prek pogojnih skokov z možnimi pogoji [14]:

- Z (angl. Zero) ob dvignjeni ničelni zastavici;
- NZ (angl. Not Zero) ob spuščeni ničelni zastavici;
- C (angl. Carry) ob dvignjeni zastavici prenosa;
- NC (angl. Not Carry) ob spuščeni zastavici prenosa;
- M (angl. Minus) ob dvignjeni zastavici znaka;
- P (angl. Positive) ob spuščeni zastavici znaka;
- PE (angl. Parity Even) ob dvignjeni zastavici parnosti;
- PO (angl. Parity Odd) ob spuščeni zastavici parnosti.

6.1.4 Nedokumentirane zastavice

Uradna dokumentacija procesorja Z80 bita 5 in 3 registrov zastavic F in F', ki ju bomo označevali kot YF in XF, pogosto obravnava kot neuporabljena, vendar ju dejanski procesor vseeno nastavlja in uporablja. Ti zastavici nista pogojni zastavici v enakem smislu kot ničelna zastavica Z ali zastavica prenosa C, saj Z80 nima pogojnih skokov, klicev ali vrnitev, ki bi neposredno preverjali YF ali XF. Zaradi tega se zdita bita za običajno programiranje

nepomembna, pri emulaciji pa sta pomembna za programe, ki uporabljajo stranske učinke ukazov. Zato ju emulator, ki želi natančno posnemati procesor, kljub pomanjkljivi uradni dokumentaciji ne sme obravnavati kot neuporabljena [30].

Pri običajnih aritmetično-logičnih ukazih z 8-bitnim rezultatom se YF in XF praviloma nastavita iz bitov 5 in 3 tega rezultata. Pri ukazih primerjave se rezultat ne shrani v akumulator, zato se YF in XF nastavita iz bitov 5 in 3 primerjanega operanda. Pri 16-bitni aritmetiki se YF in XF nastavita tako, kot da bi ju določila operacija nad višjima zlogoma operandov. Pri ukazih preverjanja stanja posameznega bita se YF nastavi, kadar ukaz preverja bit 5 in je ta bit postavljen, XF pa, kadar preverja bit 3 in je ta bit postavljen. A to ne velja pri indeksnem naslavljanju, kjer se YF in XF nastavita iz bitov 5 in 3 višjega zloga efektivnega naslova indeksiranega naslavljanja, ter pri ukazu `BIT n,(HL)`, kjer se YF in XF nastavita iz vrednosti nedokumentiranega notranjega registra, povezanega z nekaterimi predhodnimi 16-bitnimi aritmetičnimi ukazi ter uporabljenimi efektivnimi naslovi. Pri bločnih ukazih prenosov in primerjav se YF in XF nastavita iz vmesnih vrednosti prenosov ali izračunov primerjav. Pri bločnih vhodno-izhodnih ukazih se po prenosu zloga najprej zmanjša register B, nato pa se YF nastavi iz bita 5 nove vrednosti registra B, XF pa iz njenega bita 3 [30].

6.2 Ukazi

ZX Spectrum omogoča uporabo visokonivojskega jezika BASIC. Jezik je bil leta 1964 razvit na Dartmouth College v New Hampshiru in je bil v svojem času zelo razširjen na osebnih računalnikih. Ima več različic, med katerimi obstajajo manjše razlike, kot na primer zahteva po zapisu ukazne besede LET pri dodeljevanju vrednosti spremenljivkam, kar lahko opustimo v mnogih drugih različicah jezika BASIC. A različica BASICa za ZX Spectrum vseeno ni daleč od (neobstoječega standarda) jezika BASIC, zato ne bi smeli imeti preveč težav pri prilagajanju katerega koli BASIC programa za delovanje na

ZX Spectrumu [24].

Za programiranje na Spectrumu lahko uporabljamo tudi strojni jezik, ki v večini primerov omogoča hitrejše izvajanje, učinkovitejšo izrabo pomnilnika, krajše programe (za hrambo v pomnilniku omejene velikosti) ter neodvisnost od nadzornega programa, ki je potreben za izvajanje programov v BASICu. A prinaša tudi nekatere slabosti, kot so slabša berljivost programov, omejenost na izvajanje programa v sistemih z istim procesorjem ter nasploh daljši programi (za hrambo v pomnilniku) [14].

Berljivost strojnega programa lahko izboljšamo s pisanjem v zbirnem jeziku, ki predstavlja preslikavo strojnega jezika v ljudem razumljivejšo obliko, v kateri vsak zbirni ukaz ustreza enemu strojnemu ukazu (in obratno). Zbirni jezik je tako razumljivejši ljudem, a ga moramo pred izvajanjem prevesti v zaporedje zlogov strojnega koda, bodisi ročno bodisi z uporabo zbirnika. Najbolj razširjen zbirnik za uporabo v Spectrumu je bil program GENS [14, 23].

6.2.1 Število ukazov in sklopi, njihove dolžine

Procesor Z80 lahko izvaja 158 različnih dokumentiranih in mnogo nedokumentiranih ukazov, vključno z vsemi 78 ukazi procesorja 8080A [31].

Vsak ukaz je sestavljen iz zaporedja osnovnih strojnih M-stanj, ta pa so sinhrona s časovnimi T-stanji (angl. Transition state), ki predstavljajo čas ene urine periode kremenčevega kristala v CPE. Pri frekvenci 3,5 MHz procesorja Z80 eno stanje T traja približno 0,2857 mikrosekunde in predstavlja osnovno merilo hitrosti operacij [14].

Ukaze delimo na več skupin [31]:

- ukaze nalaganja in izmenjav, ki premikajo podatke med izvorno in ciljno lokacijo, bodisi med posameznimi registri bodisi registri in pomnilnikom;
- aritmetične in logične ukaze, ki jih procesor izvaja nad podatki v akumulatorju in drugih splošnonamenskih registrih ali pomnilniških lokacijah. Rezultat dane operacije se razen v nekaj izjemah shrani v akumulator,

glede na rezultat operacije pa se nastavijo tudi ustrezne zastavice v registru zastavic;

- ukaze rotacije in zamikov, ki omogočajo levo ali desno, aritmetično ali logično, rotacijo ali zamik vrednosti poljubnega registra ali pomnilniške lokacije z ali brez prenosa;
- ukaze manipulacije bitov, ki omogočajo nastavitev, ponastavitev ali preveritev vrednosti poljubnega bita vrednosti v akumulatorju, drugem splošnonamenskem registru ali pomnilniški lokaciji;
- ukaze skokov, klicev in vrnitev, ki se uporabljajo za skoke med več lokacijami v programu. Edinstvena vrsta klicev so ukazi skupine ponovnih zagonov, ki lahko kot nov naslov izvajanja določi le enega od 8 ločenih naslovov, ki se nahajajo na prvi strani pomnilnika;
- ukaze prenosov in iskanja blokov, ki se uporabljajo za prenos celih blokov podatkov iz enega mesta pomnilnika na drugo oziroma iskanje poljubnega 8-bitnega znaka v bloku podatkov v pomnilniku. Ob dosegu konca definiranega bloka ali najdenega znaka se ukaz samodejno prekine, prav tako se lahko izvajanje takega ukaza kadar koli prekine in kasneje nadaljuje, kar omogoča vmesno uporabo procesorja za druge operacije. Ukazi te skupine pridejo prav pri obdelavi velikih nizov podatkov;
- ukaze vhoda in izhoda, ki omogočajo prenose podatkov (posameznih ali blokov) med pomnilniškimi lokacijami ali splošnimi registri procesorja in zunanjimi V/I napravami;
- nadzorni oziroma kontrolni ukazi, ki omogočajo upravljanje različnih načinov delovanja procesorja, med drugim omogočanje in onemogočanje prekinitev, nastavljanje načina odziva na prekinitve itd.

Podskupina ukazov ponovnih zagonov RST (angl. ReSTart) opravlja isto nalogo kot brezpogojni klici, le da lahko z njimi kličemo le 8 naslovov v začetku ROMa [14]:

- RST `0x00` učinkuje kot izklop in ponoven vklop računalnika;
- RST `0x08` služi za prikaz sporočila o napaki BASICa, katere 8-bitni kod napake sledi ukazu kot operand, ter za uporabljanje mikrotračnih enot in omrežja Spectrumov preko vmesnika na zadnji strani računalnika;
- RST `0x10` na zaslon izpiše znak, katerega kod je v registru A;
- RST `0x18` register A napolni z vrednostjo zloga, katerega naslov je v sistemski spremenljivki CH-ADD;
- RST `0x20` se uporablja pri tolmačenju programov v BASICu;
- RST `0x28` se uporablja za računanje s plavajočo vejico na računskem skladu, zlogi operandi pa določajo kode operacij, ki se morajo izvesti;
- RST `0x30` preuredi delovni pomnilniški prostor tako, da dobimo v njem toliko prostih zlogov, kolikor je vrednost registrskega para BC;
- RST `0x38` preleti tipkovnico (kontrola pritisnjenih tipk) in poveča števec časa.

6.2.2 Tipi naslavljanj

Naslavljanje se nanaša na način definicije naslova podatkov v registrih, pomnilniku ali V/I vratih, ki se bodo uporabili pri izvedbi določenega ukaza [31]:

- registrsko naslavljanje (oblike `LD r, r`) določi enega ali več registrov, ki se uporabijo za dano operacijo [14, 31];
- sprotno ali takojšnje naslavljanje (oblike `LD r, n`) v zlogu, ki sledi št. strojnega ukaza, vsebuje 8-bitno vrednost operanda [14, 31];
- razširjeno takojšnje naslavljanje (oblike `LD rr, nn`) v zlogih, ki sledita št. strojnega ukaza, vsebuje 16-bitno vrednost operanda [31];

- zunanje ali razširjeno naslavljanje (oblike `LD A, (nn)`) v zlogih, ki sledita št. strojnega ukaza, vsebuje pomnilniški naslov operanda, kar omogoča prenašanje vrednosti iz registra v pomnilniško celico in obratno ter skok ali prenos podatkov med poljubnima pomnilniškima lokacijama [14, 31];

- registrsko posredno naslavljanje (oblik `LD r, (rr)` in `LD (rr), r`) določi registrski par, katerega vrednost predstavlja pomnilniški naslov operanda [14, 31];

- relativno naslavljanje (oblike `JR d`) v zlogu, ki sledi št. strojnega ukaza, vsebuje odmik v obliki dvojiškega komplementa, ki se prišteje vrednosti registra PC za določitev pomnilniške lokacije operanda. To poleg prenosljivosti programa omogoča tudi skoke na bližnje pomnilniške lokacije z le dvozložnim ukazom [31];

- indeksno naslavljanje (oblike `LD r, (IX+dis)` in `LD (IX+dis), r`) je le oblika registrskega posrednega naslavljanja z dodanim odmikom. V zlogu, ki sledi št. strojnega ukaza, vsebuje odmik v predstavitvi dvojiškega komplementa, ki se prišteje vrednosti izbranega indeksnega registra za določitev pomnilniške lokacije operanda. To poleg prenosljivosti programa omogoča tudi poenostavitev dela s tabelami podatkov [14, 31];

- bitno naslavljanje (oblike `BIT b, r`) s pomočjo registrskega, registrskega posrednega ali indeksnega naslavljanja lahko določi poljubni bit poljubne pomnilniške lokacije ali registra za izvedbo bitne operacije [31];

- spremenjeno naslavljanje ničelne strani (oblike `RST p`) se nanaša na posebne enozložne ukaze RST za skok na poljubno od 8 lokacij v strani 0 pomnilnika, kjer se nahajajo pogosto uporabljeni podprogrami ROMa [31];

- implicitno naslavljanje (oblike `RLA`) se uporablja pri mnogih ukazih, ki implicitno določijo registre z operandi, potrebnimi za dano operacijo

(npr. rezultat aritmetičnih operacij se implicitno shrani v akumulator) [31].

Ker mnoge operacije zahtevajo več kot en operand, se lahko hkrati uporabi tudi več tipov naslavljanj [31].

6.2.3 Nedokumentirani ukazi

Poleg uradno navedenega nabora ukazov ima Z80 tudi množico operacijskih kod, ki so v uradni dokumentaciji izpuščene ali označene kot neuporabljene. Na pravem procesorju lahko te operacijske kode kljub temu izvedejo nove ali podvojene že dokumentirane ukaze. To se zgodi zato, ker procesor tudi pri nedokumentiranih kombinacijah bitov uporabi isto dekodirno logiko kot pri uradnem naboru ukazov. Predpona določi skupino ukazov, naslednji zlogi pa določijo operacijo, register ali način naslavljanja [30].

Pri predponi `0xCB` nedokumentirane kode od `0xCB30` do `0xCB37` med drugim definirajo ukaze logičnih pomikov v levo `SLL`. Pri predponi `0xDD` se uporabe registra HL v ukazih praviloma nadomestijo z IX, pri predponi `0xFD` pa z IY, vključno z dostopi do posameznih polovic oziroma zlogov indeksnih registrov. Pri predponi `0xED` nekatere kode predstavljajo dvojnike uradnih ukazov, preostale neuporabljene kode pa se obnašajo kot zaporedja brez učinka. Posebnost sta ukazni kodi `0xED70` in `0xED71`, od katerih prva prebere vrednost iz V/I vrat in z njo nastavi samo zastavice, druga pa na V/I vrata zapiše vrednost nič. Pri štirizložnih skupinah s predponama `0xDD 0xCB` in `0xFD 0xCB` se bitne operacije izvedejo nad indeksirano pomnilniško lokacijo, pri delu ukazov pa zadnji zlog določi še dodatni ciljni register [30].

6.3 Prekinitve

Prekinitve omogočajo prekinitev operacij CPE, izvajanja PSP in vrnitev v prvotno stanje izvajanje programa. Prekinitev lahko sproži zunanja naprava ali nadzorni program računalnika. Na primer, CPE vsako 1/50 sekunde

pogleda stanje posebne notranje zastavice IFF1. Če je njena vrednost 1, sproži prekinitev in izvrši ukaz RST 38H, s katerim pridobi stanje tipkovnice [14, 31].

Ker se zaradi tega delovna hitrost procesorja nekoliko zmanjša, lahko program z ukazom DI prekinitve onemogoči in kasneje z ukazom EI ponovno omogoči. Program lahko prav tako z ukazom HALT prekine izvajanje strojnega programa do naslednje prekinitve. Med tem CPE izvaja operacije NOP za namen delovanja osveževanja pomnilnika [14, 31].

6.3.1 Načini prekinitev in njihovo delovanje

Z80 CPE podpira dve vrsti prekinitev:

Maskirne prekinitve INT, ki jih je mogoče programsko onemogočiti, z nastavljanjem vrednosti IFF1 prek ukazov EI in DI, saj se uporabljajo kot zahteva za prekinitev s strani zunanjih naprav. CPE ima poleg notranje zastavice IFF1, ki se dejansko uporablja pri izvajanju, še IFF2 kot začasno shrambo vrednosti IFF1. Glavna stanja obeh notranjih zastavic in spremembe ob pomembnih ukazih so vidne v tabeli 6.2 [31].

CPE je lahko nastavljen v enega izmed treh možnih načinov obravnavanja maskirnih prekinitev [31].

V načinu 0 lahko prekinitvena naprava na podatkovno vodilo postavi poljuben ukaz, ki ga CPE izvede. Ker mora izbran ukaz imeti le en zlog, je to pogosto ukaz klica PSP. CPE pri izvedbi tega ukaza samodejno doda dve čakalni periodi, da bi zagotovil dovolj časa za izvedbo zunanje verižne povezave za nadzor prioritet [31].

V načinu 1 CPE začne izvajati ukaz na pomnilniškem naslovu `0x0038`. Posledično je odziv razen lokacije klica identičen odzivu na nemaskirne prekinitve, ki kliče podprogram na pomnilniškem naslovu `0x0066`. CPE tudi v tem primeru doda dve čakalni periodi [31].

Način 2 omogoča klic podprograma v uporabniško vzdrževani tabeli 16-bitnih začetnih naslovov PSP na poljubnem mestu v pomnilniku. Naslovi PSP v tabeli se morajo nahajati na sodih naslovih, ker vsak zavzema dva

Dejanje	**IFF1**	**IFF2**	**Opombe**
Ponastavitev CPE	0	0	Prekinitve so onemogočene.
Izvedba ukaza `DI`	0	0	Prekinitve so onemogočene.
Izvedba ukaza `EI`	1	1	Prekinitve so omogočene.
Izvedba ukaza `LD A,I`	*	*	Vrednost IFF2 se shrani v zastavico paritete.
Izvedba ukaza `LD A,R`	*	*	Vrednost IFF2 se shrani v zastavico paritete.
Sprejem NMI	0	*	Prekinitve so onemogočene.
Izvedba ukaza `RETN`	IFF2	*	Dvignjena IFF2 označi konec rutine za obdelavo nemaskirnih prekinitev.

Tabela 6.2: Tabela stanj in sprememb stanj IFF1 in IFF2 [31].

zloga. Zgornji zlog 16-bitnega naslova PSP določa vrednost registra I, spodnji zlog pa sama prekinitvena naprava. Ko prekinitvena naprava zagotovi spodnji del kazalca, CPE samodejno potisne PC na sklad, pridobi naslov PSP iz tabele in izvede skok nanj. Ta način odziva zahteva 19 čakalnih period [31].

Prekinitve NMI imajo višjo prioriteto kot maskirne prekinitve INT in jih ni mogoče programsko onemogočiti, saj se uporabljajo predvsem za takojšnje odzive na najpomembnejše signale, ki se morajo obravnavati s čim manj zakasnitve. CPE ga zato obravnava že po koncu izvajanja trenutnega ukaza. Pri tem se najprej onemogoči IFF1 (in njegova prejšnja vrednost shrani v IFF2), kar za čas izvajanja PSP prepreči navadne prekinitve, nato se vrednost registra PC shrani na sklad, izvajanje pa preusmeri na pomnilniški naslov `0x0066` [31].

6.4 Implementacija v emulatorju

Procesor je v emulatorju predstavljen z objektom `Processor`. Ta ne hrani vseh podatkov procesorja neposredno v sebi, ampak povezuje ločene modele registrov, pomnilnika, V/I vrat in časovnika. Časovni model je implementiran v objektu `ULATiming`, ki hrani skupno število izvedenih T-stanj, trenutni položaj znotraj slikovnega okvira in števec prikazanih okvirjev. Emulator privzeto uporablja frekvenco procesorja 3,5 MHz in dolžino okvirja 69888 T-stanj, kar ustreza modelu računalnika 48K z evropskim video standardom. V objektu `Processor` so shranjena predvsem stanja, ki vodijo izvajanje, kot so omogočenost maskirnih prekinitev `IFF1` in `IFF2`, trenutni prekinitveni način, stanje `HALT`, prekinitvene točke in števec izvedenih ukazov.

Registri so zbrani v objektu `Registers`. Ta vsebuje glavni in zamenljiv registrski nabor ter posebne registre `I`, `R`, indeksna registra `IX` in `IY`, `SP`, `PC` in nedokumentirani notranji register `MEMPTR`. Zastavice niso ločen podatkovni tip, ampak posamezni biti v registru `F`. Dostop do njih je zato implementiran z metodami za branje in nastavljanje posameznih bitov. Vpogled v stanje registrov in zastavic je prikazan na sliki 6.3.

Procesor Z80 podpira več kot tisoč različnih dokumentiranih in nedokumentiranih strojnih ukazov. Posamezen strojni ukaz je predstavljen z vmesnikom `Instruction`. Ukaz pozna naslov, zloge, iz katerih je bil dekodiran, število T-stanj in metodo `execute`, ki izvede vse potrebne spremembe na stanju sistema ob izvedbi. Emulator ukaze z isto namembnostjo, a različnimi operandi, implementira z 230 definicijami ukazov. Definicija ukaza je ločena od njegove izvedbe z vmesnikom `InstructionDefinition`, ki opiše, kateri biti v prebranih zlogih tvorijo kodo ukaza oziroma opcode in kateri biti predstavljajo operande. Na primer, vzorec `00rrr110 nnnnnnnn` ustreza ukazu `LD r,n`, pri katerem fiksni biti določijo vrsto ukaza, biti `r` se dekodirajo v ciljni register, biti `n` pa v neposredno enozložno vrednost.

```
override val bitPattern = BitPattern.of("00rrr110 nnnnnnnn")
val destinationRegister = bitPattern.getRegisterCode(word, 'r')
```

Registers

Main register set			
A	00	F	5C
B	17	C	4B
D	00	E	06
H	10	L	7F

S	Z	F5	H	F3	PV	N	C
0	1	0	1	1	1	0	0

Alternate register set			
A'	00	F'	44
B'	00	C'	00
D'	5C	E'	B9
H'	00	L'	00

S	Z	F5	H	F3	PV	N	C
0	1	0	0	0	1	0	0

Special purpose registers			
I	3F	IX	0000
R	31	IY	5C3A
		SP	FF4E
		PC	15E1

Slika 6.3: Posnetek zaslona emulatorja z vpogledom v stanje registrov in zastavic.

```
val sourceUByte = bitPattern.getUByte(word, 'n')
```

Bitni vzorci so opisani z razredom `BitPattern`. V njem so fiksni biti zapisani kot ničle in enice, ostali znaki pa predstavljajo zajeta polja, na primer kodo registra, pogoj skoka, neposredni zlog ali 16-bitni operand. Dekodirnik v objektu `Instructions` iz pomnilnika prebere toliko zlogov, kot jih zahteva posamezen vzorec, prebrano vrednost primerja z registriranimi vzorci in ob ujemanju ustvari konkreten objekt ukaza. Vpogled v dekodirane ukaze je prikazan na sliki 6.4.

Posebna obravnava je potrebna pri predponah kodov ukazov. Predponi `0xDD` in `0xFD` pomenita, da se nekateri dostopi do `HL`, `H` in `L` začasno preusmerijo na `IX` oziroma `IY`. To je v emulatorju izvedeno z začasnim načinom indeksne predpone v objektu `Registers`. Dekodirnik zna obravnavati tudi kombinacije s predponama `0xCB`, ki označuje bitnih, rotacijskih in zamikalnih ukazov, ter `0xED`, ki označuje razširjene ukaze Z80, med katere spadajo tudi nekateri blokovni in vhodno-izhodni ukazi. Podpira tudi več zaporednih

indeksnih predpon. Tak ukaz se ovije v pomožni objekt, ki ohrani izvorne zloge, prišteje dodatna T-stanja za predpone in nastavi pravilen način dostopa do indeksnih registrov le za čas izvedbe takega ukaza.

Izvajanje enega koraka (naslednjega ukaza) se začne z branjem trenutnega programskega števca `PC`. Če je na tem naslovu nastavljena prekinitvena točka, se neprekinjen tek ustavi. Nato ima prednost morebitna nemaskirna prekinitev. Če je ni, se preverijo še pogoji za hitro nalaganje kasetnih podatkov. Šele nato se ukaz dekodira iz pomnilnika, števec osveževanja `R` se poveča za število dejansko prebranih ukaznih zlogov, programski števec pa se pomakne za dolžino ukaza.

Po predpripravi ukaza se izvede metoda `execute`. Ta lahko spremeni registre, zapisuje v pomnilnik, bere ali piše V/I vrata. Po izvedbi se poveča števec izvedenih ukazov za namene razhroščevalnika, časovniku ULA pa se prišteje število T-stanj ukaza. Ko število T-stanj preseže dolžino okvirja, časovnik poveča števec okvirjev, označi čakajočo prekinitev in o tem obvesti dele uporabniškega vmesnika, ki so vezani na osveževanje slike. Pogoji za izvedbo maskirne prekinitve se preverjajo ob koncu izvajanja trenutnega ukaza, zato časovnik povezuje hitrost izvajanja, periodične prekinitve in osveževanje zaslonskega prikaza.

Pri vseh maskirnih prekinitvah se trenutna vrednost registra `PC` najprej zapiše na sklad, nato se onemogočita `IFF1` in `IFF2`. Maskirna prekinitev ponastavi vrednost le `IFF1`, na sklad shrani vrednost registra `PC` in nadaljuje izvajanje na naslovu `0x0066`. Ukaza `EI` in `DI` nastavita še notranjo zastavico, zaradi katere se maskirna prekinitev ne obdela takoj za njima.

Ukaz `HALT` je izveden tako, da programski števec vrne na naslov ukaza `HALT` in nastavi stanje zaustavitve. Dokler ne pride do prekinitve, se ob vsakem koraku znova izvaja isti ukaz, v tem primeru ukaz `NOP`. Ob prekinitvi se stanje `HALT` zaključi, na sklad pa se zapiše naslov naslednjega ukaza, kot pri pravem procesorju.

Poleg izvajanja ukazov po korakih emulator podpira tudi neprekinjen tek. V tem načinu delovanja procesor ponavlja izvajanje ukazov, dokler uporabnik

emulatorja ne zaustavi. V načinu delovanja emulatorja ob standardni hitrosti primerja število izvedenih T-stanj z realnim časom izvajanja in po potrebi počaka z izvajanjem nadaljnjega ukaza, da se izvajanje približa frekvenci 3,5 MHz. V načinu delovanja emulatorja ob najvišji možni hitrosti tega čakanja ni, zato emulator izvaja ukaze tako hitro, kot dopušča gostiteljski sistem.

Disassembly

ROM

```
THE 'WAIT-KEY' SUBROUTINE
15D8    20 04        JR NZ, +04h
15DA    FD
15DE    CD E6 15     CALL 15E6h
15E1    D8           RET C
15E2    28 FA        JR Z, -06h
15E4    CF           RST 08h
15E5    07           RLCA
THE 'INPUT-AD' SUBROUTINE
15E6    D9           EXX'
15E7    E5           PUSH HL
15E8    2A 51 5C     LD HL, (5C51h)
15EB    23           INC HL
15EC    23           INC HL
15ED    18 08        JR +08h
THE 'MAIN PRINTING' SUBROUTINE
15EF    1E 30        LD E, 30h
15F1    83           ADD A, E
15F2    D9           EXX'
15F3    E5           PUSH HL
15F4    2A 51 5C     LD HL, (5C51h)
15F7    5E           LD E, (HL)
15F8    23           INC HL
15F9    56           LD D, (HL)
15FA    EB           EX DE, HL
```

Call the input subroutine indirectly via INPUT_AD.

Slika 6.4: Posnetek zaslona emulatorja z anotiranim vpogledom v dekodirane ukaze.

Poglavje 7

Vezje ULA

Vezje ULA je predhodnik modernejših večjih uporabniško programirljivih vezij in predstavlja vmesnik med procesorjem in video krmilnikom, tipkovnico in zvočnikom.

Procesorja pri branju in pisanju iz pomnilniških lokacij ne zanima, s katerim delom pomnilnika dela, pa naj bo to ROM ali RAM. Ve le, da obstaja 65536 pomnilniških celic, v katerih lahko obdeluje shranjene podatke. Na popolnoma enak način dela tudi s 65536 V/I naslovi, od katerih je odvisno delovanje zunanjih naprav, kot so tipkovnica, zvočnik, kasetofon prek vtičnic EAR in MIC, tiskalnik in drugi robni vmesniki in naprave pa prek robnega priključka, dostopni na zadnji strani računalnika [23] [14].

V/I naslovi so pri procesorju Z80 sestavljeni iz 16 bitov (A15, A14 ... A1, A0), vendar imajo posamezni biti različen pomen. Nižji zlog naslova označuje eno izmed 256 možnih V/I vrat. Ker se v večini primerov za določitev ciljne naprave operacije uporablja le ta spodnji zlog, se lahko različni V/I naslovi nanašajo na isto napravo. Če je najnižji bit naslova V/I vrat nastavljen na 0, torej je naslov sod, se podatki preusmerjajo prek ULA, vrata z lihimi naslovi pa so namenjena V/I operacijam z drugimi zunanjimi napravami [14, 23].

Vrednosti na V/I naslovih, ki prehajajo med V/I vrati in CPE, so sestavljene iz 8 bitov [14, 23].

Naslov V/I vrat 251 se nanaša na tiskalnik, naslovi vrat 254, 247 in 239

pa na dodatne zunanje naprave, med drugim se naslov V/I vrat 254 nanaša na zvočnik (D4), MIC vtičnico (D3) in nastavlja barvo obrobe zaslona (D2, D1 in D0) [23].

7.1 Zaslon

V času nastanka računalnika ZX Spectrum so se uporabljali UHF televizorji, kateremu je računalnik oddajal zaslonski signal na UHF kanalu 36. Pomembna je bila tudi različica računalnika, saj so se standardi televizorskih sistemov razlikovali po državah in celinah – evropski in britanski televizorji so takrat prikazovali 50 slik na sekundo, sistemi v ZDA, Kanadi in Japonski pa 60 slik na sekundo. Zaradi tega so zanje delali drugačno verzijo računalnika [24].

7.1.1 Razdelitev

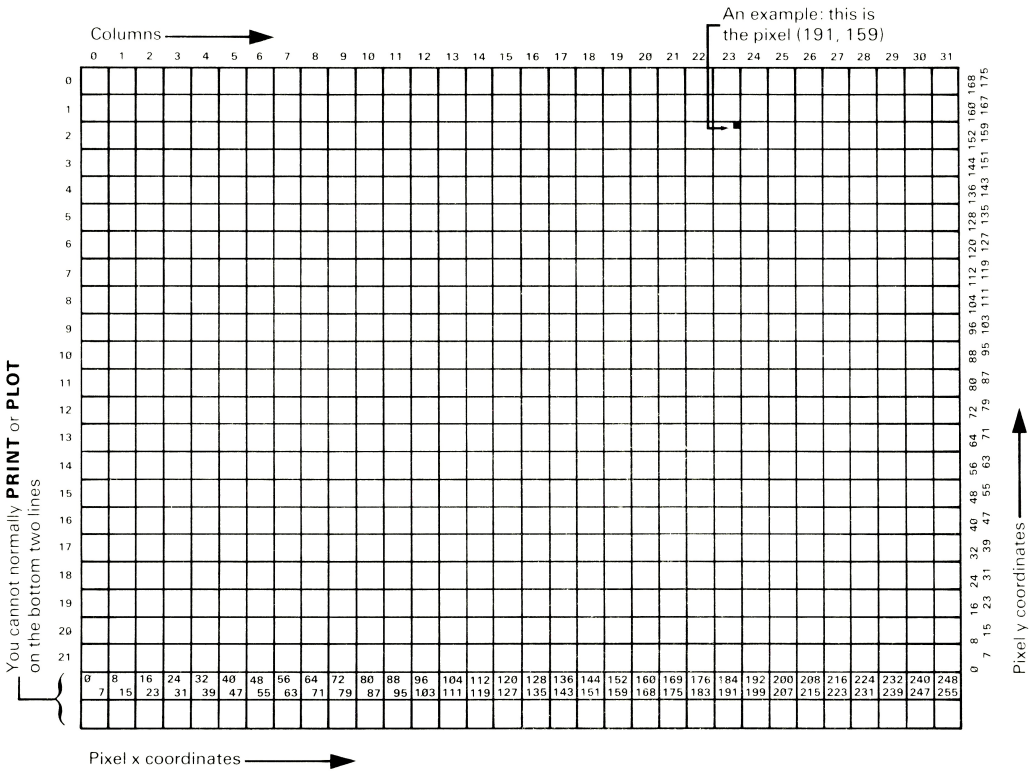
Televizijski zaslon ima zunanji del, ki se imenuje obroba zaslona, in osrednji del, ki ga imenujemo papir oziroma zaslonska slika. Slednja je razdeljena na 768 položajev znakov (prek 24 vrstic in 32 stolpcev), na katerih se prikazujejo znaki. Vsak znak je tako predstavljen kot 8 x 8 kvadrat točk, ki ga lahko shranimo v 8 zlogih (vsak za eno vrstico točk), vsaka točka pa z enim bitom, ki določa barvo točke – barva papirja, če ima pripadajoč bit vrednost 0, ali pa v barvo črnila pri vrednosti bita 1 [23, 24].

Ker je vsak položaj znaka velik 8 x 8 pik, bi lahko pričakovali, da so v pomnilniku skupaj shranjeni vsi podatki za en položaj znaka – vseh osem vrstic osmih pik prvega položaja znaka, nato vseh osem vrstic osmih pik naslednjega položaja znaka itd. Vendar pri računalniku ZX Spectrum vezje ULA podatke zaslonske slike v pomnilniku pretvarja v televizijski video signal, ki je namenjen rastrskemu grafičnemu prikazu na CRT televizorju, pri katerem elektronski snop sliko izrisuje po vodoravnih vrsticah točk, od leve proti desni. Zato so skupaj shranjeni zlogi, ki predstavljajo isto vrstico pik pri zaporednih položajih znakov čez celotno širino zaslona.

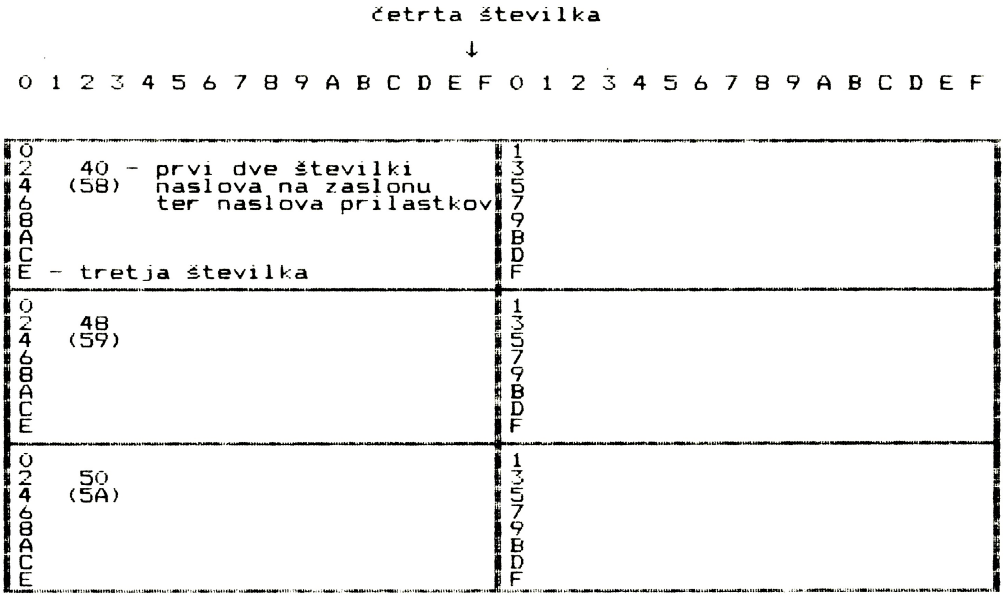
V pomnilniku so zato najprej shranjene prve vrstice pik zaporednih položajev znakov, nato druge vrstice pik istih položajev in tako naprej do osme vrstice pik. A to zaporedje ne poteka čez vseh 24 vrstic znakov zapored, temveč je zaslonska slika razdeljena na tri večje dele. Prvi del vsebuje podatke za vrstice znakov 0 do 7, drugi za vrstice 8 do 15, tretji pa za vrstice 16 do 23. Zaradi take razporeditve so biti naslova zloga za točko (X, Y) v naslovu premešani, kot prikazuje tabela 7.1. Razdelitev zaslona je prikazana na slikah 7.1 in 7.2 [23].

Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 9**	**Bit 8**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0	1	0	Y_7	Y_6	Y_2	Y_1	Y_0	Y_5	Y_4	Y_3	X_7	X_6	X_5	X_4	X_3

Tabela 7.1: Bitna sestava naslova zloga zaslonske slike za točko (X, Y).



Slika 7.1: Prikaz delitve zaslona na vrstice in stolpce [23].



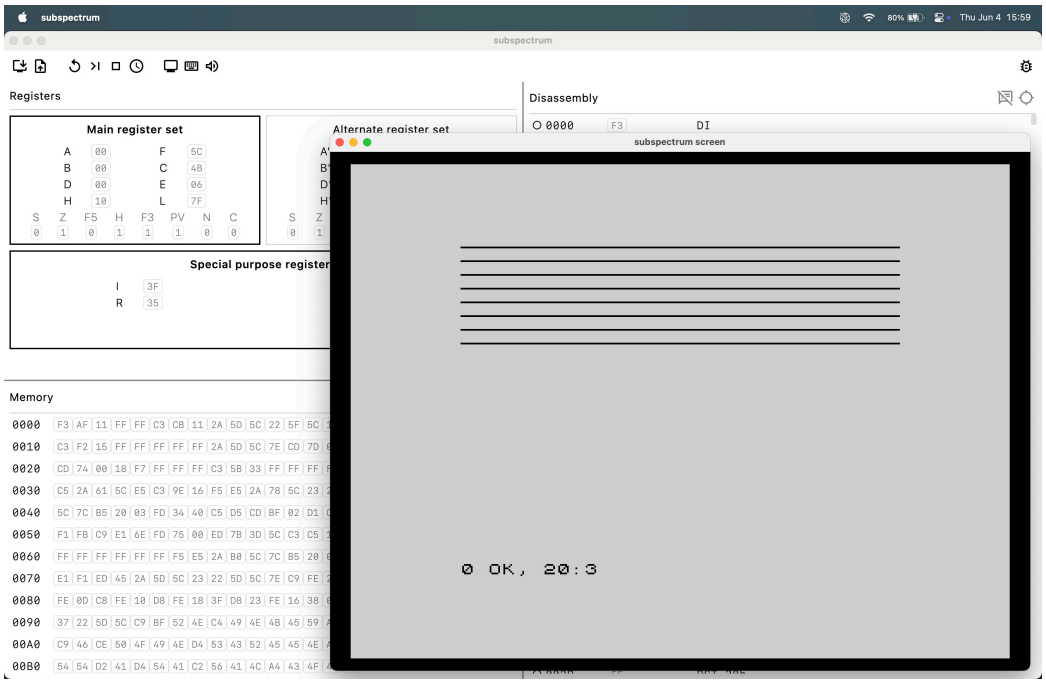
Slika 7.2: Diagram razbiranja naslovov položajev znakov na zaslonu v pomnilniški zaslonski datoteki [14].

Za lažje razumevanje vzemimo kot primer program v programskem jeziku BASIC:

```
10 CLS
20 FOR i=0 TO 255: POKE 16384+i,255: NEXT i
```

Program naredi naslednje: najprej pobriše zaslon, nato pa v pomnilnik na začetek zaslonske datoteke zapiše 256 zlogov, vsak od njih ima vrednost 255 oziroma 8 bitov vrednosti 1. Ob zagonu programa se na zaslonu izriše vzorec, viden na sliki 7.3: 8 vrstic točk, razporejenih v prvih 8 vrhnjih vrstic položajev znakov, ki so se izrisovale po vrsticah od leve proti desni. Če bi zanko izvajali dlje, bi se obarvale vrstice točk neposredno pod temi vrsticami, in tako dalje. Primer tega je viden na sliki 7.4, kjer smo zanko podaljšali iz 255 na 495 zaporednih zlogov, s čimer se obarvajo nadaljnje vrstice točk do sredine osme

vrstice položajev znakov. Če bi zanko še podaljševali, bi se obarvale vse vrstice točk položajev znakov zgornje tretjine zaslona, nato pa bi se vzorec ponovil še v srednji in spodnji tretjini zaslona.

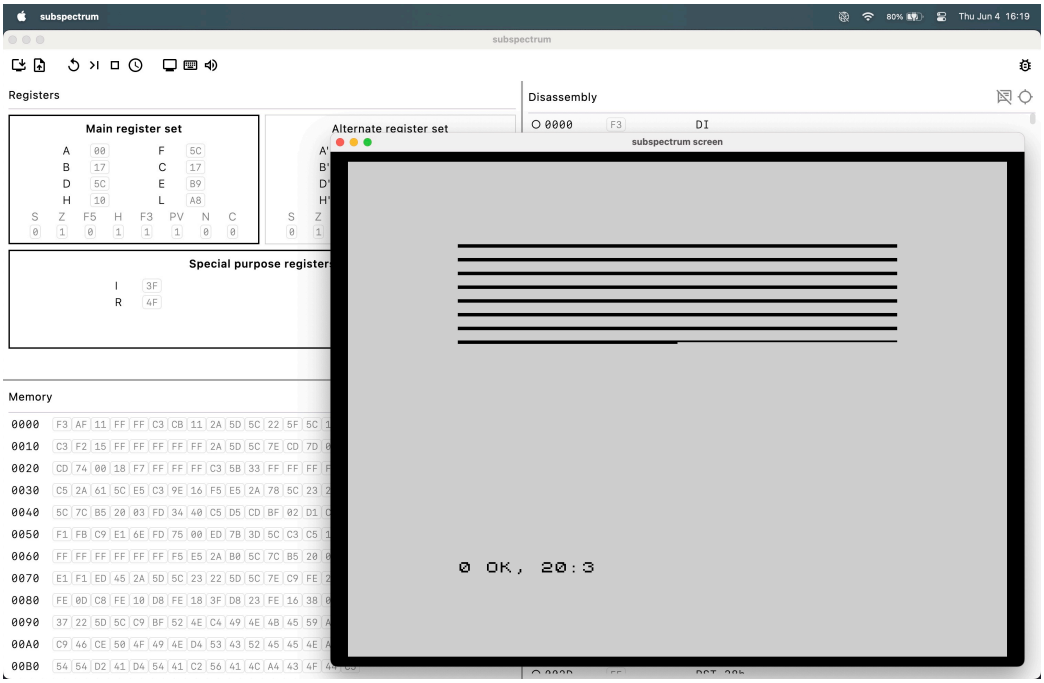


Slika 7.3: Prikaz zagona programa zaporednega pisanja v pomnilniško zaslonsko datoteko.

7.1.2 Nabor znakov

Nabor 256 znakov, ki ga uporablja ZX Spectrum, sestavljajo [23]:

- posamezni znaki iz standarda ASCII (razen simbolov £ in ©) ter grafični simboli, značilni le za ZX Spectrum. Prvih 16 grafičnih simbolov je sistemsko definiranih (vidimo jih na sliki 7.5), drugih 21 pa je uporabniško definiranih;
- žetoni, ki predstavljajo bodisi kontrolne znake z učinki na zunanje naprave, bodisi navodila računalniku v obliki celih besed (npr. PRINT,



Slika 7.4: Prikaz zagona programa nadaljnjega zaporednega pisanja v pomnilniško zaslonsko datoteko.

STOP ipd.).

V ROMu so med naslovoma 3D00 in 3FFF shranjeni točkovni vzorci vseh znakov, razen grafičnih simbolov. Naslov posameznega znaka, ki ima kod x (npr. znak 'a' ima kod 0x61), izračunamo po formuli $8x + 3C00$. Definicije uporabniško definiranih grafičnih znakov pa se nahajajo med naslovoma `0xFF58` in `0xFFFF` [14, 23].

7.1.3 Atributi znakov

Vsak znak ima določene naslednje atribute v enem zlogu, njihova bitna razporeditev pa je prikazana v tabeli 7.2 [14, 23, 24]:

- bit utripanja ima dve možni vrednosti: 0 za stalne barve ali 1 za utripanje, ki se prikaže kot zaporedno zamenjavanje barve črnila in papirja;

Symbol	Code	How obtained	Symbol	Code	How obtained
	128	G 8		143	G shift 8
	129	G 1		142	G shift 1
	130	G 2		141	G shift 2
	131	G 3		140	G shift 3
	132	G 4		139	G shift 4
	133	G 5		138	G shift 5
	134	G 6		137	G shift 6
	135	G 7		136	G shift 7

Slika 7.5: Tabela sistemsko definiranih grafičnih simbolov [23].

- bit svetlosti ima dve možni vrednosti: 0 za normalne ali 1 za svetlejše barve;
- trojica bitov koda barve papirja (privzeto bela);
- trojica bitov koda barve črnila (privzeto črna).

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Utripanje	Svetlost	Papir_2	Papir_1	Papir_0	Črnilo_2	Črnilo_1	Črnilo_0

Tabela 7.2: Bitna sestava zloga atributov za položaj znaka.

ZX Spectrum lahko na (barvnem) zaslonu prikazuje osem barv, označenih s kodi od 0 do 7 [23, 24]:

- 0, ki označuje črno;
- 1, ki označuje modro;
- 2, ki označuje rdečo;
- 3, ki označuje vijolično ali magenta;
- 4, ki označuje zeleno;
- 5, ki označuje svetlo modro ali cian;
- 6, ki označuje rumeno;
- 7, ki označuje belo.

Čeprav so barve na prvi pogled razporejene v naključnem vrstnem redu, je tako zaradi dveh razlogov. Prvi razlog se nanaša na uporabo računalnika na črno-belih zaslonih, kjer se barve v danem vrstnem redu prikažejo v lestvici odtenkov sivin. Drugi razlog se opira na dejstvo, da lahko človeško oko resnično vidi le tri primarne barve (modro, rdečo in zeleno), druge barve pa so le njihove mešanice. Tak vrstni red barv omogoča "mešanje"barv s seštevanjem kodov primarnih barv. Na primer magenta je pravzaprav mešanica modre in rdeče barve, zato je njen kod (3) vsota kodov za modro (1) in rdečo (2) [23, 24].

Kot barvo črnila ali papirja lahko nastavimo tudi vrednosti [23]:

- 8, ki označuje "prozorno", pri kateri bo barva ob izpisu znaka ostala nespremenjena glede na barvo prejšnjega znaka;
- 9, ki pomeni "kontrast", pri kateri bo barva nastavljena v kontrastu z drugo – bela, če je druga barva temna (črna, modra, rdeča ali magenta), ali črna, če je druga barva svetla (zelena, cian, rumena ali bela).

Enozložna vrednost atributa se nanaša na vseh 64 pik položaja znaka, ne le na posamezne pike znaka. Ko znak natisnemo na zaslon, poleg vzorca pik spremenimo tudi nastavitve atributov položaja znaka [23].

7.1.4 Implementacija v emulatorju

Zaslonski del emulatorja je razdeljen na logiko za razlago pomnilniškega zapisa in na prikaz v uporabniškem vmesniku. Objekt `ULAScreen` vsebuje stalne vrednosti za velikost slike, velikost obrobe, začetne in končne naslove zaslonske datoteke ter datoteke atributov. V njem je definirana tudi preslikava Spectrumovih barv v barve uporabniškega vmesnika ter funkcije za razlago bitov enozložne vrednosti atributa.

Okno zaslona (vidno na sliki 7.6) ob periodičnih prekinitvah, ki jih proži ULA za posodobitev izrisa zaslonske slike, iz pomnilnika prebere območji zaslonske datoteke in datoteke atributov na naslovih `0x4000-0x57FF` in `0x5800-0x5AFF`. Funkcija za izračun indeksa zloga zaslonske datoteke upošteva nelinearen razpored vrstic, ki ga uporablja ZX Spectrum. Prikaz je narejen z grafično komponento platna ogrodja Compose, pri čemer se velikost navidezne točke prilagodi velikosti okna, da ostane nespremenjeno razmerje slike, uporabljano v zaslonih po standardnih televizijah v času nastanka računalnika. Tako se izognemo nepravilnim proporcijam navideznih točk in posledično popačenju slike.

7.2 Tipkovnica

Znaki, ki jih ZX Spectrum prepozna in uporablja, ne vključujejo le posameznih znakov, kot so črke, številke in simboli, ampak tudi sestavljene znake, kot so ključne besede in imena funkcij [23].

Tipkovnica Spectruma je na prvi pogled po položaju tipk črk in številk podobna tipkovnici navadnega pisalnega stroja, vendar pa ima vsaka tipka Spectruma več funkcij. Razporeditev tipk je vidna na sliki 7.7 [24].

Da bi lahko z omejenim številom tipk na tipkovnici dostopali do vseh znakov in ukazov, imajo nekatere tipke več različnih pomenov, do katerih pridemo z uporabo modifikacijskih tipk CAPS SHIFT ali SYMBOL SHIFT ter menjavo načina delovanja tipkovnice. Povezava med simboli na tipkah in načini vnosa je vidna na sliki 7.8 [23].

Slika 7.6: Posnetek zaslona emuliranega zaslona.

7.2.1 Načini delovanja

Kurzor, predstavljen kot utripajoča črka, hkrati označuje položaj izpisa naslednjega vnesenega znaka in trenutni način delovanja tipkovnice s črko kurzorja [23, 24].

Ko Spectrum prvič vklopite, ta po sporočilu o avtorskih pravicah pričakuje vnos ukaza, ki se začne s ključno besedo, katere so natisnjene pod glavnimi simboli na tipkah. Za razliko od večine drugih računalnikov Spectrum omogoča vnos ključnih besed z enim samim pritiskom na tipko. Kurzor s simbolom K (angl. Keyword) predstavlja ta način delovanja za vnos ključnih besed, ki samodejno nadomesti način L, ko naprava pričakuje vnos ukaza ali programske vrstice, kar je odvisno od pozicije kurzorja v vrstici [23, 24].

Slika 7.7: Shema razporeditve tipk tipkovnice ZX Spectruma [13].

V večini drugih primerov bomo v računalnik želeli vnašati besedilo, kar naredimo v načinu delovanja za vnos navadnega besedila. V njem ima kurzor simbol L (angl. Letter) [24] [23].

Na načinu K in L velja, da je za vse kontrolne funkcije, ki so nad tipkami obarvane belo, potrebna pritisnjena tipka CAPS SHIFT, za vse pomožne znake, ki so pod tipkami obarvani rdeče, pa tipka SYMBOL SHIFT. Če želimo vnesti veliko velikih črk, ne da bi držali tipko CAPS SHIFT, lahko vse črke vnašamo kot velike tako, da najprej pritisnemo tipko CAPS LOCK (CAPS SHIFT in 2). CAPS SHIFT z drugimi tipkami ne vpliva na ključne besede v načinu K, v načinu L pa pretvori male črke v velike [23, 24].

Način delovanja za vnos besedila z velikimi črkami, v katerem ima kurzor simbol C (angl. Capitals), je različica načina L, v katerem se vse črke prikažejo kot velike. Pritisk na tipko CAPS LOCK povzroči prehod iz načina L v način C ali obratno [23, 24].

Poleg ključnih besed in besedila (malih in velikih črk ter številk) ima tipkovnica tudi osem vnaprej definiranih grafičnih znakov, ki so na tipkovnici označeni na številčnih tipkah od 1 do 8, ali pa uporabniško določene grafične znake. Ti se lahko natisnejo na zaslon na podoben način kot črke in številke. Za to je treba tipkovnico preklopiti v način delovanja za vnos grafičnih znakov,

MODE	SHIFT										
E	SYMBOL	DEF FN	FN	LINE	OPEN #	CLOSE #	MOVE	ERASE	POINT	CAT	FORMAT
	CAPS	Ink blue	Ink red	Ink magnta	Ink green	Ink cyan	Ink yellow	Ink white	Flash off	Flash on	Ink black
	NONE	Paper blue	Paper red	Paper magnta	Paper green	Paper cyan	Paper yellow	Paper white	Normal brite	Extra brite	Paper black
G	EITHER	◘	◙	◚	◛	◧	◨	◩	■	EXIT GRAPH	DELETE
	NONE	◔	◕	◖	◗	◨	◩	◪	□	EXIT GRAPH	DELETE
K, L or C	CAPS	EDIT	CAPS LOCK	TRUE VIDEO	INV. VIDEO	⇦	⇩	⇧	⇨	GRAPH MODE	DELETE
	SYMBOL	!	@	#	$	%	&	'	(	)	_
	NONE	1	2	3	4	5	6	7	8	9	Ø

Slika 7.8: Tabela simbolov in potrebnih pritiskov tipk zanje [23].

kar se naredi s pritiskom tipke GRAPHICS (kombinacija tipk CAPS SHIFT in 9). Ta način traja, dokler se ponovno ne pritisne bodisi tipka GRAPHICS bodisi le tipka 9. Kurzor bo v tem načinu delovanja označen s simbolom G (angl. Graphics) [23, 24].

Obstaja še en zadnji – razširjeni – način, ki ga označuje kurzor s simbolom E (angl. Extended). Do njega pridemo s hkratnim pritiskom tipk CAPS SHIFT in SYMBOL SHIFT in traja le do vključno naslednjega pritiska tipke. Ta način omogoča uporabo večine znanstvenih in programskih funkcij ter pridobitev dodatnih znakov oz. simbolov [23, 24].

- zgoraj prikazan v zeleni barvi, če med tem ni pritisnjena tipka SHIFT;
- spodaj prikazan v rdeči barvi, če je pritisnjena ena od tipk SHIFT;
- simbol na številčni tipki, če je pritisnjena tipka SYMBOL SHIFT;
- sicer da barvno kontrolno zaporedje.

Kurzor s simbolom vprašaja pomeni, da je pri vnosu ukaza ali znaka prišlo do napake [24].

7.2.2 Branje stanja tipkovnice

Za branje tipkovnice lahko uporabimo ROM-podprograme `KEY SCAN`, `KEY TEST` in `KEY CODE`. Podprogram `KEY SCAN` najprej pregleda stanje tipkovnice in vrne podatek o pritisnjenih tipkah, pri čemer loči med navadnimi tipkami, kombinacijami s tipkama `CAPS SHIFT` in `SYMBOL SHIFT` ter primeri, ko ni pritisnjena nobena uporabna tipka. Rezultat nato obdela `KEY TEST`, ki zavrže neuporabne ali nesmiselne vrednosti in pripravi podatke za nadaljnjo obdelavo. Na koncu `KEY CODE` glede na osnovno kodo tipke, pritisnjeno SHIFT tipko in trenutni način delovanja tipkovnice, določi dejanski znak ter njegovo ASCII-kodo. Kodi tipk so zapisani v tabeli `KEY TABLE` na pomnilniškem naslovu `0x0205`, vidni so tudi na sliki 7.9 [14].

```
[ 1 ][ 2 ][ 3 ][ 4 ][ 5 ][ 6 ][ 7 ][ 8 ][ 9 ][ 0 ]
 24   1C   14   0C   04   03   0B   13   1B   23

 [ Q ][ W ][ E ][ R ][ T ][ Y ][ U ][ I ][ O ][ P ]
  25   1D   15   0D   05   02   0A   12   1A   22

  [ A ][ S ][ D ][ F ][ G ][ H ][ J ][ K ][ L ][ENTER]
   26   1E   16   0E   06   01   09   11   19   21

 [CAPS][ Z ][ X ][ C ][ V ][ B ][ N ][ M ][SYMB][SPACE]
  27   1F   17   0F   07   00   08   10   18   20
```

Slika 7.9: Skica tipkovnice Spectruma in kodi tipk (razvrščeni v spirali v nasprotni smeri urinega kazalca) [14].

Podatke iz tipkovnice lahko namesto z uporabo podprogramov v ROMu pridobimo tudi neposredno prek V/I vrat `0xFE`. Tipkovnica je razdeljena v 8 polvrstic s 5 tipkami. Podatke za pritisnjene tipke lahko zato pridobimo le v teh petericah tipk naenkrat. Prikazane so na sliki 7.10 [14].

```
3 = ( 1  2  3  4  5 )  ( 6  7  8  9  0  ) = 4
2 = ( Q  W  E  R  T )  ( Y  U  I  O  P  ) = 5
1 = ( A  S  D  F  G )  ( H  J  K  L  enter) = 6
0 = (Caps Z  X  C  V )  ( B  N  M  Sym space) = 7
     shift                          shift
```

Slika 7.10: Skica skupin tipk in njihovih kodov [14].

Skupino tipk pri tem določa zgornji zlog V/I naslova, v katerem mora zaporedni bit, ki predstavlja številko želene skupine, imeti vrednost 0, vsi ostali pa 1. Naslov se tako izračuna kot $254 + 256 * (255 - 2^n)$; $n \in [0, 7]$, ti so vidni tudi v tabeli 7.3 [14, 23]:

- naslov `0xFEFE` predstavlja tipke med tipkama CAPS SHIFT in V;
- naslov `0xFDFE` predstavlja tipke med tipkama A in G;
- naslov `0xFBFE` predstavlja tipke med tipkama Q in T;
- naslov `0xF7FE` predstavlja tipke med tipkama 1 in 5;
- naslov `0xEFFE` predstavlja tipke med tipkama O in 6;
- naslov `0xDFFE` predstavlja tipke med tipkama P in 7;
- naslov `0xBFFE` predstavlja tipke med tipkama ENTER in H;
- naslov `0x7FFE` predstavlja tipke med tipkama SPACE in B.

Register A po izvršenem ukazu branja stanja tipkovnice vsebuje zlog podatkov, pri čemer spodnjih pet bitov označuje pritisnjeno tipko. Najnižji bit ustreza najbolj zunanji tipki, za pritisnjeno tipko ima bit vrednost 0, sicer 1 [14, 23].

Sicer lahko, z nastavitvijo zgornjega zloga V/I naslova z večimi ničlami, beremo tudi več skupin tipk hkrati, a posamezen bit rezultata v tem primeru predstavlja zaporedno pritisnjeno tipko katere koli izmed izbranih skupin tipk [14, 23].

V/I naslov	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
`0x7FFE`	B	N	M	Symbol Shift	Space
`0xBFFE`	H	J	K	L	Enter
`0xDFFE`	Y	U	I	O	P
`0xEFFE`	6	7	8	9	0
`0xF7FE`	5	4	3	2	1
`0xFBFE`	T	R	E	W	Q
`0xFDFE`	G	F	D	S	A
`0xFEFE`	V	C	X	Z	Caps Shift

Tabela 7.3: Tabela naslovov posameznih tipk računalnika ZX Spectrum.

7.2.3 Implementacija v emulatorju

Tipkovnica je v emulatorju predstavljena z objektom `ULAKeyboard`. Trenutno stanje hrani v polju `keyboardRows`, ki vsebuje osem petbitnih vrednosti, po eno za vsako polovico vrstice oziroma skupino tipk tipkovnice. Začetna vrednost posamezne skupine je `11111`. Ob pritisku tipke na dejanskem računalniku, ki poganja emulator, metoda `setMatrixKeyState` vrednost ustreznega bita nastavi na 0, ob spustu pa ga nastavi na 1. S tem je stanje tipkovnice shranjeno v enaki obliki, kot jo pri branju pričakuje emulirani Spectrum.

Branje tipkovnice je vključeno v splošni model uporabe V/I vrat. Vsa branja gredo skozi razred `IOPortSet`. Če je spodnji zlog dvozložnega V/I naslova ukaza enak `0xFE`, razred med drugim pokliče metodo `getKeyboardPortValue`. Ta iz zgornjega zloga naslova določi izbrane skupine tipk: vsak bit z vrednostjo 0 izbere eno izmed osmih skupin. Vrednosti izbranih skupin združi z operacijo AND, kar ohrani ničle pritisnjenih tipk tudi pri branju več skupin hkrati. Rezultat dopolni do osmih bitov: biti na položajih med 0 in 4 so biti tipk, bita na položajih 5 in 7 nista uporabljena in sta vedno nastavljena na vrednost 1, bit na položaju 6 pa predstavlja vhod vtičnice EAR, ki ga emulator ne emulira in ga nastavi na vrednost 0.

Vhod iz uporabniškega vmesnika se v matriko prenese prek dogodkov dejanske tipkovnice na oknih emulatorja. Metoda `handlePreviewKeyEvent` dogodke pritiska in spusta prevede v spremembe stanja matrike, pri besedilnih dogodkih brez običajnega spusta tipke pa uporabi kratek programski pritisk.

Emulator podpira dva načina preslikave razporeditve tipk dejanske tipkovnice. V "avtentičnem" načinu se fizične tipke preslikajo na položaje tipk Spectruma, zato sta na primer `Shift` in `Ctrl` uporabljena kot `CAPS SHIFT` in `SYMBOL SHIFT`. Za lažjo uporabo emulator podpira še "dejanski" način, v katerem se namesto položaja fizične tipke za določitev vnosa uporabi znak, ki ga posreduje gostiteljski operacijski sistem (z upoštevanjem uporabnikove razporeditve tipk). Metoda `getMatrixKeysForCharacter` ga nato preslika v eno ali več tipk, na primer pri velikih črkah in simbolih v matrike emuliranih tipk.

7.3 Kasete

ZX Spectrum za trajno hrambo uporabniškega programa uporablja navadne zvočne kasete. Programe lahko na kaseto shranimo, preverimo za možne napake pri prenosih ter jih iz nje kasneje naložimo v pomnilnik in zaženemo [23, 24].

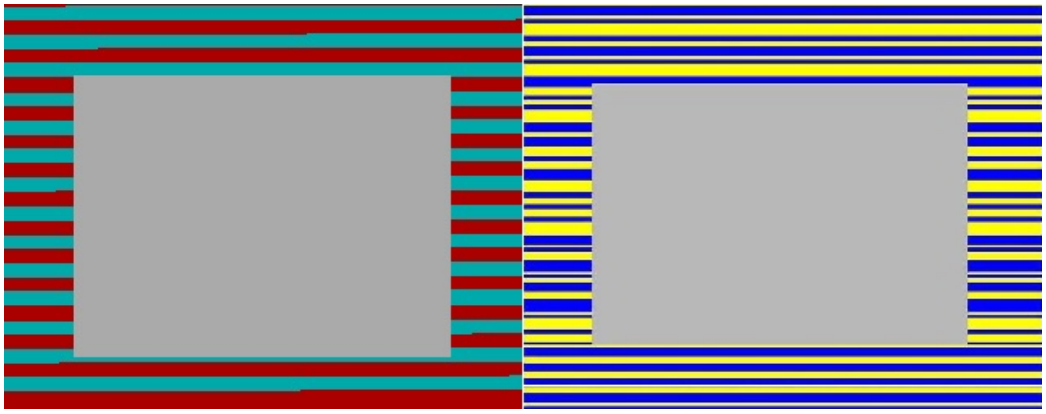
Kasetne posnetke ločimo od posnetkov stanja, kot sta datotečna formata SNA in Z80. Posnetek stanja hrani le stanje pomnilnika, registrov in drugih indikatorjev stanja sistema, kasetni format pa hrani le podatke kasete v obliki, v kakršni bi jih računalnik prebral iz kasete. S tem se ohrani ne le vrstni red nalaganja blokov, temveč tudi morebitne posebne nalagalnike programov [27, 29].

Posnetki kaset v formatih, primernih za uporabo v takšni emulaciji, so razdeljeni na bloke. Pri običajnem nalaganju se najprej prebere glava zapisa z osnovnimi podatki o programu, nato pa še podatkovni bloki z dejansko vsebino. Vsak blok vsebuje zastavični zlog, podatke in kontrolno vsoto [27].

Format TAP je neposreden zapis takih blokov. Datoteka `.tap` je sesta-

vljena iz zaporedja enako sestavljenih blokov, pri čemer ima vsak blok najprej dva zloga dolžine, nato pa sledijo zastavični zlog, zlog s kontrolno vsoto in nato dejanski podatki. Format je zato preprost in primeren za programe, shranjene s standardnimi podprogrami ROMa, ne ohrani pa natančnih časov impulzov, pavz ali delovanja posebnih nalagalnikov [27].

Pri nalaganju kaset s standardnim podprogramom iz ROMa se branje uvodnega dela bloka na zaslonu prikazuje kot rdeče in svetlo modre črte, branje dejanskih podatkov pa kot modre in rumene črte v območju obrobe zaslona. Vzorca sta vidna na sliki 7.11 [23, 24].



Slika 7.11: Vzorci, ki se med nalaganjem programa iz kasete prikazujejo na zaslonu, ko se za nalaganje uporablja standardni podprogram iz ROMa [3].

Format TZX je namenjen čim natančnejšemu hranjenju kaset za ZX Spectrum, predvsem za uporabo v emulatorjih. Vsebina datoteke formata `.tzx` se začne z glavo `ZXTape!` in številko različice formata, za tem pa sledi en ali več blokov različnih tipov. Če je bil program shranjen z navadnimi ROMovimi rutinami, datoteka v formatu TZX načeloma vsebuje enake standardne podatkovne bloke kot TAP. Njegova prednost je, da lahko poleg osnovnih tipov blokov hrani tudi tudi pavze, časovne podatke o impulzih in druge posebne bloke, ki jih uporabljajo posebni nalagalniki ali zaščite. Zato je format TZX primernejši takrat, ko želimo v emulatorju čim bolj natančno ponoviti potek izvirnega nalaganja s kasete [28] [27].

7.3.1 Implementacija v emulatorju

Nalaganje kaset je v emulatorju razdeljeno na razčlenjevanje datoteke in na izvajanje nalaganja med tekom programa. Razčlenjevanje izvaja objekt `ULATapeParser`. Če se vsebina datoteke začne z značilno glavo `ZXTape!`, jo obravnava kot datotečni format TZX, sicer pa kot TAP.

Razčlenjen posnetek kasete je predstavljen z razredoma `SpectrumTapeImage` in `SpectrumTapeDataBlock`. Posamezen blok hrani zastavični zlog, koristne podatke in kontrolno vsoto. Kontrolna vsota se preveri z operacijo XOR nad zlogom z zastavicami in vsemi podatkovnimi zlogi.

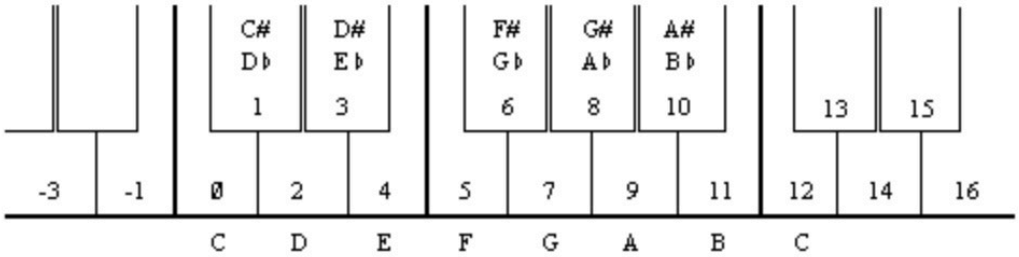
Objekt `ULATapeDeck` hrani vsebino trenutno vstavljene kasete, položaj bralne glave in morebitni podatkovni blok, ki pripada že prebrani glavi. Pri formatu TAP zaporedno bere dolžino bloka in surove podatke bloka. Pri formatu TZX prepozna standardne, hitrejše in čiste podatkovne bloke, druge s strani formata podprte bloke pa preskoči, kadar ne vplivajo na potek nalaganja. Razčlenjevalnik pri veljavnih standardnih zapisih označi tudi povezavo med blokom glave in naslednjim podatkovnim blokom, kar kasneje poenostavi izbiro pravega bloka pri nalaganju več zaporednih blokov.

Ko program v emuliranem Spectrumu pokliče ROMov podprogram `LD-BYTES` na naslovu `0x0556`, emulator to zazna in obravnavo preda kasetnemu podsistemu emulatorja. Ta iz registrov prebere pričakovani tip bloka, dolžino in ciljni naslov. Pri nalaganju iz kasete se koristni podatki iz izbranega bloka zapišejo v pomnilnik, pri preverjanju pravilnosti vsebine pa se enak obseg samo prebere in primerja z vsebino bloka. Po koncu se registri nastavijo tako, kot bi se po vrnitvi iz podprograma ROMa, nato se izvajanje nadaljuje pri skupnem izhodu iz nalagalnega podprograma.

Tak pristop "hitrega nalaganja"vsebine kasetne datoteke ne emulira zvočnih impulzov kasete v realnem času, temveč posnema učinek standardnih podprogramov ROMa za nalaganje. Prednost takega pristopa je bistveno hitrejše in zanesljivejše nalaganje programov v emulatorju, predstavlja pa omejitev, da posebni nalagalniki, ki so odvisni od natančnega trajanja impulzov ali od neposrednega branja signala EAR, niso polno podprti.

7.4 Zvok

ZX Spectrum ima vgrajen le en piezoelektrični piskač oziroma enobitni zvočni izhod. Ta podpira izvajanje le enega tona naenkrat fiksne amplitude oziroma glasnosti, določimo pa tako dolžino izvajanja tona v sekundah kot frekvenco oziroma višino tona v poltonih nad srednjim C. Osrednja podprta frekvenca je srednji C s kodom 0, lahko pa pridobimo kateri koli polton nad ali pod to osrednjo frekvenco. Kodi tonov so vidni na sliki 7.12 [23, 24].



Slika 7.12: Shema tonov in pripadajočih kod tonov [23].

Na zvočnik lahko poleg ukaza BEEP prek podprogramov ROMa vplivamo tudi z ukazi OUT preko V/I vrat `0xFE`. Bit 4 v podatkovnem zlogu na vratih določa stanje zvočnika, pri čemer vrednost 0 pomeni tišino, 1 pa izvajanje tona. Ker lahko z enim bitom določamo le stanje proizvajanja zvoka, njegovo frekvenco določamo s hitrostjo preklapljanja stanja bita [14].

7.4.1 Implementacija v emulatorju

Zvočni del je vezan na pisanje v V/I vrata `0xFE`. Vsa pisanja na V/I vrata gredo skozi razred `IOPortSet`, ki zapisano vrednost posreduje objektu `ULABeeper`. Ob vsakem zapisu na V/I vrata `0xFE` primerja trenutno število izvajanih T-stanj procesorja s časom prejšnjega zapisa in iz razlike izračuna, koliko vzorcev mora zapisati v zvočni izhod, implementiran z neposredno zvočno linijo `SourceDataLine`. Kadar je bit 4 nastavljen, zapisuje vzorce z visoko amplitudo, sicer tišino. S tem se signal zvočnika gradi iz istega

časovnega modela, ki ga uporablja procesor.

Poglavje 8

Evalvacija emulatorja

Za raziskovanje delovanja računalnika ZX Spectrum in programiranje emulatorja je bilo potrebnih približno osem mesecev. Nastali emulator lahko ovrednotimo z vidika obsega izvorne kode, pravilnosti izvajanja programov za ZX Spectrum, uporabnosti razhroščevalnika in časovnih omejitev. Pri pravilnosti emulacije je pomembno, da se kot na izvornem računalniku programi pravilno naložijo, izvajajo pričakovane ukaze procesorja, izpisujejo pravilno sliko, proizvajajo pravilne zvoke in omogočajo uporabo tipkovnice. Pri notranjem vpogledu pa je pomembno, da uporabniški vmesnik prikazuje dejansko stanje sistema, ki ga emulator uporablja za izvajanje, in da posegi uporabnika v izvajanje vplivajo na izvajanje programov.

8.1 Obseg izvorne kode

Izvorna koda emulatorja je javno objavljena v repozitoriju GitHub [19]. V tabeli 8.1 je prikazan obseg celotnega repozitorija in glavnih delov aplikacijske kode.

Po uporabljenih programskih jezikih oziroma podpornih formatih izrazito prevladuje Kotlin kot osnovni programski jezik ogrodja Kotlin Multiplatform, v katerem je 511 datotek oziroma 70.727 vrstic kode, kar predstavlja približno 99,6 % vse kode v repozitoriju. Preostanek predstavljajo predvsem podporne

Del	Datotek	Vrstic	Vrstic kode	Komentarjev	Praznih vrstic
Celoten repozitorij	518	79.843	71.000	492	8.351
`composeApp/src`	511	79.275	70.667	330	8.278
`commonMain`	281	50.546	47.927	60	2.559
`commonTest`	214	28.344	22.425	270	5.649
`jvmMain`	5	279	233	0	46
Druge datoteke	11	106	82	0	24
Podporne datoteke	7	568	333	162	73

Tabela 8.1: Obseg izvorne kode razvitega emulatorja.

datoteke: zagonska skripta `gradlew` v jeziku Shell obsega 106 vrstic kode, `gradlew.bat` 73 vrstic kode, datoteka XML 44 vrstic kode, konfiguracija TOML 25 vrstic kode, datoteki HTML in CSS pa skupaj 19 vrstic kode.

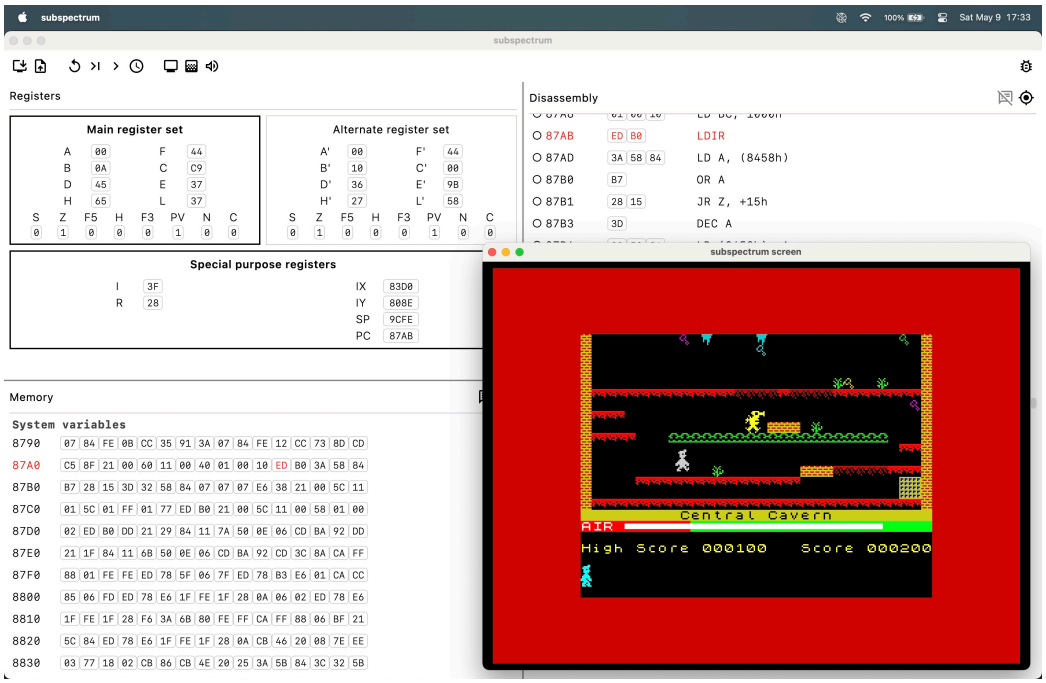
Tabela 8.2 prikazuje še razdelitev glavnih modulov skupne kode. Največji del predstavlja modul `base`, vendar je njegova velikost predvsem posledica velike količine podatkov anotacij ROMa in sistemskih delov pomnilnika, ne le izvajalne logike emulatorja. `ROMInstructionRegistry.kt` kot največja datoteka anotacij ROMa obsega 32.489 vrstic, `ROMSectionRegistry.kt` obsega 1.607 vrstic, `SystemVariableRegistry.kt` pa 351 vrstic kode.

Modul	Datotek	Vrstic	Kode
`base`	15	36.464	36.127
`proc`	233	11.403	9.355
`ui`	24	2.412	2.213
`units`	4	163	135

Tabela 8.2: Obseg glavnih modulov skupne kode emulatorja.

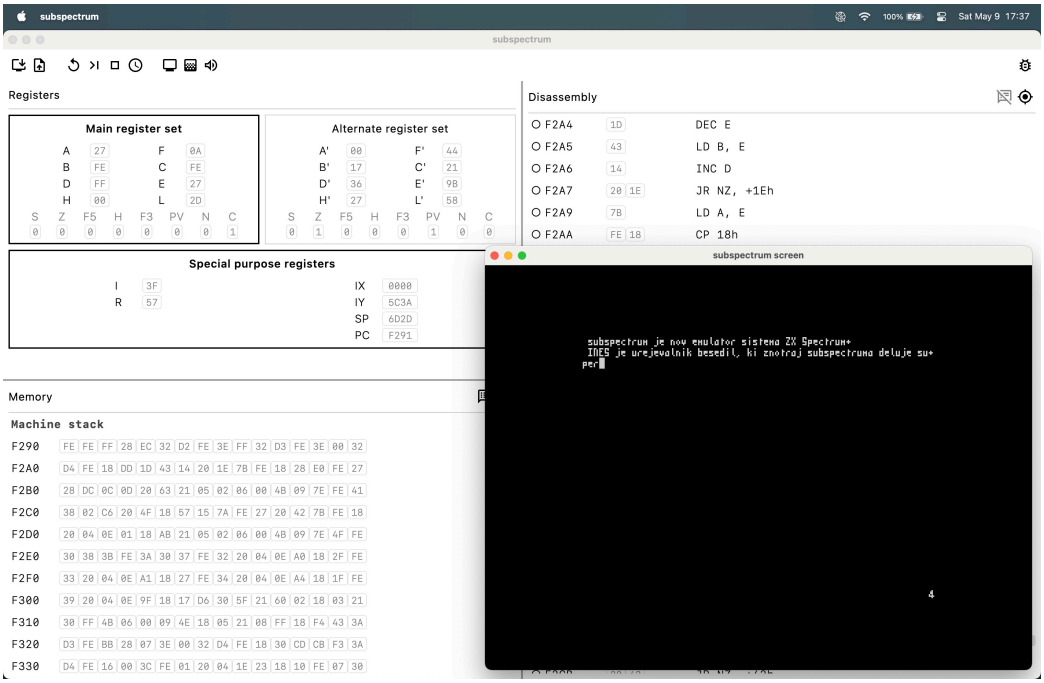
8.2 Pravilnost emulacije

Vsi trije izbrani programi se naložijo iz datotek posnetkov kaset ter delujejo brez opaznih napak, z delujočo sliko, zvokom in upravljanjem preko tipkovnice. Preizkušeni so bili popularna arkadna platformna igra Manic Miner, urejevalnik besedil INES ter slovenska besedilna pustolovska igra Kontrabant. Zaslonske slike delovanja emulatorja med izvajanjem izbranih programov vidimo na slikah 8.1, 8.2 in 8.3.



Slika 8.1: Posnetek zaslona emulatorja med izvajanjem igre Manic Miner.

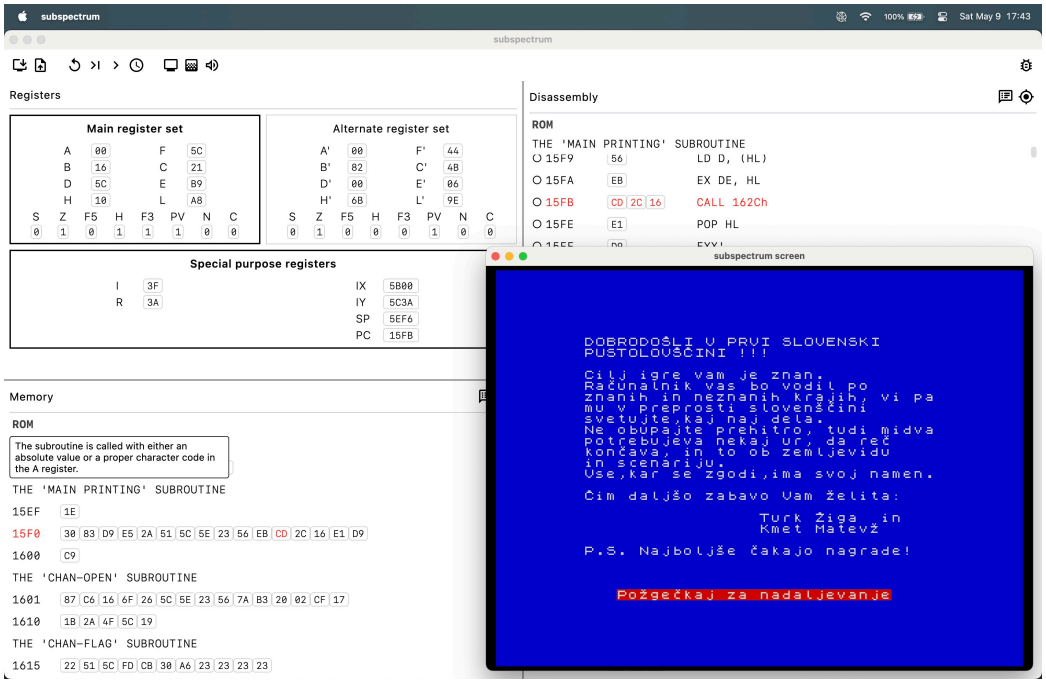
Pravilnost delovanja procesorja, natančneje implementacije ukazov in njihovih učinkov na registre, je bila preverjana tudi s programoma `zexdoc` in `zexall`, različicama preizkuševalnika ukazov Z80, prilagojenima za ZX Spectrum. `zexdoc` preverja dokumentirane zastavice, registre in pomnilnik, `zexall` pa poleg tega preverja tudi nedokumentirani zastavici XF in YF. Programa izvajata skupine ukazov z različnimi začetnimi stanji, iz končnega stanja izračunata kontrolno vsoto CRC in jo primerjata z referenčno vre-



Slika 8.2: Posnetek zaslona emulatorja med izvajanjem urejevalnika besedil INES.

dnostjo, pridobljeno na pravem procesorju Z80. Preizkus ne pove, katera podrobnost ukaza je napačna, temveč samo, da se rezultat razlikuje od referenčnega izvajanja. Izvajanje le prvih petih od 68 skupin ukazov testov programa `zexall` zaradi obsežnosti, kljub nastavitvi emulatorja na najhitrejšo možno hitrost emulacije procesorja na računalniku MacBook Pro M4 Pro s 24 GB pomnilnika, traja približno 10 minut, kar je prikazano na sliki 8.4 [1, 7].

Preizkusi z izbranimi programi preverjajo pravilnost vedenja celotnega sistema: nalaganje programa, izvajanje ukazov, spreminjanje pomnilnika, prikaz slike, uporabo tipkovnice in osnovni zvočni izhod. Preizkusa `zexdoc` in `zexall` pa sta ožje usmerjena v pravilnost ukazov CPE, predvsem v končne vrednosti registrov, zastavic in pomnilnika po izvedbi ukazov. Noben od teh preizkusov sam zase ne potrjuje popolne časovne natančnosti celotnega računalnika, potrjujeta pa, da so najpomembnejši funkcionalni deli emulatorja za izbrane programe izvedeni dovolj pravilno.

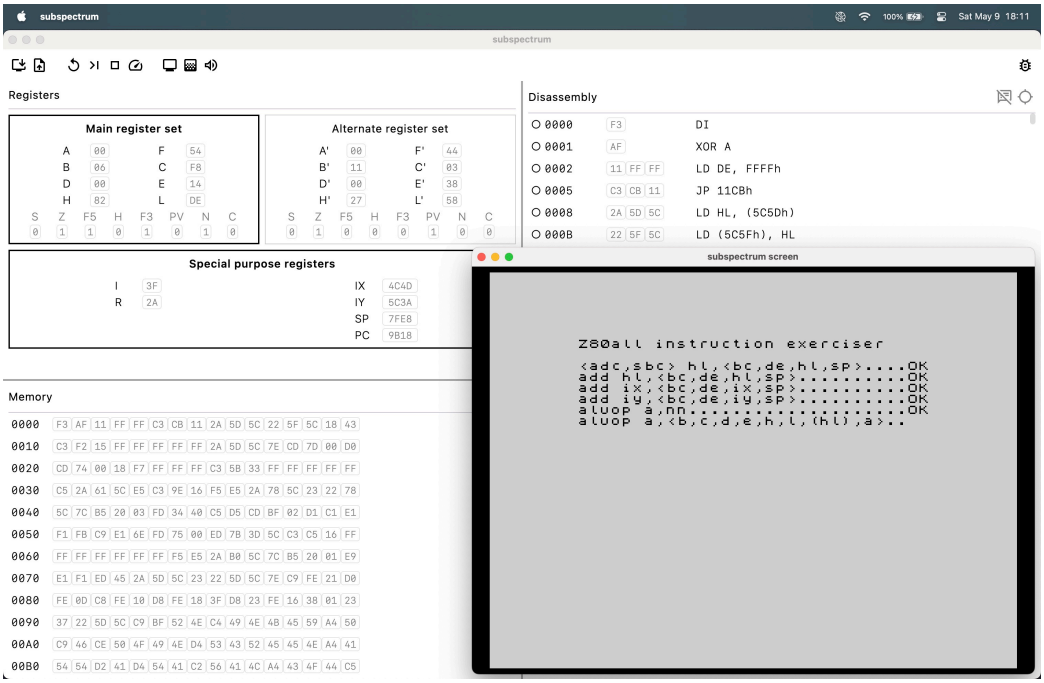


Slika 8.3: Posnetek zaslona emulatorja med izvajanjem igre Kontraband.

8.3 Notranji vpogled in razhroščevanje

Pomemben del evalvacije je tudi preverjanje vpogleda v stanje sistema, saj je razhroščevalnik eden od glavnih ciljev diplomskega dela. Med izvajanjem programov se vmesnik pravilno posodablja glede na stanje procesorja in pomnilnika. Podokno z registri prikazuje spremembe splošnih registrov, posebnih registrov in zastavic po posameznih ukazih, podokno s pomnilnikom prikazuje spremembe vsebine pomnilniških celic, razstavljeni ukazi pa sledijo vsebini pomnilnika. To omogoča opazovanje izvajanja programa na ravni, ki je pri uporabi pravega računalnika brez dodatne opreme težje dosegljiva.

Pravilnost razhroščevalnega vpogleda je bila preverjana tudi med ročnim korakanjem skozi ukaze in z uporabo prekinitvenih točk. Pri korakanju skozi ukaze se po vsakem izvedenem ukazu spremenijo registri in morebitne pomnilniške lokacije, ki jih je ukaz dosegel. Uporabnik lahko ob zaustavljenem procesorju spreminja vrednosti registrov, pomnilniških celic in ukaznih zlogov,



Slika 8.4: Posnetek zaslona emulatorja med izvajanjem testnega programa `zexall`.

te spremembe pa se upoštevajo pri nadaljnjem izvajanju. S tem emulator poleg izvajanja programov podpira tudi raziskovanje delovanja posameznih ukazov in celotnih programov ter delovanja osnovnega sistema ROMa.

Posebna vrednost razvitega emulatorja je v tem, da razhroščevalni vmesnik ni dodan le kot zunanje orodje za ustavljanje programa, temveč neposredno prikazuje podatke, ki jih uporablja emulacija. Uporabnik lahko opazuje registre in zastavice, pomnilnik, razstavljene ukaze, označena območja ROMa in sistemske spremenljivke. To je glavna razlika v primerjavi z emulatorji, ki so osredotočeni predvsem na igranje, najširšo združljivost ali udobno nalaganje programov.

8.4 Časovno delovanje in omejitve

Evalvacija pokaže tudi omejitve razvitega emulatorja. Najpomembnejša omejitev je, da časovni model temelji na T-stanjih izvedenih ukazov, emulacija pa ni popolnoma časovno natančna na ravni vseh posameznih dostopov do pomnilnika, V/I vrat in vezja ULA. Za običajne programe in obravnavane preizkuse to zadostuje, pri programih, ki so odvisni od zelo natančnega časovnega usklajevanja z ULA, od učinkov zadrževanja vodil ali od posebnih strojnih pojavov, pa bi lahko prišlo do razlik, čeprav te med testiranjem mnogih programov niso bile opažene. Podobno omejitev predstavlja nalaganje kaset. Nalaganje vsebine kasetnih datotek deluje preko "hitrega nalaganja", ki posnema učinek standardnega ROMovega podprograma za nalaganje, ne emulira pa celotnega zvočnega signala kasete in njegovega branja prek V/I vrat `0xFE`. Zato posebni nalagalniki in zaščite, ki neposredno merijo dolžine impulzov ali berejo signal EAR mimo standardnih ROMovih rutin, niso polno podprti, prav tako se pri nalaganju programov nalagalni zasloni ne vidijo, temveč delovanje skoči neposredno na izvajanje programa.

Poglavje 9

Zaključek

Cilj diplomskega dela je bil razviti delujoč emulator računalnika ZX Spectrum 48K, ki poleg zaganjanja programov omogoča tudi vpogled v stanje sistema in s tem služi kot razhroščevalno, učno in predstavitveno orodje. V okviru dela so bili predstavljeni zgodovinski pomen računalnika ZX Spectrum, njegova strojna zgradba in ključni deli sistema, nato pa še zasnova ter implementacija emulatorja. Posebej so bili obravnavani pomnilnik, procesor Z80, vezje ULA, zaslon, tipkovnica, kasetni podsistem in zvok, saj prav ti deli skupaj določajo obnašanje emuliranega računalnika.

Razviti emulator izpolnjuje glavne cilje diplomskega dela, saj omogoča nalaganje in izvajanje programov za ZX Spectrum 48K, prikaz slike, uporabo tipkovnice, osnovni zvočni izhod ter uporabo datotek posnetkov kaset. Implementacija procesorja Z80 podpira dokumentirane in nedokumentirane dele nabora ukazov, njena pravilnost pa je bila preverjana z namenskim testnim programom. Delovanje emulatorja je bilo dodatno ovrednoteno z izvajanjem izbranih programov in iger, ki so se pravilno naložili in uporabljali različne dele sistema.

Poleg same emulacije je bil dosežen tudi cilj vpogleda v notranje stanje sistema. Uporabniški vmesnik omogoča pregled registrov, zastavic, pomnilnika in razstavljenih ukazov, uporabnik pa lahko izvajanje zaustavi, izvede posamezen ukaz, uporabi prekinitvene točke ter po potrebi spremeni vredno-

sti registrov ali pomnilnika. S tem emulator ni le način poganjanja stare programske opreme, ampak tudi orodje za razumevanje, kako ti programi uporabljajo zmogljivosti procesorja, pomnilnika in vhodno-izhodnih naprav. Tak pristop je pomemben tudi za digitalno ohranjanje, saj ne ohranja samo možnosti izvajanja programov, temveč tudi znanje o njihovem delovanju.

Delo hkrati pokaže, da popolna emulacija računalnika ni odvisna samo od pravilne izvedbe ukazov procesorja. Za še natančnejšo emulacijo bi bilo treba izboljšati časovni model dostopov do pomnilnika, procesorja in ULA. S tem bi bilo mogoče bolje podpreti zadrževanje pomnilnika pri dostopih do zaslonskega dela pomnilnika, zadrževanje V/I operacij prek vrat `0xFE` glede na stanje naslovnega vodila ter strojne pojave, kot je učinek šneženja"na zaslonu. Druga pomembna nadgradnja bi bila prava emulacija nalaganja kaset prek zvočnih impulzov in branja signala prek V/I vrat `0xFE`, namesto trenutnega "hitrega nalaganja"prek ROMovega podprograma. Tak pristop bi povečal združljivost s posebnimi nalagalniki in zaščitami, vendar bi zahteval tudi natančnejši ter zanesljivejši časovni model, ki pa bi lahko bil tudi manj zanesljiv kot trenutna implementacija "hitrega nalaganja".

Možna izboljšava je tudi ponoven razmislek o izbrani tehnologiji, v kateri je emulator implementiran. Kotlin Multiplatform omogoča delitev emulacijske logike in uporabniškega vmesnika med platformami, vendar se je pri namizni razhroščevalni aplikaciji pokazalo, da medplatformnost ni nujno najpomembnejša lastnost. Ker je osrednja funkcionalnost projekta razhroščevanje, bi bilo pri nadaljnjem razvoju smiselno razmisliti tudi o tehnologijah, ki omogočajo več neposrednega nadzora nad namiznim okoljem in izrisovanjem uporabniškega vmesnika, na primer o ogrodju Electron in uporabi jezika TypeScript, pri katerem pa bi bilo treba preveriti hitrost delovanja emulacije napram drugim jezikom, ali pa ogrodji Dioxus oziroma Tauri in uporabo jezika Rust, pri katerem bi bilo treba preveriti prijaznost uporabe teh ogrodij za razvijalca.

Kljub navedenim omejitvam razviti emulator uspešno poveže delujočo emulacijo računalnika ZX Spectrum 48K z vpogledom v notranje stanje sistema. Rezultat dela je zato uporaben tako za izvajanje izbrane programske

opreme kot za raziskovanje delovanja računalnika, procesorja Z80 in programov, napisanih za računalnik ZX Spectrum 48K.

Literatura

[1] Jeff Avery. *Z80 emulation: Using Z80 Instruction Exerciser (Zexall/Zexdoc).* 2011. URL: https://jeffavery.ca/computers/macintosh_z80exerciser.html (pridobljeno 26. 4. 2026).

[2] Fabrice Bellard. „QEMU, a Fast and Portable Dynamic Translator“. V: *Proceedings of the 2005 USENIX Annual Technical Conference.* 2005, str. 41–46. URL: https://www.usenix.org/conference/2005-usenix-annual-technical-conference/qemu-fast-and-portable-dynamic-translator (pridobljeno 26. 5. 2026).

[3] Mike Dailly. *How Tape Loading Works.* 2024. URL: https://lemmings.info/how-tape-loading-works/ (pridobljeno 26. 4. 2026).

[4] Rodney Dale. *The Sinclair Story.* London: Duckworth, 1985.

[5] Guy Fernando. *C Programming on Sinclair ZX Machines.* Created May 2023, last modified Nov 2024. 2023. URL: https://www.i4cy.com/sinclair/ (pridobljeno 28. 4. 2026).

[6] Fuse emulator project. *Fuse - the Free Unix Spectrum Emulator.* URL: https://fuse-emulator.sourceforge.net/ (pridobljeno 27. 4. 2026).

[7] Jonathan Graham Harston. *Z80 Instruction Exerciser for the Spectrum.* URL: https://mdfs.net/Software/Z80/Exerciser/Spectrum/ (pridobljeno 26. 4. 2026).

[8] Primož Jakopin. „INES“. V: *Moj Mikro* 6 (1985), str. 67.

[9] Primož Jakopin. *Ines Steve Eva.* 2023. URL: https://jakopin.net/ines_steve_eva/index_en.php (pridobljeno 26. 4. 2026).

[10] Primož Jakopin. *O mikroračunalnikih in o INESu*. 2022. URL: https://www.jakopin.net/papers/memoirs/On_microcomputers_and_on_INES_si.php (pridobljeno 26. 4. 2026).

[11] Primož Jakopin. *On STEVE*. 2023. URL: https://www.jakopin.net/papers/memoirs/On_STEVE_en.php (pridobljeno 26. 4. 2026).

[12] Ivan Kanič. „INES, sistem za obdelavo besedil, slik in majhnih podatkovnih zbirk“. V: *Moj Mikro* 6 (1984), str. 10.

[13] Magnus Krook. *Home of SoftSpectrum 48*. 2019. URL: https://softspectrum48.weebly.com/ (pridobljeno 26. 4. 2026).

[14] Jaro Lajovic. *Strojni jezik za procesor Z80*. Ljubljana: Mladinska knjiga, 1985.

[15] Sinclair Research Limited. *ZX Spectrum 128 Introduction*. 1st. Sinclair Research Limited. Cambridge, 1986.

[16] Ian Logan in Frank O'Hara. *The Complete Spectrum ROM Disassembly*. 2012. URL: http://www.primrosebank.net/computers/zxspectrum/docs/CompleteSpectrumROMDisassemblyThe.pdf (pridobljeno 26. 4. 2026).

[17] Jurij Mihelič in Mojca Ferle. „Računalniška emulacija kot strategija digitalnega ohranjanja računalniških sistemov in programske opreme“. V: *Uporabna informatika* 29.4 (2021), str. 196–207. DOI: 10.31449/upinf.133.

[18] Jonathan Needle. *Spectaculator, Sinclair ZX Spectrum Emulator Home*. URL: https://www.spectaculator.com/spectaculator/ (pridobljeno 27. 4. 2026).

[19] Klemen Remec. *Subspectrum*. URL: https://github.com/kremec/subspectrum (pridobljeno 26. 5. 2026).

[20] Iztok Saje. „INES“. V: *Moj Mikro* 4 (1985), str. 16–17.

[21] SkoolKid. *Spectrum ROM: Memory map*. URL: https://skoolkid.github.io/rom/maps/all.html (pridobljeno 26. 4. 2026).

[22] James E. Smith in Ravi Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*. San Francisco: Morgan Kaufmann, 2005.

[23] Steven Vickers in Robin Bradbeer. *ZX Spectrum BASIC Programming*. 3rd. Sinclair Research Limited. Cambridge, 1983.

[24] Steven Vickers in Robin Bradbeer. *ZX Spectrum Introduction*. 1st. Sinclair Research Limited. Cambridge, 1982.

[25] Matt Westcott. *JSSpeccy 3*. URL: `https://github.com/gasman/jsspeccy3` (pridobljeno 27. 4. 2026).

[26] Wikipedia contributors. *ZX Spectrum*. URL: `https://en.wikipedia.org/wiki/ZX_Spectrum` (pridobljeno 28. 4. 2026).

[27] World of Spectrum. *Emulator File Formats*. URL: `https://worldofspectrum.org/faq/reference/formats.htm` (pridobljeno 22. 4. 2026).

[28] World of Spectrum. *TZX technical specifications*. 2006. URL: `https://worldofspectrum.net/TZXformat.html` (pridobljeno 22. 4. 2026).

[29] World of Spectrum. *Z80 File Format*. 2004. URL: `https://worldofspectrum.org/faq/reference/z80format.htm` (pridobljeno 26. 4. 2026).

[30] Sean Young. *The Undocumented Z80 Documented*. Version 0.91. 2005. URL: `http://www.z80.info/zip/z80-documented.pdf` (pridobljeno 26. 4. 2026).

[31] Zilog. *Z80 CPU User manual*. 2016. URL: `https://www.zilog.com/docs/z80/um0080.pdf` (pridobljeno 1. 2. 2026).