UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Horvat

Emulacija računalnika Iskra Delta Partner

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Jurij Mihelič

Ljubljana, 2017

Besedilo je bilo napisano z avtorjevim lastnim urejevalnikom besedila in pretvorjeno v zapis PDF z lastnim pretvornikom. Oba sta bila razvita v jeziku NewtonScript in izvedena z avtorjevo lastno implementacijo.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V 80. letih prejšnjega stoletja je bila slovenska proizvodnja računalnikov v razcvetu. Eden izmed računalnikov, ki se je tedaj izdeloval, je bil Partner podjetja Iskra Delta. Ni bilo izdelanih veliko, vseeno pa je eden izmed njih ohranjen v zbirki Fakultete za računalništvo in informatiko Univerze v Ljubljani. Njegovo nadaljnje izpostavljanje uporabi bi lahko pripeljalo do okvare, zato je potrebno izvesti ukrepe za ustrezno ohranitev.

V okviru diplomske naloge izvedite arhiviranje Partnerjeve programske opreme, vključno s pripadajočo dokumentacijo. Nato z obratnim inženiringom preštudirajte njegovo arhitekturo in izbrano programsko opremo. Na podlagi zbranih informacij izdelajte emulator računalnika, ki bo kar se da istovetno posnemal delovanje originala. Končni rezultat naj bo na voljo uporabnikom za preprosto uporabo na navadnem PC računalniku.

Zahvaljujem se:

- mentorju, ki je omogočil projekt in me usmerjal pri pisanju diplomske naloge,
- Janezu Kožuhu, ki je fakulteti doniral računalnik Partner,
- Tadeju Pečarju za odkrivanje skrivnosti disket in disketnih pogonov,
- doc. dr. Tomažu Dobravcu za nadomestni disketni pogon,
- izr. prof. dr. Veselku Guštinu za stik z Janezom Kožuhom,
- članom Laboratorija za algoritme in podatkovne strukture in Laboratorija za tehnologijo programske opreme – za prostor in potrpežljivost.

Kazalo

Seznam uporabljenih kratic

- ASCII · American standard code for information interchange · standardno ameriško kodiranje za izmenjavo informacij
- AVDC · advanced video display controller · napredni zaslonski krmilnik
- BCD · binary coded decimal · dvojiško kodirano desetiško število
- BDOS · basic disk operating system · osnovni diskovni operacijski sistem
- CCP · console command processor · konzolski ukazni procesor
- CTC · counter/timer channels · večkanalni števec/časovnik
- DCGG · display character and graphics generator · generator zaslonskih znakov in grafike
- DMA · direct memory access · neposredni dostop do pomnilnika
- DRAM · dynamic random access memory · dinamični pomnilnik
- EPROM · erasable programmable read-only memory · izbrisljivi programirljivi bralni pomnilnik
- GDP · graphic display processor · grafični procesor
- PIO · parallel input/output · vzporedni vhod/izhod
- RAM · random access memory · bralno-pisalni pomnilnik
- ROM · read-only memory · bralni pomnilnik
- SIO · serial input/output · zaporedni vhod/izhod
- TPA · transient program area · začasno programsko območje
- VAC · video attributes controller · krmilnik grafičnih atributov

Povzetek

Naslov: Emulacija računalnika Iskra Delta Partner

Avtor: Matej Horvat

V diplomski nalogi je predstavljen postopek razvoja emulatorja računalnika Partner, ki ga je proizvajalo podjetje Iskra Delta v 80. letih prejšnjega stoletja. Opisane so njegove glavne komponente, kako so emulirane, pa tudi njegova sistemska programska oprema in postopki, s katerimi je bila arhivirana.

Ključne besede: Iskra Delta, Partner, emulator, emulacija, obratni inženiring, arhiviranje, ohranitev, navidezni stroj, CP/M, Z80.

Abstract

Title: Emulation of the Iskra Delta Partner computer

Author: Matej Horvat

This diploma thesis presents the development process of a program that emulates a Partner computer, which was produced by the Slovenian/Yugoslav company Iskra Delta in the 1980s. It describes its main components and how they are emulated, as well as its system software and the methods and processes used to archive it.

Keywords: Iskra Delta, Partner, emulator, emulation, reverse engineering, archiving, preservation, virtual machine, CP/M, Z80.

Poglavje 1

Uvod

Slovensko oz. jugoslovansko podjetje Iskra Delta je v 80. letih prejšnjega stoletja proizvajalo tako namizne kot velike računalnike, namenjene poslovni rabi, in programsko opremo zanje [14]. V primerjavi s hišnimi računalniki tistega časa jih je uporabljalo relativno malo ljudi, zato je te računalnike in informacije o njih danes težko najti.

Eden izmed teh računalnikov je Partner. To je namizni računalnik, ki je bil prvič predstavljen leta 1983 [17] in mišljen kot mali poslovni oz. razvojni sistem [14]. Pozneje so se pojavili modeli, ki so lahko prikazovali tudi grafiko: 1F/G (z enim disketnim pogonom), 2F/G (z dvema disketnima pogonoma) in WF/G (s trdim diskom in enim disketnim pogonom – na sliki 1.1) [1]. (Skupaj se zanje uporablja tudi ime Partner GDP. Kdaj točno so bili ti modeli predstavljeni, ne vemo; preliminarna izdaja uporabniškega priročnika je iz marca 1987 [1], nekatere aplikacije zanje pa so datirane 1985 oz. 1986.)

Ker strojna oprema nima neskončne življenjske dobe, se pojavlja nevarnost, da v prihodnosti ne bo več mogoče v živo videti in uporabljati teh računalnikov in njihove programske opreme, pa tudi vedno težje bo najti ljudi, ki kaj vedo o njih. S tem bi izginil del slovenske zgodovine.

Kako ohraniti strojno opremo, je zapleten problem, ohraniti programsko opremo pa je relativno lahko. Za to poskrbi program, ki ga imenujemo emulator. Ta omogoča izvajanje programske opreme na drugi strojni opremi kot na tisti, za katero je bila prvotno razvita.

Emulirati pomeni implementirati vmesnik in funkcionalnost nekega sistema na sistemu z drugačnim vmesnikom in funkcionalnostjo [13]. Če to definicijo apliciramo na celoten računalniški sistem, kot je Partner, pomeni, da na nekem drugem (novejšem) računalniku implementiramo Partnerjev vmesnik (strojno kodo njegove centralne procesne enote) in funkcionalnost

(vhodno-izhodne naprave), da se lahko programska oprema, razvita za Partnerja, izvaja na novejšem računalniku.

Slika 1.1: Partner WF/G, uporabljen pri izdelavi diplomske naloge.

Emulator je sicer podzvrst navideznega stroja. Ti omogočajo izvajanje iste kode na različnih procesorskih arhitekturah in operacijskih sistemih, ne da bi jo bilo potrebno posebej prilagoditi. Ta koda ni nujno strojna koda nekega fizičnega procesorja; več programskih jezikov se zaradi prenosljivosti prevaja v kodo za navidezni stroj, ki se nato interpretira ali še enkrat prevede, da se lahko izvaja [13]. Vendar pa se izraz "emulator" navadno uporablja za navidezne stroje, ki poustvarjajo fizične računalnike – vključno z vhodno-izhodnimi napravami. To so celosistemski navidezni stroji (ang. whole-system virtual machines) [13].

Emulator deluje tako, da strojno kodo "gostujočega" programa izvaja tako, kot bi jo procesor izvirnega računalnika, in ko ta navidezni procesor poskuša komunicirati s strojno opremo, se na njegove ukaze odzove tako (oz. čim

bolj podobno), kot bi se prvotna strojna oprema. Program torej ne "ve", da se ne izvaja v svojem "domačem okolju". Če npr. uporabnik pritisne tipko na tipkovnici, bo program na to reagiral enako, kot bi na prvotni strojni opremi, če pa poskuša na zaslon izrisati sliko ali besedilo, pa bo uporabnik videl enak rezultat, kot bi ga na prvotni strojni opremi – v tem primeru na pravem računalniku Partner.

V tej diplomski nalogi si bomo ogledali razvoj programa PartEm, ki emulira Partnerja – natančneje model WF/G.

V drugem poglavju je predstavljenih nekaj glavnih značilnosti Partnerjeve strojne in programske opreme, potem pa so predstavljeni postopki, s katerimi smo Partnerjevo programsko opremo prenesli na sodoben računalnik, da smo sploh lahko začeli razvijati emulator.

V tretjem poglavju je predstavljeno, kako je potekal razvoj emulatorja, kakšna je njegova zgradba, katere so preostale komponente Partnerjeve strojne opreme, kako delujejo in kako so emulirane.

V četrtem poglavju si bomo ogledali končni izdelek, ga primerjali s pravim Partnerjem in navedli možne izboljšave.

Poglavje 2

Arhiviranje

Brez programske opreme, ki bi jo izvajali v emulatorju, bi emulator bil sam sebi namen, v vsakem primeru pa smo želeli arhivirati vse, kar smo imeli pri roki:

- Partnerjev uporabniški priročnik (tega smo skenirali; vseboval je veliko ključnih informacij o delovanju Partnerja),
- zagonsko disketo,
- trdi disk v Partnerju,
- Partnerjev bralni pomnilnik.

Predvsem vsebina trdega diska se nam je zdela koristna, saj bi tako lahko sproti preverjali natančnost emulatorja proti pravemu Partnerju. (Pozneje smo na njem odkrili tudi nekaj dokumentacije in izvorne kode; oboje nam je pomagalo pri projektu.)

Ti postopki so potekali sočasno z obratnim inženiringom. Posebej izdelava slike trdega diska in slike bralnega pomnilnika sta zahtevali nekaj predznanja o Partnerjevem procesorju, pomnilniku, operacijskem sistemu in vhodno-izhodnih napravah, zato jih bomo opisali v tem poglavju, njihovo emulacijo pa v naslednjem.

2.1 Centralna procesna enota

Partner kot svojo centralno procesno enoto uporablja mikroprocesor Zilog Z80A, ki ga poganja ura s frekvenco 4 MHz [1]. Z80 je 8-bitni mikroprocesor, ki ga je predstavilo ameriško podjetje Zilog leta 1976 [15]. Njegov nabor inštrukcij je nadmnožica tistega od Intelovega mikroprocesorja 8080A [16]. Z80A je nekoliko novejša verzija, ki je zmožna operirati pri višjih frekvencah.

Procesor ima dva nabora osmih 16-bitnih registrov oz. štirih registrskih parov,

ki so sestavljeni iz dveh 8-bitnih registrov, ki ju je mogoče neodvisno uporabljati [15, 16]:

- AF oz. A in F. A je akumulator; uporablja se kot implicitni vhodni in/ali izhodni operand za večino aritmetičnih operacij. F je register z zastavicami; spreminja se implicitno glede na aritmetične operacije in uporablja večinoma za pogojne skoke.
- BC oz. B in C. To sta splošnonamenska registra, za nekatere inštrukcije pa imata poseben pomen, npr. B je implicitni operand za inštrukcijo DJNZ (ang. decrement and jump if not zero), s katero se navadno implementirajo zanke, za katere je število iteracij vnaprej znano, BC pa služi kot števec za LDIR (ang. load, increment, repeat) in druge bločne inštrukcije.
- DE oz. D in E.
- HL oz. H in L. HL se večinoma uporablja kot kazalec.

Registrski par AF se zamenja z alternativnim z inštrukcijo EX AF, AF', ostali trije pa naenkrat z inštrukcijo EXX. Znotraj istega nabora registrov se lahko zamenjata tudi DE in HL z inštrukcijo EX DE, HL.

Obstajajo še sledeči 16-bitni registri, ki niso podvojeni:

- programski števec PC, ki ni neposredno dostopen,
- skladovni kazalec SP,
- indeksna registra IX in IY, ki se uporabljata kot bazna naslova pri načinih naslavljanja z 8-bitnim predznačenim odmikom.

Procesor ima 16-bitni pomnilniški naslovni prostor, ki ga zasedata RAM in ROM, in 8-bitni vhodno-izhodni naslovni prostor, ki ga zasedajo naprave.

Na prekinitve se lahko procesor odziva na tri načine. Vsi trije najprej shranijo PC na sklad, razlikujejo pa se v tem, na kaj ga nastavijo [15, 16]:

- V načinu 0 procesor prebere inštrukcijo s podatkovnega vodila. Navadno je to inštrukcija RST, ki nastavi PC na vrednost od 0 do 38h.
- V načinu 1 se PC vedno nastavi na 38h. Programska oprema mora z izpraševanjem ugotoviti, katera naprava je povzročila prekinitev.

- V načinu 2 procesor najprej prebere 8-bitni podatek s podatkovnega vodila in ustvari 16-bitni pomnilniški naslov tako, da zgornjih 8 bitov dobi iz posebnega registra I. Nato nastavi PC na 16-bitno besedo, ki jo prebere z dobljenega pomnilniškega naslova. Ker vsaka naprava na vodilo postavi drugo vrednost, se samodejno začne izvajati njena rutina za odzivanje na prekinitev. Ta način uporablja Partner; v njegovem priročniku so zapisani naslovi kazalcev na te rutine relativno na register I [1].

2.2 Operacijski sistem

Partner za svoj operacijski sistem uporablja CP/M 3 (imenovan tudi CP/M Plus), ki ga je razvilo ameriško podjetje Digital Research.

CP/M je enoopravilni diskovni operacijski sistem za mikroračunalnike s procesorjem Intel 8080 ali združljivim (npr. Zilog Z80). Programom zagotavlja funkcije za delo z datotekami in osnovne vhodno-izhodne funkcije za naprave [11].

Sestavljen je iz dveh delov. BIOS je del, ki se razlikuje glede na računalnik in ga priskrbi izdelovalec računalnika (v tem primeru Iskra Delta). Njegove funkcije kliče BDOS, ki je neodvisen od strojne opreme in implementira višjenivojske funkcije, njega pa kličejo uporabniški programi [11].

Partnerjev BIOS poleg vseh funkcij, ki jih zahteva CP/M, vsebuje še funkcijo za risanje grafičnih primitivov [1]. Ta funkcija obstaja samo na Partnerju in samo na grafični verziji. Kličejo jo aplikacije, da jim ni treba neposredno upravljati grafičnega procesorja.

Poleg naštetega vsebuje CP/M še ukazni interpreter (CCP) za zaganjanje programov in delo z datotekami in tudi druge programe, vendar ti niso stalno naloženi v pomnilniku, ampak se naložijo na isto mesto kot ostali uporabniški programi [11].

2.3 Delovni pomnilnik

Partner ima 125 kilobajtov delovnega pomnilnika (RAM). Ker lahko procesor s 16-bitnim naslovnim vodilom naenkrat naslavlja le 64 kilobajtov, je njegov pomnilniški naslovni prostor razdeljen na dva dela. Spodnjih 61 kilobajtov pripada trenutno izbrani izmed dveh t.i. bank, zgornji 3 kilobajti pa so stalni (ne pripadajo nobeni banki).

CP/M prvo banko uporablja zase, druga pa je na voljo aplikacijam (CP/M to območje imenuje TPA [11]). Skupni del pomnilnika je prav tako rezerviran za operacijski sistem; v njem so rutine za odzivanje na prekinitve in vstopne točke v različne funkcije operacijskega sistema, od katerih je večina v prvi banki.

2.4 Izdelava slike zagonske diskete

Poleg Partnerja in njegovega priročnika smo imeli na voljo tudi zagonsko disketo. Njeno vsebino smo želeli arhivirati, saj bi lahko bila uporabna za testiranje emulatorja, dokler ne bi naredili slike trdega diska.

Partner uporablja 5,25-palčne diskete visoke kapacitete. Format je drugačen kot pri drugih računalnikih, zato diskete ne bi mogli prebrati npr. na klonu PC AT zaradi drugačnega krmilnika. Partnerjev format je [1]:

- dvostranski,
- na vsaki strani je 73 sledi,
- v vsaki sledi je 18 sektorjev,
- v vsakem sektorju je 256 bajtov,
- sektorji so prepleteni (ang. interleaved) v razmerju 2:1 (vendar BIOS to prikrije),
- skupna kapaciteta je 657 kilobajtov.

Za branje smo uporabili krmilnik KryoFlux [4]. To je krmilnik, ki se priključi na sodoben računalnik preko vrat USB in lahko prebere "surovi" magnetni zapis

na disketi. Dekodiranje se opravi s programsko opremo; mi smo uporabili program HxCFloppyEmulator (sicer mišljen za uporabo z istoimensko napravo, ki nadomešča disketni pogon z uporabo USB ali kartice SD) [8]. Tako smo dobili sektorsko sliko, ki je primerna za analizo in uporabo v emulatorju. To je datoteka, ki vsebuje en bajt za vsak bajt vsakega sektorja na disketi (brez prepletanja). Programska oprema oz. oseba, ki dela s tako sliko, mora vnaprej poznati njene "dimenzije", a v večini primerov to ni potrebno, saj datotečni sistemi podatkovne nosilce organizirajo kot seznam enako velikih blokov.

Emulator smo začeli razvijati tako, da smo analizirali nalagalnik operacijskega sistema na disketi in ga poskušali "pripeljati čim dlje" – idealno do naloženega operacijskega sistema. Pozneje smo skozi podrobnejšo analizo sicer ugotovili, da je zagonska disketa namenjena starejši (negrafični) verziji Partnerja (dejanske fizične diskete nismo mogli preizkusiti zaradi težav z disketnim pogonom v našem Partnerju, zato tega nismo takoj vedeli). Vendar smo kmalu potem že imeli narejeno sliko trdega diska, s katero smo nadaljevali razvoj.

2.5 Izdelava slike trdega diska

Model WF/G vsebuje trdi disk Seagate ST-412 s kapaciteto približno 10 megabajtov [1]. V našem Partnerju je po 30 letih na srečo še vedno deloval in je vseboval veliko programske opreme in drugih zanimivih datotek (npr. datoteko z dokumentacijo o samem disku in njegovem krmilniku in izvorno kodo za eno od nameščenih aplikacij), zato smo želeli narediti kopijo podatkov na njem. (Koliko časa bo še deloval, ne moremo predvideti.)

Preprosto kopiranje vseh datotek (npr. na diskete) ne bi zadoščalo, ker:

- na ta način ne bi mogli skopirati nalagalnika operacijskega sistema; ta ni shranjen v datoteki, ampak v posebnem območju, ki zaseda prvih nekaj sektorjev diska,
- s tem ne bi dobili morebitnih izbrisanih datotek, ki operacijskemu sistemu niso več vidne, vendar njihovi fragmenti še vedno ležijo na sektorjih, ki niso

bili ponovno uporabljeni za kakšno drugo datoteko,
- bi za rabo v emulatorju morali rekonstruirati datotečni sistem; emulator namreč tako kot fizični računalnik ne ve ničesar o datotekah, temveč ima na voljo le (navidezni) disk, ki je množica sektorjev, njihov pomen pa določa operacijski sistem.

Poleg tega se je izkazalo tudi, da disketni pogon v našem Partnerju ni v delujočem stanju, računalnika pa nismo želeli razstavljati iz strahu, da bi ga poškodovali, oz. vsaj ne brez že narejene kopije podatkov na trdem disku.

Treba je bilo torej najti način, kako bi celotno vsebino trdega diska (torej od prvega do zadnjega sektorja) prenesli na novejši računalnik brez uporabe disket in, če je le možno, ne da bi zapisali karkoli novega na disk, saj bi s tem izgubili fragmente izbrisanih datotek. Po naključju smo na samem disku našli datoteko, ki je vsebovala veliko podatkov o njem in njegovem krmilniku; za nas je bila najbolj ključna kapaciteta: 1224 sledi, od katerih ima vsaka 32 sektorjev, ki so dolgi 256 bajtov.

Vse konfiguracije Partnerja imajo vgrajena vsaj ena serijska (RS-232) vrata, zato smo poskusili najti način, kako jih uporabiti za prenos podatkov. Sprva nismo niti vedeli, ali delujejo; vsakič, ko smo poskusili preusmeriti izhod kakšnega programa nanje (namesto na zaslon), kot je pisalo v priročniku, se je Partner nehal odzivati, na računalniku na drugi strani kabla pa nismo prejeli nobenega podatka ne glede na to, kako smo konfigurirali serijska vrata na obeh straneh.

Potem smo na trdem disku našli program GRMT20, ki je bil omenjen v priročniku (pod imenom RMT20 – sam program ob zagonu izpiše svoje ime kot "RMT 2.0") kot program, ki omogoča rabo Partnerja kot terminal za oddaljen računalnik. S tem programom smo lahko pošiljali podatke; če smo kaj natipkali na Partnerjevi tipkovnici, smo to videli na drugem računalniku, ne pa obratno. (Pri tem je bilo presenetljivo, da je bila hitrost prenosa enkrat višja, kot smo jo določili. BIOS dovoli izbiro hitrosti 1200, 2400 ali 4800 bitov na sekundo, v praksi pa je to 2400, 4800 ali 9600 bitov na sekundo. Priročnik glede dovoljenih hitrosti ni dosleden.) Vendar je bilo že to dovolj za naše potrebe.

V priročniku so bili podani naslovi vhodno-izhodnih naslovov serijskih vrat. Tudi tipkovnica je priključena skozi serijska vrata, vendar uporablja drugačen konektor [1]. Z analiziranjem kode na zagonski disketi, ki komunicira s tipkovnico, smo ugotovili, kako se pošiljajo podatki na serijska vrata. Odkrili smo, da če GRMT20 "nasilno" končamo (s kombinacijo tipk SHIFT in BRK; na Partnerju to prekine izvajajoči se program in naloži ukazni interpreter), serijska vrata ostanejo še vedno uporabna in lahko iz lastne kode pošiljamo podatke skozi njih.

S tem znanjem smo lahko na Partnerju napisali program v jeziku BASIC, ki je z uporabo funkcij, napisanih v zbirnem jeziku, prebral vsak sektor trdega diska in ga poslal preko serijskih vrat:

```
10 DEFINT A-Z              'Vse spremenljivke so celoštevilske.
20 A = &H7000              'Začetni naslov strojne kode.
30 DEF USR0 = A            'Vstopna točka za SETDMA.
31 DEF USR1 = A + 2        'Vstopna točka za SETTRK.
32 DEF USR2 = A + 4        'Vstopna točka za SETSEC.
33 DEF USR3 = A + 6        'Vstopna točka za READ.
34 DEF USR4 = A + 8        'Vstopna točka za PosljiSek.
39 I = A
40 READ B                  'Preberemo bajt iz ukaza DATA.
50 IF B = -1 THEN GOTO 90
60 POKE I, B               'Zapišemo ga.
70 I = I + 1
80 GOTO 40
90 Z = USR0(0)             'Kličemo SETDMA.
100 FOR T = 0 TO 1223      'Števec sledi.
110 POKE A + 10, T AND 255
115 POKE A + 11, T \ 256
120 Z = USR1(0)            'Kličemo SETTRK.
130 FOR S = 1 TO 32        'Števec sektorjev.
140 POKE A + 10, S
145 POKE A + 11, 0
150 Z = USR2(0)            'Kličemo SETSEC.
160 PRINT "Sled"; T; "sektor"; S
```

```
170 Z = USR3(0)              'Kličemo READ.
180 IF PEEK(A + 10) = 0 THEN GOTO 210
190 PRINT "Napaka pri branju!"
200 END
210 Z = USR4(0)              'Kličemo PosljiSek.
220 NEXT S
230 NEXT T
```

V vrsticah 30 do 34 določimo vstopne točke v strojno kodo. Vsaka kaže na inštrukcijo JR (relativni skok), ker je bilo tako lažje popravljati kodo.

V vrsticah 39 do 80 preberemo podatke iz ukazov DATA, ki vsebujejo strojno kodo (več o njih pozneje), in jih zapišemo na naslove od 7000h dalje.

V vrstici 90 kličemo zbirniško funkcijo SETDMA. Ta kliče BIOS funkcijo istega imena in ji posreduje naslov medpomnilnika, kamor bomo brali sektorje. Nato se začne zanka, ki iterira skozi sledi in sektorje (koliko jih je, smo izvedeli iz dokumentacije na trdem disku).

Znotraj zanke nastavimo sled in nato za vsak sektor v sledi izberemo tisti sektor, ga preberemo (če branje ne uspe, izpišemo napako) in pošljemo.

Branje in pošiljanje sektorja smo implementirali v zbirnem jeziku, saj BASIC ne omogoča dostopa do diska na nivoju sektorjev (v resnici bi lahko vse implementirali v zbirnem jeziku, vendar bi si s tem otežili razhroščevanje):

```
        org 7000h
        jr SETDMA       ; 7000h
        jr SETTRK       ; 7002h
        jr SETSEC       ; 7004h
        jr READ         ; 7006h
        jr PosljiSek    ; 7008h
Param   dw 0            ; 700Ah (prostor za parametre)
SETDMA  ld a, 12        ; SETDMA (nastavi...
        ld bc, Buffer   ; ... naslov medpomnilnika za sektorje)
        jr BIOS
```

```
SETTRK  ld a, 10          ; SETTRK (nastavi sled)
        jr BCBIOS
SETSEC  ld a, 11          ; SETSEC (nastavi sektor)
BCBIOS  ld bc, (Param)
BIOS    ld (.par), a      ; (številka BIOS funkcije)
        ld (.par+2), bc   ; Posredujemo parameter.
        ld c, 50          ; (BDOS funkcija "use the BIOS")
        ld de, .par
        jp 5              ; Kličemo BDOS in se vrnemo.
.par    db 0, 0, 0, 0     ; (prostor za parametre)
READ    ld a, 13          ; READ (prebere sektor)
        call BIOS
        ld (Param), a     ; Sporočimo, ali je uspelo ali ne.
        ret
PosljiSek ld hl, Buffer
        ld b, 0           ; (256 ponovitev notranje zanke)
        ld c, b           ; Kontrolna vsota := 0.
.nasl   ld a, (hl)        ; Preberemo bajt iz medpomnilnika...
        call PosljiBajt   ; ... in ga pošljemo.
        add a, c          ; Prištejemo h kontrolni vsoti.
        ld c, a
        inc hl
        djnz .nasl
        ld a, c           ; Pošljemo kontrolno vsoto.
PosljiBajt push bc
        ld b, 3           ; (število ponovitev)
.znova  push af
.cakaj  in a, (0DBh)
        and 4             ; (zastavica za pripravljenost)
        jr z, .cakaj      ; Čakamo, dokler SIO ni pripravljen.
        pop af
        out (0DAh), a     ; Pošljemo bajt na vrata LPT.
        djnz .znova
        pop bc
        ret
Buffer                    ; (medpomnilnik)
```

Večina rutin le kliče BIOS funkcije. To storijo tako, da register A nastavijo na številko funkcije [7], nato pa pokličejo ali skočijo (če jim ni treba storiti ničesar več – če zaporedje CALL in RET spremenimo v JR, prihranimo dva bajta, torej pretipkavanje traja manj časa) v rutino BIOS oz. BCBIOS (slednja še prebere parameter z naslova 700Ah). V vsakem primeru se klic BIOS funkcije izvede z BDOS funkcijo, ki ji je treba posredovati številko BIOS funkcije in parametre zanjo v določeni strukturi. Aplikacije namreč ne morejo klicati BIOS funkcij neposredno, ker niso v isti banki [8]. BDOS funkcije se kličejo preko naslova 5; na tistem naslovu je skok v BDOS (ta se lahko razlikuje glede na računalnik in verzijo CP/M, na naslovu 5 pa je dokumentirana vstopna točka [8]).

Idealno bi za pošiljanje podatkov uporabili že obstoječ protokol, npr. XMODEM, ki je preprost in široko podprt med terminalskimi programi, vendar mora pri njem prejemnik preveriti kontrolno vsoto vsakega poslanega paketa in pošiljatelju sporočiti, če se ta ne ujema s podatki. Ker na našem Partnerju nismo mogli prejemati nobenih podatkov, smo se raje odločili za lasten počasnejši protokol, ki je za vsak bajt vsakega sektorja poslal trikrat, izračunal kontrolno vsoto in tudi to poslal trikrat. Na računalniku, ki je podatke prejel, smo potem napisali program, ki je ta zapis dekodiral z "glasovanjem" (če se vsaj dva bajta v neki trojici ujemata, se uporabi tista vrednost, drugače je prišlo do napake). Pošiljanje celotne vsebine trdega diska je tako trajalo cel dan (v resnici več zaradi razhroščevanja). Verjetno bi se to dalo opraviti veliko hitreje, vendar raje nismo tvegali.

Pošiljanje sektorja (vsak je dolg 256 bajtov) poteka tako, da se sprehodimo skozi medpomnilnik in za vsak bajt kličemo rutino, ki ga pošlje trikrat. Vmes še prištejemo bajt h kontrolni vsoti in na koncu sektorja še to pošljemo trikrat.

Pošiljanje enega bajta skozi serijska vrata se opravi s pisanjem na njihov podatkovni naslov (0DAh), še prej pa je treba počakati, da je SIO na to pripravljen. To naredimo tako, da v zanki preverjamo zastavico za pripravljenost, ki jo preberemo s statusnega naslova (0DBh).

Zbirniško kodo smo napisali in prevedli na novejšem računalniku, nato pa smo

jo pretipkali na Partnerja v obliki ukazov DATA, s katerimi se v jeziku BASIC v program vstavi vnaprej pripravljene podatke:

```
240 DATA 24, 10, 24, 15, 24, 17, 24, 40
250 DATA 24, 47, 0, 0, 62, 12, 1, 90
260 DATA 112, 24, 10, 62, 10, 24, 2, 62
270 DATA 11, 237, 75, 10, 112, 50, 44, 112
280 DATA 237, 67, 46, 112, 14, 50, 17, 44
290 DATA 112, 195, 5, 0, 0, 0, 0, 0
300 DATA 62, 13, 205, 29, 112, 50, 10, 112
310 DATA 201, 33, 90, 112, 6, 0, 72, 126
320 DATA 205, 73, 112, 129, 79, 35, 16, 247
330 DATA 121, 197, 6, 3, 245, 219, 219, 230
340 DATA 4, 40, 250, 241, 211, 218, 16, 244
350 DATA 193, 201, -1
```

Ko je bila slika narejena, smo jo lahko uporabili v emulatorju. Napisali smo tudi program, ki je iz nje prebral vse datoteke (za lažje pregledovanje podatkov s sodobnimi programi), vključno z izbrisanimi – v njihovem primeru je bilo to možno le, če ni bil prostor, ki so ga zasedale, ponovno uporabljen za novejše datoteke. Nekaj datotek smo lahko dobili s hevurističnim iskanjem po nerabljenih blokih datotečnega sistema, na katere ni bilo več nobenih kazalcev v imeniški strukturi. Med izbrisanimi datotekami je bilo nekaj programov, izvorne kode in dokumentacije.

2.6 Izdelava slike bralnega pomnilnika

V bralnem pomnilniku tipa EPROM [1] je shranjen začetni nalagalnik, ki ob vključitvi računalnika preveri delovanje delovnega pomnilnika in naloži nalagalnik operacijskega sistema z diskete ali trdega diska. Ta je shranjen na začetnih sektorjih in naloži operacijski sistem.

Ob resetu računalnika se bralni pomnilnik vklopi in zaseda prvih 8 kilobajtov naslovnega prostora ne glede na izbrano banko. Izklopi se z dostopom do vhodno-izhodnega naslova 80h (tisti del pomnilniškega naslovnega prostora,

ki ga je zasedal, potem zasede RAM), ni pa ga mogoče nazaj vklopiti brez resetiranja celotnega računalnika.

Predvidevali smo, da ga izklopi operacijski sistem oz. njegov nalagalnik. Ta teorija ni bila napačna, saj smo pri pregledovanju strojne kode opazili, da nalagalnik operacijskega sistema na zagonski disketi izklopi ROM na samem začetku svojega izvajanja, na trdem disku pa ne, vendar tam to opravi operacijski sistem, ko je naložen, tik preden začne inicializirati strojno opremo. V vsakem primeru je rezultat enak; ko je operacijski sistem naložen in lahko v njem izvajamo programe, je že prepozno, da bi naredili sliko bralnega pomnilnika.

Sklepali smo torej, da lahko sliko naredimo le tako, da ne zaženemo operacijskega sistema, ampak namesto njega program, ki prebere ROM in njegovo vsebino pošlje po serijskih vratih. Ker nismo želeli izgubiti možnosti zagona operacijskega sistema s trdega diska, je ostala le še ena možnost: da zamenjamo disketni pogon in ustvarimo svojo zagonsko disketo, ki bi vsebovala ta program.

To sprva ni uspelo, ker Partner ni prepoznal, da je disketa zagonska. Zato smo naredili kopijo zagonske diskete in "povozili" del nalagalnika operacijskega sistema s svojim programom, ki je poslal sliko celotnega naslovnega prostora (ker nismo vedeli, katere naslove zaseda ROM) skozi serijska vrata na enak način kot program za branje vsebine trdega diska (za inicializacijo serijskih vrat smo zagnali program GRMT20, nato pa resetirali Partnerja s stikalom na zadnji strani).

Večina nastale slike pomnilnika je bila "prazna" – vsebovala je vzorec "55 AA", ki je rezultat preizkusa delovnega pomnilnika. Ponekod so ostali še podatki izpred reseta. Bralnega pomnilnika očitno ni bilo (prepoznali bi ga po kateremkoli izmed sporočil, ki jih izpiše na zaslon), opazili pa smo sledeči fragment kode:

```
F600    in a, (80h)
F602    ld hl, 0E000h
F605    ld de, 100h
```

```
F608    ld bc, 1600h
F60B    ldir
F60D    jp 100h
```

Ta fragment z branjem vhodno-izhodnega naslova 80h izklopi ROM, nato pa kopira 1600h bajtov z naslova 0E000h na 100h in skoči tja. Vedeli smo, da se nalagalnik operacijskega sistema, ko je naložen, začne na tej lokaciji, zato smo sklepali, da ROM naloži nalagalnik na naslov 0E000h in zgornji fragment kode na 0F600h, nato pa ga izvede. Takoj na naslov 100h ga ne bi mogel naložiti; procesor namreč ob vklopu začne izvajati kodo z naslova 0, zato se mora ROM začeti na tem naslovu, v Partnerjevem priročniku pa piše, da je velik 4 kilobajte [1], zato naslov 100h zagotovo leži v njegovem območju.

Potem smo opazili, da je ta fragment prisoten na zagonski disketi in na trdem disku na enaki lokaciji. Sklepali smo, da je pravzaprav to vstopna točka v nalagalnik. Program za branje naslovnega prostora smo premaknili na to lokacijo (pri tem smo povozili inštrukcijo, ki izklopi ROM) in ga še enkrat pognali. Nova slika je vsebovala tudi ROM. To je končna verzija programa:

```
            org 0F600h
Zacetek     di                  ; Onemogočimo prekinitve.
            ld sp, 0FF00h       ; SP := vrh našega prostora.
            call Zapiskaj       ; Obvestimo uporabnika o začetku.
            call PosljiBan      ; Na začetku je izbrana prva banka.
            ld d, 0Ch           ; (3 kilobajti)
            ld hl, 0F400h       ; (začetek skupnega območja)
            call PosljiOb       ; Pošljemo skupno območje.
            in a, (90h)         ; Izberemo drugo banko...
            call PosljiBan      ; ... in jo pošljemo.
            call Zapiskaj       ; Obvestimo uporabnika o koncu.
            halt                ; Ustavi procesor do reseta.
PosljiBan   ld d, 0F4h          ; (61 kilobajtov)
            ld h, 0
            ld l, h             ; HL := 0.
PosljiOb    call PosljiSek      ; Pošljemo "sektor" s kontrolno vsoto.
            dec d               ; To ponovimo D-krat.
```

```
            jr nz, PosljiOb
            ret
Zapiskaj    in a, (0D9h)
            and 4               ; (zastavica za pripravljenost)
            jr z, Zapiskaj
            out (0D8h), a       ; (A je zdaj 4, kar je tudi koda za dolg pisk)
            ret
```

Opomba: rutini PosljiSek in PosljiBajt sta enaki kot v prejšnjem programu.

Nadomestni disketni pogon smo našli šele nekaj mesecev po začetku projekta; vsi, ki smo jih pred njim preizkusili, niso delovali ali pa niso bili združljivi s Partnerjem. Emulator je torej že na začetku kar sam naložil nalagalnik operacijskega sistema v RAM na naslov 100h in tudi drugače opravil potrebno inicializacijo emulirane strojne opreme. Bralni pomnilnik torej sploh ni nujno potreben, vendar smo ga želeli arhivirati in uporabiti v emulatorju zaradi pristnosti, saj ob zagonu z velikimi črkami izpiše "Delta Partner GDP". Ta napis je prikazan na sliki 2.1.

Slika 2.1: Besedilo, ki ga ROM izpiše na zaslon ob zagonu računalnika.

Poglavje 3

Emulator

V tem poglavju so opisani preostali sestavni deli Partnerja in kako so implementirani v emulatorju.

3.1 Postopek razvoja

Razvoj emulatorja je potekal iterativno. Ko smo imeli narejeno sliko zagonske diskete in nato sliko trdega diska, je bil naš prvi cilj spraviti emulator do faze, kjer bi lahko naložil operacijski sistem (CP/M).

Pri tem smo analizirali njegov nalagalnik, za katerega je izvorna koda javno dostopna [10]. Strojno opremo smo sprva emulirali le, kolikor je bilo potrebno, da je postopek nalaganja in inicializiranja prišel čim dlje; če je npr. kakšen del kode v zanki čakal, da bo neka naprava pripravljena, smo preprosto vedno vrnili rezultat, s katerim je bil "zadovoljen". Emulaciji krmilnika disketnega pogona in krmilnika trdega diska smo se povsem izognili s prestrezanjem BIOS funkcij.

Po približno mesecu dela, med katerim smo tudi analizirali sliko zagonske diskete, izvajali poskuse na pravem Partnerju (da bi ugotovili, kako deluje strojna oprema) in ugotavljali, kako bi naredili sliko trdega diska, je emulator prišel do faze, kjer je lahko naložil operacijski sistem. Takrat je naš cilj postal čim bolj natančno emulirati vse sestavne dele Partnerja.

Nekatere informacije o strojni opremi smo dobili iz Partnerjevega priročnika (brez njega razvoj emulatorja ne bi bil mogoč), v primeru grafičnega vezja pa iz dokumentacije grafičnih krmilnikov. Vse ostalo je bilo treba odkriti empirično: da smo ugotovili, kako nek kos strojne opreme deluje, smo navadno postavili hipotezo, nato pa v programskem jeziku BASIC napisali kratek program, s katerim smo jo preverili. Rezultate smo nato zabeležili in enako oz. čim bolj podobno obnašanje implementirali v emulatorju. Če

dokumentacija ni bila razumljiva, je bilo velikokrat koristno pognati program v emulatorju in zraven opazovati, kako krmili strojno opremo in kakšne podatke od nje pričakuje. Le pri emulaciji miške smo si lahko pomagali z izvorno kodo aplikacije in tako implementirali emulacijo naprave, ki je niti nismo imeli in je sicer ne bi mogli emulirati.

3.2 Splošna zgradba emulatorja

Strojna oprema deluje sočasno in svoje delovanje usklajuje in nadzoruje z različnimi kontrolnimi signali. Koda, ki sestavlja nek program (v tem primeru emulator), pa se izvaja zaporedno. Kot je bilo omenjeno v uvodu, je emulator najbolj smiselno gledati s perspektive procesorja. Ta izvaja kodo, ko pa je treba izvesti vhodno-izhodno operacijo, pa izvede to, nato nadaljuje s kodo. Taka zanka je torej na najvišjem nivoju emulatorja. Naprave, s katerimi dela človek (tipkovnica, miška, zaslon), pa zahtevajo posebno obravnavo. Oglejmo si najprej zaslon.

Na pravem Partnerju se slika na zaslonu osvežuje sočasno z delovanjem procesorja, v emulatorju pa se zaradi enostavnosti emulacija procesorja in izrisovanje izvajata izmenično. V vsaki iteraciji glavne zanke se izvede 80000 procesorskih ciklov (to je rezultat deljenja frekvence procesorja, 4 MHz, s frekvenco osveževanja zaslona, 50 Hz), nato se izriše slika, če se je spremenila. Sprožijo se tudi morebitne čakajoče prekinitve. Potem emulator čaka 20 ms (1/50 s) ali manj (odvisno od tega, kako dolgo je trajalo izrisovanje slike) in zanka se ponovi. Kot stranski učinek v nekem času emulirani procesor izvede približno toliko inštrukcij kot v pravem Partnerju (približno zato, ker ne upoštevamo čakalnih stanj, ki jih zahteva delovni pomnilnik, ki je tipa DRAM).

Hkrati emulator v isti zanki preverja, ali je bila pritisnjena tipka na tipkovnici, ali je bil pritisnjen ali spuščen gumb na miški in ali je bila miška premaknjena. Te dogodke posreduje ustreznemu modulu, ki ustrezno posodobi stanje naprave, ki bo programski opremi vidno, ko ga bo naslednjič prebrala, in v primeru tipkovnice sproži prekinitev, da obvesti programsko opremo o pritisku tipke.

3.3 Centralna procesna enota

Za emulacijo procesorja se uporablja knjižnica Zymosis [5], ki smo jo nekoliko prilagodili za večjo prenosljivost. To je knjižnica, ki natančno emulira procesor Z80, vključno z nedokumentiranimi inštrukcijami in drugimi podrobnostmi.

Emulirani procesor se z ostalimi napravami povezuje s povratnimi klici (ang. callback). Ko mora procesor dostopati do pomnilnika ali naprave, pokliče podano funkcijo za bralno oz. pisalno operacijo. Za pomnilnik (RAM in ROM) skrbi en par funkcij, za naprave pa se zahteva po branju ali pisanju preusmeri na modul, ki implementira tisto napravo. Vsaka naprava implementira vsaj sledeče funkcije:

- Funkcijo za ponastavitev. Ta se pokliče ob začetku emulacije in vsakič, ko je emulirani računalnik resetiran.
- Funkcijo za branje podatkov. Ta vzame naslov vrat in vrne vrednost.
- Funkcijo za pisanje podatkov. Ta vzame naslov vrat in vrednost. Pri večini naprav tudi sproži stranske učinke.

Naprave lahko tudi sprožajo prekinitve. Ker je vsaka naprava drugačne narave, za to ni enotnega mehanizma.

3.4 Pomnilnik

V emulatorju je pomnilnik (tako RAM kot ROM) le polje, do katerega emulirani procesor dostopa preko povratnih klicev. Ob vsakem dostopu se naslov prevede iz naslovnega prostora emuliranega procesorja v naslovni prostor gostitelja, nato pa se izvede bralna oz. pisalna operacija.

3.5 Grafično vezje

Za sliko, ki se prikaže na zaslonu, skrbita dva čipa. V grobem lahko rečemo, da je en zadolžen za besedilo, drug pa za grafiko.

Zaslon, ki je vgrajen v Partnerja, je monokromatski in zelene barve. Nekatere konfiguracije Partnerja omogočajo tudi prikaz slike na televizijskem zaslonu [1]. Naš te možnosti nima, zato emulator sliko prikazuje tako, kot bi jo vgrajeni zaslon.

3.5.1 AVDC

AVDC (Signetics SCN2674) krmili zaslon in prikazuje besedilo.

Uporablja se lahko na različne načine, odvisno od tega, kako je povezan s preostankom sistema [2]. V Partnerju se uporablja t.i. samostojni način (ang. independent mode), v katerem ima AVDC svoj pomnilnik, do katerega glavni procesor ne more dostopati neposredno, ampak ga upravlja s pošiljanjem ukazov.

Pomnilnik je sestavljen iz dveh delov. Vsak je velik 4 kilobajte. Eden vsebuje kode znakov, drugi pa njihove atribute [1]. Zapisovanje in branje poteka na obeh delih hkrati.

Nekateri izmed ukazov so:

- izbira inicializacijskega registra,
- vklop oz. izklop kurzorja,
- vklop oz. izklop prekinitev,
- zapis znaka in atributa na naslovu kurzorja,
- premik kurzorja na naslednji naslov,
- zapis več enakih znakov in atributov.

Lastnosti slike, ki jo generira AVDC, se nastavijo s 15 inicializacijskimi registri. Vsi se nastavljajo skozi isti vhodno-izhodni naslov, izbirajo pa se s pošiljanjem ustreznega ukaza. Po zapisu vsakega od njih se samodejno izbere naslednji, tako da lahko programska oprema nastavi vse naenkrat (potem jih navadno več ne spreminja). Nekatere lastnosti, ki se lahko nastavijo, so:

- Način dela. Na Partnerju je to vedno samostojni način.
- Velikost znakov. Na Partnerju je to vedno 8×11 [1].

- Dolžine raznih intervalov, s katerimi se krmili zaslon.
- Ali kurzor utripa, in če utripa, kako hitro.
- Hitrost utripanja znakov, ki imajo nastavljen atribut utripanja.
- Število znakov v vrstici. Na Partnerju lahko uporabnik izbere 80 ali 132 znakov.
- Število vrstic na zaslonu. Na Partnerju je to vedno 26.
- Višina kurzorja.
- Katera vrstica znaka se v celoti prižge, če ima znak nastavljen atribut podčrtavanja.

Obstajajo tudi drugi registri, ki imajo svoje vhodno-izhodne naslove, npr. naslov kurzorja in registri, ki sliko razdelijo na več delov (Partnerjev BIOS tega ne uporablja).

Pri generiranju slike AVDC lahko bere znake na dva načina:

- V linearnem načinu AVDC bere znake iz pomnilnika po vrsti in jih v istem vrstnem redu prikazuje (od leve proti desni).
- V načinu vrstične tabele (ang. row table mode) programska oprema nek del znakovnega pomnilnika rezervira za tabelo, ki za vsako vrstico na zaslonu vsebuje naslov vrstice v pomnilniku AVDC. Naslov te tabele je shranjen v registru. Za prikaz vsake vrstice AVDC najprej prebere njen naslov, nato pa od tam bere znake po vrsti. Partnerjev BIOS uporablja ta način, saj ima dve prednosti: pomikanje vrstic je enostavno (treba je le zamenjati njihove naslove), omogoča pa tudi preklapljanje med 80-stolpčnim in 132-stolpčnim načinom brez težav; BIOS vrstice v pomnilniku AVDC vedno organizira tako, da je vsaka dolga 132 znakov. V 80-stolpčnem načinu preprosto ne uporablja preostalih stolpcev, AVDC pa jih pri prikazu ignorira. Če uporabnik nato preklopi v 132-stolpčni način, vsi znaki ostanejo na svojem mestu, le da vrstice postanejo daljše.

AVDC ne generira slike popolnoma sam; za to potrebuje še DCGG in VAC [2]. Ta dva čipa procesorju nista vidna, zato ju ni treba posebej emulirati; pomembno je le, da je končni rezultat čim bolj pravilen. Emulator njune funkcije implementira v istem modulu kot AVDC. V Partnerjevi dokumentaciji ta dva čipa nista omenjena, sta pa omenjena v dokumentaciji AVDC.

Generiranje slike poteka približno takole:

1. Slika je sestavljena iz več vrstic besedila, vsaka od njih pa iz več vrstic pikslov. Ker se vrstice na zaslon izrisujejo od vrha proti dnu in vsaka vrstica od leve proti desni, AVDC v tem vrstnem redu iz svojega pomnilnika prebira znake, ki sestavljajo vrstico, in njihove atribute. (Ker je vsaka vrstica znakov sestavljena iz več vrstic pikslov, mora vsako vrstico znakov iz pomnilnika prebrati večkrat. V emulatorju to ne drži, saj emulator nima neposrednega dostopa do strojne opreme gostitelja, ampak sliko izriše v celoti in šele nato posreduje na zaslon.)
2. Kode znakov, ki jih AVDC prebere iz pomnilnika, prejme DCGG. Ta ima svoj ROM z oblikami znakov, ki so prikazane na sliki 3.1, in (konceptualno) proizvede sliko 1-bitnih pikslov.
3. VAC od AVDC prejme atribute znakov in na podlagi njih določi barve pikslov.

Vsak znak v atributnem pomnilniku zaseda en bajt (8 bitov). Atributi vsakega znaka so poljubna kombinacija teh bitov. To so:

- bit za utripanje,
- bit za podčrtavanje,
- bit za močnejšo osvetlitev,
- trije biti za barvo ozadja (na Partnerjevem zaslonu proizvedejo različne odtenke zelene; ni znano, kako bi barve izgledale na televizijskem zaslonu),
- dva bita, katerih funkcija ni povsem razumljena.

Vsaki vrstici je mogoče dodeliti tudi atribut, ki povzroči, da so znaki v njej dvojno široki ali dvojno široki in visoki. V obeh primerih je treba v pomnilnik zapisati vsak znak dvakrat; npr. "beseda" postane "bbeesseeddaa", saj AVDC še vedno bere znake na isti način, le izriše se njihova leva ali desna polovica, odvisno od tega, ali je znak v lihem ali sodem stolpcu. V primeru vrstic dvojne višine je treba to ponoviti še pri naslednji vrstici in določiti, katera je zgornja in katera je spodnja polovica. Vsi ti atributi se določijo z zgornjima dvema bitoma naslova vrstice v vrstični tabeli.

V emulatorju smo seveda želeli pri izrisu slike AVDC uporabiti isto pisavo kot

Partner, toda ker ROM, v katerem je shranjena, procesorju ni dostopen, je nismo mogli dobiti drugače, kot da smo jo prerisali. Da smo lažje videli posamezne piksle, smo napisali program, ki je izpisal vseh 256 znakov v dvojni širini in višini. Slika 3.1 prikazuje celoten nabor znakov. Partnerjev BIOS pri izpisovanju znakov sicer upošteva le spodnjih 7 bitov; kateri znak predstavljajo, je odvisno od trenutno izbranega nabora (poleg jugoslovanskega in ameriškega jih je še enajst; od teh so trije nedokumentirani). BIOS interpretira tudi ubežna zaporedja (ang. escape sequence); nekatera izmed njih omogočajo menjavo nabora, druga pa spreminjajo atribute nadaljnjih znakov, premikajo kurzor itd.

Mogoče naj bi bilo tudi definiranje lastnih znakov [1], vendar nismo našli nobene dokumentacije, ki bi razložila, kako, niti nismo do časa pisanja tega besedila tega ugotovili sami.

Primer zapisovanja utripajočega znaka "A" na trenutno pozicijo kurzorja, ki se nato poveča:

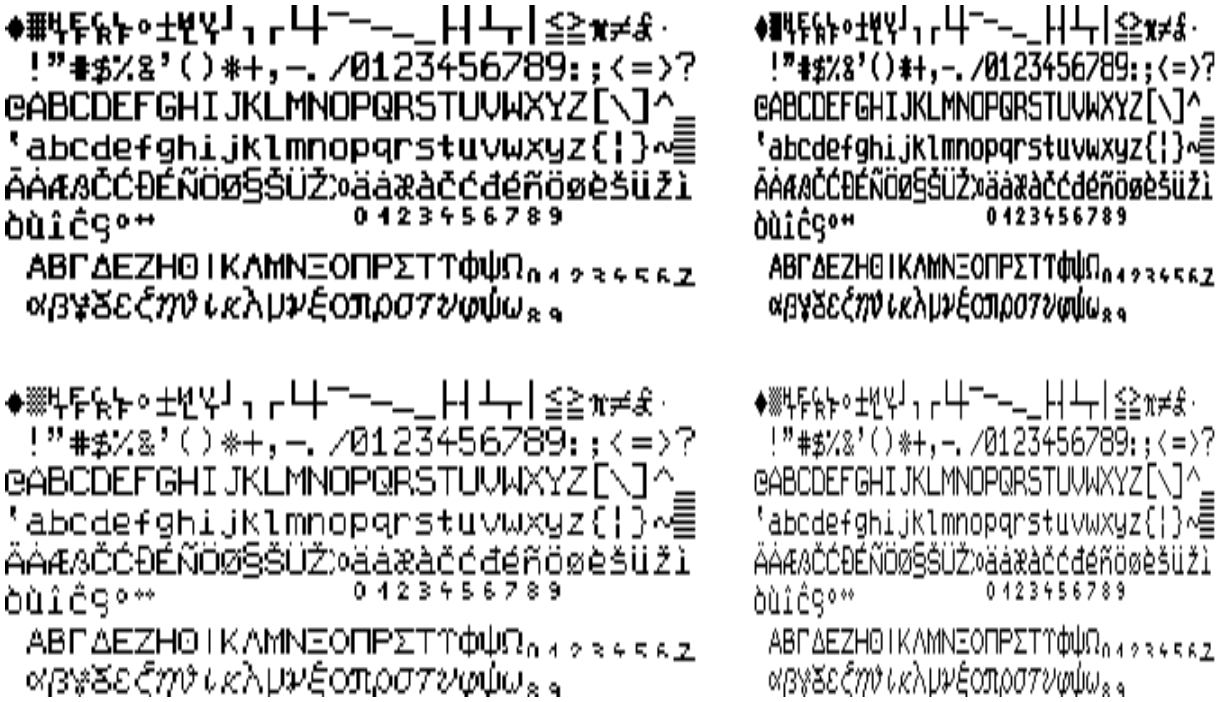
```
.cakaj1   in a, (39h)
          and 20h           ; Zastavica za pripravljenost AVDC...
          jr z, .cakaj1     ; ... mora biti 1.
.cakaj2   in a, (36h)
          and 10h           ; Zastavica, ali AVDC bere pomnilnik...
          jr nz, .cakaj2    ; ... mora biti 0.
          ld a, 'A'         ; Znak "A"...
          out (34h), a      ; ... zapišemo v znakovni register.
          ld a, 1           ; Atribut za utripanje...
          out (35h), a      ; ... zapišemo v atributni register.
          ld a, 0ABh        ; "Zapiši znak na kurzorju in ga povečaj".
          out (39h), a      ; Pošljemo ukaz.
```

Poleg omenjenih registrov na Partnerju obstaja še register z zastavicami, ki vplivajo na sliko, ki jo generira AVDC. Te zastavice niso povsem dokumentirane, zato bomo našteli le tiste, ki jih emulator implementira:

- Zastavica, ki raztegne "prižgane" piksle v znakih še za en piksel v desno.

Znaki se tako odebelijo, ločljivost slike pa ostane enaka. BIOS to zastavico v 80-stolpčnem načinu nastavi, v 132-stolpčnem načinu pa jo pobriše, vendar obe stanji delujeta v obeh načinih. To prikazuje slika 3.1.

- Zastavica, ki negira sliko; svetli piksli postanejo temni in obratno.
- Zastavica, ki jo emulator uporablja za razločevanje med 80-stolpčnim in 132-stolpčnim načinom, drugače pa njen pomen ni znan. V slednjem so znaki ožji.



Slika 3.1: Nabor znakov DCGG v 80- in 132-stolpčnem načinu, z odebelitvijo in brez.

V emulatorju so registri in pomnilnik AVDC shranjeni v poljih. Slika se izriše kot na pravem AVDC (z izjemo, da procesor medtem stoji); za vsako vrstico se preberejo znaki z naslova, ki je določen v vrstični tabeli, za vsak stolpec se glede na njegove atribute določi barva ospredja (torej barva samega znaka) in ozadja (ki zapolni preostanek pravokotne celice). Potem se znak (oz. polovica ali četrtina znaka, če so znaki v vrstici dvojne širine ali višine) izriše na tistem mestu. Pri tem se tudi skalira in odebeli, če je treba.

AVDC lahko tudi sproža prekinitve procesorju. Partnerjev BIOS omogoči le

prekinitev ob intervalu navpičnega vračanja (ang. vertical blanking interval). Emulator to prekinitev emulira, ne emulira pa drugih, npr. prekinitve, ki se lahko sproži, ko AVDC začne izrisovati prvo vrstico pikslov vsake vrstice besedila, ali pa prekinitve, ko je izvajanje ukaza zaključeno.

3.5.2 GDP

GDP (Thomson EF9367) prikazuje grafiko, lahko pa tudi besedilo.

Za to ima na voljo 128 kilobajtov pomnilnika tipa DRAM [1]. V njem sta shranjeni dve t.i. strani ločljivosti 1024×512 pikslov. Vsak piksel je lahko "prižgan" ali "ugasnjen". Medtem ko programska oprema spreminja eno stran, lahko GDP prikazuje drugo (ali isto) stran.

Za večjo hitrost lahko programska oprema izbere ločljivost 1024×256, kjer piksli niso več kvadratni, ampak so enkrat višji (strani sta v tem primeru še vedno dve, ne štiri), ali pa uporabi t.i. način hitrega pisanja, ki začasno onemogoči prikazovanje slike; prikazovanje namreč zahteva stalno izvajanje bralnih operacij, med katerimi pisanje ni mogoče.

Do grafičnega pomnilnika centralna procesna enota nima neposrednega dostopa. Pisalne operacije sproža s pošiljanjem ukazov. Ti so treh vrst:

- Ukazi za risanje črt. GDP ima v strojni opremi implementiran Bresenhamov algoritem za risanje črt [3]. Da programska oprema nariše črto, mora le določiti začetno in končno točko črte in poslati ukaz.
- Ukazi za risanje besedila. GDP vsebuje ROM, v katerem so shranjene oblike vseh znakov ASCII. S temi ukazi se te oblike kopirajo v sliko in postanejo del nje. Besedilo se lahko izrisuje vodoravno (od leve proti desni) ali navpično (od spodaj navzgor; znaki so obrnjeni za 90° v levo). V vsakem primeru je lahko pokončno ali ležeče, mogoče pa ga je tudi skalirati v širino in višino s celoštevilskima faktorjema od 1 do 16 [3]. (Pisavo bi lahko prerisali na podoben način kot tisto od AVDC/DCGG, vendar nam je ni bilo treba, ker je prikazana v dokumentaciji GDP.)
- Nadzorni ukazi, npr. brisanje celotne slike, preklapljanje med načinom risanja

in brisanja ipd.

Ukazi se pošiljajo s pisanjem na poseben vhodno-izhodni naslov. Branje istega naslova vrača statusne informacije. Od teh sta najbolj pomembna bita, ki povesta, ali je GDP pripravljen na nov ukaz in ali ravnokar poteka izrisovanje na zaslon.

Brisanje narisanih črt in besedila se doseže tako, da programska oprema namesto "peresa", ki piksle prižge, uporabi "radirko", ki jih ugasne, in z njo izvede zaporedje risarskih operacij, identično tistemu, s katerimi je piksle prižgala. (Ta način ni popoln, če se elementi, ki jih brišemo, prekrivajo z drugimi elementi.)

Na Partnerju je možen še drugi način: način pisanja v grafični pomnilnik je lahko "normalen", kjer se piksli nastavljajo na eno ali drugo stanje (prižigajo, če je izbrano pero in ugašajo, če je izbrana radirka), ali pa "XOR", ki na pikslih izvaja operacijo XOR (če je izbrano pero, se vrednost pikslov negira, drugače se ne spremeni).

Z ukazom, ki začasno dovoli neposreden dostop do grafičnega pomnilnika, je mogoče tudi branje. Partner dovoli branje enega piksla naenkrat. Programska oprema najprej določi koordinati, pošlje ukaz, nato pa prebere bit iz posebnega registra, ki ga implementira Partner (in ne GDP).

Ostale funkcije se upravljajo preko registrov, od katerih ima vsak svoj naslov. Registri služijo tudi za podajanje parametrov ukazom:

- Prvi kontrolni register določa, ali je izbrano pero ali radirka, ali je vključeno pisanje (če ne, risarske opracije nimajo učinka razen premikanja peresa), ali je vključen način hitrega pisanja, ali je uporabljen t.i. ciklični zaslon (če je, se zgornji biti koordinat ignorirajo tako, da je pero vedno znotraj slike) in katere prekinitve so omogočene.
- Drugi kontrolni register določa smer in obliko besedila in tip črte (polna, pikčasta, črtkana, črta-pika).
- Register za velikost znakov določa skalirna faktorja za izrisovanje znakov.
- Registra ΔX in ΔY, ki sta 8-bitna, določata razliko v koordinatah med

začetno in končno točko črte. Njuna predznaka se določita kot del ukaza za risanje.
- Registra X in Y, ki sta 16-bitna, določata koordinati začetne točke črte ali znaka. Izhodišče koordinatnega sistema je spodaj levo. Ta dva registra se po risarski operaciji samodejno posodobita; v primeru črte se jima prištejeta ΔX in ΔY, v primeru besedila pa se X oz. Y (za navpično besedilo) posodobi za risanje naslednjega znaka.
- Obstajata še registra, ki vsebujeta trenutno pozicijo svetlobnega peresa, vendar tega na Partnerju ni mogoče uporabljati.

Na Partnerju obstajata še dva registra, ki vplivata na delovanje GDP. Eden določi, za koliko vrstic navzdol je pomaknjena celotna grafična slika. Drugi vsebuje zastavice za:

- stran, ki se prikazuje,
- stran, na kateri poteka risanje,
- način risanja (normalen ali XOR),
- ločljivost (1024×256 ali 1024×512),
- način pomikanja slike; vrstice na dnu, ki jih pomikanje izrine iz slike, se lahko pojavijo na vrhu, ali pa tiste vrstice ostanejo prazne.

Primer risanja črte in besedila na GDP – rezultat je na sliki 3.2:

```
        ld a, 18h       ; Izberemo visoko ločljivost.
        out (30h), a
        ld a, 6         ; Pobrišemo sliko, X := 0, Y := 0.
        call UkaziGDP
        xor a           ; A := 0; izberemo vodoravne pokončne znake.
        out (22h), a
        ld a, 43h       ; Skaliranje znakov: 4× v X, 3× v Y smer.
        out (23h), a
        call Abc        ; Izpišemo "Abc".
        call CakajGDP
        ld a, 0Fh       ; Navpični ležeči znaki, črtni vzorec "črta-pika".
        out (22h), a
        xor a
```

```
            out (29h), a        ; Spodnjih 8 bitov X := 0.
            ld a, 100
            out (2Bh), a        ; Spodnjih 8 bitov Y := 100.
            out (27h), a        ; ΔY := 100.
            ld a, 200
            out (25h), a        ; ΔX := 200.
            ld a, 15h           ; Izrišemo črto; ΔX pozitivna, ΔY negativna.
            call UkaziGDP
            call CakajGDP
            ld a, 20
            out (2Bh), a        ; Spodnjih 8 bitov Y := 20.
            ld a, 34h           ; Skaliranje znakov: 3× v X, 4× v Y smer.
            out (23h), a
Abc         ld a, 'A'
            call UkaziGDP
            ld a, 'b'
            call UkaziGDP
            ld a, 'c'
UkaziGDP    call CakajGDP
            out (20h), a        ; Pošljemo ukaz.
            ret
CakajGDP    push af
.cakaj      in a, (2Fh)         ; (statusni register GDP)
            and 4               ; (zastavica za pripravljenost)
            jr z, .cakaj
            pop af
            ret
```

Slika 3.2: Primer črte in besedila na GDP.

Primer branja piksla (registra X in Y morata biti že nastavljena, rutina CakajGDP je enaka kot zgoraj):

```
call CakajGDP       ; Če GDP izvaja ukaz, čakamo.
ld a, 15            ; (ukaz za neposreden dostop do pomnilnika)
out (20h), a        ; Pošljemo GDP ukaz.
call CakajGDP       ; Čakamo, dokler ga ne izvede.
in a, (36h)         ; Beremo register, ki vsebuje piksel...
and 80h             ; ... v najvišjem bitu.
ret                 ; A je 0, če je prižgan in 80h, če je ugasnjen.
```

Emulirati GDP je dokaj enostavno. Ob pisanju na ukazni naslov se ukazi interpretirajo in izvedejo, ostali naslovi pa ali nastavijo register ali pa ga preberejo. Risanje se izvaja v polje, kjer vsak piksel zaseda en bit. Ko je treba vsebino zaslona osvežiti, se vsebina vidne strani izriše – za prižgane piksle to pomeni, da se svetlosti tistega piksla na zaslonu prišteje neka konstanta. Tako se slika združi s tisto, ki jo je prej narisal AVDC.

Risanje slike GDP se ne izvede, če je vključen način hitrega pisanja (ki v emulatorju ne omogoča nič hitrejšega pisanja kot sicer, a vseeno onemogoči prikaz slike kot na pravi strojni opremi), ali pa če je stran, ki je izbrana za prikaz, prazna. To je optimizacija, ki pohitri osveževanje slike pri aplikacijah, ki ne uporabljajo grafike. Ali je prazna, si emulator beleži z uporabo zastavice (za vsako stran); ta se nastavi, ko je izveden ukaz za brisanje slike, in pobriše, ko je izvedena risarska operacija.

3.5.3 Prikaz slike na zaslonu

AVDC in GDP generirata vsak svojo sliko. Ti dve sliki sta med seboj neodvisni. Emulator mora sliko celotnega zaslona izrisati v polje kvadratnih pikslov enakih velikosti. Na vprašanje, kakšno ločljivost uporablja Partner, ni enotnega odgovora, saj:

- AVDC generira (v teoriji) poljubno veliko sliko z znaki po 8×11 pikslov. Število vrstic in stolpcev je prilagodljivo; Partnerjev BIOS uporablja le 80×26 in 132×26. To pomeni 640×286 oz. 1056×286 pikslov, pri čemer so v prvem primeru piksli še dodatno raztegnjeni v širino. (AVDC sicer omogoča do 128 vrstic in 256 stolpcev; emulator podpira le velikosti, ki ju

uporablja BIOS.)
- GDP generira grafično sliko velikosti 1024×256 ali 1024×512 pikslov; le v drugem primeru so piksli kvadratni (gostota je približno 128 dpi v obe dimenziji). Ta slika vedno prekrije sliko, ki jo generira AVDC tako, da se svetlosti prekrivajočih se pikslov seštejejo.

Če vzamemo maksimum obeh dimenzij, je to 1056×512. V resnici pa je slika, ki jo generira AVDC, nekoliko večja (približno za višino ene vrstice besedila v vsako smer). Treba jo je torej skalirati, težava pa je, kako, da bo besedilo še vedno dovolj estetsko in da to ne bo vzelo preveč časa, saj je treba v najslabšem primeru zaslonsko sliko osvežiti 50-krat na sekundo (za grafiko to ni problem, ker so piksli ali kvadratni ali pa podvojeni v višino).

Odločili smo se za ločljivost 1056×572 oz. 1024×572 pikslov, če je zaslon gostiteljskega računalnika ožji od 1056 pikslov. Pri tem velja sledeče:

- Višina slike AVDC se v vsakem primeru podvoji (od tod višina 572 pikslov).
- Če AVDC prikazuje 80 stolpcev, se vsak drugi piksel podvoji v širino; slika AVDC je potem široka 960 pikslov, kar je manj kot slika GDP, vendar večina aplikacij še vedno izgleda dovolj dobro. Slika je tudi vodoravno centrirana.
- Če AVDC prikazuje 132 stolpcev in je zaslon ožji od 1056 pikslov, je vsak četrti znak širok samo 7 pikslov namesto 8; AVDC slika je v tem primeru široka 1023 pikslov.
- Slika GDP se podvoji v višino, če je izbrana ločljivost 1024×256, drugače pa se ne skalira. V vsakem primeru se navpično centrira, ker je nižja od slike AVDC.
- Če je zaslon širši od 1024 pikslov, se slika GDP vodoravno centrira.

3.6 Serijski vmesnik

Partner ima lahko največ tri serijska vrata, ki ustrezajo standardu RS-232-C in uporabljajo konektor DB-25 [1]. (Naš ima le ena, ki so standardna v vseh konfiguracijah Partnerja.) Tudi tipkovnica je priključena na serijska vrata, vendar uporablja konektor DIN 5.

Zanje skrbita dva krmilnika (SIO), od katerih ima vsak dva kanala [16]. Vsak kanal zaseda dva vhodno-izhodna naslova [1, 16]; prikazani so v tabeli 3.1.

Ime	Podatki	Status	Opomba
CRT	0D8h	0D9h	tipkovnica
LPT	0DAh	0DBh	v vseh konfiguracijah Partnerja
VAX	0E0h	0E1h	opcijsko (skupaj z MOD)
MOD	0E2h	0E3h	opcijsko (skupaj z VAX)

Tabela 3.1: Serijska vrata in pripadajoči naslovi.

V tabeli 3.1 je v prvem stolpcu ime vrat v CP/M. V drugem stolpcu je podatkovni naslov; s pisanjem nanj se sproži pošiljanje, če je krmilnik na to pripravljen, z branjem pa se dobi čakajoči podatek, če je na voljo [16].

V tretjem stolpcu je naslov, ki se uporablja za branje statusnih informacij in spreminjanje nastavitev (npr. hitrosti komuniciranja). Preko tega naslova je dostopnih več registrov serijskega krmilnika [16]. BIOS po inicializaciji oz. spreminjanju nastavitev vedno pusti izbran statusni register. Aplikacije navadno berejo le njega, zato emulator emulira le dva njegova bita, brez katerih aplikacije ne bi mogle pošiljati oz. prejemati podatkov: bit 2 pove, ali je krmilnik pripravljen na pošiljanje novega podatka, bit 0 pa, ali je prejel podatek, ki ga mora aplikacija prebrati. Razen tega emulator serijskih vrat samih ne emulira, emulira pa nekatere naprave, ki se priključijo preko njih.

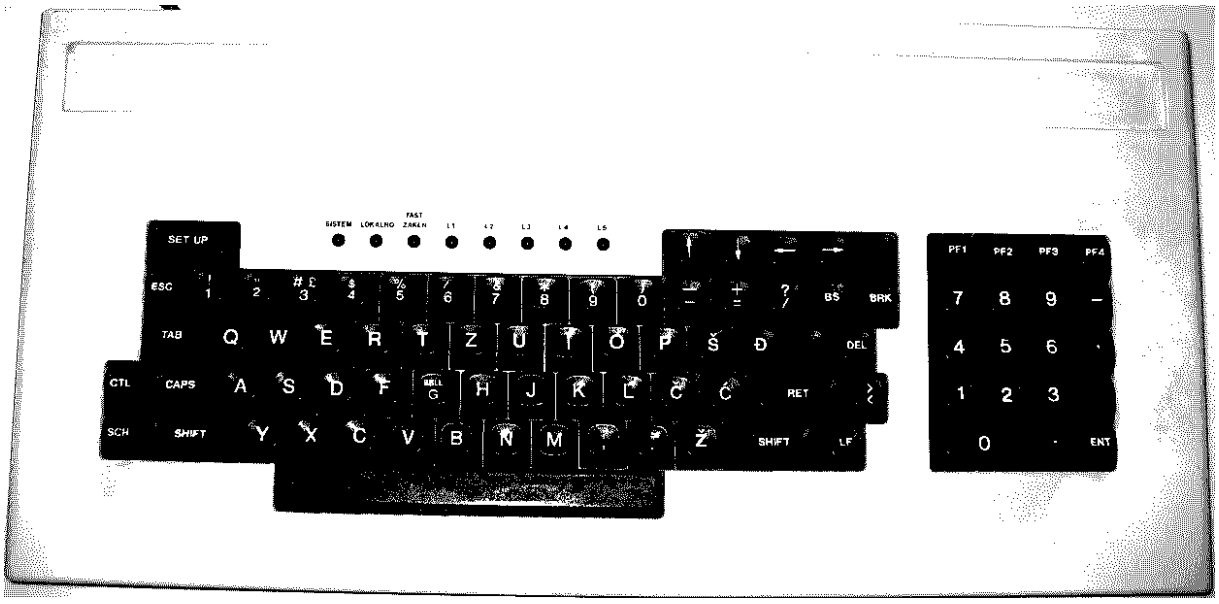
Partnerjev BIOS ob zagonu preveri prisotnost vrat VAX in MOD tako, da poskusi spremeniti enega od njunih registrov in potem prebere rezultat. Na podlagi tega sklepa, ali so ta vrata prisotna (oboja ali nobena).

Program za izdelavo slike trdega diska v rutini ZapisiBajt pokaže, kako se pošiljajo podatki preko serijskih vrat. Sinhrono branje poteka na enak način, le da je treba pred branjem preverjati bit 0 statusnega registra.

3.7 Tipkovnica

Partnerjeva tipkovnica se priključi na serijska vrata s konektorjem DIN 5. Z računalnikom komunicira s hitrostjo 300 bitov na sekundo [1]. Izdelalo jo je podjetje Gorenje.

Poleg alfanumeričnih tipk in numeričnega dela ima še nekaj posebnih tipk, ki jih danes standardne (PC) tipkovnice nimajo; prikazane so na sliki 3.3. Emulator slednje preslika na PC tipke na čim bolj podobnih mestih.



Slika 3.3: Partnerjeva tipkovnica.

Od PC tipkovnice se razlikuje tudi v tem, da je "samostojna". Kode, ki jih generirajo nekatere tipke, se spreminjajo glede na stanje tipk SHIFT, CTL in CAPS, slednje pa ne generirajo nobenih kod. Računalnik torej ne zazna, če pritisnemo ali spustimo katero od teh tipk, niti ne loči npr. kombinacije SHIFT+A od pritiska tipke A s pritisnjeno tipko CAPS. Računalnik je obveščen o pritisku (vendar ne spustu) tipke s prekinitvijo, na katero se odzove z branjem podatkovnega registra serijskih vrat. Če je tipka pritisnjena dalj časa, se ista koda pošlje večkrat; ponavljanje torej implementira tipkovnica, ne računalnik. Že kode tipk so v večini primerov enake znakom ASCII, ki jih

predstavljajo, tako da programski opremi ni treba prevajati vseh kod, ampak samo tiste za funkcijske tipke.

Tipkovnica ima tudi osem lučk (označene "SISTEM", "LOKALNO", "TAST ZAKLN." in "L1" do "L5"; pomen oznak v priročniku ni razložen, navadno pa je prižgana le lučka "SISTEM") in zvočnik, ki lahko proizvede kratek ali dolg pisk. Oboje je možno upravljati programsko s pošiljanjem 8-bitnih vrednosti:

- Če sta spodnja dva bita vrednosti "10", se sproži kratek pisk, če so spodnji trije biti "100", pa dolg pisk.
- Neodvisno od tega se spremeni tudi stanje lučk. Čeprav jih je osem, nima vsaka preprosto svojega bita: če sta spodnja dva bita vrednosti "00", se najprej od vrednosti odšteje 2, če sta "10" ali "11", se odšteje 1, drugače pa nič. Šele potem se lučke spremenijo glede na stanje bitov, pri čemer najnižji bit predstavlja najbolj levo lučko ("SISTEM"), najvišji pa najbolj desno ("L5").

Nekatere vrednosti spreminjajo delovanje tipkovnice. Uporabnik to doseže z uporabo programa Set up, ki je vgrajen v Partnerjev BIOS in se lahko prikliče kadarkoli s tipko SET UP. (Uporabnik tam tudi preklaplja AVDC med 80-stolpčnim in 132-stolpčnim načinom in po možnosti negira njegovo sliko.) Vendar ta program ne more spreminjati nastavitev na naši tipkovnici; očitno obstajata vsaj dva modela, ki se razlikujeta v ukazih za spreminjanje lastnosti. Na tipkovnici, ki jo podpira Set up, se vrednosti interpretirajo takole:

- Bit 3, če je nastavljen, izklopi pisk ob pritisku tipke.
- Bit 5, če je nastavljen, izklopi ponavljanje tipk.
- Bit 7, če je nastavljen, izbere jugoslovanski razpored "QWERTZ", drugače pa ameriški razpored "QWERTY".

Pri vseh ukazih morajo biti nastavljeni biti 0, 1, 2, 4 in 6, drugače se lastnosti tipkovnice ne spremenijo (v vsakem primeru pa se spremeni stanje lučk).

Emulator za boljšo uporabniško izkušnjo razume ukaze za model tipkovnice, ki ga predvideva Set up, čeprav ga mi nismo imeli. Večina aplikacij tipkovnico

samo bere (preko BDOS funkcij) in ji ničesar ne pošlje, tako da razlika v modelih ni velik problem.

Program za izdelavo slike naslovnega prostora v rutini Zapiskaj pokaže, kako se pošiljajo podatki na tipkovnico.

Zvočnik tudi proizvede zelo kratek pisk ob pritisku vsake tipke, če programska oprema tega ne izključi.

Ko emulator zazna pritisk tipke, jo preslika v ustrezno kodo, ki si jo zapomni. Nato sproži prekinitev (katero, smo izvedeli iz priročnika) in predvaja zvok, če je treba. BIOS se potem odzove z branjem podatkovnega registra serijskih vrat, kjer mu emulator vrne kodo tipke. Če uporabnik tipko drži še nekaj časa in ponavljanje tipk ni izključeno, emulator še naprej periodično sproža prekinitve, dokler tipka ni spuščena.

3.8 Miška

Nekatere aplikacije, razvite posebej za Partnerja, lahko uporabljajo miško, ki se priključi na serijska vrata. Mi je nimamo, vendar smo iz izvorne kode programa VIGRED ugotovili, kakšen protokol uporablja.

Program inicializira oz. zazna miško tako, da ji pošlje bajt z vrednostjo 99 in nato poskuša prebrati 60 bajtov. Če v teh 60 bajtih najde ASCII niz "LOGI", sklepa, da je miška priključena.

Miška podatkov ne vrača stalno, ampak je treba po njih periodično poizvedovati s pošiljanjem bajta z vrednostjo 80. Miška odgovori s petimi bajti:

1. bajt vsebuje stanje gumbov; bit 5 je levi gumb, bit 4 je srednji gumb, bit 3 je desni gumb.
2. bajt v spodnjih 6 bitih vsebuje spodnjih 6 bitov razlike v koordinati X.
3. bajt v spodnjih 6 bitih vsebuje zgornjih 6 bitov razlike v koordinati X (dvojiški komplement).

4. in 5. bajt naredita enako za razliko v koordinati Y. Ta narašča navzgor.

Miška je v emulatorju implementirana kot končni avtomat. Na začetku je ta v stanju, ko čaka na ukaz. Ko prejme vrednost 99, se premakne v stanje, ki ob naslednjem branju vrne ASCII znak "L", nato podobno "O", "G", "I" in za njimi ničelne bajte, ker ni znano, kakšni podatki bi morali zasedati preostalih 56 bajtov, ki jih programska oprema pričakuje. Potem gre miška spet v čakalno stanje. Na podoben način vrednost 80 sproži vračanje zgoraj opisanih petih bajtov.

3.9 Tiskalnik

Emulator lahko v datoteko preusmeri podatke, zapisane na serijska vrata. Na tak način je mogoče "tiskati" besedilne datoteke, npr. iz programa WordStar.

To sicer ni popolna rešitev, ker je večina tiskalnikov, ki so se uporabljali s Partnerjem, razen tiskanja besedila omogočala tudi spreminjanje lastnosti besedila (npr. pisave in njene velikosti) ali celo grafiko. Emulator tega ne podpira, vendar to za večino besedilnih datotek ne predstavlja težav.

3.10 Disketni pogon in trdi disk

Vsi modeli Partnerja imajo vsaj en disketni pogon. Model WF/G ima še trdi disk, ki je priključen na krmilnik Xebec S1410. ROM omogoča zaganjanje operacijskega sistema z obeh.

V želji, da bi emulator čim prej dosegel delujoče stanje (to pomeni, da bi se v njem lahko naložil CP/M in uporabljali vsaj nekateri programi), pa tudi zato, ker nismo imeli nobenih informacij o krmilniku disketnega pogona, smo se odločili za alternativen pristop: namesto emuliranja teh naprav smo v emulatorju implementirali prestrezanje diskovnih funkcij. To je virtualizacija na nivoju gonilnika [13].

CP/M namreč vedno kliče BIOS skozi dokumentirane vstopne točke, zato emulator pred izvedbo vsake inštrukcije preveri vrednost v PC. Če se ujema s katero od vstopnih točk za diskovne funkcije, emulator izvede želeno operacijo in vrne inštrukcijo RET, ki povzroči, da se prava BIOS funkcija ne izvede. Diskovne funkcije so sledeče [7]:

- HOME. Določi sled 0 za naslednjo bralno oz. pisalno operacijo.
- SELDSK. Izbere napravo za vse nadaljnje diskovne operacije.
- SETTRK. Določi sled za naslednjo bralno oz. pisalno operacijo.
- SETSEC. Določi sektor za naslednjo bralno oz. pisalno operacijo.
- SETDMA. Določi 16-bitni naslov medpomnilnika, ki bo prejel podatke naslednjega prebranega sektorja oz. ki vsebuje podatke, ki bodo zapisani v sektor.
- READ. Prebere sektor, določen s SETTRK in SETSEC z naprave, določene s SELDSK, v medpomnilnik, določen s SETDMA in SETBNK.
- WRITE. Zapiše vsebino medpomnilnika v sektor; oba se določita enako kot pri READ.
- SETBNK. Določi banko, v kateri je medpomnilnik za naslednjo bralno oz. pisalno operacijo.

Ta metoda večinoma deluje, ker je CP/M edini operacijski sistem za Partnerja in večina aplikacij ne dostopa do teh dveh naprav neposredno, ampak skozi CP/M. Glavni izjemi sta programa WF in DISKETTE, ki sta namenjena prav temu, da preverita delovanje trdega diska in njegovega krmilnika oz. trenutno vstavljene diskete, in program FORMAT, ki formatira diskete.

Tudi ROM do teh dveh naprav dostopa neposredno (saj drugače ne more začeti nalaganja operacijskega sistema), zato emulator prestreže tudi nekatere njegove funkcije.

Isto metodo bi se dalo uporabiti tudi za nekatere druge naprave, vendar se je v želji po čim boljši emulaciji nismo posluževali, če ni bilo potrebno.

3.11 Ura realnega časa

Da operacijski sistem ve, kateri dan je in koliko je ura, Partner vsebuje uro realnega časa, ki jo napaja baterija.

Ura je procesorju dostopna kot množica registrov, ki vsebujejo časovne komponente (tisočinke, sekunde, minute, ure, dan, mesec, zadnji dve števki leta) v obliki "packed BCD". Vsak register ima svoj vhodno-izhodni naslov, skozi katerega ga je mogoče poljubno brati in spreminjati.

Emulator dobi datum in čas od gostiteljevega operacijskega sistema. Ker Partnerjev BIOS in CP/M ne podpirata datumov po letu 1999 (zaradi hroščev se ti datumi ne izpišejo pravilno; to bi se teoretično dalo popraviti), emulator od trenutnega leta odšteje večkratnik 28, da je leto manjše od 2000 (npr. 2017 – 28 = 1989). Gregorijanski koledar se namreč ponovi na vsakih 28 let; npr. 1. julij 2017 je bila sobota, 31. julij 2017 pa ponedeljek – leta 1989 je bilo enako. (Ta rešitev deluje do vključno 28. februarja 2100, če gostiteljev operacijski sistem nima druge omejitve.)

K uri realnega časa spada tudi majhen pomnilnik, ki se uporablja za shranjevanje uporabniških nastavitev. Te se urejajo s programom Set up. Emulator vsebino tega pomnilnika ob zagonu prebere iz datoteke (oz. uporabi privzete vrednosti), ob izhodu pa ga shrani, če se je spremenil.

Če se baterija izprazni in ta pomnilnik izgubi napajanje, njegova vsebina ni več veljavna; vsi biti se preberejo kot enice. Med njimi BIOS štiri bite uporablja, da si zapomni privzeti nabor znakov. Ta vrednost je navadno med 0 in 8; ko smo prvič vključili računalnik, je bila 15 (dvojiško 1111), zaradi česar so se v pomnilnik AVDC zapisovali napačni znaki; BIOS namreč ob branju te vrednosti ne preveri, ali spada v dovoljeni interval. Slika 3.4 prikazuje ta učinek. Program Set up k sreči izgleda pravilno ne glede na izbrani nabor znakov, zato smo to težavo še prvi dan odpravili tako, da smo izbrali jugoslovanski nabor.

```
A>  vic

PhysiculrD vic s:r
IwInpde,OwOdepde,§w§ aiul,XwXrn-Xrff
CRTrrrr9600rrIO§rrrrLPTrrrr§800rrIO§Xrrrrrrrrrr9600rrrrrrrrr
rrrrrrr300rrrrrrrrrrrrrrrrrNONErrrrrrrrrGDPrrrrNONErrOrrrrrr


Cdaa nerAssignm nes:r
CONIN:rrwrCRTrrrr
CONOUT:rwrGDPrrrr
AUXIN:rrwrLPTrrrr
AUXOUT:rwrLPTrrrr
L§T:rrrrwrLPTrrrr

Ene arn !russignm nerrarhierRETURNr

A>device

Physical Devices:
I=Input,O=Output,S=Serial,X=Xon-Xoff
CRT     9600  IOS     LPT     4800  IOSX          9600
        300                   NONE          GDP     NONE  O


Current Assignments:
CONIN:   = CRT
CONOUT:  = GDP
AUXIN:   = LPT
AUXOUT:  = LPT
LST:     = LPT

Enter new assignment or hit RETURN
```

Slika 3.4: Besedilo ob prvem vklopu (zgoraj) in po pravilni nastavitvi nabora (spodaj).

3.12 Ostale naprave

V Partnerju je še nekaj čipov, ki jih ne emuliramo:

- Krmilnik DMA. Uporablja se za prenose med delovnim pomnilnikom in

krmilnikom disketnih pogonov oz. krmilnikom trdega diska. Ne emuliramo ga, ker tudi njiju ne. Ni znano, ali se lahko uporablja tudi za kaj drugega.

- CTC. Uporablja se v istih situacijah kot krmilnik DMA. Teoretično bi ga lahko uporabljale tudi aplikacije za svoje potrebe, vendar tega nismo opazili.
- Krmilnik paralelnih vrat (PIO). Na trdem disku je sicer izvorna koda za pošiljanje in branje podatkov z njim, vendar ne vemo, kakšne naprave so ga v praksi uporabljale.

Poglavje 4

Ovrednotenje rešitve

V tem poglavju bomo ocenili pravilnost emulatorja in navedli možne izboljšave.

4.1 Natančnost

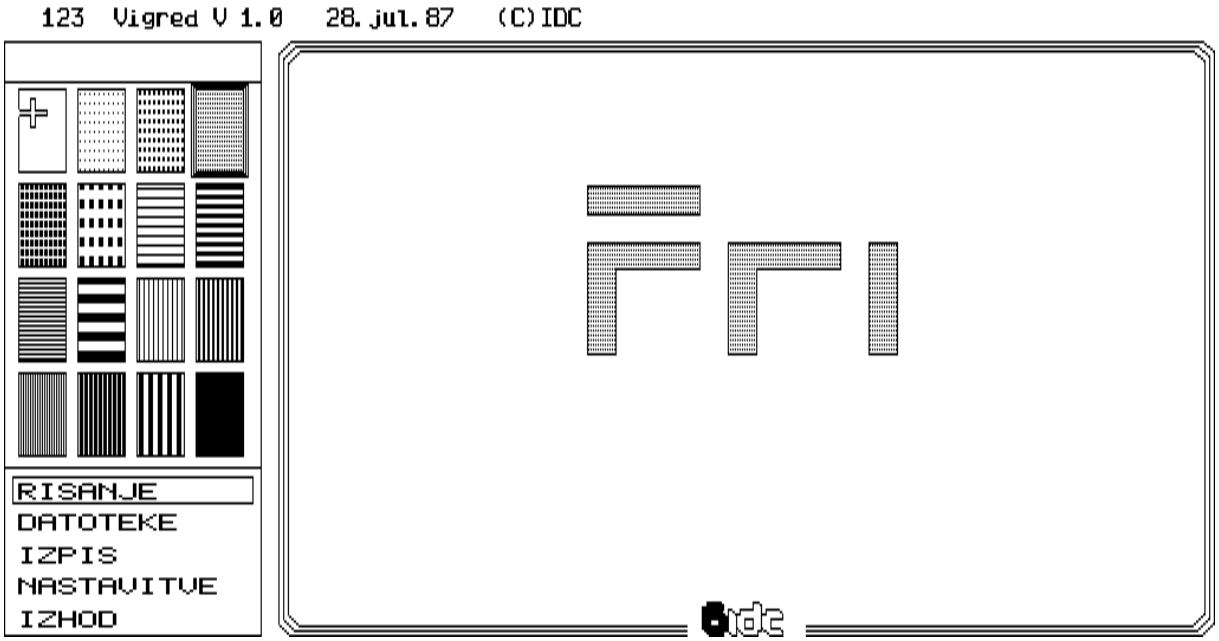
V splošnem smo poskušali narediti emulator čim bolj podoben pravemu Partnerju, če le to ni zahtevalo preveč dela. S perspektive uporabnika bi aplikacije v njem morale delovati enako kot na Partnerju, tudi če se emulator razlikuje v nekaterih podrobnostih. Slika 4.1 prikazuje aplikacijo VIGRED, ki je zahtevala nekoliko več dela kot ostale, preden je delovala povsem pravilno.

Te podrobnosti so večinoma povezane s časom in so posledica tega, da v pravem računalniku vsa strojna oprema deluje sočasno, v emulatorju pa zaradi enostavnosti vse poteka zaporedno. To se najbolj pozna pri emulaciji AVDC in GDP, kjer se spremembe v njunih registrih oz. pomnilniku uveljavijo le periodično namesto v realnem času, saj emulator ne poustvari delovanja zaslona s katodno cevjo.

Druga stvar, povezana s časom, so razne zakasnitve, ki so prav tako posledica sočasnega delovanja strojne opreme. AVDC, GDP in druge naprave večine operacij ne izvedejo takoj (s perspektive procesorja "takoj" pomeni hitreje, kot lahko procesor izvede inštrukcijo, ki sledi tisti, ki je sprožila operacijo na napravi), ampak šele po nekem času, konec operacije pa procesorju sporočijo s prekinitvijo ali spremembo statusnega registra, ki ga mora procesor periodično preveriti. Podobno je s pomnilnikom tipa DRAM, ki se bere in zapisuje z zakasnitvijo in se mora periodično osveževati, medtem pa procesor do njega ne more dostopati.

V emulatorju se te operacije izvedejo takoj; ko procesor dostopa do naprave, se pokliče ustrezna funkcija, ki tisto operacijo takoj izvede, in šele nato lahko

emulirani procesor naprej izvaja inštrukcije. To bi se dalo popraviti, vendar ni bistveno; navsezadnje se (pravilno napisana) programska oprema ne bi smela zanašati na take podrobnosti, saj dokumentacija za večino naprav niti ne omenja dolžine teh zakasnitev; edini način, da ugotovimo njihovo trajanje, je empirično, torej da jih izmerimo.



Slika 4.1: Aplikacija VIGRED v emulatorju.

4.2 Možne izboljšave

Za "popolno" emulacijo bi morali emulirati še naprave, ki jih trenutno ne – naštete so bile v prejšnjem poglavju.

Zaželene so tudi sledeče funkcije:

- Emulacija grafične tablice. Imamo izvorno kodo, ki komunicira z njo; morda bi si lahko z njo pomagali, kot smo si pomagali z izvorno kodo za miško.
- Emulacija vsaj enega izmed tiskalnikov, ki jih je Iskra Delta podprla v svojih aplikacijah.

- Emulacija vsaj enega izmed risalnikov oz. tiskalnikov s podporo za grafiko, ki jih je Iskra Delta podprla v svojih aplikacijah.

Nekatere nebistvene funkcije bi olajšale delo z emulatorjem in premikanje datotek med emulatorjem in gostiteljem, npr.:

- integracija z datotečnim sistemom gostitelja,
- uvažanje datotek v diskovne slike; trenutno je to treba storiti ročno s šestnajstiškim urejevalnikom (ang. hex editor),
- lažje izvažanje datotek iz diskovnih slik,
- kopiranje in lepljenje besedila med emulatorjem in gostiteljem.

4.3 Uporabniška izkušnja

Ker je Partnerja uporabljalo malo ljudi in se ta številka v prihodnosti ne bo povečala, je smiselno, da se emulator tudi navzven poskuša obnašati čim bolj kot pravi Partner. To doseže z uporabo zvočnih posnetkov trdega diska, tipkovnice in stikal za vklop oz. izklop in resetiranje. Tudi slika na zaslonu je takoj po "vklopu" (zagonu emulatorja) črna, šele nato doseže polno svetlost, medtem pa se predvaja zvok zagona trdega diska.

4.4 Prenosljivost

Emulator, ki podpira le enega ali peščico operacijskih sistemov, ni preveč uporaben, ker je potem tudi sam v nevarnosti, da ga nekega dne v prihodnosti ne bo več mogoče uporabljati.

Prenosljivost je dosežena z uporabo jezika C (kodo v tem jeziku se da prevesti praktično povsod, pa tudi sam jezik se dobro prilega pisanju emulatorjev) in knjižnice SDL (Simple DirectMedia Layer) [6]. Slednja priskrbi enoten vmesnik za upravljanje z vhodnimi in izhodnimi napravami (npr. branje tipkovnice in miške, predvajanje zvoka, risanje na zaslon) na številnih operacijskih sistemih. Emulator je z različnimi prevajalniki mogoče prevesti za

različne operacijske sisteme in arhitekture.

Prevesti ga je mogoče tudi s prevajalnikom Emscripten, ki kodo iz jezika C prevede v JavaScript [12]. Emscripten implementira tudi večji del vmesnika SDL. Emulator se lahko tako izvaja tudi v spletnem brskalniku na domeni matejhorvat.si, vendar to zahteva veliko več računske moči, kot če je preveden v strojno kodo.

Poglavje 5

Zaključek

Na začetku projekta zaradi težav s strojno opremo (disketnim pogonom in serijskimi vrati) in malo dokumentacije nismo vedeli, ali nam bo uspelo narediti delujoč emulator. Z malo sreče in obratnega inženiringa pa smo prišli kar daleč; s perspektive uporabnika se emulator obnaša skoraj enako kot pravi Partner.

Seveda bi se ga dalo še izboljšati; kot je bilo omenjeno v prejšnjem poglavju, nekatere naprave sploh niso emulirane in aplikacije delujejo le zato, ker do njih ne dostopajo neposredno ali pa jih sploh ne uporabljajo, nekatere naprave pa so emulirane le, kolikor je treba, da večina aplikacij deluje brez težav.

Projekt je trajal približno 4 mesece. Emulator je bil narejen v približno enem mesecu, vendar je to potekalo sočasno z raziskovanjem in obratnim inženiringom. Njegova koda obsega malo več kot 3500 vrstic. Sem nista šteti knjižnica za emulacijo procesorja Zymosis in knjižnica SDL.

Za projekt je bilo dodatno napisanih skoraj 1000 vrstic kode v različnih jezikih. To vključuje npr. kodo za izdelavo slike trdega diska in slike bralnega pomnilnika, dekodiranje teh slik in razno eksperimentalno kodo za obratni inženiring strojne opreme.

Rezultati projekta so dostopni na spletni domeni matejhorvat.si.

S podobnimi pristopi bi se najbrž dalo emulirati tudi druge slovenske računalnike, npr. Dialog in Triglav.

Literatura

[1] Iskra Delta. Partner WFG, 2FG, 1FG – priročnik za uporabnike. 1987.

[2] Signetics. SCN2674/SCN2674T Advanced Video Display Controller (AVDC). Dosegljivo: http://www.datasheet4u.com/datasheet-pdf/Signetics/SCN2674/pdf.php?id=524408, 1987. [Dostopano: 1. 5. 2017]

[3] SGS-Thomson Microelectronics. EF9367 MOS graphic display processor (GDP). Dosegljivo: http://www.datasheet4u.com/datasheet-pdf/SGS-Thomson/EF9367/pdf.php?id=604475, 1988. [Dostopano: 1. 5. 2017]

[4] KryoFlux. Dosegljivo: https://www.kryoflux.com/. [Dostopano: 12. 12. 2016]

[5] Zymosis. Dosegljivo: http://repo.or.cz/w/zymosis. [Dostopano: 8. 4. 2017]

[6] Simple DirectMedia Layer 1.2.15. Dosegljivo: https://www.libsdl.org/download-1.2.php. [Dostopano: 2. 4. 2017]

[7] CP/M information archive: BIOS. Dosegljivo: http://seasip.info/Cpm/bios.html. [Dostopano: 20. 4. 2017]

[8] CP/M information archive: BDOS system calls. Dosegljivo: http://seasip.info/Cpm/bdos.html. [Dostopano: 20. 4. 2017]

[9] HxC Floppy Emulator. Dosegljivo: http://hxc2001.com/download/floppy_drive_emulator/. [Dostopano: 28. 3. 2017]

[10] Digital Research Source Code. Dosegljivo: http://www.cpm.z80.de/source.html. [Dostopano: 2. 4. 2017]

[11] Digital Research. An introduction to CP/M features and facilities. 1978.

[12] Emscripten. Dosegljivo: https://kripken.github.io/emscripten-site/. [Dostopano: 1. 7. 2017]

[13] J. E. Smith, R. Nair. Virtual Machines – Versatile Platforms for Systems and Processes. Morgan Kaufmann Publishers, 2005.

[14] A. Vilfan, J. Vilfan. Iskra Delta med zahodom in vzhodom. Bit, 1984, št. 1.

[15] J. C. Nichols, E. A. Nichols, P. R. Rony. Z-80 Microprocessor Programming & Interfacing: Book 2. Howard W. Sams & Co., 1980.

[16] L. A. Leventhal. Z80 assembly language programming. Osborne/McGraw-Hill, 1979.

[17] A. P. Železnikar. Evropska mikroračunalniška industrija. Informatica, 1984, št. 1.