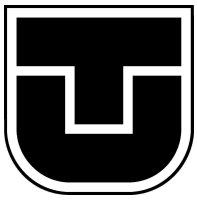


TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

emuStudio

Peter Jakubčo

Príloha A
Diplomová práca
Košice, 2009

Používateľský manuál

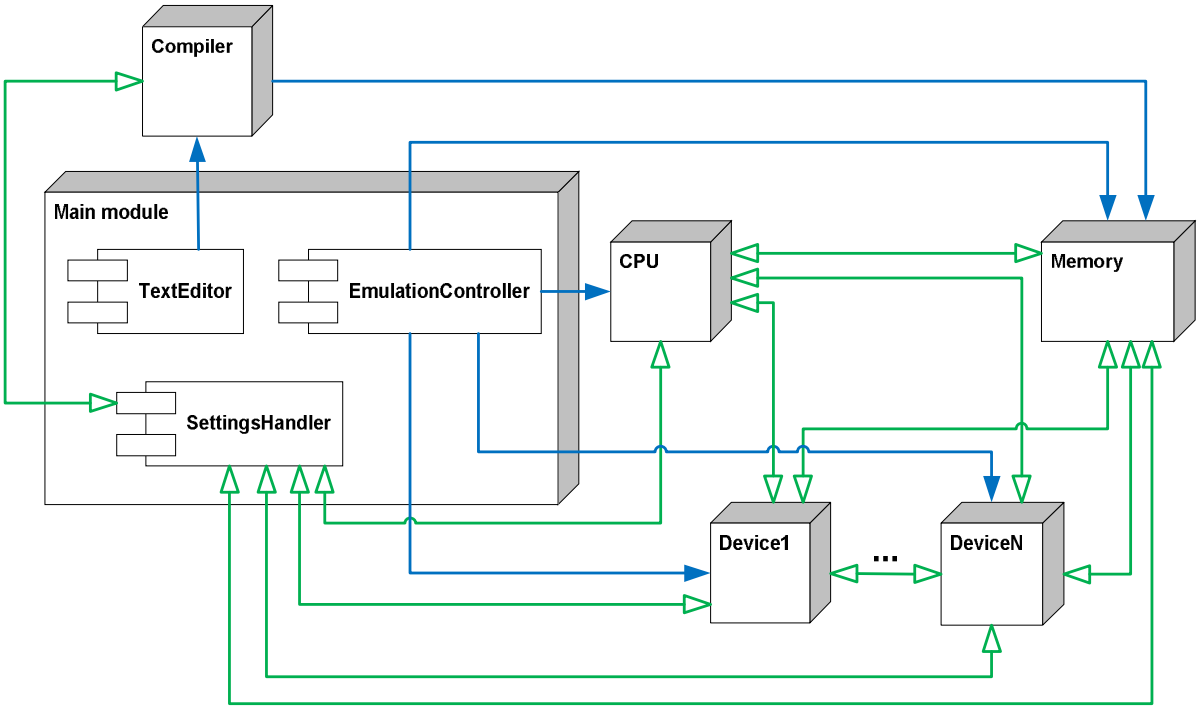
Vedúci práce a konzultant: *ing. Slavomír Šimoňák, PhD.*

Obsah

Úvod

Príručka si kladie za cieľ popísať spôsob programovania a prácu v emulačnej platforme ***emuStudio***. Účel tejto platformy je pomôcť študentom a laikom pochopiť prácu starších (nie nevyhnutne 8-bitových) počítačov, čo znamená, že pri jej tvorbe bola preferovaná interakcia (komunikácia s používateľom a vizualizácia prebiehajúcich procesov) pred samotným výkonom platformy.

Štruktúru platformy je možné vidieť na Obr.1. Uzol **Main module** sa skladá z niekoľkých komponentov, pričom na obrázku sú uvedené len tri (*EmulationController*, *TextEditor* a *SettingsHandler*). Komunikačné linky medzi jednotlivými uzlami sú buď jednosmerné (modrá farba, plné šípky), alebo obojsmerné (zelená farba, prázdne šípky). Keď sa zahľadíme len na uzly **CPU**, **Memory** a **Device**$_{1..n}$, štruktúra pripomína Von-Neumanovskú architektúru (čo bol, aj je úmysel); kde procesor, pamäť a periférne zariadenia sú oddelené, ale prepojené. Snaha o zachovanie podobnosti s touto architektúrou vyústila do riešenia, v ktorom sú všetky spomenuté moduly realizované formou zásuvných modulov.



Obr. 1: *Štruktúra platformy emuStudio*

Inštalácia programu

Na to, aby bolo možné spustiť platformu ***emuStudio***, je potrebné mať nainštalované prostredie *Java Runtime Environment* (JRE) aspoň verziu **1.6**. Dá sa stiahnuť zo stránky `http://www.java.com/en/download/manual.jsp`

Na samotné spustenie a používanie programu už nie je potrebná žiadna ďalšia, špeciálna inštalácia. Súbory z inštalačného balíka rozbaľte do ľubovoľného adresára. Štruktúra podadresárov musí byť nasledovná:

`\compilers`	→ Zásuvné moduly kompilátorov
`\config`	→ Súbory konfigurácií
`\cpu`	→ Zásuvné moduly všetkých CPU
`\devices`	→ Zásuvné moduly zariadení
`\lib`	→ Potrebné knižnice
`\mem`	→ Zásuvné moduly operačných pamätí

Program sa dá spustiť nasledovným príkazom v príkazovom riadku:

```
java -jar emuStudio.jar
```

Bližší popis nájdete v súbore `README.TXT`.

Časť I

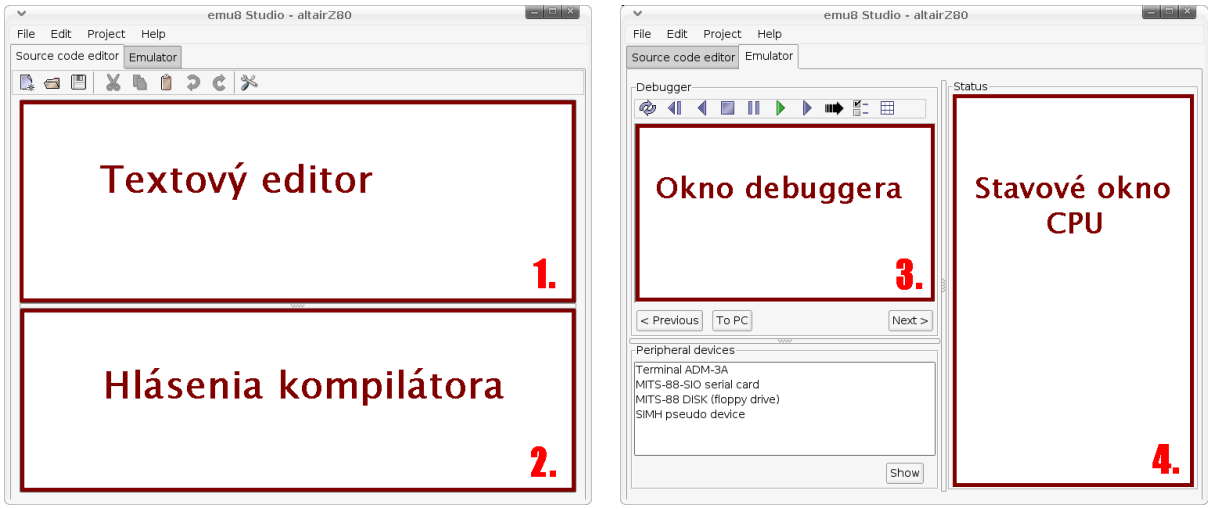
Hlavný modul

Kapitola 1

Prehľad

Hlavný modul je samostatný program vyvíjaný nezávisle od zásuvných modulov. Jeho úloha je komunikovať s používateľom - umožniť používateľovi riadiť emuláciu ľahko a pohodlne, a tiež integrovať zásuvné moduly - aby všetko spolu vyzeralo ako jeden celok. Výhodou nezávislého modulu je v tom, že je univerzálny a používateľ sa nemusí učiť používať stále nový spôsob ovládania emulátora. Základné funkcie pre písanie, kompilovanie zdrojových textov, a ovládanie emulácie sú rovnaké pre všetky zvolené konfigurácie.

Teraz sa budeme zaoberať štruktúrou hlavného okna (Obr. 1.1), ktoré je najdôležitejším prvkom hlavného modulu. Síce trochu predbiehame, ale je dobré základné veci vysvetliť už tu.



Obr. 1.1: *Štruktúra hlavného okna*

Obrázok hlavného okna obsahuje štyri orámované časti. Tieto časti môžu mať po spustení programu „premenlivý“ obsah — môže sa meniť vzhľadom na zvolenú konfiguráciu počítača.

Celkovo je hlavné okno rozdelené do dvoch panelov: *Source code* (ľavá časťObr. 1.1) a *Emulator* (pravá časťObr. 1.1).

1.1 Panel „Source code“

Panel *Source code* obsahuje nástroje na manipuláciu so zdrojovým kódom. Skladá sa z dvoch veľmi dôležitých častí - textový editor (na obrázku označený číslom 1) a okno s hlásením kompilátora (číslo 2).

Voľba konfigurácie počítača (popísaná v časti 2) zahŕňa aj voľbu kompilátora (prekladača). Tento dokáže preložit' zdrojový kód (napísaný v textovom editore) do strojového kódu zvolenej CPU. Pre jednu CPU môže existovat' aj viacero kompilátorov (ktoré nemusia byt' nevyhnutne assemblermi[1]), preto má zmysel si kompilátor volit'.

Textový editor sa správa a vyzerá na prvý pohľad rovnako pre všetky zvolené kompilátory. Používateľ v ňom môže písat' zdrojový kód presne podľa syntaxe jazyka zvoleného kompilátora. Má k dispozícii číslovanie riadkov a zvýrazňovanie syntaxe jazyka. Pri snahe zvýrazňovat' syntax textový editor komunikuje priamo s kompilátorom, ktorý mu poskytuje lexikálne jednotky a preto zvýrazňovanie syntaxe funguje rovnako dobre pre každý jazyk.

O priebehu kompilovania zdrojového kódu informuje kompilátor používateľa zobrazovaním jeho hlásení, v časti 2 na Obr. 1.1. Hlásenia sa môžu (a spravidla sa aj) pre každý kompilátor líšit'. V každom prípade by sa po kompilovaní malo vypísat' hlásenie o úspechu/neúspechu prekladu. Dobrý kompilátor by mal vediet' identifikovat' chyby/varovania číslom riadku a stĺpca s popisom chyby.

1.2 Panel „Emulator“

Panel *Emulator* napravo na Obr. 1.1 obsahuje tri časti - okno debuggera (ladiace okno, označené číslom 3), stavové okno CPU (časť 4) a okno periférnych zariadení[2]. Obsah označených okien (grafická reprezentácia) je závislý na zvolenej konfigurácie počítača.

Napríklad: pri emulácii skutočných CPU ladiace okno vypisuje určitú oblast' operačnej pamäte, do ktorej ukazuje programové počítadlo[3]. Táto oblast' je obvykle reprezentovaná krajšou formou ako iba výpisom čísel - hodnôt buniek v pamäti. Väčšinou je okno rozdelené do niekoľkých stĺpcov a každý riadok predstavuje jednu alebo viac adries operačnej pamäte. Každý stĺpec potom reprezentuje hodnotu na tejto adrese inak - jeden stĺpec môže zobrazovat' mnemonický tvar inštrukcie, iný, či je na adrese tzv. breakpoint (bod pozastavenia emulácie), atď. (Obr 4.2). Štruktúra tohto okna však nemusí byt' rovnaká pre všetky architektúry, ktoré sú k dispozícii. Pre reálne CPU takáto štruktúra vyhovuje, no pri reprezentácii napr. abstraktného Turingovho stroja (ktorý však zatiaľ nie je implementovaný), by takáto štruktúra vhodná nebola (keďže inštrukcie nie sú reprezentované rovnakým spôsobom ako dáta na tej istej pamäti/páske).

Ak si však na Obr. 1.1 všimneme nástrojový panel umiestnený nad časťou 3 (zväčšený na Obr. 4.3), zistíme, že ovládanie emulácie je jednotné pre ľubovoľnú zvolenú konfiguráciu, čiže

[1] v podstate môžu existovat' prekladače rôznych programovacích jazykov, ktorých výstupom je strojový kód zvolenej CPU

[2] nie je označené číslom, jeho obsah má jednotnú štruktúru - zoznam zariadení, čiže sa dá pokladat' za nie „premenlivý“

[3] programové počítadlo, alebo `PC`, je v 8 bitových procesoroch názov registra obsahujúceho adresu nasledujúcej inštrukcie

ovládanie emulácie nezávisí od štruktúry zobrazenia aktuálnej inštrukcie a jej „okolia“.

Ďalej časť 4, predstavujúca stavové okno CPU, má takisto meniaci sa obsah v závislosti od zvolenej CPU. Pod stavom CPU rozumieme aktuálne hodnoty jej všetkých registrov a vnútorných nastavení/signálov v jednom časovom okamihu ako aj informáciu, či CPU beží, alebo nebeží. Okno nemusí zobrazovať všetky stavy a naopak môže zobraziť aj niektoré informácie navyše.

Abstraktné stroje nemusia mať registre, preto ak predpokladáme, že stroj je dobre formálne popísaný, potom pojem stav takéhoto stroja budeme chápať v zmysle jeho definície pre tento stroj. Pretože samotný stav môže byť reprezentovaný rôznymi informáciami, resp. reálne CPU majú iný počet a typ registrov a nastavení, obsah stavového okna nemôže byť rovnaký pre všetky CPU. Obvykle toto okno pre reálne CPU zobrazuje práve hodnoty jeho registrov, nastavení; prípadne obsahuje možnosti aj pre nastavenie rýchlosti CPU.

Nakoniec neorámované okno na Obr. 1.1 dole obsahuje zoznam zariadení, ktoré sú načítané v aktuálnej konfigurácii (čiže mení sa iba aktuálny „zoznam“, nie celá štruktúra okna, ako to je v prípade označených okien). Položka zoznamu popisuje zariadenie svojim interným menom, ako je definované vo vnútri zásuvného modulu, teda nie názov súboru. Používateľ môže zobraziť grafické rozhranie zariadenia (ak ho zariadenie má), kliknutím na tlačidlo *Show*.

1.3 Základný postup, ako vyvíjať a emulovať

Je pravdou, že emulátor bol vyvíjaný so zreteľom na jednoduchosť a pohodlie pre používateľa. Napriek tomu, možno je to vplyvom aj samotných emulovaných architektúr, sa môže cítiť používateľ na prvý pohľad zmätený a nemusí hneď zo štruktúry emulátora (Obr. 1) pochopiť, ako pod emulátorom správne pracovať.

Z toho dôvodu je táto časť venovaná práve základnému postupu práce na emulátore. Pod pojmom „práca na emulátore“ rozumieme jednu z dvoch činností - vývoj programu, alebo samotné emulovanie. Z takejto definície vychádza aj diagram na Obr. 1.2.

Ide o diagram aktivít, ktorý sa podobá vývojovému diagramu. Sú v ňom zahrnuté základné procesy v postupnosti, aká je pre platformu prirodzená. Poradie niektorých krokov v postupnosti sa môže zmeniť, niektoré možno dokonca aj vynechať a okrem uvedených krokov môžu existovať aj iné, ktoré na diagrame uvedené nie sú. Tie vznikajú už z potrieb konkrétneho emulovania (napríklad nastavenie dodatočných parametrov operačnej pamäte).

Práca na emulátore začína formálnym rozhodnutím, čo chceme robiť - či vyvíjať program, alebo rovno emulovať. Podľa rozhodnutia sa postupuje do danej vetvy v diagrame. Vývoj programu znamená napísať jeho zdrojový kód. Používateľ samozrejme nemusí vychádzať zo striktného pojmu *napísať* zdrojový kód, môže otvoriť aj existujúci zo súboru alebo vložiť ho do textového editora iným spôsobom. Aktivita *Písanie zdrojového kódu* znamená vytváranie nového kódu, dopisovanie, modifikáciu, atď. Používateľ však musí kód písať so zreteľom na syntax a sémantiku zvoleného kompilátora vo virtuálnej architektúre.

Nasleduje aktivita *Kompilovanie zdrojového kódu*, ktorej výstupom je buď hotový program pripravený na emuláciu, alebo chybové hlásenia kompilátora. V prípade, že kód nebol preložený z dôvodu chýb, je na používateľovi, aby kód opravil a znova prekompiloval. V opačnom prípade

sa prejde na jeho emuláciu[4].

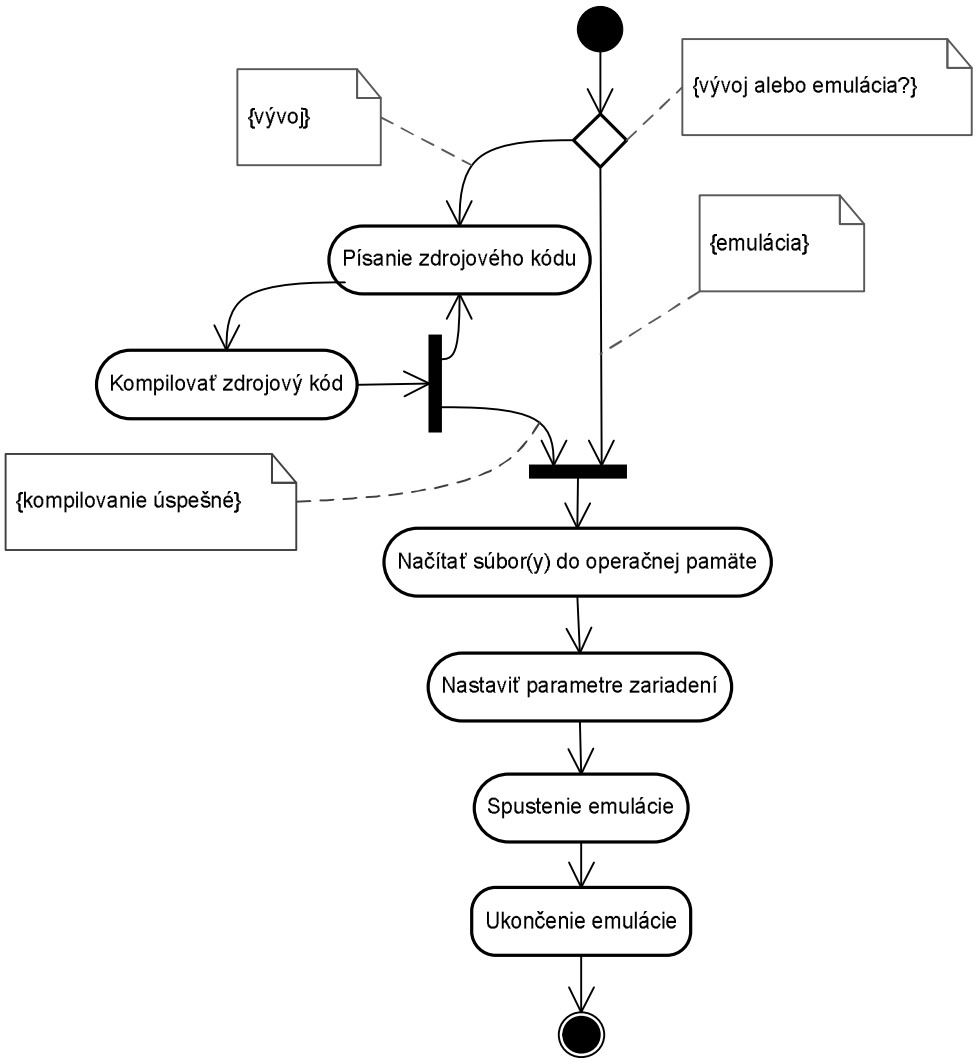
Emulovať sa môže začať, až keď bude všetko pripravené - hlavne treba emulátoru povedať, čo chceme emulovať. Všetky architektúry na emulátore pracujú podobne ako von-neumannovská architektúra, preto odpoveď na otázku čo sa má emulovať je v operačnej pamäti. Operačná pamäť obsahuje program aj dáta. Procesor z nej číta hodnoty jednotlivých buniek sekvenčne, ktorých význam rozpoznáva ako inštrukcie a tie potom vykonáva. Preto je potrebné, aby prvým krokom v emulácii bolo načítanie vyvíjaného programu (alebo externého(-ých) súboru(-ov), tzv. obrazu) do operačnej pamäte. Takže ak má používateľ k dispozícii už hotový, skompilovaný program vo formáte podporovanom operačnou pamäťou, môže písanie zdrojového kódu a jeho kompilovanie preskočiť.

V prípade, že virtuálna architektúra obsahuje aj prídavné zariadenia, teraz je vhodný čas nastaviť im parametre. Parametre môžu mať rôzny charakter, napr. pre terminál môže ísť o nastavenie parametra „always on top“, pre diskové zariadenie to môže byť namontovanie obrazov diskiet (virtuálne vloženie diskety do zariadenia). Zariadenia môžu podporovať aj uloženie/znovuačítanie svojich nastavení a preto niekedy netreba nič nastavovať, pretože si nastavenia zariadenie vie z danej architektúry zistiť samo.

Nezáleží na tom, či sa najskôr pripraví operačná pamäť a až potom parametre zariadení, alebo naopak. Emulácia je teraz pozastavená, preto príprava môže prebiehať v kľude, „bez stresu“ a bez nároku na poradie prípravných krokov. Výnimkou je však poradie načítavania obrazov do operačnej pamäte. Poradie nie je podstatné, ak sa obrazy v operačnej pamäti neprekrývajú. V opačnom prípade musí mať používateľ na zreteli, že oblasť, v ktorej obrazy kolidujú, bude mať platné hodnoty posledného obrazu, ktorý oblasť prekryl.

Poslednou aktivitou je *Spustenie emulácie*. Znova upozorňujem, že nejde o striktné stlačenie tlačidla *Run*. Používateľ sa môže rozhodnúť, či emuláciu bude krokovať, alebo emuláciu spustí. Pred samotným spustením môže používateľ nastaviť adresu v pamäti, od ktorej sa emulácia spustí (od ktorej začne procesor čítať inštrukcie).

[4] ak používateľ vôbec chce kód emulovať, pretože táto aktivita nie je povinná, aj keď podľa diagramu nemá inú možnosť



Obr. 1.2: *Základný postup pre vývoj a emulovanie na platforme*

Kapitola 2

Konfigurácie a virtuálne architektúry

Emulátor vďaka schopnosti akceptovať rôzne zásuvné moduly toho istého typu umožňuje používateľovi emulovať rôzne počítačové architektúry. Vychádzajúc pritom zo známej schémy Von-Neumanna, každá emulovaná architektúra musí obsahovať procesor, operačnú pamäť a voliteľne aj niekoľko prídavných zariadení.

Nová verzia emulátora so sebou prináša schopnosť tvoriť hierarchické prepojenia zariadení medzi sebou, operačnou pamäťou a procesorom. Používateľ prepojenia kreslí v abstraktnej schéme. Každý blok schémy predstavuje určitý zásuvný modul, resp. prvok v architektúre. Viac o vytváraní konfigurácií sa čitateľ dozvie v časti 2.2.

Konfigurácie sú uložené v konfiguračných súboroch, pričom každý súbor odpovedá jednej architektúre. Konfigurácie je samozrejme možné aj vytvárať a preto je možné vytvoriť aj takú konfiguráciu počítača, aká existovala (alebo existuje) v skutočnosti. Z toho vyplýva, že programy napísané pre pôvodný hardvér sa dajú spustiť aj na tejto platforme, ak jej konfigurácia odpovedá pôvodnej (a naopak).

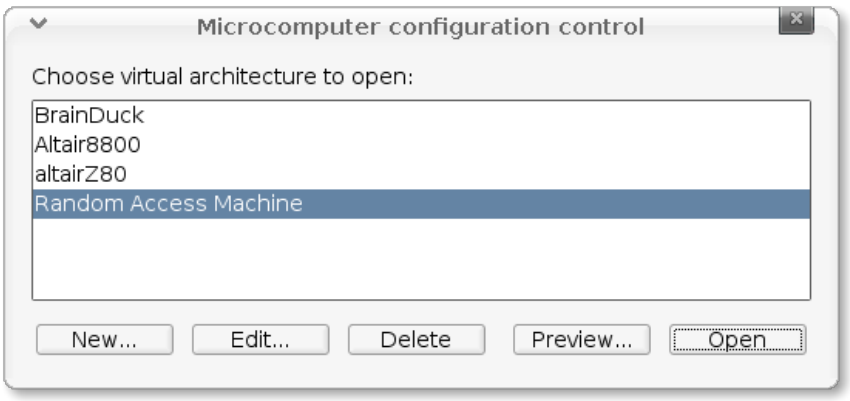
Pojem *virtuálna architektúra* definujem ako inštanciu konkrétnej konfigurácie počítača. Konfigurácie sú uložené v konfiguračných súboroch, pričom každý súbor odpovedá práve jednej konfigurácii.

Súčasná verzia emulátora dokáže (počas behu) pracovať len s jednou virtuálnou architektúrou. Z tohto dôvodu je potrebné *zvoliť si*, s ktorou. Používateľ volí konfiguráciu virtuálnej architektúry hneď po spustení programu.

2.1 Voľba konfigurácie

Okno voľby konfigurácie je možné vidieť na Obr. 2.1. Ide o klasický zoznam konfigurácií, pričom každá má svoj názov. Používateľ najprv klikne na názov konfigurácie a potom na tlačidlo *Open*. Ak okno používateľ zruší, zavrie sa aj celý program (voľba architektúry je nutná). V tomto okamihu sa začne načítavať konfigurácia a postupne z nej vytvárať virtuálna architektúra.

Proces sa začne zobrazením informačného okna (Obr. 2.2). Je aktivované po výbere architektúry a zmizne, keď je architektúra úspešne načítaná.



Obr. 2.1: *Voľba konfigurácie počítača*

Obr. 2.2: *Informačné okno - načítavanie konfigurácie*

POZNÁMKA:

Prípadné chyby pri vytvaraní architektúry (alebo prepojení) vyskakujú, len keď je toto okno aktívne, pričom je na ňom uvedené upozornenie, čo treba robiť, ak vyskočí chyba (overiť abstraktnu schému alebo samotné zásuvné moduly).

Hlavný modul pri vytváraní architektúry nekontroluje celú sémantiku abstraktnej schémy, teda nevie určiť, či schéma bude fungovať. Počas tohto procesu môžu vzniknúť chyby dvojakého druhu:

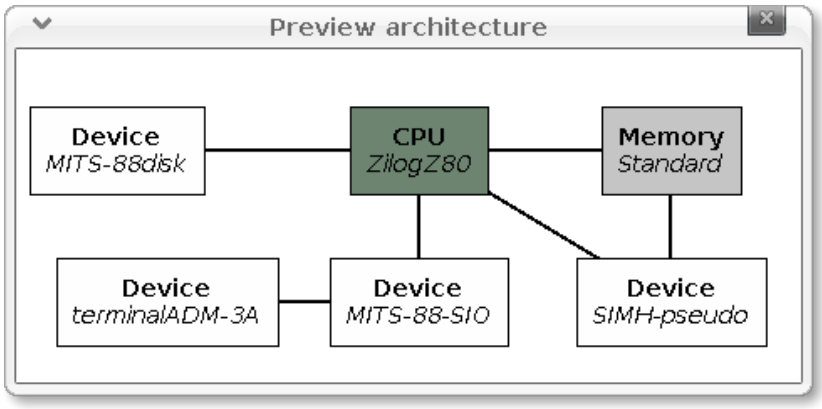
- Chyby pri načítavaní zásuvných modulov (neexistujúci súbor zásuvného modulu, ...)
- Chyby pri realizácii prepojení (ak sa zásuvné moduly medzi sebou nedajú prepojiť v žiadnom smere)

Ak sa objaví ľubovoľná chyba, virtuálna architektúra sa prestane vytvárať a emulátor sa ukončí. Ak všetko prebehne v poriadku, tak sa konfigurácia premení na virtuálnu architektúru, ktorá sa už dá emulovať. Počas behu emulátora sa už nedá meniť architektúra, ani prepínať medzi inými architektúrami.

2.2 Vytváranie a editácia konfigurácií

Konfigurácie je samozrejme možné aj vytvárať a preto je možné vytvoriť aj takú konfiguráciu počítača, aká existovala (alebo existuje) v skutočnosti. Z toho vyplýva, že programy napísané pre pôvodný hardvér sa dajú spustiť aj na tejto platforme, ak jej konfigurácia odpovedá pôvodnej (a naopak).

Oproti predošlej verzii sa vytváranie konfigurácií zásadne líši. Používateľ namiesto strohého výberu zásuvných modulov teraz musí nakresliť abstraktnú schému celej architektúry. Schéma je v podstate „pekný" graf, skladajúci sa z *uzlov* - typových zásuvných modulov a z *hrán* - prepojení medzi nimi. Príklad abstraktnej schémy je možné vidieť na Obr. 2.3



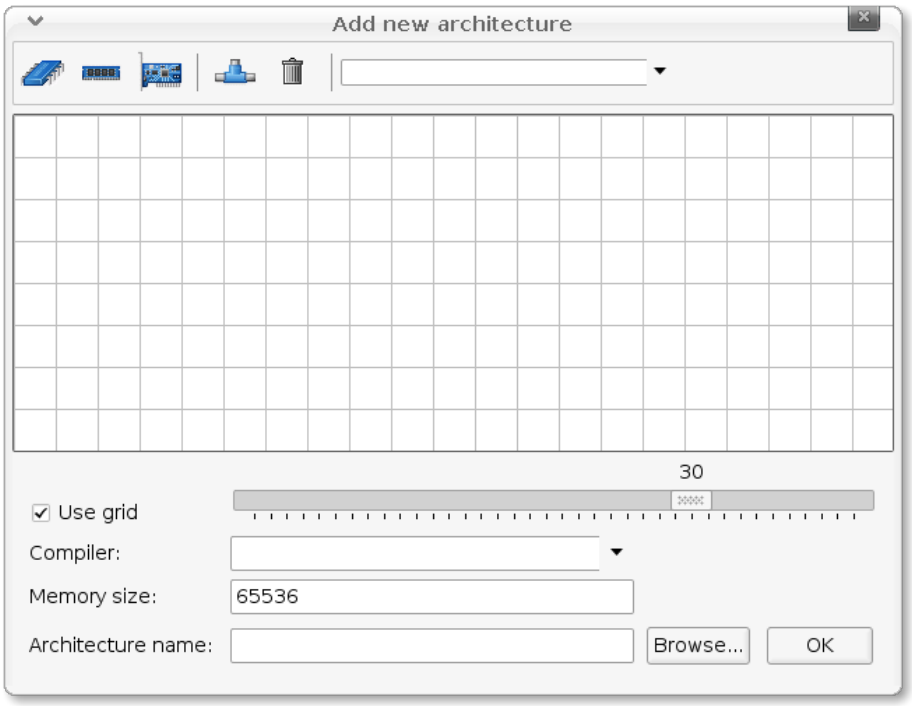
Obr. 2.3: *Príklad abstraktnej schémy*

Emulátor rozoznáva štyri typy zásuvných modulov: kompilátory, CPU, operačné pamäte a zariadenia. Všetky tieto typy sa v abstraktnej schéme môžu nachádzať. Z toho okrem zariadení, ktorých môže byť 0 a viac, sa každý typ zásuvného modulu v schéme môže nachádzať len raz. To znamená práve jeden kompilátor, práve jedna CPU, a práve jedna operačná pamäť. Editor abstraktných schém nedovolí v schéme použiť väčší, ani menší počet týchto zásuvných modulov.

V schémach neexistujú žiadne zbernice. Cieľom bolo vyhnúť sa schémam a prepojeniam zasahujúcich do viac elektrotechnického charakteru. Je lepšie, aby mali schémy viac logický charakter (formu Von-Neumanovskej schémy bez zbernice). Aj takto je stále možné prepájať skoro ľubovolne medzi sebou CPU, zariadenia a operačnú pamäť. Aj keď to editor schém nevyžaduje, väčšinou vždy existuje prepojenie medzi CPU a operačnou pamäťou, ktoré treba explicitne vytvoriť.

2.2.1 Vytváranie nových konfigurácií

Novú architektúru vytvoríme kliknutím na tlačidlo *New* v okne voľby konfigurácie (Obr. 2.1). Zobrazí sa editor abstraktnej schémy, v ktorom používateľ nakreslí schému architektúry, vyberie kompilátor, nastaví veľkosť operačnej pamäte a samozrejme tiež názov novej konfigurácie. Okno editora je možné vidieť na Obr. 2.4.



Obr. 2.4: *Editor novej konfigurácie*

2.2.1.1 Mriežka

Kreslenie schém je možné trochu uľahčiť použitím mriežky (*Use grid*), čo spôsobí uchytenie kreslených prvkov na pravidelné pozície kresliacej plochy. Mriežka sa zobrazuje vo fome križujúcich sa vodorovných a zvislých čiar na pozadí. Pevné body, v ktorých sa prvky uchycujú, sú priesečníkmi týchto čiar. Veľkosť mriežky je možné meniť (posúvací panel napravo od checkboxu *Use grid*, Obr. 2.4) a tým sa zvyšuje/znižuje hustota pevných bodov na kresliacej ploche. V prípade, že sa mriežka nepoužije, pozícia prvkov na ploche nie je korigovaná.

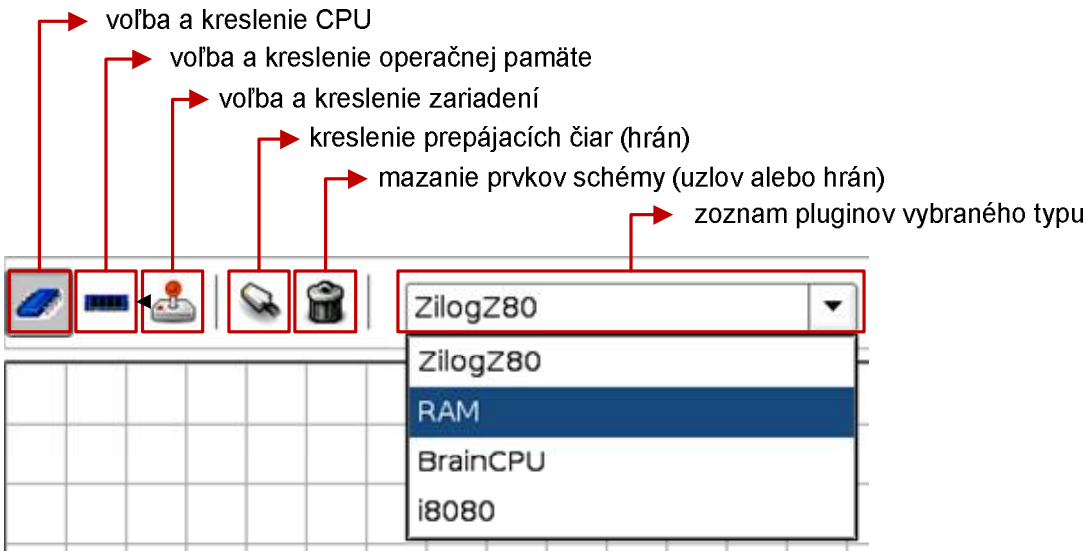
2.2.1.2 Voľba a kreslenie prvkov schémy

Panel nástrojov editora schém je možné vidieť na Obr. 2.5

Používateľ umiestňuje na kresliacu plochu vybrané prvky schémy - CPU, operačnú pamäť a zariadenia. Postup pri ich kreslení je jednoduchý: najprv je potrebné vybrať, čo ideme kresliť (čiže kliknúť na príslušnú ikonu na nástrojovom paneli, Obr. 2.5). Na to sa aktualizuje zoznam dostupných zásuvných modulov pre vybraný typ prvku.

POZNÁMKA:

Aby sa pri kreslení schémy dali použiť všetky zásuvné moduly vybraného typu, musia byť uložené v príslušných podadresároch adresára, kde je program nainštalovaný (viď časť Inštalácia programu na str. 2).



Obr. 2.5: *Panel nástrojov editora konfigurácie*

Používateľ si zo zoznamu vyberie už konkrétny zásuvný modul, ktorý chce použit'a umiestni ho do schémy jedným kliknutím ľavým tlačidlom. Tým istým postupom umiestni do schémy všetky ostatné prvky.

Pri voľbe operačnej pamäte je tiež potrebné určit' jej veľkost' - túto potom už nie je možné dynamicky menit' (po vytvorení virtuálnej architektúry). Osembitové počítače mali túto veľkost' štandardne od $64 \div 256$ *kB*. Veľkost' operačnej pamäte sa zadáva v spodnej časti okna (*Memory size*, Obr. 2.4).

Názvy zásuvných modulov v zozname dostupných modulov sú vlastne ich názvy súborov bez prípony, a teda neodpovedajú názvom v hlavnom okne (po vytvorení virtuálnej architektúry). Je to tak preto, lebo interné názvy zariadení ešte nie sú známe (zásuvné moduly ešte neboli načítané).

2.2.1.3 Presúvanie prvkov

V prípade, že používateľ chce zmenit' pozíciu niektorého prvku na schéme, musí najprv zrušit' funkciu vytvárania prvku - „odkliknutím“ vybranej ikony z nástrojového panela[1]. Tým sa zruší výber dostupných zásuvných modulov a teraz môže používateľ prvky presúvat'. Prvky sa presúvajú metódou „drag & drop“ - klikneme na prvok ľavým tlačidlom myši, a za súčasného držania tlačidla presunieme prvok na požadované miesto. Potom tlačidlo pustíme a prvok už ostane na novom mieste (v prípade, že je použitá mriežka, sa prvok umiestni na najbližší pevný bod).

[1] v skutočnosti musí byt' zrušená každá funkcia - vytváranie prvkov, mazanie, kreslenie prepájacích čiar

2.2.1.4 Prepájacie čiary

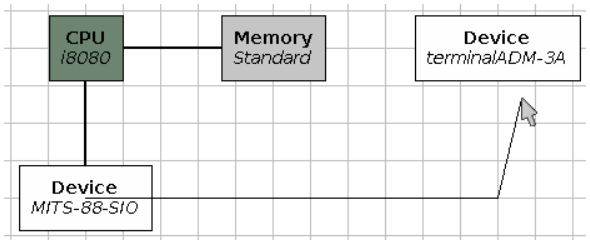
Po umiestnení prvkov do schémy nasleduje ich vzájomné prepájanie prepájacími čiarami (hranami). Najprv je potrebné kliknúť na správnu ikonu nástrojového panela (Obr 2.5). Prepájacia čiara sa skladá nutne z dvoch koncov, ktorými sú prepájané uzly a môže sa skladať aj z ďalších pomocných bodov, ktoré lomia čiaru na viac častí.

Čiary sa nekreslia metódou „drag & drop“. „Rovné“ prepojenia vznikajú jedným kliknutím ľavým tlačidlom myši na prvý prvok a potom ďalším kliknutím na druhý prvok. Pozícia čiary je korigovaná automaticky tak, že jej konce sú umiestnené v strede pod prepojenými prvkami. Pritom nezáleží na tom, od ktorého prvku sa začne čiara kresliť, pretože používateľ v schéme (ani nikde inde) nedefinuje smer prepojenia.

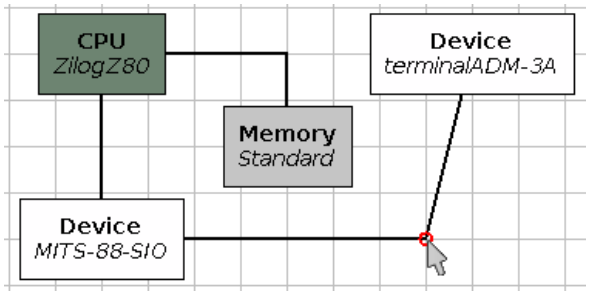
POZNÁMKA:

Emulátor na začiatku predpokladá obojsmerné prepojenie a pri vytváraní virtuálnej architektúry ho skúša realizovať. Ak sa mu to nepodarí, tak skúša realizovať aspoň jednosmerné prepojenie (najprv zapája prvý prvok (prvý koniec čiary) do druhého, ak sa to nepodarí, tak opačne). Ak sa nepodarí prepojiť zariadenia vôbec (ani obojsmerne, ani jednosmerne), vyskočí chybové hlásenie a emulátor sa ukončí.

Počas vytvárania prepájacej čiary môže používateľ definovať pomocné body čiary, od ktorých jedna čiara môže pokračovať iným smerom. Vznikne tak lomená čiara. Vytváranie lomených čiar je zobrazené na Obr. 2.6. Pomocné body pri vytváraní prepájacej čiary vytvárame jedným kliknutím ľavým tlačidlom myši na miesto, kde bod má ležať.



Obr. 2.6: *Ukážka kreslenia lomenej prepájacej čiary*



Obr. 2.7: *Ukážka presúvania pomocného bodu*

Ak nie je v nástrojovom paneli vybratá žiadna funkcia, je možné pomocné body presúvať metódou „drag & drop“ - klikneme na bod ľavým tlačidlom myši, a počas súčasného držania tlačidla presunieme bod inam. Bod, ktorý presúvame je zvýraznený - okolo neho je nakreslený červený krúžok (Obr. 2.7).

Pomocné body sa dajú vytvárať aj keď je už prepájacia čiara vytvorená. Stačí kliknúť ľavým tlačidlom myši na ľubovoľné miesto na čiare[2] a počas súčasného držaním tlačidla pohnúť myšou. Na kliknutom mieste sa vytvorí nový pomocný bod, ktorý súčasným pohybom presúvame na požadované miesto.

[2] všetky funkcie nástrojového panela musia byť neaktívne

Rušenie pomocných bodov je jednoduché - na pomocný bod, ktorý chceme zrušiť klikneme raz pravým tlačidlom myši. Po zrušení pomocného bodu sa čiara „narovná“ - priamou úsečkou sa spoja body ležiace bezprostredne pred a za rušeným bodom.

2.2.1.5 Mazanie prvkov

V prípade, že chce používateľ odstrániť nejaký prvok alebo prepájaciu čiaru zo schémy, musí z panelu nástrojov vybrať funkciu mazania. Táto funkcia sa však nepoužíva na mazanie pomocných bodov lomenej čiary. Jedným kliknutím ľavým tlačidlom myši na prvok sa tento zo schémy odstráni. Ak odstraňujeme uzol (nie prepájaciu čiaru), odstránia sa aj všetky prepojenia s odstraňovaným prvkom.

2.2.2 Editovanie konfigurácie

Editovanie, alebo úprava existujúcej konfigurácie znamená úpravu alebo zmenu abstraktnej schémy, výberu kompilátora a veľkosti operačnej pamäte bez zmeny jej názvu. Editor použitý na úpravu existujúcej konfigurácie však neumožňuje zmenu nastavení pre jednotlivé zásuvné moduly. Ide totiž o rovnaký editor, aký sa používa pre vytváranie novej konfigurácie. Preto pre jeho bližší opis nech čitateľ pozrie časť 2.2.1.2.

Editovanie existujúcej konfigurácie aktivujeme v okne voľby konfigurácie (Obr. 2.1) kliknutím na tlačidlo *Edit*.

Kapitola 3

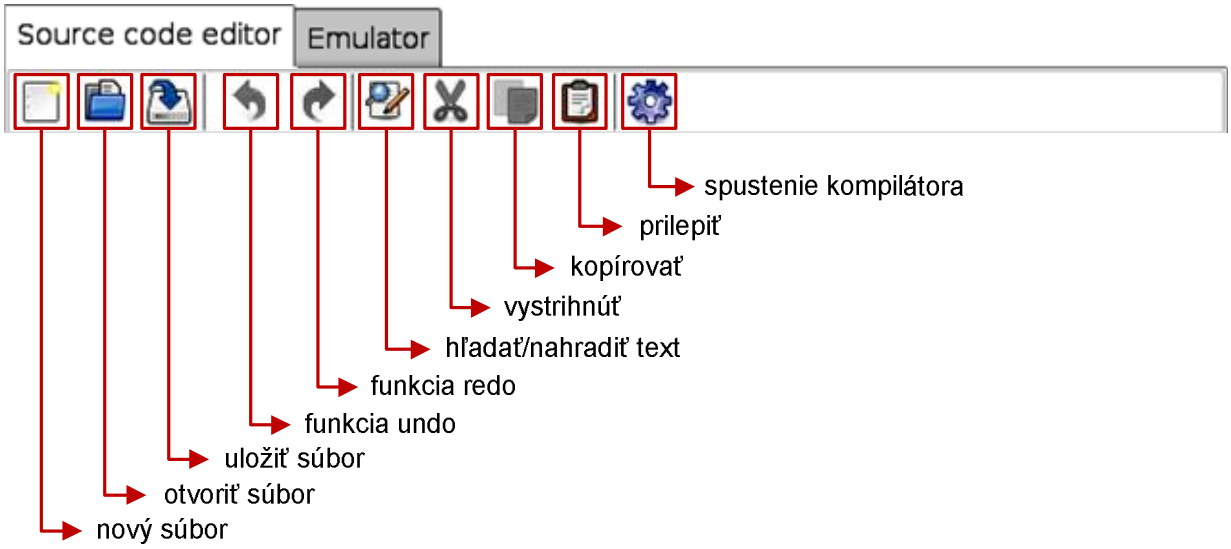
Textový editor

3.1 Práca v textovom editore

Ako už bolo spomenuté, textový editor slúži na písanie zdrojových kódov. V konečnom dôsledku použitý byť vôbec nemusí — môžme emulovať kód, ktorý už je skompilovaný.

Písanie textu je veľmi jednoduché. Textový editor automaticky podporuje zvýrazňovanie syntaxe jazyka zvoleného kompilátora, vyznačuje čísla riadkov. Hlásenia kompilátora sú tak prehľadnejšie (riadok a stĺpec sa dá rýchlejšie nájsť v zdrojovom kóde). Funkcie `undo` (vráti späť jeden napísaný znak) a `redo` (znova napíše znak, ktorý bol predtým vrátený) sú samozrejmosťou.

Napísaný kód nie je možné skompilovať, pokiaľ nebol predtým uložený. Nástrojový panel textového editora aj s popisom jednotlivých ikon je možné vidieť na Obr. 3.1.



Obr. 3.1: *Nástrojový panel textového editora*

Nástrojový panel na Obr. 3.1 obsahuje ikony reprezentujúce iba tie základné, najpoužívanejšie funkcie textového editora a spoluprácu s kompilátorom. Všetky funkcie programu zahŕňajúce aj funkcie spomínaného nástrojového panela, sú k dispozícii v ponuke programu — textové menu umiestnené celkom navrchu okna.

Niektoré funkcie (najčastejšie používané) je možno spustiť aj klávesovými skratkami, ako to ukazuje Tab. 3.1.

Názov funkcie	**Umiestnenie v ponuke**	**Skratka**
Nový súbor	File → New	`CTRL+N`
Otvoriť súbor	File → Open	`CTRL+O`
Uložiť súbor	File → Save	`CTRL+S`
Späť	Edit → Undo	`CTRL+Z`
Dopredu	Edit → Redo	`CTRL+Y`
Vystrihnúť výber textu	Edit → Cut selection	`CTRL+X`
Kopírovať výber textu	Edit → Copy selection	`CTRL+C`
Prilepiť do výberu textu	Edit → Paste selection	`CTRL+V`
Hľadať/nahradiť text	Edit → Find/replace text	`CTRL+F`
Hľadať ďalej	Edit → Find next	`F3`
Nahradiť ďalej	Edit → Replace next	`F4`

Tabuľka 3.1: *Klávesové skratky v hlavnom module*

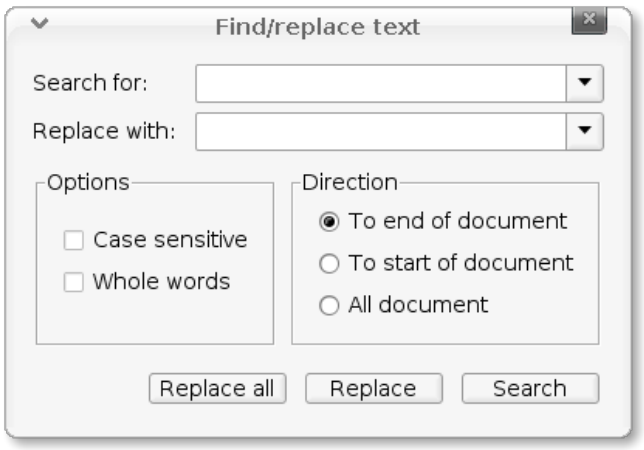
Význam väčšiny spomenutých funkcií je intuitívne jasný, preto sa budeme ďalej zaoberať menej jasným funkciám alebo funkciám, ktorých význam je širší.

3.2 Vyhľadávanie, nahrádzanie textu

Vyhľadávanie/nahrádzanie textu vie pracovať v dvoch režimoch: klasický a rýchly.

V *klasickom* režime po aktivácii funkcie vyhľadávania/nahrádzania sa zobrazí dialógové okno (Obr. 3.2).
V tomto okne používateľ zadáva/vyberá:

- povinne text, ktorý chce nájsť/zameniť (textové okno *Search for*)
- v prípade nahrádzania textu povinne text, ktorý sa použije ako náhrada (textové okno *Replace with*)
- v sekcii *Options* sa vyberajú detaily vyhľadávania — podpora rozlišovania veľkosti znakov (*Case sensitive*) a či hľadaný text predstavuje samostatné slovo (*Whole words*)
- sekcia *Direction* umožňuje vybrať jeden z troch typov smerov vyhľadávania:
  - smerom ku koncu dokumentu (*To end of document*)



Obr. 3.2: *Dialógové okno vyhľadávania/nahrádzania textu*

- smerom k začiatku dokumentu (*To start of document*)
- prehľadá sa celý dokument, začne sa smerom ku koncu dokumentu; ak sa hľadaný text nenájde, pokračuje hľadaním od začiatku (*All document*)

Výber samotnej funkcie je realizovaný kliknutím na jedno z troch tlačidiel: Vyhľať (*Search*), Nahradiť (*Replace*) alebo Nahradiť všetko, čo sa nájde vo vybranom smere vyhľadávania (*Replace All*). Vykonanie vybranej funkcie je jednorázové; dialógové okno sa následne zavrie.

Hľadaný/nahrádzajúci text môže mať aj tvar regulárneho výrazu (teda akéhosi vzoru, podľa ktorého by mal vyzerať hľadaný/nahrádzajúci text). V implementácii je použitý java-ovský formát regulárnych výrazov. Podrobnejšie sa štruktúra regulárnych výrazov a práca s nimi rozoberá tu: `http://java.sun.com/docs/books/tutorial/essential/regex/index.html`

Ako príklad vyhľadávania ľubovoľného reťazca uzavretého v jednoduchých úvodzovkách `''`, by sme napísali napr. takýto vyhľadávací text:

```
'([^']*)'
```

Preto ak chceme vyhľadať text, ktorý obsahuje špeciálne riadiace znaky regulárnych výrazov, je potrebné dať pred nich spätné lomítko \.

Špeciálne znaky sú tieto:

```
$ ^ . * + ? [ ] ( ) " \
```

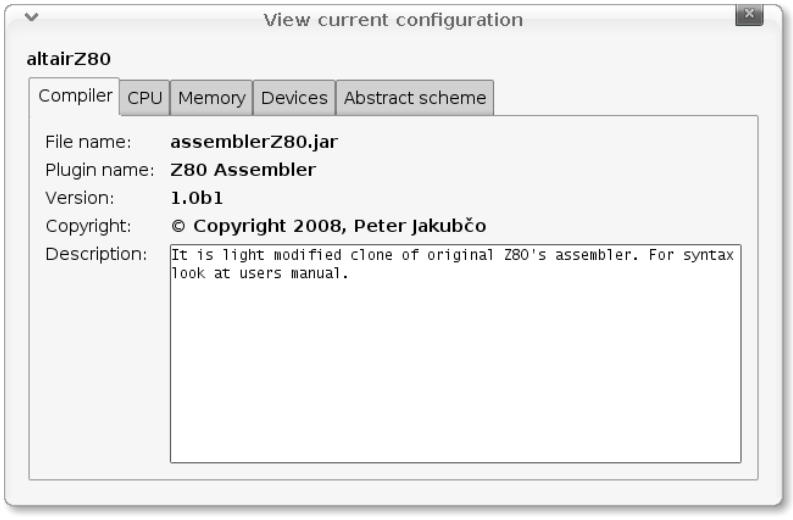
Práca vyhľadávania/nahrádzania textu v *rýchlom* režime predstavuje rýchle vyvolanie funkcie vyhľadávania alebo nahrádzania textu, pričom sa nezobrazí žiadne dialógové okno a parametre vyhľadávania/nahrádzania sa použijú presne také isté ako pri poslednej takejto funkcii. Rýchly režim sa vyvoláva položkou z menu *Edit* alebo klávesovými skratkami, ako je uvedené v Tab. 3.1 v položkách *Edit* → *Find Next* resp. *Edit* → *Replace Next*. Klasický režim však vždy predchádza rýchlemu režimu, v prípade ak bol rýchly režim vyvolaný pred použitím klasického režimu, klasický režim sa automaticky vyvolá sám.

Kapitola 4

Emulátor

4.1 Prehľad aktuálnej konfigurácie

Konfigurácia zvolená na začiatku programu sa už počas jeho behu meniť nedá. Je však možnosť si ju podrobnejšie prezrieť, zmysel to má pri pozeraní detailov jednotlivých zásuvných modulov, ako je napr. verzia zásuvného modulu alebo abstraktná schéma virtuálnej architektúry. Okno zobrazujúce vybranú konfiguráciu sa aktivuje položkou v menu *Project* → *View configuration*. Okno je zobrazené na Obr. 4.1

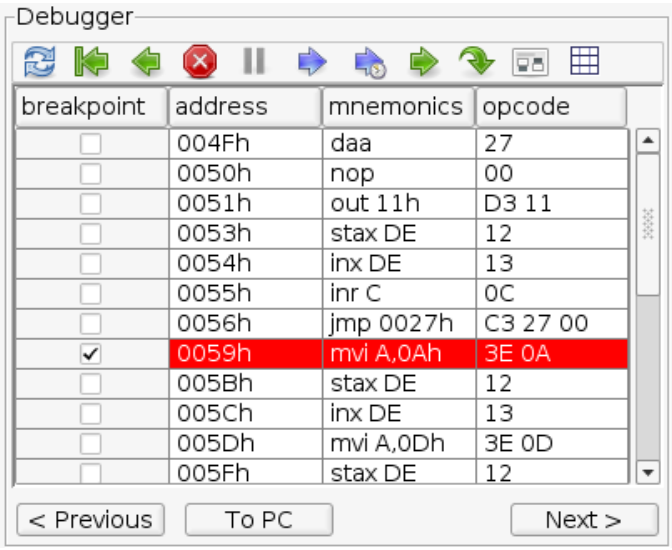


Obr. 4.1: *Dialógové okno zobrazujúce aktuálnu konfiguráciu*

4.2 Okno debuggera

Panel emulátora obsahuje tri základné časti, ako boli popísané na začiatku kapitoly 1. V tejto časti sa budeme zaoberať oknom debuggera. Pri štúdiu práce emulovaného procesora je to azda najdôležitejšie okno, ktoré poskytuje hlavne informáciu o tom, ktorá inštrukcia pri vykonávaní práve nasleduje. Rôzne implementácie rôznych procesorov môžu v tomto okne zobrazovať aj iné informácie, určite vždy však pôjde o akýsi zoznam zložený z riadkov a stĺpcov.

Na Obr. 4.2 je zobrazené okno procesora Intel 8080 zobrazujúce inštrukcie programu.



Obr. 4.2: *Okno debuggera „v akcii“*

Všeobecne nejde o nič iné, ako o zobrazenie obsahu operačnej pamäte v určitej oblasti, v určitom rozsahu (25 riadkov = 25 inštrukcií). Každá inštrukcia je uložená, rovnako ako aj dáta, v operačnej pamäti. Inštrukcie spomínaného procesora Intel 8080 zaberajú v operačnej pamäti od jedného po tri bajty. Z tohto dôvodu na Obr. 4.2 adresy sa nezväčšujú stále o rovnakú hodnotu. V prípade, že nasledujúca adresa je o napr. dve čísla väčšia ako aktuálna, znamená to, že inštrukcia na aktuálnej adrese zaberá dva bajty. Všimnite si na Obr. 4.2 napr. riadok s adresou $0022h$. Na tejto adrese v operačnej pamäti je uložená inštrukcia `in 08h`. Táto inštrukcia zaberá dva bajty: `DB 08`. V takomto tvare je inštrukcia zobrazená v stĺpci *opcode*, ako aj v operačnej pamäti.

Implementovaná CPU procesora Intel 8080 umožňuje pozastaviť emuláciu, ak programové počítadlo (`PC`) dosiahne určitú adresu. Takýto bod pozastavenia sa nazýva *breakpoint* a v okne ho možno jednoducho nastaviť kliknutím na takto označený stĺpec v príslušnom riadku (viď riadok s adresou $0022h$ na Obr. 4.2). Potom v prípade, že spustíme emuláciu (bez krokovania), v tomto bode sa emulácia pozastaví a používateľ tak môže sledovať stav CPU pred spustením inštrukcie nachádzajúcej sa na adrese pozastavenia.

4.2.1 Stránkovanie inštrukcií

Nová verzia emulátora prináša možnosť stránkovania zoznamu inštrukcií v okne debuggera. Ide o schopnosť, pomocou ktorej sa používateľ vie pozrieť na ďalšie inštrukcie, nezobrazené v aktuálnom zozname inštrukcií. Prvá stránka je zoznam inštrukcií, v ktorom sa nachádza aktuálna inštrukcia. Prepínanie stránok dopredu resp. dozadu spôsobí posun zoznamu inštrukcií dopredu resp. dozadu.

K dispozícii má používateľ tlačidlá:

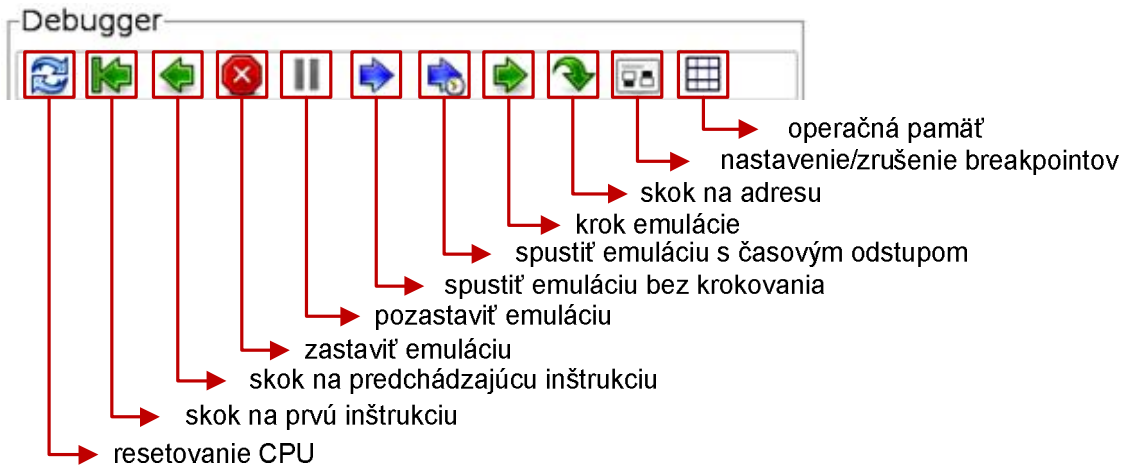
- *Next* - Zobrazí nasledujúcu stránku (ak sme na poslednej stránke, nič sa nestane). Pár inštrukcií na konci zoznamu na aktuálnej stránke sa budú zhodovať zo začiatočnými inštrukciami na nasledujúcej stránke, teda aktuálna stránka nekončí na mieste, kde potom začína nasledujúca. Má to výhodu v tom, že ak používateľ sleduje inštrukcie v nejakej postupnosti, časť tejto postupnosti sa zachová aj na ďalšej stránke („nestratí nit", tj. sa nemusí vracať o stránku späť a pozerať na poslednú inštrukciu).

- *Previous* - Zobrazí predchádzajúcu stránku (ak sme na prvej stránke, nič sa nestane). Pár inštrukcií na konci zoznamu na predchádzajúcej stránke sa budú zhodovať zo začiatočnými inštrukciami na aktuálnej stránke, teda predchádzajúca stránka nekončí na mieste, kde potom začína aktuálna. Je to z podobného dôvodu, ako pri zobrazovaní inštrukcií nasledujúcich stránok.

- *To PC* - Vráti sa na prvú stránku (tam, kde sa nachádza aktuálna inštrukcia)

4.2.2 Ovládanie emulácie

Nad samotným oknom zobrazujúcim inštrukcie programu sa nachádza panel na ovládanie emulácie. Tento panel, aj s krátkym popisom, je zobrazený na Obr. 4.3.



Obr. 4.3: *Panel ovládania emulácie*

Funkcia **Reset CPU** môže mať pre každú CPU iný význam. Vo všeobecnosti je to funkcia, ktorá vráti CPU do počiatočného stavu, teda do stavu, v akom sa nachádzala pred prvým spustením emulácie. Pre reálne 8 bitové CPU to väčšinou znamená inicializácia registrov, príznakov a nastavenie programového počítadla `PC` na nejakú počiatočnú hodnotu, väčšinou 0. Táto funkcia má v reálnych počítačoch skôr širší význam. Napríklad stlačením tlačidla reset na počítači sa neinicializuje iba CPU, ale aj všetky ostatné zariadenia a iné obvody v priamom kontakte s CPU. Obsah operačnej pamäte však ostáva zachovaný.

Ďalšia sada funkcií zobrazených na paneli na Obr. 4.3 vyzerá ako ovládanie nejakého prehrávača. Používateľ už intuitívne môže zistiť, že sa jedná o priame ovládanie emulácie.

Funkcia **nastavenia PC na začiatok** znovu inicializuje programové počítadlo na počiatočnú hodnotu. Niektoré CPU môžu PC inicializovať na hodnotu adresy výskytu prvej inštrukcie programu (pokiaľ je táto adresa dobre známa).

Nasledujúca funkcia **dekrementácie PC** odpočíta od programového počítadla jednotku. Nemení registre, ani výsledok predchádzajúcej inštrukcie. Dekrementácia `PC` môže, ale nemusí znamenať posun o inštrukciu späť. Ak bola predchádzajúca inštrukcia na adrese a veľkosti napr. 3 bajty, `PC` sa dekrementuje a bude ukazovať na inštrukciu (možno známu, možno nie) na adrese $a - 1$. To spôsobí zmenu v chápaní aj nasledujúcich inštrukcií, pretože známa inštrukcia začínajúca na adrese $a - 1$ nemusí nutne zaberať 2 bajty (aby sa posun vyrovnal). Situácia je zobrazená v Tab. 4.1. Ak chceme posunúť `PC` späť o celú inštrukciu, musíme zavolať funkciu dekrementácie `PC` toľkokrát, aká je inštrukcia veľká v bajtoch.

		Interpretácia inštrukcií			
Addresa (hex)	**Hodnota bajtu (hex)**	**pre PC = 0**	**Op. kód**	**pre PC = 1**	**Op. kód**
`0000`	`3E`	mvi A, 06h	`3E 06`	—	
`0001`	`06`			mvi B, 04h	`06 04`
`0002`	`04`	inr B	`04`		
`0003`	`0B`	dcx BC	`0B`	dcx BC	`0B`

Tabuľka 4.1: *Ukážka rozdielnej interpretácie inštrukcií pri nepatrnom posunutí PC*

POZNÁMKA:
Ak nastane takáto zmena v interpretácii, prejaví aj v okne debuggera, to znamená, že inštrukcie sa začnú interpretovať od aktuálnej adresy dozadu, aj dopredu.

Funkcia **zastavenia emulácie** zastaví prácu CPU bez možnosti pokračovania. Stav CPU ostane zachovaný. Jediným možným východiskom z tejto situácie (ako je možné vidieť na Obr. 1.2) je resetovanie CPU, čo spôsobí jej re-inicializáciu. Do tohto stavu príde CPU veľakrát aj automaticky, dôvodov môže byť niekoľko:

- inštrukciou programu (`halt`)
- výpadkom inštrukcie (ak CPU narazí na neznámu alebo neúplnú inštrukciu)
- výpadkom pamäte (ak sa program pokúša pristúpiť k pamäti mimo jej adresnej oblasti)

Pozastavením emulácie pozastavíme prácu CPU, s možnosťou ďalšieho pokračovania. Situácia je podobná, ako keby sme emulácii zastavili úplne, len s tým rozdielom, že je ešte možné v nej pokračovať. Ak CPU narazí na bod pozastavenia (*breakpoint*), dostane sa presne do tohoto stavu. Zavolanie tejto funkcie je ako vyvolanie umelého bodu pozastavenia.

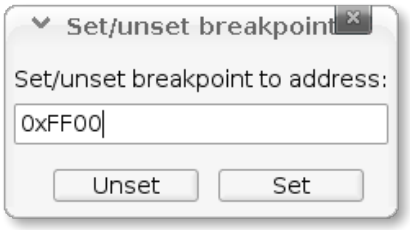
Spustením emulácie ju spustíme od miesta, na ktoré ukazuje programové počítadlo. Zo stavom CPU sa nič nerobí. Emuláciu nie je možné spustiť, ak je CPU zastavená úplne. V prípade, že emuláciu spustíme týmto spôsobom, prestanú sa automaticky zobrazovať inštrukcie v okne debuggera. Je to z dôvodu urýchlenia emulácie — implementácia samotného spustenia emulácie môže byť iná (výkonovo rádovo rýchlejšia, ale bez možnosti podrobnej interakcie), ako implementácia krokovania. Procesy tu prebiehajú veľmi rýchlo, plynule - tak, ako keby sme spustili skutočný počítač. Niektoré CPU môžu implementovať presnú časovú synchronizáciu, aby bolo možné regulovať rýchlosť (frekvenciu) CPU.

Ďalšou funkciou je **krok** emulácie. Po vyvolaní tejto funkcie CPU dostane priestor na vykonanie jediného kroku (inštrukcie) programu. Zmeny stavu CPU spôsobené vykonanou inštrukciou sa samozrejme zobrazia. Po vykonaní kroku sa štandardne CPU uvedie do stavu pozastavenia (teda ak vykonaný krok nespôsobil úplné zastavenie emulácie).

Ak je emulácia pozastavená, má zmysel uvažovať o „ručnom skoku“ na nejakú adresu. Tento skok znamená v podstate prepísanie hodnoty programového počítadla na hodnotu zadanú používateľom. Po kliknutí na funkciu **skoku** sa zobrazí dialógové okno, kde je používateľ vyzvaný k zadaniu adresy, na ktorú sa má „skočiť“. Okrem programového počítadla `PC` sa nič iné v CPU nemení.

4.2.3 Breakpointy na neviditeľnej adrese

Body pozastavenia, alebo *breakpointy* sa dajú nastaviť aj na „neviditeľnú“ adresu. Kliknutím na príslušnú ikonu na Obr. 4.3 sa zobrazí dialógové okno (Obr. 4.4) s výzvou na zadanie adresy operačnej pamäte, na ktorej sa nastaví/zruší breakpoint.

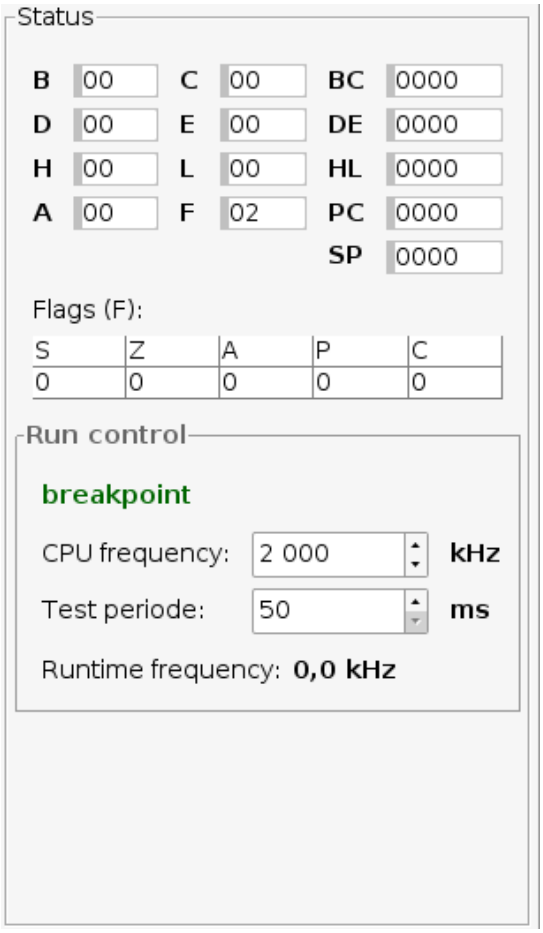


Obr. 4.4: *Dialógové okno na nastavenie/zrušenie breakpointu*

Čísla adries sa dajú zapisovať vo formátoch: dekadický (klasické čísla od $0-9$), hexadeximálny (tvar 0x*číslo*, kde *čislo* je od $0-F$), osmičkový (tvar 0*číslo*, kde *číslo* je od $0-7$). Takýto zápis je možný vo všetkých výzvach v celom hlavnom module.

4.3 Stavové okno

Stavové okno zobrazuje stav CPU. Podľa implementácie CPU sa toto okno aktualizuje, keď nie je emulácia spustená permanentne. Príklad zobrazenia stavu CPU procesora Intel 8080 ukazuje Obr. 4.5.



Obr. 4.5: *Stavové okno procesora Intel 8080*

Okno na Obr. 4.5 je rozdelené na dve časti. Horná časť zobrazuje obsah všetkých registrov a príznakov procesora. Spodná časť umožňuje používateľovi manipulovať s frekvenciou CPU a umožňuje tak koordinovať jej rýchlosť.

Bližšie informácie o význame jednotlivých registrov a príznakov si môže čitateľ nájsť v nejakej príručke k procesoru Intel 8080.

Teraz trochu popíšem spôsob, akým sa nastavuje rýchlosť CPU. Keďže práca CPU je diskretizovaná, každý jej krok sa udeje v určitom diskrétnom čase. Počet krokov za určitú periódu sa nazýva frekvencia CPU. Čím je počet krokov za jednu periódu väčší, tým CPU pracuje rýchlejšie. Tieto kroky sú elementárne, každá inštrukcia sa skladá z niekoľkých takýchto krokov. Na

nastavenie frekvencie CPU stačí, ak používateľ zadá hodnotu od 100 do 99999 kHz do vyznačeného textového políčka. Nastaviť frekvenciu je možné iba keď je emulácia pozastavená, alebo zastavená. Pri krokovaní emulácie nastavovanie frekvencie nemá význam, pretože je náročné a zbytočné presne simulovať rýchlosť vykonania jedinej inštrukcie. Ak je emulácia spustená, zobrazuje sa frekvencia, s akou CPU pracuje v reálnom čase (*Runtime frequency*).

Má to význam preto, lebo zabezpečenie frekvencie sa deje na základe vzorkovania s konštantnou vzorkovacou periódou a teda presnosť zachovania frekvencie je dané touto vzorkovacou periódou. Čím je perióda menšia, tým častjšie sa koná kontrola, resp. usmerňovanie frekvencie na požadovanú hodnotu. Táto vzorkovacia perióda sa tiež dá nastaviť, v textovom poli označenom ako *Test periode*. Čím je jej hodnota menšia, tým častejšie prebieha vzorkovanie a tým je frekvencia presnejšia. Pre nižšie frekvencie sa odporúča, aby táto hodnota bola nastavená na nižšie hodnoty, a naopak pre vyššie frekvencie na vyššie hodnoty (vzorkovanie takisto stojí určitý výkon).

4.4 Okno zariadení

Okno zariadení, ktoré je možné vidieť na Obr. 1.1 vpravo, je iba akýmsi zoznamom obsahujúcim všetky načítané zariadenia v danej konfigurácii. Zariadenia môžu podporovať svoje vlastné, individuálne funkcie. Tieto funkcie sa aktivujú interaktívne pomocou grafického rozhrania. Na zobrazenie tohto GUI stačí kliknúť na tlačidlo *Show*. Ak ho zariadenie nepodporuje, mala by vyskočiť chybová hláška.

Časť II

Popis zásuvných modulov

Kapitola 5

Operačná pamäť

Táto kapitola sa zaoberá popisom zásuvného modulu operačnej pamäte s názvom *Standard linear byte operating memory with variable size*, súbor `Standard.jar` a verzia `2.8b1`.

Svojou povahou spĺňa požiadavky pre emuláciu reálnych počítačov, ktoré využívajú operačnú pamäť, ktorej:

- adresy rastú lineárne
- jedna bunka má veľkosť jedného bytu (8 bitov)

Nezáleží pritom, či CPU používa malý alebo veľký bytový endián. Ďalšie charakteristiky zásuvného modulu:

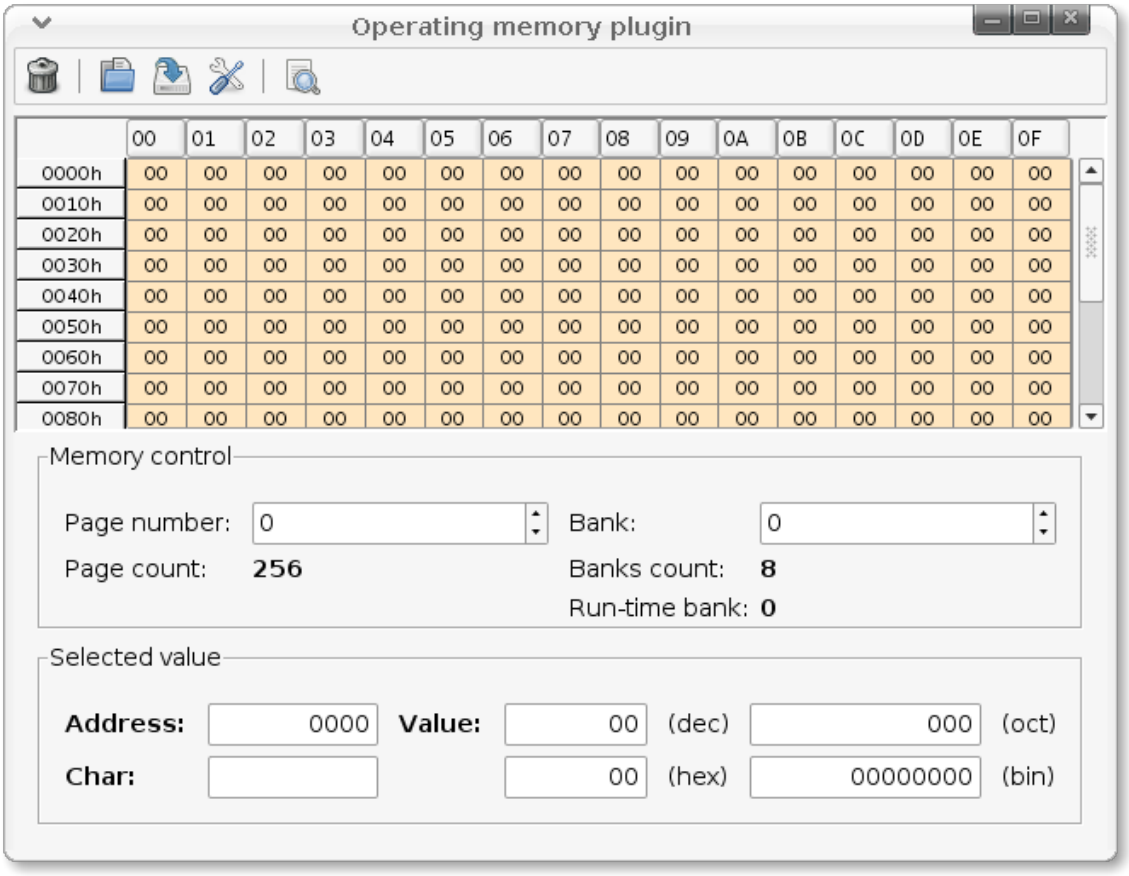
- má premenlivú (nastaviteľnú) veľkosť
- dokáže zmeniť správanie vybraných oblastí pamäte na typ ROM
- podporuje bankovanie

5.1 Popis zásuvného modulu

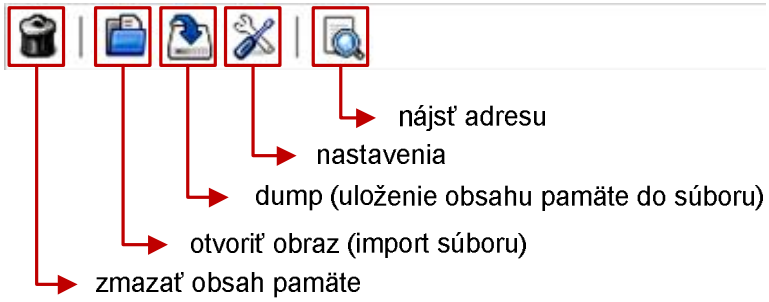
Zásuvný modul operačnej pamäte môžte vidieť na Obr. 5.1. Jej GUI spustíte kliknutím na poslednú ikonu panela emulácie, ako ukazuje Obr. 4.3.

Operačná pamäť je implementovaná ako pole bajtov preddefinovanej veľkosti. Veľkosť operačnej pamäte sa definuje pri voľbe konfigurácie, viď časť 2. Ovládací panel operačnej pamäte aj s jeho popisom je zobrazený na Obr. 5.2.

Aby bolo zobrazenie operačnej pamäte prehľadné, je rozdelená do niekoľkých stránok. Ovládanie zobrazenia stránok sa nachádza v rámčeku *Memory control*. Každá stránka zobrazuje maximálne 256 bajtov. Tak operačná pamäť o veľkosti $65536b$ bude mať $65536/256 = 256$ stránok (na Obr. 5.1). Stránky sa dajú prepínať buď presným napísaním čísla do príslušného textového okienka (*Page number*), alebo klikaním na šípky napravo od tohto okienka.



Obr. 5.1: *Zásuvný modul operačnej pamäte*



Obr. 5.2: *Ovládací panel operačnej pamäte*

Hlavný obsah GUI pamäte je samotná stránka. Každá stránka je zobrazená formou tabuľky, ktorá má 16 stĺpcov a 16 riadkov. Teda počet zobrazených hodnôt v tabuľke je (ako už bolo spomenuté) $16 * 16 = 256$. Každý riadok hýbe adresou na cifre desiatok, teda od $00h$ do $F0h$. Stĺpce predstavujú offset od adresy na danom riadku, teda od $0h$ do Fh. Má to svoju výhodu, pretože

hľadanú adresu adr nájdeme veľmi rýchlo (predpokladáme, že číslovania stránok, riadkov aj stĺpcov začínajú od 1):

- číslo riadka je adresa na cifre desiatok + 1, formálne: $row = ((adr \texttt{ AND } \texttt{F0h}) \texttt{ SHR } 4) + 1$
- číslo stĺpca je adresa na cifre jednotiek + 1, formálne: $col = (adr \texttt{ AND } \texttt{0Fh}) + 1$
- číslo stránky sú zvyšné horné cifry adresy v hexadecimálnom tvare, formálne $page = \lfloor adr \ / \ 256 \rfloor$

Majme napr. adresu $4566h$. Z tejto adresy rovno vieme vyčítať: číslo stránky je $45h = 69$, číslo riadka je 7 a stĺpca tiež 7. Keďže riadky aj stĺpce sú pekne označené, nájdenie adresy nepredstavuje žiadny problém, jediné, čo treba riadne zistiť je číslo stránky, zvyšná adresa sa dá pekne nájsť v rámci tabuľky. Ak by sa však niekomu aj toto lenilo urobiť, na nástrojovom paneli (Obr. 5.2) môžme vidieť ikonu predstavujúcu ďalekohľad. Ide o funkciu, ktorá presne nájde zadanú adresu, takže je len na používateľovi, ktorý spôsob si vyberie alebo ktorý sa mu zdá rýchlejší.

Na Obr. 5.1 je vyznačená adresa $0016h$, na ktorej má bunka hodnotu `41h`. Vyznačená hodnota sa pod tabuľkou zobrazí v rôznych tvaroch: najprv je tam celá adresa, potom hodnota so základom: dekadickým (*dec*), hexadecimálnym (*hex*), oktálnym alebo osmičkovým (*oct*) a binárnym (*bin*). Okrem toho je pod adresou zobrazený symbol, ktorý je reprezentovaný touto hodnotou (v našom prípade písmeno `A`).

5.1.1 Editovanie buniek

Hodnoty buniek v operačnej pamäti sa dajú meniť. Stačí dvakrát kliknúť na bunku a zobrazí sa textové okno, kde sa pôvodná hodnota prepíše. Formát zápisu novej hodnoty je trojaký: dekadický (klasické čísla od $0-9$), hexadeximálny (tvar 0x*číslo*, kde *číslo* je od $0-F$), osmičkový (tvar 0*číslo*, kde *číslo* je od $0-7$)[1].

Zásuvný modul operačnej pamäte umožňuje tiež pamäť úplne vymazať (ikona odpadkového koša na Obr. 5.2), čím sa priradí každej bunke pamäti hodnota 0 a tým sa prepíše jej pôvodná hodnota. Funkcia má vplyv aj na ROM oblasť.

5.1.2 Import súborov

Operačnú pamäť môžme čiastočne alebo úplne naplniť obsahmi externých súborov, ktoré sa dajú importovať. Pre import sú podporované dva formáty: klasický binárny, alebo špeciálny HEX formát (od firmy Intel). V minulosti, ale aj v súčasnosti sa formát HEX dosť často využíva aj mimo oblasti emulovania. V skratke ide o reprezentáciu binárneho súboru hexadecimálnymi číslicami. Okrem toho výstup kompilátorov by mal byť práve vo formáte HEX. Funkciou importu tak používateľ nie je nútený znova a znova prekompilovávať kód, akonáhle chce niečo emulovať, ale môže už skompilovaný kód „načítať" do operačnej pamäte (po kompilácii vyskočí otázka hlavného modulu, či skompilovaný má program načítať do pamäte).

[1] tento tvar už bol popísaný v časti 4.2 pri popise nastavovania breakpointov, a stretneme sa s ním skoro všade

POZNÁMKA:
Importovaný súbor obsahuje kód dĺžky menšej nanajvýš rovnej veľkosti celej pamäte. Časť operačnej pamäte, ktorá nie je zasiahnutá obsahom importovaného súboru, ostane nezmenená.

Súbory formátu HEX majú pre každý bajt definovanú jeho absolútnu adresu v pamäti (a teda nemusí ísť o súvislú zmenu obsahu v pamäti). Preto import takéhoto súboru si nevyžaduje žiadny ďalší vstup od používateľa. Naopak binárny formát sa operačnej pamäti javí ako úplne chaotický bez nejakej štruktúry a informácií o jeho umiestnení. Preto operačná pamäť nevie, na ktorú adresu má súbor načítať. Z tohoto dôvodu sa zobrazí výzva pre používateľa vo forme dialógového okna na zadanie adresy. Následne sa tento binárny súbor importuje od zadanej adresy súvisle.

V prípade, že veľkosť importovaného súboru presahuje veľkosť pamäte, bude jeho zvyšok „odstrihnutý".

5.1.3 Dump pamäte

Obsah operačnej pamäte je možné exportovať, tj. uložiť do externého súboru. Táto funkcia sa po anglicky nazýva **dump**. Podporované sú dva formáty súborov:

- *textový* - každá bunka (byte) pamäte je uložená na samostatnom riadku. Každý riadok má tvar: `n: mm`, kde `n` je poradie bunky (adresa od 0), za ňou nasleduje dvojbodka (`:`), tabulátor a samotná hodnota bunky v textovom, hexadecimálnom tvare (`mm`). Výhodou tohto formátu je jednoduché automatizované spracovanie (nejakým externým programom).
- *binárny* - každá bunka pamäte je uložená v binárnom formáte.

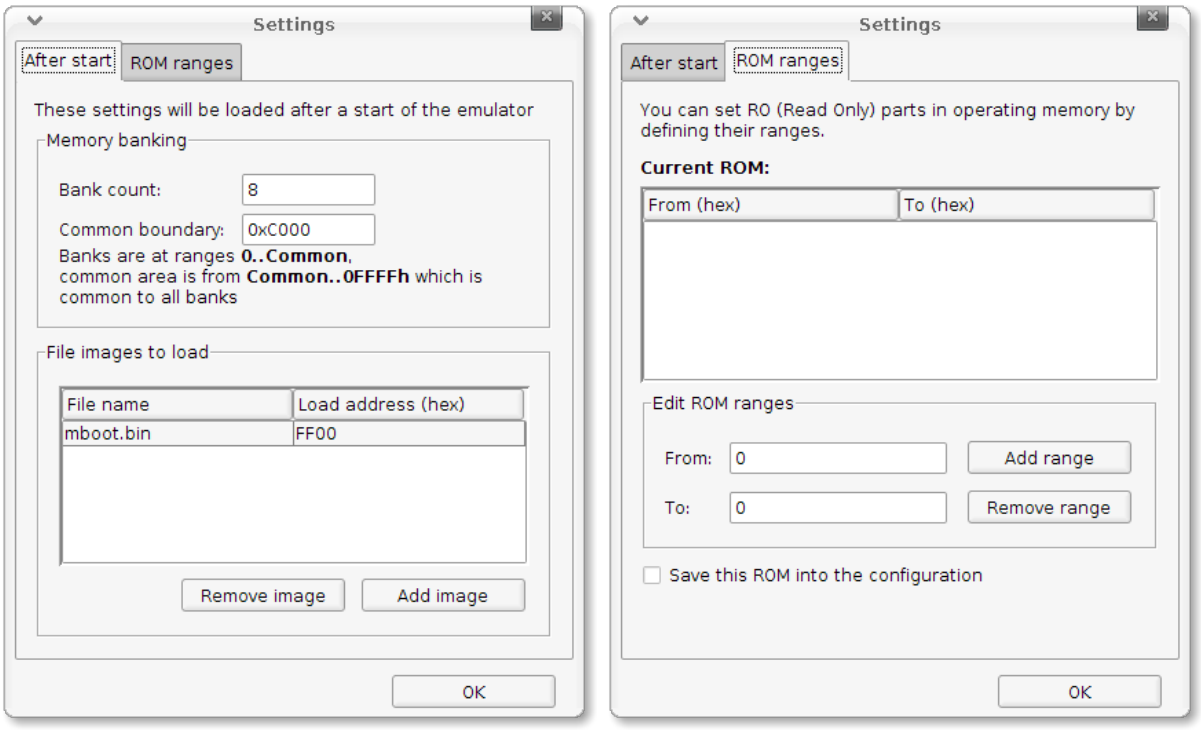
5.2 Nastavenia zásuvného modulu

Z dôvodu prispôsobovania sa práci používateľa umožňuje zásuvný modul svoje niektoré už spomenuté schopnosti nastaviť, aby sa aktivovali hneď po štarte. Okrem toho, aj svoju ďalšiu (ešte nespomenutú) nastaviteľnú funkcionalitu dopĺňa do jediného okna s nastaveniami. Používateľ klikne na príslušnú ikonu v ovládacom paneli (Obr. 5.2).

Toto okno (Obr. 5.3) má dva panely: *After start* a *ROM ranges*. Nastavenia prvého panelu sa po kliknutí na tlačidlo *OK* automaticky uložia do konfiguračného súboru virtuálnej architektúry a prejavia sa až po novom štarte emulátora.

POZNÁMKA:
Uložené nastavenia budú však platiť len pre zvolenú architektúru, v inej sa neprejavia.

Druhý panel sa týka novej funkcionality, ktorá umožní určitú oblasť operačnej pamäte premeniť na pamäť typu ROM (*Read Only Memory*). Keď je niektorá oblasť takto vyznačená, nie je možné v tejto oblasti zmeniť hodnoty buniek operačnej pamäte, iba čítať. Je možné vytvárať viacero takýchto oblastí, na ľubovoľných miestach a v ľubovoľných veľkostiach.



Obr. 5.3: *Dialógové okno nastavení operačnej pamäte*

5.2.1 After start

Okno v rámci panelu je rozdelené do dvoch častí. V prvej časti sa nastavuje bankovanie - spoločná adresa, od ktorej budú bunky pamäte spoločné pre všetky banky, a počet bánk. Problematika bankovania je popísaná v časti 5.3.

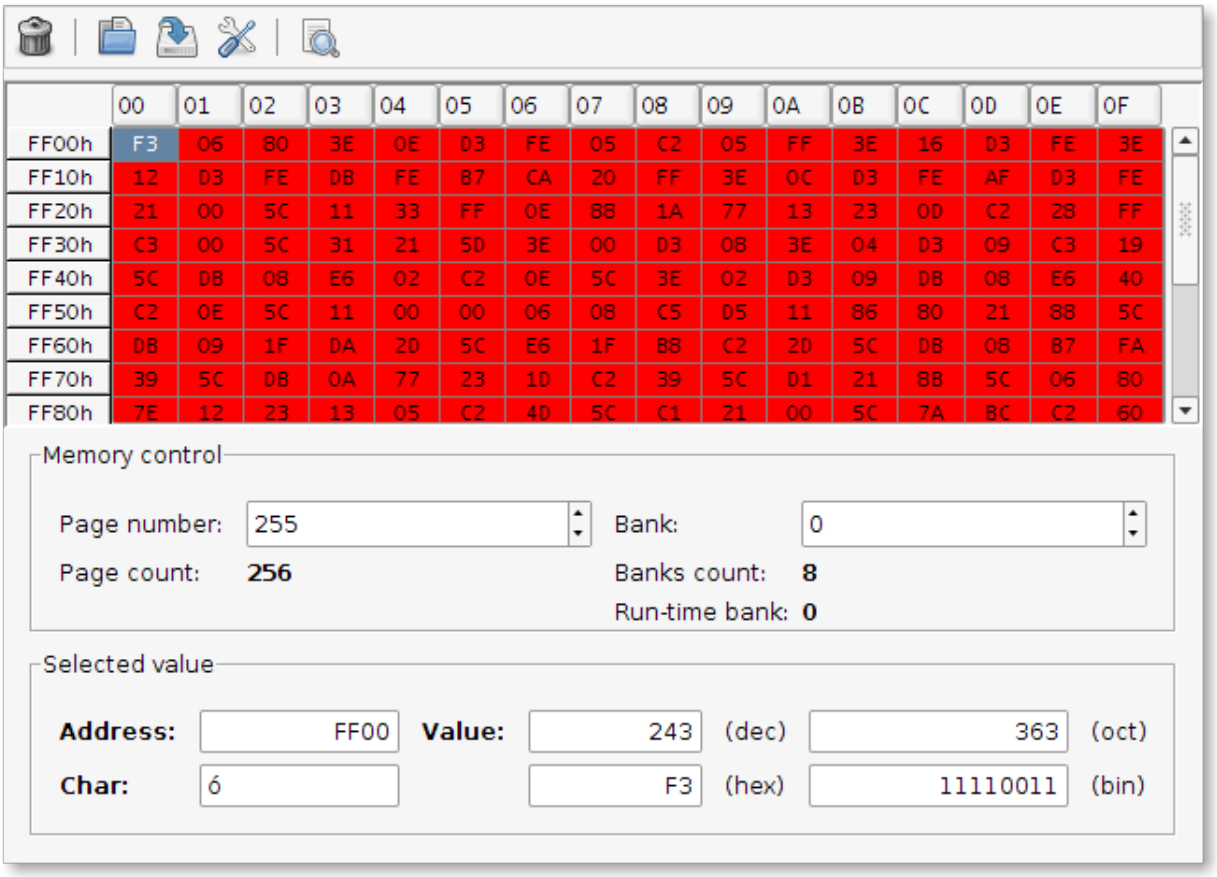
Aby používateľ nemusel neustále po každom štarte emulátora načítavať do operačnej pamäte notorické obrazy, ktoré v nej chce mať, môže si ich pridať do zoznamu obrazov, ktoré sa po štarte emulátora načítajú, kliknutím na tlačidlo *Add image*. Tlačidlom *Remove image* obraz zo zoznamu odoberie. V prípade, že sa obraz po štarte nepodarilo načítať, vypíše sa chybová hláška, ale emulátor pokračuje v činnosti ďalej.

5.2.2 ROM ranges

Jednotlivé oblasti sa zadávajú vo forme od-do do príslušných textových oknien. Po zadaní rozsahu je možné kliknúť buď na tlačidlo *Add range* (pridá zadaný rozsah ako ROM), alebo na tlačidlo *Remove range* (odoberie rozsah z ROM). Všetky rozsahy ROM sú zobrazené formou zoznamu. Zadávané rozsahy sa môžu rôzne križovať, pretože po kliknutí na ktorékoľvek tlačidlo je urobený inteligentný výpočet na zlúčenie rozsahov, s dôrazom na zachovanie konzistencie.

POZNÁMKA:
Nastavenia ROM oblastí sa automaticky do konfiguračného súbory neuložia. Ak si to používateľ želá, potom je potrebné zaškrtnúť checkbox Save this ROM into configuration.

Na Obr. 5.4 je možné vidieť, ako sa zmení zobrazenie ROM oblasti v tabuľke operačnej pamäte (bunky označené ako ROM sú vyznačené červenou farbou).



Obr. 5.4: *Operačná pamäť s ROM oblasťou*

5.3 Technika bankovania

Technika bankovania sa využívala v rozkvete 8-bitových procesorov, vďaka ktorej bolo možné pristúpiť k pamäti o veľkosti väčšej ako 64 kB.

Adresová zbernica bola 16-bitová, preto sa dalo adresovať maximálne 64 kB. Aj registre boli najviac 16 bitové, kde sa zmestilo číslo najviac $FFFh = 65535$. Nič však nebránilo tomu, aby boli operačné pamäte väčsie. Problém však bol ten, že vyššie oblasti sa prakticky nedali adresovať. Riešením bolo, že sa na rovnakých adresách operačnej pamäte prepínali „banky“,

teda operačná pamät na dostupné adresy namapovala nedostupnú pamäť (v reálnom svete išlo o akýsi podporný obvod pripojený na IO port procesora, ovládateľný používateľom).

Napríklad: Majme fyzickú pamät veľkosti 1 MB, teda veľkosť je $100000h$ bytov, fyzické adresy sú teda v rozsahu $0 - FFFFFh$. My však máme k dispozícii adresný priestor od $0 - FFFFh$. Preto, aby sme mohli pristupovať ku všetkým bunkám, musíme si pamäť „rozdelit"' na menšie časti. Nezáleží pritom na tom, ako si pamäť rozdelíme. Dôležité je však brať do úvahy fakt, že banky (resp. logické časti pamäte) musia byť rovnako veľké.
Rozdeľme si pamäť do 16-tich bánk:

```
1. banka = 0      - 10000h
2. banka = 10000h - 20000h
3. banka = 30000h - 40000h
...
```

V každom okamihu môže byť aktívna len jedna banka a operačný systém (napr. CPM v.3) si ju vedel v prípade potreby prepnúť. Ak je aktívna prvá banka, náš adresný priestor adresuje pamäť v rozsahu od $0-FFFFh$. V prípade aktívnej druhej banky adresujeme pamäť od $10000-1FFFF$ (konkrétne adresa 0 bude odpovedať fyzickej adrese $10000h$, atď).

Existovala možnosť rozdeliť adresný priestor do dvoch častí. Prvá časť (od adresy 0 až po jej hranicu) mala podporu bankovania (teda v rámci tejto oblasti pamäte prepínanie bánk hralo úlohu), a druhá časť bola spoločná pre všetky banky (prepínanie bánk nemalo na obsah tejto pamäte vplyv). Túto funkciu podporuje aj zásuvný modul emulátora, hranica spoločnej pamäte sa dá nastaviť v paneli *After start* v okne nastavení (popísané v časti 5.2.1).

5.3.1 Výpočet veľkosti pamäte

Použitím rôzneho počtu bánk (ktoré majú vždy rovnakú veľkosť), sa mení celková veľkosť pamäte. Veľkosť operačnej pamäte sa síce nastavuje pri vytváraní konfigurácie (popísané v časti 2.2.1.2). Toto nastavenie hovorí o veľkosti celej pamäte len v prípade, ak neexistujú banky (teda ak existuje len jediná banka - celá operačná pamäť). Vo všeobecnosti však táto hodnota definuje platný adresný priestor pamäte - rozsah platných adries, z ktorých bude možné čítať, resp. na ne zapisovať (ak nepôjde o pamäť typu ROM). Výpočet celkovej veľkosti je o to zložitejší, že na ňu má vplyv aj hraničná adresa spoločnej pamäte. Samozrejme je podmienkou, že adresa spoločnej pamäte musí byť v rozsahu platných adries.

Definujme si maximálnu platnú adresu pamäte, a_{max}, ktorú vypočítame z počiatočného nastavenia veľkosti pamäte $memsize$:

$$a_{max} = memsize - 1 \tag{5.1}$$

Predpokladajme počet bánk n a adresu spoločnej pamäte a_{common}. Celková veľkosť operačnej pamäte je potom definovaná ako:

$$size = a_{common} \cdot n + a_{max} + 1 - a_{common} \tag{5.2}$$

$$size = a_{common} \cdot (n-1) + memsize \tag{5.3}$$

Kapitola 6

Operačná pamäť RAM stroja

Táto kapitola je venovaná popisu zásuvného modulu operačnej pamäťe `RAMMemory.jar`, s názvom *RAM Memory* a verziou `1.0b1`. Táto operačná pamäť je použiteľná len v spojení s odpovedajúcim zásuvným modulom procesora - `RAM.jar`. Táto architektúra je bližšie popísaná v časti 15 a ako celok predstavuje emulátor abstraktného stroja RAM (*Random Access Machine*).

Architektúra RAM stroja bola prispôsobená požiadavkám Von-Neumannovej architektúry - zásuvný modul procesora predstavuje riadiacu jednotku stroja, operačná pamäť RAM stroja je programová páska a ostatné pásky sú realizované ďalšími zásuvnými modulmi zariadení.

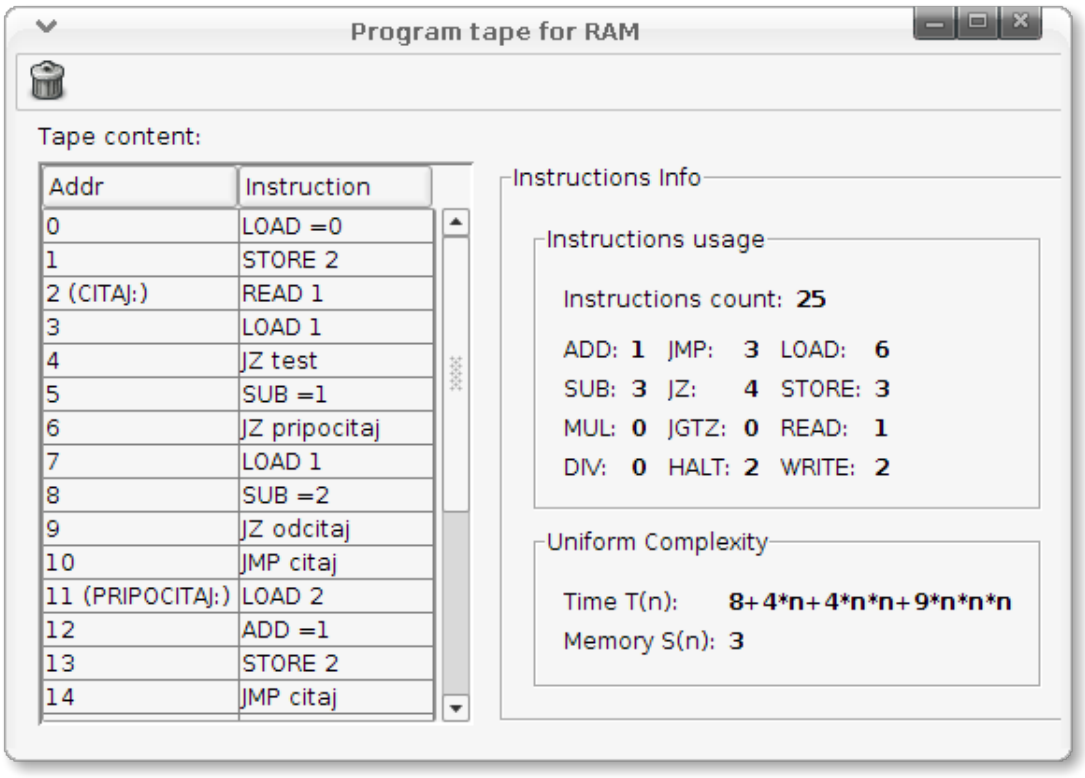
Čiže táto operačná pamäť neobsahuje všeobecne ľubovoľné binárne bunky o veľkosti napríklad jeden byte, ale obsahuje len *inštrukcie programu*. Každá inštrukcia sa nachádza na jednej adrese, teda každá inštrukcia zaberá len jednu bunku pamäte, či pásky.

Pamäť sa napĺňa len po úspešnom kompilovaní. Používateľ *nemôže* editovať jej bunky, iba zmazať celú pamäť naraz. To odpovedá architektúre RAM stroja, ktorého popis môžte nájsť v [11].

GUI tejto pamäte (Obr. 6.1) zobrazuje pásku ako zoznam lineárne usporiadaných inštrukcií. Adresy, ktoré sú cieľmi skokov sú zobrazené okrem číselného označenia aj textovým označením, aké bolo použité v programe.

Veľkou prednosťou tejto pamäte je jej schopnosť spočítať uniformnú časovú a pamäťovú zložitosť programu. Pre výpočet časovej zložitosti je cena každej z inštrukcií rovnaká. Výpočet dokáže identifikovať aj cykly. Pamäťová zložitosť nie je počítaná celkom správne, je počítaná ako súčet jasne použitých registrov. Jej asymptotická zložitosť bude preto vždy $O(1)$, a to nemusí byť vždy pravda.

Okrem výpočtu uniformnej zložitosti dokáže pamäť spočítať početnosť využitia jednotlivých inštrukcií.



Obr. 6.1: *GUI operačnej pamäte (programová páska) RAM stroja*

Kapitola 7

Assemblery procesorov Intel 8080 a Zilog Z80

Táto kapitola sa zaoberá popisom zásuvných modulov kompilátorov:

- *Intel 8080 Compiler*, súbor `compiler8080.jar` a verzia `2.9b1`.
- *Z80 Assembler*, súbor `assemblerZ80.jar` a verzia `1.5b1`.

Jej cieľom nie je naučiť čitateľa programovať v assembleri a nepopisuje inštrukčný súbor procesorov. Z toho dôvodu je možné popísať naraz dva kompilátory, ktoré majú skoro rovnaké vlastnosti a možnosti (vzhľadom na podobnosť samotných procesorov).

Kompilátor pre procesor i8080 je podobný kompilátoru [2], no sú v ňom rozdiely. Nebolo dôležité zachovať všetky aspekty pôvodnej oficiálnej špecifikácie, naopak bolo potrebné zabezpečiť, aby bola špecifikácia dostatočná z hľadiska možností a zároveň jednoduchá, aby neviedla programátora k zbytočným chybám.

V ďalšom texte budem na označenie oboch kompilátorov miesto množného čísla „kompilátory“, resp. „assemblery“ používať len jednotné číslo „kompilátor“, resp. „assembler“ a budem tým myslieť oba popisované zásuvné moduly, ak to explicitne neuvediem inak.

V prípade v texte sa vyskytujúcich príkladov s inštrukciami platí, že ak to nie je explicitne uvedené, jedná sa o príklady pre kompilátor procesora i8080. Ak sa nablízku nenachádza aj príklad pre procesor Z80, tak je predpoklad, že to v tomto procesore bude fungovať rovnako, ak pozmeníme inštrukciu na jej ekvivalent v procesore Z80.

7.1 Lexikálne jednotky jazyka

Assembler nerozlišuje medzi malými a veľkými písmenami (je necitlivý na veľkosť písmen). Jeho lexikálny analyzátor rozlišuje tieto skupiny symbolov:

- kľúčové slová — názvy inštrukcií, direktívy preprocesora (`org`, `equ`, `set` (i8080) resp. `var` (Z80), `macro`, `endm`, `include`, `if`, `endif`), definície dátových výrazov (`db`, `dw`, `ds`),

registre procesora

- identifikátory
- návestia
- konštanty — znakové (resp. reťazce), celočíselné
- operátory —
  - spoločné: `+`, `-`, `*`, `/`, `=`
  - len i8080: `mod`, `and`,`or`, `not`, `xor`,`shl`, `shr`
  - len Z80: `%`, `&`, `|`, `!`, `~`, `<<`, `>>`, `>`, `<`, `>=`, `<=`
- komentáre — text za bodkočiarkou (`;`) vrátane bodkočiarky

7.1.1 Konštanty

Číselné konštanty môžu byť zapísané v jednej z niekoľkých číselných sústav:

dvojková: $n_k n_{k-1} ... n_0 B$, pričom $n_i \in \{0, 1\}$

desiatková: $n_k n_{k-1} ... n_0 D$, písmeno D je nepovinné, $n_i \in \{0, 1, ..., 9\}$

osmičková: $n_k n_{k-1} ... n_0 O$ resp. $n_k n_{k-1} ... n_0 Q$, $n_i \in \{0, 1, ..., 7\}$

šestnástková: $n_k n_{k-1} ... n_0 H$, $n_i \in \{0, 1, ..., 9, A, B, ..., F\}$ a *musí* začínať číslicou

Znaky, resp. reťazce musia byť uzavreté do:

- i8080 — jednoduchých úvodzoviek, príklad: `MVI E,'*'`
- Z80 — normálnych úvodzoviek, príklad: `LD E,"*"`

7.2 Syntax inštrukcií

Program sa skladá z inštrukcií. Tieto sú oddelené novým riadkom. Formát inštrukcie sa skladá z týchto polí:

- návestie (LABEL) - je to identifikátor použitý ako referencia na adresu nasledujúcej inštrukcie.
- kód (CODE). Určuje operáciu, ktorá sa má vykonať (názov inštrukcie).
- operand (OPERAND). Poskytuje ľubovoľnú adresovú alebo dátovú informáciu potrebnú pre pole CODE. Niekedy je toto pole nepovinné, to záleží od typu inštrukcie.

- komentár (COMMENT). Je prítomné pre pohodlie programátora a assemblerom je ignorované. Programátor môže použit' komentáre na popis operácií a to robí program prehľadnejším.

Pole CODE a OPERAND musia byt' oddelené minimálne jednou medzerou. Príklady:

Návestie	Kód	Operand	Komentár
`HERE:`	`MVI`	`C,0`	; Vlož do registra C hodnotu 0
`THERE:`	`DB`	`3AH`	; Vytvor dátovú konštantu veľkosti
			; 1 byte
`LOOP:`	`ADD`	`E`	; Pridaj hodnotu regitra E do akumulátora
	`RLC`		; Rotuj akumulátor doľava

7.2.1 Pole návestia

Toto je voliteľné pole. Návestie sa skladá z mena návestia, za ktorým nasleduje dvojbodka (`:`). Meno návestia musí spĺňat' požiadavky na identifikátor. Teda prvý znak návestia musí byt' písmeno abecedy alebo jeden zo špeciálnych znakov: `@` (zavináč), _ (podčiakovník) alebo `?` (otáznik). Ďalšie znaky môžu obsahovat' aj číslice od `0-9`.

Operačné kódy, názvy pseudoinštrukcií, a mená registrov sú vyhradené pre assembler a nemôžu byt' použité ako návestia. Keďže návestia vystupujú ako adresy inštrukcií, ich definície sa nemôžu opakovat'.

7.2.2 Pole operanda a adresné módy

Pole operanda upresňuje pole kódu a teda danú inštrukciu. V závislosti na type poľa kódu môže pole operanda chýbat', alebo môže pozostávat' z jednej alebo dvoch položiek oddelených čiarkou (`,`). Pri vytváraní operanda sa používajú adresné módy (spôsoby, akými inštrukcia bude pristupovat' k dátam pre ňu potrebným). Pritom druh inštrukcie definuje, aký adresný mód sa môže použit'.

Adresné módy:

Implicitné adresovanie - niektoré inštrukcie pracujú s dátami (registre, adresy,...), ktoré sú dané implicitne touto inštrukciou (napr. inštrukcia `RAL` pracuje implicitne s akumulátorom `A`, inštrukcia `PCHL` načíta do registra `PC` hodnotu registrového páru `HL`)

Registrové adresovanie - ako operand je použitý názov registra alebo registrového páru a inštrukcia pracuje s hodnotou tohto registra v čase svojho volania (napr. `INR B`)

Registrové nepriame adresovanie - hodnotu registra použitého v operande inštrukcia chápe ako adresu a pracuje s hodnotou operačnej pamäte uloženej na tejto adrese. Procesor 8080 používa na adresovanie register `M`, ktorý určuje, že ide o adresu danú registrovým párom `HL`. Procesor Z80 register `M` neobsahuje, namiesto toho pre nepriame adresovanie používa zátvorky `()`, do ktorých uzavrie daný registrový pár, napr. `INC (HL)`.

Bezprostredné adresovanie - operand je daný priamou 8-bitovou hodnotou. Táto hodnota je v zdrojovom kóde napísaná ako aritmeticko-logický výraz, ktorého vyhodnotenie musí byť 8-bitová hodnota

Bezprostredné rozšírené adresovanie *(len Z80)* - operand je daný priamou 16-bitovou hodnotou. Táto hodnota je v zdrojovom kóde napísaná ako aritmeticko-logický výraz, ktorého vyhodnotenie nesmie byť viac ako 16-bitová hodnota

Priame adresovanie - operand je daný priamou 8 alebo 16 bitovou hodnotou, ktorá je chápaná ako adresa. Aj táto hodnota môže byť v zdrojovom kóde napísaná ako aritmeticko-logický výraz. Príklad (8080): `SHLD 1234h`, (Z80): `LD (1234h), HL`.

Relatívne adresovanie *(len Z80)* - operand je 8-bitová hodnota (offset), ktorá sa pripočíta k adrese nasledujúcej inštrukcie. Tento offset je celé číslo (so znamienkom) v rozsahu $-127 - +128$. Príklad (Z80): `JR 4`

Modified page zero - operand inštrukcie je daný ako 3-bitová hodnota $(0-7)$ N. Inštrukcia však pracuje s adresou, ktorá je odvodená od tejto hodnoty. Táto adresa je vypočítaná ako $N * 8$. Príklad: inštrukcie `RST`.

Indexové adresovanie *(len Z80)* - operand je v tvare `(IX+d)`, resp. `(IY+d)`, čo označuje adresu danú týmto výrazom. Symbol `d` označuje 8-bitový offset, ktorý je pripočítaný k indexovému registru (`IX` alebo `IY`) a výsledná hodnota je smerník (ukazovateľ) na miesto, kde sa nachádzajú dáta pre inštrukciu. Príklad: `ADD A, (IX+45h)`

Bitové adresovanie *(len Z80)* - operand je špecifikovaný hodnotou $0 - 7$. Táto hodnota je chápaná ako označenie čísla bitu druhého operanda, ktorý môže byť rôzneho druhu. Toto adresovanie používajú inštrukcie `BIT`, `SET` a `RES`, ktoré pracujú nad označeným bitom daného operanda.

Priame dáta a adresa môžu byť určené rôznym spôsobom. Zoznam možných spôsobov ich určenia je tu:

- dekadické, hexadecimálne, osmičkové a binárne konštanty
- programové počítadlo (`$`) - reprezentuje aktuálnu hodnotu adresy práve kompilovanej inštrukcie. Príklad: `JMP $+6`. Táto inštrukcia spôsobí skok na adresu o 6 bytov vyššiu ako je adresa tejto inštrukcie v čase prekladu.
- reťazec - jedná sa o jeden alebo viac ASCII znakov uzavretých v:
  - i8080 — jednoduchých úvodzovkách, príklad: `'hello'`
  - Z80 — normálnych úvodzovkách, príklad: `"world"`
- premenné, konštanty — ide o identifikátory, ktorým boli assemblerom pridelené numerické hodnoty. Príklad: Predpokladajme, že premennej `VALUE` bola pridelená hexadecimálna hodnota `9FH`. Potom obe nasledujúce inštrukcie načítajú register `D` s touto hodnotou:

```
A1: MVI D, VALUE
A2: MVI D, 9FH
```

- návestia - ich definícia je v poli LABEL, môže ísť aj o doprednú referenciu. Príklad:

```
HERE:  JMP THERE ; Skok na inštrukciu na adrese THERE
       ...
THERE: MVI D, 9FH
```

- výrazy

7.2.2.1 Výrazy

Aritmetické a logické výrazy predstavujú v podstate vhodné spojenie priamych dát (popísaných vyššie) a operátorov, vyhodnotením ktorých vznikne výsledná číselná hodnota. Jedinou výnimkou sú reťazce, ktoré nesmú byť použité v aritmeticko-logických výrazoch[1]. V assembleri musia byť výrazy „vyhodnotiteľné“ už pri preklade.

Assemblery využívajú spolu niekoľko operátorov:

aritmetické - + (unárne plus a sčítanie), − (unárne mínus a odčítanie), `*` (násobenie), `/` (delenie), `mod` resp. `%` (modulo - zvyšok po delení)

bitové (dvojkové) - `not` resp. `!` (negácia), `and` resp. `&` (AND), `or` resp. `|` (OR), `xor` resp. `~` (exkluzívne OR), `shr` resp. >> (bitový posun doprava), `shl` resp. << (bitový posun doľava),

porovnávacie - > (väčší), < (menší), = (rovný), >= (rovný alebo väčší), <= (rovný alebo menší)

ostatné - `( )` (ľavá a pravá zátvorka)

Operátory sú buď unárne (majú jeden parameter), alebo binárne (majú dva parametre). Nasledujúca tabuľka poskytuje prehľad operátorov (sú v nej obsiahnuté operátory oboch kompilátorov a operátory sú zoradené podľa priority ich vyhodnocovania):

Priorita	Operátor	Počet parametrov
1.	`( )`	unárny
2.	`*`, `/`, `mod`, `%`, `shl`, `<<`, `shr`, `>>`,`<`,`>`,`<=`,`>=`	binárne
3.	+, −	unárne aj binárne
4.	=	binárny
5.	`not`	unárne
6.	`and`	binárny
7.	`or`, `xor`	binárne

[1]Okrem jednoznakových reťazcov, ktoré sú ľahko previediteľné na číslo - ASCII kód znaku

Všetky operátory chápu svoje argumenty ako 16-bitové kvantity, a generujú 16-bitové kvantity ako ich výsledok. V prípade, že operand očakáva 8 bitovú hodnotu, je výsledok výrazu automaticky „orezaný“ (teda je na neho aplikovaný operátor `and 0FFh`). To môže byť niekedy nežiadúce, preto musí programátor zabezpečiť, aby výsledok výrazu zodpovedal potrebám operácie, ktorá sa programuje. Výrazy sa vyhodnocujú pri preklade.

Význam niektorých operátorov:
Binárne a infixné `shr` resp. `shl` operátory sú lineárne posuvy, ktoré posúvajú ich prvé operandy doprava resp. doľava o počet bitov špecifikovaných ich druhým operandom.

Príklady:

```
HERE: MVI C, HERE SHR 8
```

Inštrukcia načíta číslo `2EH` (16-bitová adresa návestia HERE posunutá doprava o 8 bitov) do registra C.

```
NEXT: MVI D, 34 + 40H / 2
```

Inštrukcia načíta hodnotu $34 + (64/2) = 34 + 32 = 66$ do registra `D`.

7.2.3 Pole komentára

Jediné pravidlo tohto poľa je, že musí začínať bodkočiarkou (`;`) a je samozrejme nepovinné.

```
HERE: MVI C, 0ADH ; Toto je komentár
```

7.3 Dátové výrazy

Aritmetické inštrukcie oboch procesorov predpokladajú, že byty dát, s ktorými pracujú, sú reprezentované špeciálnym formátom, nazvanom „dvojkový doplnok“ a operácie vykonávané nad týmito bytami sa volajú názvom „aritmetika dvojkového doplnku“.

Byte, v ktorom je uložené číslo so znamienkom v dvojkovom doplnku je interpretovaný takto: nižších 7 bitov obsahuje hodnotu čísla a najvyšší bit reprezentuje jeho znamienko (0 - číslo je kladné, 1 - číslo je záporné). Rozsah reprezentovatených kladných čísel v jednom bajte pre dvojkový doplnok je preto od 0 do 127:

```
0 = 00000000B = 0H
1 = 00000001B = 1H
```

Pre zmenu znamienka čísla reprezentovaného v dvojkovom doplnku sa musia aplikovať nasledujúce pravidlá:

1. negovanie každého bitu čísla (vytvorenie tzv. jednotkového doplnku)
2. pripočítanie 1 k výsledku, pričom výsledný prenos sa ignoruje

Príklady:
Vytvorte dvojkový doplnok k číslu `-10D` *:*

```
+10D = 00001010B
```

Negovanie každého bitu: `11110101B`
Pripočítanie 1: `11110110B`

Dvojkový doplnok čísla `-10D` je `F6H`. (Pozn.: je nastavený aj bit znamienka, ktorý indikuje, že číslo je záporné).

Spočítajte čísla `+12D` *a* `-15D`*:*

```
  +12D = 00001100B
+(-15D)= 11110001B
         ---------
       0 11111101B = -3DH
```

Spočítanie spôsobí vynulovanie CARRY príznaku.

Odčítajte čísla `+15D` *od* `+12D`*:*

```
  +12D = 00001100B
-(+15D)= 11110001B
         ---------
       0 11111101B = -3DH
```

Odpočítanie spôsobí nastavenie CARRY príznaku.

Dáta sa definujú pomocou pseudoinštrukcií `db` (define byte), `dw` (define word), resp. `ds` (define storage).

7.3.1 DB - define byte

Syntax: `db list`

kde „list“ je zoznam buď:

1. aritmetických a logických výrazov, ktorých výsledok je 8-bitový
2. reťazce ASCII znakov uzavreté do jednoduchých úvodzoviek

Popis: 8-bitová hodnota každého výrazu, alebo 8-bitová ASCII reprezentácia každého znaku je uložená do nasledujúceho voľného bytu v pamäti.

Príklady:

```
HERE:  DB 0A3H          ; A3
W0RD1: DB 5*2, 2FH-0AH  ; 0A25
W0RD2: DB 5ABCH SHR 8   ; 5A
STR:   DB 'STRINGSpl'   ; 535452494E472031
MINUS: DB -03H          ; FD
```

7.3.2 DW - define word

Syntax: `oplab: dw list`

„list“ je zoznam výrazov, ktoré sa vyhodnotia do 16-bitových dátových kvantít.

Popis: Najmenej významných 8 bitov výrazu sú uložené v byte na nižšej adrese (`oplab`), a najviac významných 8 bitov sú uložené na vyššej adrese (`oplab + 1`). Takéto prehodenie bytov je normálne aj pri ukladaní 16-bitovej adresy do pamäte. Hovorí sa tomu malý byový endián. Tento príkaz (`db`) je obvykle použitý pri vytváraní adresových konštánt pre inštrukcie pre prenos údajov; potom LIST je obvykle zoznam jedného alebo viacerých návestí, ktorých definícia je niekde inde v programe.

Príklady:

Predpokladajme, že existuje návestie `COMP` s referenciou na adresu `3B1CH` a návestie `FILL` ukazuje na adresu `3EB4H`.

```
ADD1: dw COMP           ; 1C3B
ADD2: dw FILL           ; B43E
ADD3: dw 3C01H, 3CAEH   ; 013CAE3C
```

7.3.3 DS - define storage

Syntax: `oplab: ds exp`

„exp“ je jeden aritmetický alebo logický výraz.

Popis: Hodnota `exp` určuje počet bytov, ktoré budú rezervované. Do týchto bytov nie sú zapísané žiadne dáta: v podstate programátor by nemal predpokladať, že budú nulové, alebo že budú mať nejakú inú hodnotu. Ďalšia inštrukcia sa bude potom nachádzať na adrese `oplab + exp`.

7.4 Pseudoinštrukcie

Všeobecný formát pre zápis pseudoinštrukcie je:
`Identifikátor Operácia Operand Komentár`.

`Operand` môže byť voliteľný, `Identifikátor` môže byť potrebný, voliteľný alebo neplatný.

Všetky pseudoinštrukcie sú: `include`, `org`, `equ`, `set`, `if`, `endif`, `macro`, `endm`. Identifikátory sú potrebné pre pseudoinštrukcie: `macro`, `equ`, a `set`. Ostatné pseudoinštrukcie môžu obsahovať voliteľné návestia, presne ako návestia pre strojové inštrukcie. V tom prípade potom návestie bude odkazovať na adresu nasledujúcu za adresou predtým preloženej strojovej inštrukcie.

7.4.1 INCLUDE

Syntax: `oplab: include 'nazov_suboru'`

`nazov_suboru` - relatívna, alebo úplná cesta názvu vkladaného súboru

Popis: Na miesto výskytu tejto pseudoinštrukcie sa vloží obsah súboru, ktorého názov je daný ako parameter. Adresy inštrukcií za touto pseudoinštrukciou sa preto posunú o veľkosť vkladaného súboru po preklade.

Kompilátor odvádza relatívne cesty od cesty, kde je nainštalovaný emulátor. Je predpoklad, že bude existovať akýsi štandardný adresár „include", alebo „examples" či niečo podobné, ktorý bude obsahovať súbory so „štandardnými" makrami, ktoré budú používatelia často využívať.

Príklad:

Nech obsah súboru `C:\emu8\examples\a.asm` je:

```
mvi b, 80h
```

Súbor `C:\b.asm` nech má obsah:

```
include 'examples\a.asm'
```

Výsledok kompilovania súboru `b.asm` bude: `06 80 (mvi b, 80h)`

Assembler používa jeden globálny menný priestor pre identifikátory, preto premenné deklarované v hlavnom súbore (direktíva `set`) ktoré sa používajú aj v include súbore, budú prepísané. Konštanty (`equ`), a návestia rovnakého názvu ako v hlavnom súbore, sa v include súbore zakazuje používať.

7.4.2 ORG (Origin)

Syntax: `oplab: org exp`

`exp` - 16-bitová adresa

Popis: Počítadlo aktuálnej adresy kompilátora je nastavené na hodnotu `exp`, ktorá musí byť platnou 16-bitovou adresou v pamäti. Ďalšia strojová inštrukcia alebo dátové byty sa budú asemblovať od tejto adresy. Ak sa pred prvou strojovou inštrukciou v programe nenachádza pseudoinštrukcia `org`, asemblovanie začína od adresy 0.

Príklad:

Pseudoinštrukcia `org` môže vykonávať ekvivalentnú funkciu ako `ds` (define storage) inštrukcia (pozri časť 7.3.3). Nasledujúce dve útržky kódu sú úplne ekvivalentné:

Adresa	Inštrukcia		Asemblované dáta
2C00	`MOV A,C`	`MOV A,C`	`79`
2C01	`JMP NEXT`	`JMP NEXT`	`C3102C`
2C04	`DS 12`	`ORG $+12`	
2C10	`NEXT: XRA A`	`NEXT: XRA A`	`AF`

7.4.3 EQU (Equate)

Syntax: `Názov equ exp`

`Názov` je potrebný identifikátor, `exp` je ľubovoľný výraz.

Popis: Assembler priradí názvu hodnotu výrazu `exp`. Všade, kde sa používa tento názov, nahradí

sa hodnotou tohto výrazu. Hodnota pridelená názvu sa potom chápe ako konštanta. Názov sa teda môže vyskytovať iba v jedinej definícii, tj. `equ` symbol sa nedá predefinovať.

7.4.4 SET a VAR

Syntax: `Názov set exp` (8080), `Názov var exp` (Z80)

`Názov` je potrebný identifikátor, `exp` je ľubovoľný výraz.
Popis: Assembler priradí názvu hodnotu výrazu `exp`. Všade, kde sa používa tento `Názov`, nahradí sa hodnotou tohto výrazu, pokiaľ sa nezmení ďalšou pseudoinštrukciou `set` resp. `var`. Použitie je teda identické s `equ`, avšak názvy môžu byť redefinované viac krát. V podstate sa `Názov` dá chápať ako akýsi druh premennej.

7.4.5 IF a ENDIF (podmienené asemblovanie)

Syntax:

```
oplab: if exp
          i n š t r u k c i e
oplab: endif
```

Popis: Assembler vyhodnotí výraz `exp`. Ak je výsledok nulový, inštrukcie medzi `if` a `endif` sa ignorujú. V opačnom prípade sa tieto inštrukcie normálne skompilujú, ako keby tam `if` a `endif` vôbec neboli.

7.4.6 MACRO a ENDM (definícia makra)

Syntax:

```
Názov macro Operandy
    i n š t r u k c i e
oplab: endm
```

`Názov` je potrebný identifikátor. Operandy pri definícii makra sú zoznamom identifikátorov (ASCII reťazce, ktoré nie sú v úvodzovkách), ktoré predstavujú symbolické označenie parametrov špecifikovaných pri volaní makra, ktoré v tele makra vystupujú ako konštanty.
Popis: Definícia makra neprodukuje žiadny výstupný kód. Assembler akceptuje inštrukcie medzi `macro` a `endm` ako definíciu makra (ktoré tvoria jeho telo). Telo makra sa uchová pod označením `Názov` a môže pozostávať z ľubovoľných strojových inštrukcií, pseudoinštrukcií, komentárov alebo volaní iných makier.

7.4.6.1 Volanie makra

Makro sa potom dá volať v rámci programu napísaním jeho mena a príslušných parametrov v poli kódu inštrukcie. Keď sa pri asemblovaní zistí, že ide o volanie makra, assembler vytvorí

„myslené“ konštanty `equ` v tele makra (názvy sú názvy operandov), ktorým priradí špecifikované parametre a skompiluje inštrukcie v tele makra (ide o tzv. expanziu makra). Myslené konštanty sú platné iba v tomto tele makra. Makrá sa do seba môžu aj vnárať, hĺbka vnárania je ľubovoľná.

Parametre pri volaní makra sú zoznamom výrazov. Každý výraz je nahradený do tela makra pod názvom príslušného operadnu. Substitúcia prebieha zľava doprava; čo znamená, že prvý výraz v parametroch nahrádza všetky výskyty prvého operandu v tele makra, druhý nahrádza druhý výskyt, atď. Počet parametrov pri volaní makra musí byť rovnaký, ako je počet operandov uvedených v definícii makra. Ani pre parametre, ani pre operandy *nesmú* byť použité názvy registrov, registrových párov a žiadne kľúčové slová assembleru.

Príklad:

```
SHV MACRO
LOOP: RRC      ; Rotovanie s carry doprava
      ANI 7FH  ; Zmazanie najvyššieho bitu
      DCR D    ; Dekrementovanie počítadla rotácie
      JNZ LOOP ; Skok na ďalšie rotovanie
ENDM
```

Makro `SHV` sa potom volá nasledovným spôsobom:

```
LDA TEMP
MVI D,3  ; Určenie 3 posunov doprava
SHV
STA TEMP
```

Alebo iná definícia:

```
SHV MACRO AMT
      MVI D,AMT   ; Načíta počet rotovaní do registra
LOOP: RRC         ; Rotovanie s carry doprava
      ANI 7FH     ; Zmazanie najvyššieho bitu
      DCR D       ; Dekrementovanie počítadla rotácie
      JNZ LOOP    ; Skok na ďalšie rotovanie
ENDM
```

A použitie:

```
LDA TEMP
SHV 5
```

7.4.6.2 Lokálne vs. globálne symboly

Termíny *globálny* a *lokálny* v spojení s makrami budú pre naše účely definované nasledovne:

1. symbol je definovaný v programe **globálne**, ak jeho hodnota je známa a môže byť na neho odkazované ľubovoľnou inštrukciou v programe, a to aj keď bola alebo nebola inštrukcia vytvorená expanziou makra.

2. symbol je definovaný **lokálne**, ak jeho hodnota je známa a môže byť na neho odkazované iba v jednotlivej expanzii makra (v rámci tela makra)

Návestia: Normálne môže byť identifikátor návestia definovaný iba raz v celom programe. Ak sa definícia návestia nachádza v tele makra, je potom definované lokálne a každá ďalšia expanzia tohto makra používa nové návestie, ktoré je vnútorne odlíšené od návestia predchádzajúcej expanzie toho istého makra (teda takto sa zabraňuje konfliktom). V prípade, že je niekde v programe definované globálne návestie, lokálne návestie dočasne „prepíše" globálne návestie a všetky inštrukcie v tele makra odkazujúce na toto návestie budú odkazovať na lokálne návestie.

Konštanty EQU: Názvy konštánt v rámci tela makra sú stále lokálne a globálne definície sa nedajú prepisovať.

Premenné SET resp. VAR: Názvy premenných v rámci tela makra sú stále lokálne a globálne definície sa prepisujú a majú trvácnosť aj mimo tela makra.

7.4.6.3 Substitúcia parametrov makra

Hodnota parametrov makra je zistená a postúpená do tela makra v čase volania makra, pred jeho expanziou.

Príklad:
Predpokladajme, že nasledujúce makro `MAC4` je definované na začiatku programu:

```
MAC4 MACRO P1
ABC SET 14
    DB P1
ENDM
```

Ďalej predpokladajme, že inštrukcia `ABC SET 3` bola napísaná pred prvým volaním makra `MAC4`, pričom nastavila hodnotu premennej `ABC` na 3.
Potom nasledujúce volanie makra: `MAC4 ABC` spôsobí, že assembler najprv vyhodnotí `ABC` a nahradí parameter `P1` hodnotou 3, a potom vytvorí expanziu:

```
ABC SET 14
DB 3
```

Kapitola 8

Jazyk kompilátora pre RAM stroj

Táto kapitola sa zaoberá popisom zásuvného modulu kompilátora RAM stroja - súbor `compRAM.jar`, názov *RAM compiler* a verzia `1.0b`. Syntax a sémantika inštrukcií bola prebratá z knihy [11], ich zoznam je možné vidieť v tabuľke 8.1.

8.1 Lexikálne jednotky jazyka

Kompilátor nerozlišuje medzi malými a veľkými písmenami (je necitlivý na veľkosť písmen). Jeho lexikálny analyzátor rozlišuje tieto skupiny symbolov:

kľúčové slová — názvy inštrukcií a direktívu preprocesora (`<input>`)

identifikátory — v inštrukciách skoku

návestia

konštanty — celočíselné (označenia registrov) - zapísané v desiatkovej sústave bez označenia sústavy, alebo reťazcové (priame dáta) - jedné súvislé dáta nesmú byť oddelené prázdnymi znakmi (medzerou, tabulátorom, . . .). V tabuľke 8.1 sú predstavujú symboly `i` celočíselné konštanty, symboly `s` reťazce.

operátory — označenie priameho operanda (`=`), označenie nepriameho operanda (`*`)

komentáre — text za bodkočiarkou (`;`) vrátane bodkočiarky

8.2 Syntax inštrukcií

Program sa skladá z inštrukcií. Tieto sú oddelené novým riadkom. Formát inštrukcie sa skladá z týchto polí:

Tabuľka 8.1: *Inštrukcie RAM stroja*

Inštrukcia RAM	Možné operandy
READ	i,*i
WRITE	=s,i,*i
LOAD	=s,i,*i
STORE	i,*i
ADD	=s,i,*i
SUB	=s,i,*i
MUL	=s,i,*i
DIV	=s,i,*i
JMP	i
JZ	i
JGTZ	i
HALT	

- návestie (LABEL) - je to identifikátor použitý ako referencia na adresu nasledujúcej inštrukcie.
- kód (CODE). Určuje operáciu, ktorá sa má vykonať (názov inštrukcie).
- operand (OPERAND). Poskytuje informáciu potrebnú pre danú inštrukciu (register, nepriame dáta, priame dáta). Jedine inštrukcia HALT má toto pole prázdne.
- komentár (COMMENT). Je prítomné pre pohodlie programátora a kompilátorom je ignorované. Programátor môže použiť komentáre na popis operácií, čo robí program prehľadnejším.

8.2.1 Pole návestia

Toto je voliteľné pole. Návestie sa skladá z mena návestia, za ktorým nasleduje dvojbodka (:). Meno návestia musí spĺňať požiadavky na identifikátor. Teda prvý znak návestia musí byť písmeno abecedy alebo jeden zo špeciálnych znakov: @ (zavináč), _ (podčiakovník) alebo ? (otáznik). Ďalšie znaky môžu obsahovať aj číslice od 0–9.

Názvy inštrukcií, operátory a názov pseudoinštrukcie sú vyhradené pre kompilátor a nemôžu byť použité ako návestia. Keďže návestia vystupujú ako adresy inštrukcií, ich definície sa nemôžu opakovať.

8.2.2 Pole operanda a adresné módy

Pole operanda upresňuje pole kódu a teda danú inštrukciu. V prípade jazyka stroja RAM každá inštrukcia okrem inštrukcie HALT má pole operanda. Pri vytváraní operanda sa používajú adresné módy (spôsoby, akými inštrukcia bude pristupovať k dátam pre ňu potrebným). Pritom

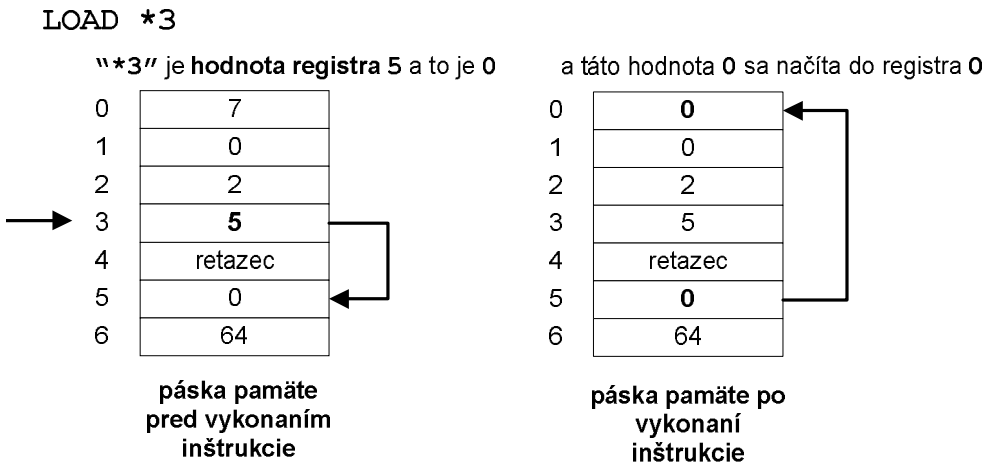
druh inštrukcie definuje, aký adresný mód sa môže použiť. Možné operandy jednotlivých inštrukcií je možné vidieť v tabuľke 8.1.

Adresné módy sú tri:

Priamy - niektoré inštrukcie vedia pracovať s priamymi dátami (reťazce, čísla). Uvedenie samotných priamych dát musí predchádzať operand =. Priame dáta sú kompilátorom vždy chápané ako reťazce, a ako reťazce môžu byť použité ľubovoľné symboly, okrem prázdnych znakov a bodkočiarky. Teda ako reťazce je možné použiť aj kľúčové slová kompilátora - názvy inštrukcií, pseudoinštrukcie a operandov. Práve prázdne znaky sú ukončovacími znakmi reťazca (napr. `LOAD=ahoj`).

Registrový - ako operand je číselné označenie registra. Inštrukcia pracuje s hodnotou tohto registra, alebo do neho uloží svoj výsledok (napr. `READ 3`). Tu je treba dávať pozor na to, že kompilátor očakáva ako operand číslo, a nie reťazec.

Nepriamy - Uvedenie nepriamych dát predhádza operand `*` a nasleduje číselná hodnota registra. Túto hodnotu registra použitého v operande inštrukcia chápe ako adresu a pracuje s hodnotou, ktorá je pod touto adresou uložená na registrove páske. Táto adresa je inými slovami číslo registra, v ktorom je uložené skutočné číslo registra, s ktorým sa má pracovať. Príklad môžte vidieť na Obr. 8.1.



Obr. 8.1: *Príklad práce inštrukcie LOAD *3*

8.3 Inicializácia vstupnej pásky

Po spustení programu RAM stroja na emulátore sa očakáva, že vstup je na vstupnej páske už zadaný. Používateľ môže vstup upravovať pred spustením programu ručne na danej páske, alebo to môže urobiť už priamo v zdrojovom kóde programu. Na tento účel slúži pseudoinštrukcia `<input>`.

Táto pseudoinštrukcia je jedinou pseudoinštrukciou, ktorú jazyk kompilátora obsahuje. Má 1 a viac voliteľných parametrov, ktoré sú reťazce (formátu ako priame dáta inštrukcií). V programe je možné použiť aj niekoľko týchto pseudoinštrukcií a to na rôznych miestach v programe. Po skompilovaní programu sa vstupná páska stroja naplní vstupnými dátami v poradí, v akom boli uvedené v tejto pseudoinštrukcii a tiež je zachované poradie jednotlivých pseudoinštrukcií. Tieto pseudoinštrukcie nie sú súčasťou programu, takže naozaj nezáleží na tom, kde budú v zdrojovom kóde umiestnené a je zaručené, že budú „zavolané“ práve raz.
Príklad:

```
; vstup : X0 - reťazec X ukončený nulou, X patrí do {1,2,3,....}*
; výstup: N1(X) - počet jednotiek v X.

<input> 1 2 3 4 5 6 7 1 1 1 2 5 0

load =0
store 2

citaj:
 read 1
 load 1
 jz vypis

 sub =1
 jz pripocitaj

 jmp citaj

pripocitaj:
 load 2
 add =1
 store 2
 jmp citaj

vypis:
 write 2
 halt
```

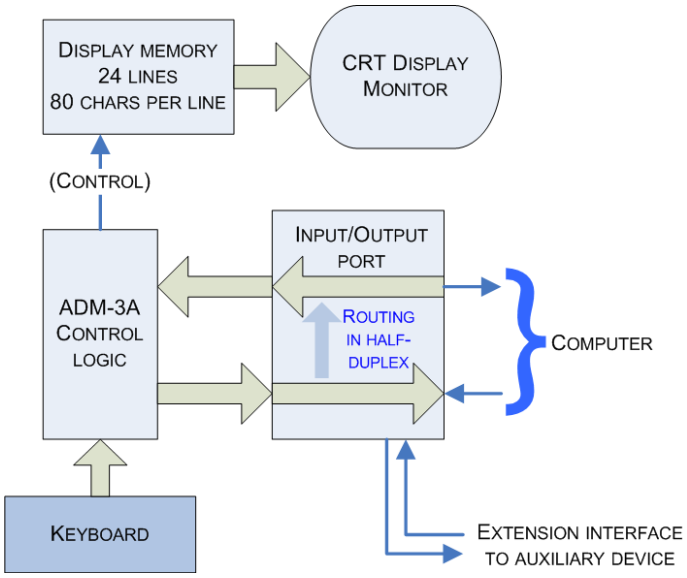
Kapitola 9

Virtuálny terminál ADM-3A

Táto kapitola sa zaoberá popisom zásuvného modulu zariadenia - virtuálneho terminálu ADM-3A od firmy LSI. Tento zásuvný modul má názov *Terminal ADM-3A*, súbor `terminalADM-3A.jar` a verzia `1.2b2`. Nová verzia zásuvného modulu prináša podporu anti-aliasingu, ale hlavne double-bufferingu, vďaka ktorému obraz už „nebliká“.

ADM-3A [3] je interaktívne zariadenie, ktoré sa používalo na zadávanie, zobrazovanie a posielanie informácií do počítača; a tiež prijímanie a zobrazovanie údajov z počítača. Používateľ môže údaje zadávať priamo klávesnicou, a tie sú okamžite posielané do počítača (a môžu byť simultánne zobrazované na obrazovke). Dáta z počítača sú na reálnom termináli prijímané a zobrazované rýchlosťou najviac 1920 znakov za sekundu. Zásuvný modul je pravdepodobne rýchlejší, ale táto rýchlosť nebola nijak testovaná, ani obmedzovaná.

Princíp práce terminálu je zobrazený na obrázku 9.1.



Obr. 9.1: *Princíp práce terminálu ADM-3A*

Niektoré parametre terminálu boli voliteľné (používateľ ich mohol určitým spôsobom aktivovať, napr. stlačením určitej kombinácie kláves), no súčasná verzia terminálu nepodporuje zmenu parametrov. Implementovaný terminál má nasledujúce parametre:

- veľkosť obrazovky je 80 x 24 riadkov
- rozlišuje 128 znakov (prvá polovica ASCII)
- pri zaplnení obrazovky sa obraz posunie (roluje) smerom hore
- pri dosiahnutí konca riadku sa automaticky pridá nový riadok

Po stlačení ľubovoľnej klávesy sú dáta okamžite vyslané do počítača. Terminál rozlišuje 2 režimy práce: poloduplexný a plneduplexný režim. V poloduplexnom režime sú dáta odoslané aj do video pamäte a následne zobrazené (sú kopírované na výstup), zatiaľ čo pri plneduplexnom režime sú dáta oba odoslané do počítača, a ich zobrazenie musí zabezpečiť volanie z počítača. Nastavenie režimu práce terminálu sa robí v jeho grafickom prostredí.

Obrázok 9.2 ukazuje vedľa seba reálny terminál ADM-3A (vľavo) a jeho softvérovú implementáciu (vpravo).

Obr. 9.2: *Terminály ADM-3A: reálny(**vľavo**), virtuálny(**vpravo**)*

Terminál mal svoju vlastnú videopamäť o veľkosti $80 \cdot 24 = 1920$ bytov. Táto pamäť je neustále premietaná na obrazovku.

Terminál je k počítaču pripojený prostredníctvom sériového rozhrania RS-232. Toto rozhranie realizuje prídavná karta MITS 88-SIO, resp. MITS 88-SIO-2, ktorá je zasadená v hlavnej zbernici počítača. Terminál je zapojený do tejto karty.

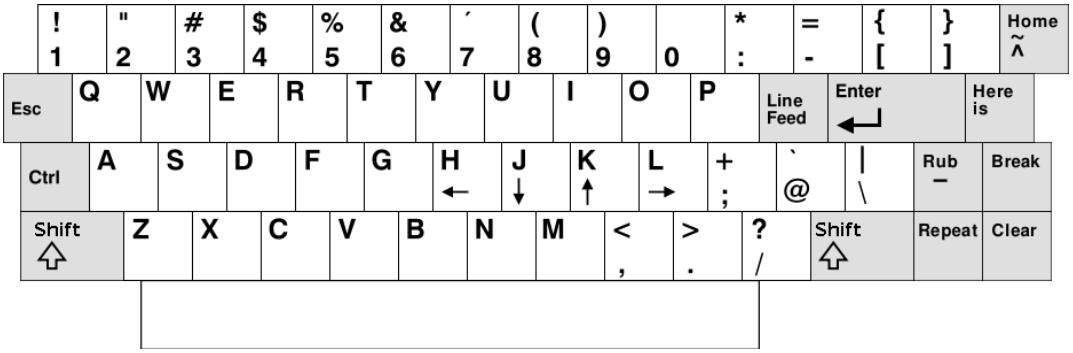
Z dôvodu, že v emulátore zatiaľ nie je možné pripájať jedno zariadenie do druhého, sú karta MITS 88-SIO a terminál ADM-3A implementované ako jedno zariadenie.

Typ	Počet
alfanumerické	47
špeciálne	7
ovládanie terminála	5

Tabuľka 9.1: *Typy kláves na klávesnici terminála ADM-3A*

9.1 Klávesnica

Dáta, ktoré sa budú spracovávať vznikajú buď v počítači, alebo sú zadané pomocou klávesnice. Vstup z reálnej klávesnice pozostával zo stlačenia jedného z 59-tich kláves. Každý stlačený kláves je zakódovaný podľa tabuľky ASCII a je okamžite odoslaný do počítača. Odoslaný znak je v poloduplexnom režime „odrazený“ späť a zobrazený na obrazovke. Aby bol znak zobrazený v plne-duplexnom režime, je potrebné znak manuálne odoslať z počítača.
Štandardná klávesnica terminálu je zobrazená na Obr. 9.3.



Obr. 9.3: *Štandardná klávesnica terminálu ADM-3A*

Tabuľka 9.1 zobrazuje typy a počet kláves terminála ADM-3A.

Implementovaný zásuvný modul rozširuje sadu znakov vyjadriteľných pomocou klávesnice, pretože berie do úvahy každý kláves, ktorý reprezentuje určitý zobraziteľný symbol. No väčšina riadiacich kláves je z dôvodu zachovania kompatibility ignorovaná (ich kódy sa vôbec neodošlú do počítača). Tabuľka 9.2 zobrazuje riadiace klávesy (prípadne ich kombináciu), ktoré sa nejakým spôsobom interpretujú, alebo ktoré *nie sú* implementovaným terminálom ignorované.

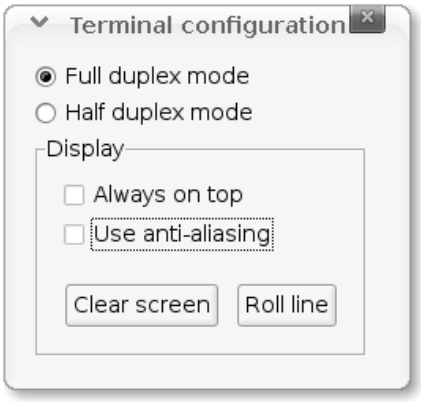
9.2 Konfigurácia zariadenia

Všimnite si tlačidlo *Config* na Obr. 9.2 (vpravo). Po kliknutí na toto tlačidlo sa otvorí dialógové okno, kde je možné nastaviť základnú konfiguráciu terminálu. Spomínané okno ukazuje Obr. 9.4.

Zmeny v nastavení sa prejavia ihneď, bez nutnosti ich potvrdenia. Prvé dve možnosti umožňujú nastaviť režim práce terminálu — poloduplexný alebo plneduplexný.

Riadiaci(e) kláves(y)	ASCII Kód	Interpretácia
DEL	7Fh	žiadna, zobrazí sa
ESC	1Bh	žiadna, zobrazí sa
BS, CTRL+H alebo ←	08h	posunie kurzor o 1 znak doľava
ENTER	0Dh	posunie kurzor na začiatok riadku
↑ alebo CTRL+K	0Bh	žiadna, zobrazí sa
↓ alebo CTRL+J	0Ah	Line Feed — kurzor sa posunie o 1 riadok dole
→ alebo CTRL+L	0Ch	žiadna, zobrazí sa

Tabuľka 9.2: *Riadiace klávesy a ich interpretácia*



Obr. 9.4: *Kofigurácia terminálu*

V okne *Display* je možnosť nastaviť GUI terminálu vždy-na-vrchu (*Always on top*), čo znamená, že sa okno nedostane do pozadia, ani keď nebude aktívne.

Ďalšou možnosťou (novinkou) je použitie techniky počítačovej grafiky anti-aliasingu, ktorou sa pixely vyhladia a zjemnia tak obraz. Pre programy, ktoré pracujú len s textom, sa neodporúča používať anti-aliasing, lebo text bude trochu rozmazaný.

Tlačidlá nachádzajúce sa pod týmto checkboxom plnia funkcie zmazania obrazovky (*Clear screen*), resp. posunutiu obrazu o jeden riadok vyššie (*Roll line*).

Ak sme ukončili nastavovanie konfigurácie, okno stačí zavrieť.

Kapitola 10

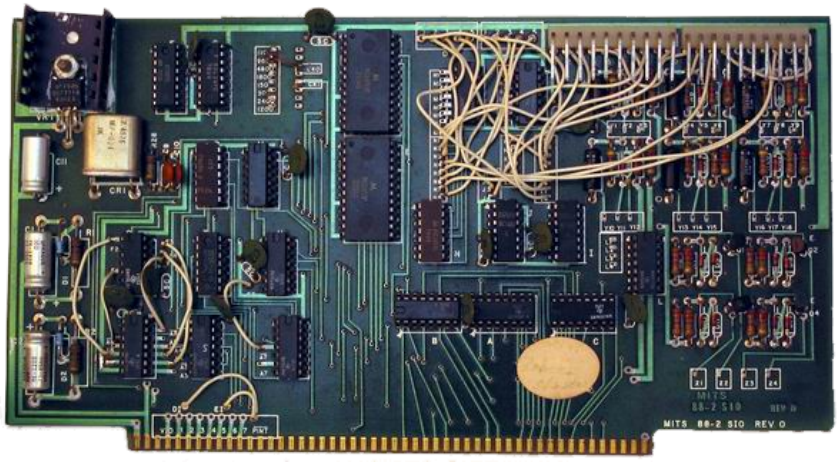
Sériová karta MITS 88-SIO

Táto kapitola popisuje zásuvný modul s názvom *MITS-88-SIO serial card*, súbor `MITS-88-SIO.jar`, verzia `1.5b1`.

Klasický počítač Altair 8800 bol vybavený touto sériovou kartou, do ktorej sa pripájali ďalšie zariadenia. Karta komunikovala so zariadeniami prostredníctvom sériového rozhrania RS-232 a bola priamo pripojená na IO porty CPU — `10h` a `11h` (to platí pre počítače Altair). Súčasná verzia zásuvného modulu už umožňuje nastavenie aj iných CPU portov, na ktoré sa môže pripojiť.

Existovali dva typy týchto sériových kariet: **a) klasická SIO-88** — mala iba jeden I/O port a umožňovala pripojiť najviac jedno zariadenie (náš prípad) a **b) karta SIO-88-2** — mala dva I/O porty a umožňovala tak zapojiť dve zariadenia naraz.

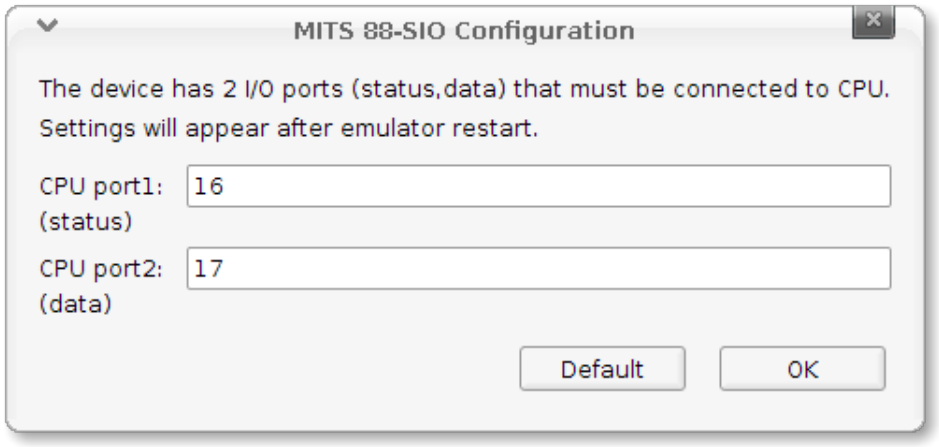
Obr. 10.1 ukazuje kartu MITS 88-SIO-2, ako vyzerala v skutočnosti.



Obr. 10.1: *Sériová karta MITS 88-SIO-2*

10.1 Nastavenie CPU portov

Súčasná verzia zásuvného modulu sériovej karty už umožňuje zmenu štandardných CPU portov, na ktoré sa táto karta pripája. To rozširuje jej využitie aj pre iné počítače, ktoré používali túto sériovú kartu. Nastavenia CPU portov sa spúšťa kliknutím na tlačidlo *Settings* v hlavnom module v paneli emulátora, pri súčasnom vyznačení tejto karty v zozname zariadení. Okno s nastaveniami je možné vidieť na Obr. 10.2.



Obr. 10.2: *Okno s nastavením CPU portov zásuvného modulu karty MITS 88-SIO*

Tlačidlo *Default* nastaví textové okná na štandardné hodnoty portov (`10h` a `11h`) a tlačidlom *OK* potvrdíme nové nastavenia portov, ktoré sa uložia do konfiguračného súboru a prejavia sa až po reštarte emulátora.

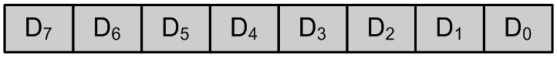
10.2 Programovanie karty a terminálu

V tejto časti sa zameriame na prácu s terminálom ADM-3A. Predpokladáme, že je zapojený do tejto sériovej karty. Na komunikáciu s terminálom potom stačí programovať sériový port. Interpretáciu riadiacich znakov terminálu zabezpečí sám terminál. Príklady v tejto časti sú napísané pre procesor i8080.

Nastavenia sériového portu (prenosová rýchlosť, parita, počet stop bitov apod.) sa realizovali ručne prepínačmi na karte, programátor s týmito vecami neprichádzal do styku. Keďže je emulátor zjednodušenou približeninou reálneho počítača, nebolo potrebné zachádzať až do takýchto detailov.

Sériová karta mala jeden fyzický I/O port, ktorý mohol byť spojený s ľubovoľným sériovým I/O zariadením. Celý vstup-výstup je programovateľný. Tento fyzický I/O port má dva vnútorné porty: stavový (číslo $10h$ (16)) a dátový (číslo $11h$ (17)), ktoré sú nastaviteľné (viď predchádzajúcu časť) takže je treba dať pozor na to, aké porty sú nastavené.

Zápis na stavový port umožňuje vybrať niektoré možnosti pre zariadenie (podporovaná je len hodnota $03h$, čo spôsobí reset portu). Čítanie zo stavového portu vráti jeho stav:



Obr. 10.3: *Usporiadanie bitov v bajte*

D_1 — 1 na tejto pozícii znamená, že bol prijatý znak zo zariadenia na dátový port a je pripravený na čítanie

D_0 — 1 na tejto pozícii znamená, že port je pripravený prijať znak na dátový port a tak ho vyslať zariadeniu

Význam ostatných bitov je nepodstatný, v emulátore sa nepoužíva. Išlo napr. o kontrolu, či boli dáta správne prijaté.
Čítanie dátového portu vráti bufferovaný znak, zápis na dátový port zapíše znak do zariadenia.
Programovanie terminálu sa tak stáva veľmi jednoduchým. Pre výpis znaku na obrazovku stačí na port $11h$ vyslať znak, napríklad:

```
MVI A, 'A'
OUT 11h
MVI A, 'h'
OUT 11h
MVI A, 'o'
OUT 11h
MVI A, 'j'
OUT 11h
```

Takéto vypisovanie znakov je však nepraktické, lepšie je napísať si vlastnú procedúru:

```
lxi h, text  ; načítaj do HL adresu návestia text
call print   ; zavolaj procedúru na výpis textu
hlt          ; zastav CPU

text: db 'Ahoj',0

; v registrovom pári HL sa očakáva adresa reťazca ASCIIZ
; reťazec teda musí byť ukončený nulovým znakom

print:
    mov a, m  ; načítanie znaku z pamäte
    inx h     ; posunutie ukazovateľa
    cpi 0     ; je znak 0 ?
    rz        ; ak áno, návrat z procedúry
    out 11h   ; inak výpis znaku
    jmp print ; a opakovanie činnosti
```

Na dotazovanie dát z klávesnice jednoducho stačí čítať stavový port dovtedy, kým nebude pripravený znak na dátovom porte. Potom stačí tento znak prečítať. Jednoduchá procedúra:

```
getchar:
    in 10h
    ani 1       ; je znak načítaný v buffri ?
    jz getchar
    in 11h      ; čítaj znak (do registra A)
```

V prípade, že chceme prečítať celý riadok znakov, ktorý uložíme na adresu špecifikovanú registrovým párom `DE`, s interpretáciou klávesy backspace, treba napísať trochu zložitejšiu procedúru:

```
lxi h, text         ; načíta adresu návestia text do HL
xchg                ; DE <-> HL
call getline        ; načíta riadok z klávesnice na adresu DE

lxi h, text         ; opäť do HL adresa textu
call print          ; vypíše načítaný text na obrazovku

hlt                 ; zastaví CPU

text: ds 30         ; tu sa uloží načítaný textprenášajú

;tu je procedúra na načítanie reťazca z klávesnice
;vstup: DE = adresa, kde sa má načítať text
getline:
    mvi c, 0        ; reg. C použijeme ako počítadlo
                    ; znakov
next_char:
    in 10h          ; vstup: stavový port
    ani 1           ; je znak načítaný v buffri ?
    jz next_char    ; ak nie, skok
    in 11h          ; inak čítaj znak

    ; interpretácia niektorých klávesov
    cpi 13          ; ENTER?
    jz getline_ret  ; áno - skok na koniec
    cpi 8           ; backspace ?
    jnz save_char   ; ak nie, skok na uloženie znaku

    mov a, c        ; testovanie, či sme na začiatku
    cpi 0           ; tj. či počet znakov je 0
    jz next_char    ; ak áno, ignorujeme backspace skokom
```

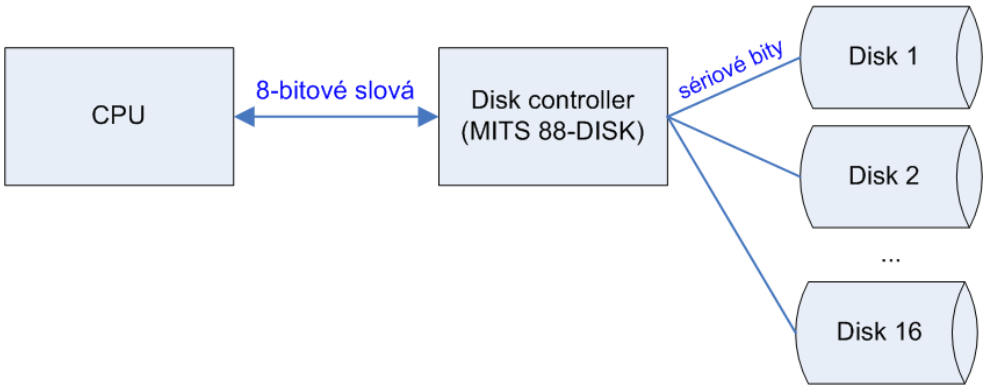
```
                    ; na ďalší znak
    dcx d           ; inak dekrementácia adresy
    dcr c           ; dekrementácia počtu znakov
    mvi a,8         ; "zobraz" backspace (posunie kurzor
                    ; doľava)
    out 11h
    mvi a, 32       ; "zmaž" znak, na ktorom je kurzor medzerou
    out 11h         ; (terminál znakom BS nemaže znak, iba
                    ; posúva kurzor)
    mvi a,8         ; a opäť "zobraz" backspace
    out 11h
    jmp next_char   ; a skok na ďalší znak

save_char:          ; tu je časť procedúry na uloženie znaku
    out 11h         ; zobraz znak
    stax d          ; ulož znak na adresu DE
    inx d           ; inkrementuj DE
    inr c           ; inkrementuj počet znakov
    jmp next_char   ; skok na ďalší znak

getline_ret:        ; tu je časť procedúry na ukončenie
                    ; čítania znakov
    mvi a, 10       ; LINE FEED
    stax d          ; ulož znak na DE
    inx d           ; inkrementácia DE
    mvi a,13        ; CARRIAGE RETURN
    stax d          ; ulož znak na DE
    inx d           ; inkrementácia DE
    mvi a, 0        ; a ukonč reťazec znakom 0
    stax d
    ret             ; návrat
```

Kapitola 11

Diskový radič MITS 88-DISK

Táto kapitola popisuje zásuvný modul s názvom *MITS-88 DISK (floppy drive)*, súbor `MITS-88disk.jar`, verzia `1.5b1`.

Altair Disk ponúkal výhodu stálej pamäte vrátane relatívne rýchleho prístupu k dátam. Dáta boli prenášané na disk a z disku rýchlosťou 250 Kb/s (zásuvný modul však toto nerieši). Disk bol napojený na diskový radič, pričom dáta sa z radiča na disk a späť prenášali sériovo, bit po bite. Diskový radič, komunikujúci na druhej strane s CPU, tieto sériové dáta transformoval na 8-bitové slová, ktoré CPU ukladala/čítala do/z operačnej pamäte v závislosti od toho, či údaje boli čítané alebo zapisované. Situácia je zobrazená na Obr. 11.1. Na jeden diskový radič mohlo byť pripojených až 16 diskových zariadení (jedno takéto zariadenie ukazuje Obr. 11.2).



Obr. 11.1: *Diagram systému Altair Disk*

Diskový radič tiež ovláda všetky mechanické funkcie disku, vrátane poskytovania informácií o stave disku pre CPU. Vedel pracovať s prerušeniami, aby programy nemuseli čakať na ukončenie diskových operácií. V mojom zásuvnom module prerušenia nie sú podporované.

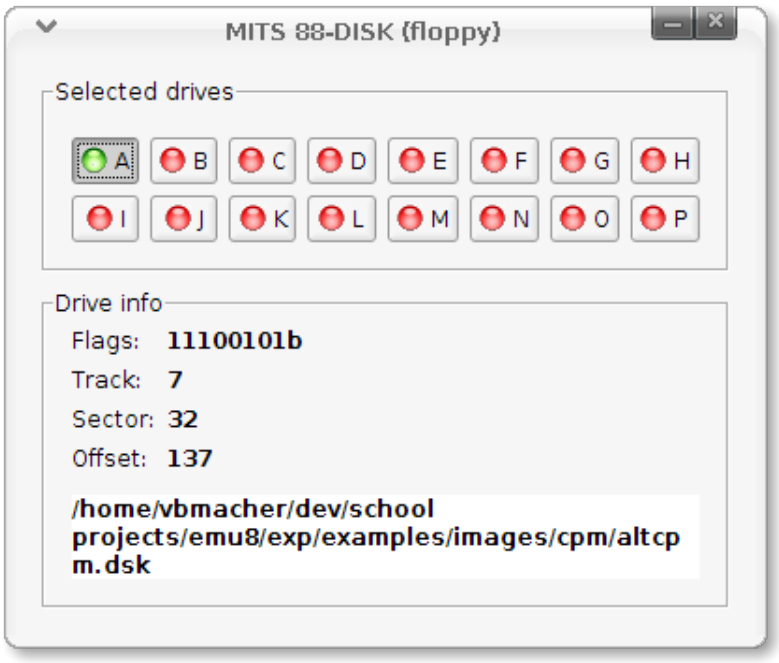
Diskety, ktoré sa vkladali do diskového zariadenia (Obr. 11.2), boli napevno naformátované; každá disketa bola rozdelená na 77 stôp (*tracks*), každá stopa mala 32 sektorov a každý sektor obsahoval 137 elementárnych dátových buniek (bytov). Keď to všetko zrátame, disketa mala kapacitu $77 \cdot 32 \cdot 137 = 337\ 568B \doteq 330$ KB. Pre programy sa však využívala menšia kapacita,

Obr. 11.2: *Diskové zariadenie Pertec FD400*

pretože 9 bytov z každého sektora sa využívali na integritnú sumu. Na sektor tak ostávajú 128 bytov pre dáta a konečná kapacita diskety je $77 \cdot 32 \cdot 128 = 315\,392B = 308$ KB.

11.1 Popis zásuvného modulu

Implementovaný zásuvný modul emuluje základnú funkcionalitu celého diskového systému počítača Altair 8800. Grafické rozhranie zásuvného modulu ukazuje Obr. 11.3.



Obr. 11.3: *Používateľské rozhranie zásuvného modulu*

Okno podáva používateľovi informácie o aktuálnom stave jednotlivých diskových zariadení počas behu emulácie. K dispozícii máme 16 tlačidiel označených ako A až P. Kliknutím na

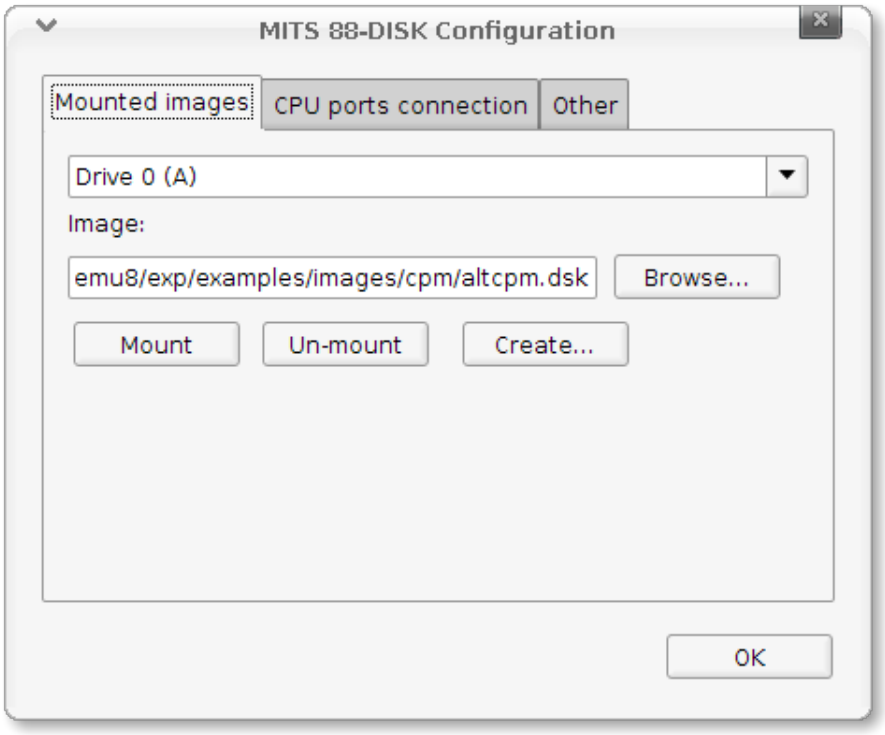
niektoré z týchto tlačidiel informujeme zásuvný modul, že chceme poznať stav vybraného disku.

Stav disku je zobrazený v spodnej časti okna. Jeho obsahom sú príznaky radiča pre vybraný disk (*Flags*), a pozícia na diskete (číslo stopy *Track*, číslo sektora *Sector*, a pozícia v rámci sektora *Offset*), na ktorej sa disk nachádza. Tieto informácie sú užitočné pre programátorov, ktorí testujú svoje riešenia na disketových obrazoch. Poslednou informáciou je názov súboru (obrazu), ktorý je namontovaný na príslušnom disku.

11.2 Práca s obrazmi diskiet

Ako už bolo povedané, okrem radiča (ktorý prichádza priamo do styku s CPU) zásuvný modul implementuje aj 16 diskových zariadení. Na každé zariadenie môže používateľ „namontovať" obraz diskety[1], čo odpovedá vloženiu diskety do mechaniky. Naopak namontované obrazy môžu byť kedykoľvek odmontované (čiže akoby „vybrané z mechaniky").

Práca s obrazmi diskiet a s inými nastaveniami zásuvného modulu musíme otvoriť grafické okno s nastaveniami, a to kliknutím na tlačidlo *Settings* v hlavnom module v paneli emulátora pri súčasnom výbere tohto zásuvného modulu zo zoznamu zariadení. Okno je zobrazené na Obr. 11.4.



Obr. 11.4: *Okno s nastaveniami, panel práce s obrazmi*

[1] musí byť veľkosti presne 337 568B

Ak chceme „vložiť disketu“ do diskového zariadenia, v reprezenácii zásuvného modulu to znamená namontovanie obrazu diskety. Obraz diskety je vlastne obyčajný binárny súbor ľubovoľného formátu. Význam údajov v rámci obrazu dáva až samotný program.

Na namontovanie obrazu je potrebné si v hornej časti okna vybrať konkrétne zariadenie, na ktoré ideme obraz namontovať (výsuvná ponuka s označeniami diskov *Drive 0 (A)* až *Drive 15 (P)*).

POZNÁMKA:

Montovanie obrazov je možné vykonávať dynamicky aj počas behu emulácie — ak je obraz na disku už namontovaný, môžme ho „premontovať“ na iný.

Ak je vybratý disk, na ktorý chceme namontovať obraz diskety, do políčka *Image* je potrebné napísať cestu k súboru obrazu, resp. si ho vybrať interaktívne kliknutím na tlačidlo *Browse*.

Samotný obraz na disk namontujeme kliknutím na tlačidlo *Mount*. Obraz z nejakého disku odmontujeme kliknutím na tlačidlo *Unmount*. To odpovedá situácii, že v diskovom zariadení nie je vložená disketa.

Nový obraz diskety môžme vytvoriť pomocou tlačidla *Create image*. Vybratím cesty a názvu súboru vytvoríme obraz o spomínanej veľkosti. Jeho obsahom sú hodnoty s ASCII kódom 0.

11.3 Nastavenie CPU portov

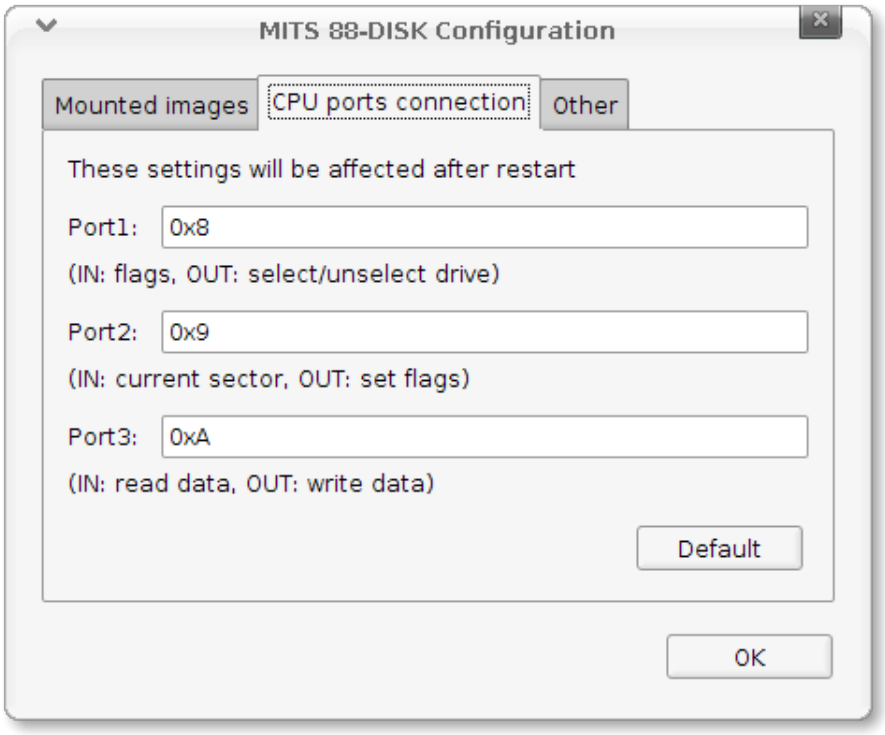
Tak ako aj sériová karta, aj radič disku sa pripája na určité porty na CPU. Štandardné čísla portov sú `08h`, `09h` a `0Ah`. Súčasná verzia zásuvného modulu disku už umožňuje zmenu štandardných CPU portov, na ktoré sa tento radič pripája. To rozširuje jeho využitie aj pre iné počítače, ktoré používali tento disk. Nastavenia CPU portov sa nachádza v paneli *CPU ports connection* v okne s nastaveniami (Obr. 11.5).
Tlačidlo *Default* nastaví textové okná na štandardné hodnoty portov (`08h`, `09h` a `0Ah`).

11.4 Ostatné nastavenia a uloženie nastavení

Posledný, tretí panel v okne s nastaveniami umožňuje nastaviť hlavné okno zásuvného modulu (Obr. 11.3), aby bolo vždy na vrchu obrazovky, aj keď sa stane neaktívne (checkbox *Always on top*). Okno je zobrazené na Obr. 11.6.

Druhým checkboxom (*Save settings*), ak ho zaškrtneme, potvrdíme, že nastavenia v každom paneli sa uložia aj do konfiguračného súboru. Pre nastavenia CPU portov je to jediná cesta, ako nastavenia realizovať (lebo sa prejavia až po reštarte emulátora).

Uloženie a potvrdenie nastavení realizujeme kliknutím na tlačidlo *OK*. Ak je zaškrtnutý checkbox *Save settings*, uložia sa nastavenia o namontovaní jednotlivých diskov, CPU porty a či má byť okno stále na vrchu.



Obr. 11.5: *Okno s nastaveniami, panel nastavení CPU portov*

11.5 Programovanie radiča

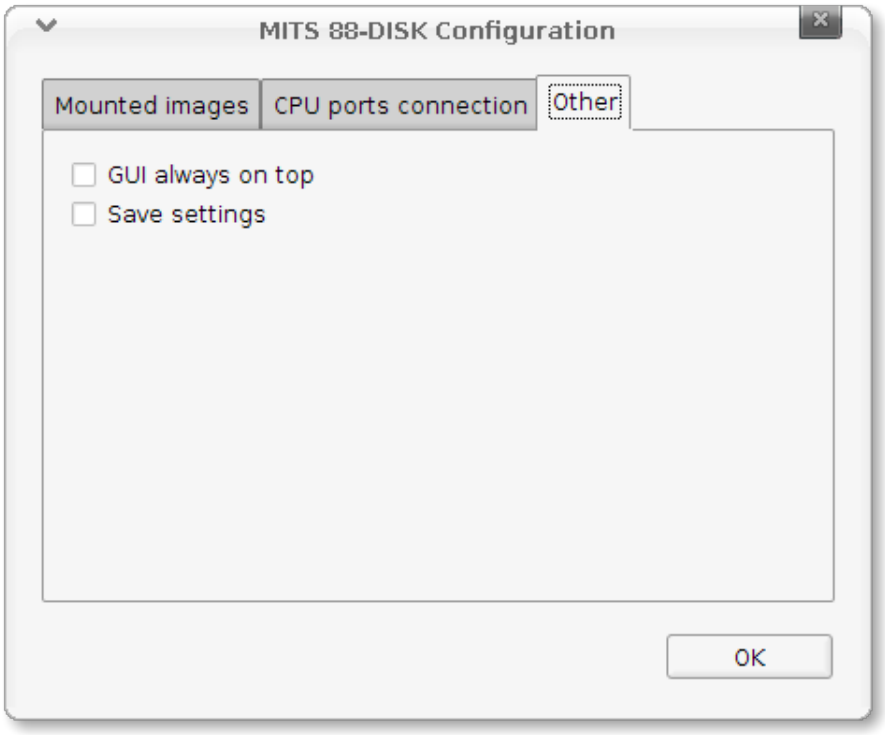
Dáta sa na disk zapisujú a z disku čítajú súvislo. Pozícia na diskete je jednoznačne popísaná číslom stopy, číslom sektora a pozíciou v rámci sektora. Číslo stopy a sektora je možné nastaviť celkom jednoducho, a to ich postupnou inkrementáciou/dekrementáciou. Pozícia v rámci sektora sa nastavuje trochu zložitejšie. Ak chce program zapísať údaj na určitú pozíciu, musí najprv nastaviť stopu, nastaviť sektor, a potom počkať, kým sa nastaví pozícia v rámci sektora na jeho začiatok. Od tohto momentu dáta z disku musí čítať, a odrátavať si počet čítaní, kým sa dostane na pozíciu, na ktorú chce zapisovať. Je to dosť zložitý prístup pre programovanie, ale má to výhodu v jednoduchšej hardvérovej realizácii. Neskôr v tejto časti uvediem presný postup, ako dáta zapisovať, alebo čítať.

Radič komunikuje s CPU prostredníctvom troch vstupno-výstupných portov na adresách `08h`, `09h` a `0Ah`. Nasledujúca tabuľka uvádza prehľad sémantiky (významu) použitia portov vo vzťahu k čítanu/zapisovaniu z nich (Tab. 11.1).

Nasleduje presný popis vstupov/výstupov z jednotlivých portov. Usporadanie bitov v prijatých/ododsielaných bytoch je podľa Obr. 10.3.

Port `08h`

zápis: Vyberá a aktivizuje jedno zo 16-tich diskových zariadení. V prípade, že na vybranom disku nie je namontovaný obraz, všetky ďalšie operácie na disku



Obr. 11.6: *Okno s nastaveniami, panel s ostatnými nastaveniami*

Adresa	Vstup (čítanie z portu)	Výstup (zápis na port)
08h	Vracia stav disku a radiča	Vyberá (aktivizuje) a povoľuje konkrétny disk
09h	Vracia číslo sektora	Ovláda funkcionalitu (nastavenia) disku
0Ah	Čítanie dát z disku	Zápis dát na disk

Tabuľka 11.1: *Prehľad sémantiky použitia I/O portov radiča*

budú ignorované.

D_0(LSB) až D_3(MSB) výber a aktivácia jedného zo 16-tich diskových zariadení, každé zariadenie má priradené jednoznačnú adresu od 0 do 15. Výberom zariadenia sa predchádzajúce zariadenie deaktivuje. Výber diskového zariadenia znamená, že všetky diskové operácie sa budú vykonávať len na vybranom diskovom zariadení.

D_4, D_5, D_6 nepoužité bity

D_7 ak má hodnotu 1, deaktivuje sa aktuálne diskové zariadenie (neostane žiadne aktívne), bez ohľadu na to, akú hodnotu majú bity D_0 až D_3.

čítanie: Prijatý bajt z tohto portu indikuje stav disku, keď je disk a radič povolený. Platí tu opačná logika, čiže pri platnej podmienke majú bity hodnotu 0, a pri neplatnej podmienke hodnotu 1.

- **D_0** *Enter new write data* — indikuje, či je radič pripravený na zápis dát. Resetuje sa, keď sú dáta zapísané na port `0Ah`. Kontrola tohto bitu má zmysel iba vtedy, keď je povolený zápis dát (viď zápis na port `09h`). V prípade, že hodnota tohto bytu je `false`, akékoľvek dáta zapísané na port `0Ah` budú ignorované.
- **D_1** *Move head* — indikuje povolenie pohybu hlavy. Má zmysel to kontrolovať pri nastavovaní stopy. Ak pohyb hlavy nie je povolený, pokus o posun stopy bude ignorovaný (viď zápis na port `09h`)
- **D_2** *Head Status* — zisťuje, či je hlava a stopa korektne nastavená. Tento bit nadobudne hodnotu `true`, po načítaní hlavy, alebo po presune stopy. Ak má tento bit hodnotu `true`, čítanie čísla sektora z portu `09h` bude platné.
- **D_3, D_4** nepoužité bity, majú stále hodnotu 0
- **D_5** *Interrupt Enabled* — indikuje povolenie prerušení, v zásuvnom module však prerušenia disku nie sú implementované
- **D_6** *Track 0* — indikuje, či sa hlava nachádza na stope číslo 0. Na diskete je táto stopa umiestnená na jej vonkajšom okraji.
- **D_7** *New read data available* — indikuje, že radič má pripravený 1 byte na čítanie z portu `0Ah`. Resetuje sa, keď sa údaj z portu `0Ah` prečíta. V prípade, že tento bit má hodnotu `false`, tak dáta prečítané z portu `0Ah` budú neplatné.

Port `09h`

zápis: Riadi radič disku, keď je vybraté a povolené diskové zariadenie. Tu platí klasická logika, hodnota 1 operáciu alebo vlastnosť nastaví, 0 zakáže/zruší.

- **D_0** *Step In* — posunie hlavu dopredu o 1 stopu (inkrementuje číslo stopy). Pred týmto príkazom je nutné overiť platnosť bitu D_1 čítaním z portu `08h` (*Move head*). Ak je tento bit 0, zmena stopy prebehne v poriadku.
- **D_1** *Step Out* — posunie hlavu späť o 1 stopu (dekrementuje číslo stopy). Aj pred týmto príkazom je nutné overiť platnosť spomínaného bitu *Move head*.
- **D_2** *Head load* — nastaví (umiestni) čítaco/zapisovaciu hlavu na povrch disku, čítanie pozície sektora z portu `09h` sa stane platné. Ak je zároveň nastavený bit D_7 čítaním z portu `08h` (*New read data available*), je možné dáta z disku čítať.
- **D_3** *Head unload* — odoberie (premiestni) hlavu z povrchu disku, čítanie pozície sektora sa stane neplatné. Rovnako sa zruší platnosť spomínaného bitu D_7 z portu `08h` (*New read data available*).
- **D_4** *Interrupt Enable* — povolenie prerušení, v mojom zásuvnom module prerušenia disku nie sú implementované
- **D_5** *Interrupt Disable* — zakázanie prerušení
- **D_6** *Head Current Switch* — v reálnych diskoch tento bit by mal byť nastavený na 1, ak program zapisuje dáta na stopy $43 \div 76$. V mojom zásuvnom module je tento bit ignorovaný, a nemusí byť nastavený.

D_7 *Write Enable* — iniciuje zapisovaciu sekvenciu (povoľuje zápis na disk). Zásuvný modul po obdržaní tohto príkazu nastaví číslo sektora na 0 a dá do platnosti bit D_0 (*Enter new write data*) pri čítaní z portu `08h`. Pri opačnej logike bude bit platný, ak má hodnotu 0. Podľa manuálu [5] táto sekvencia trvá len určitý čas, maximálne však do dosiahnutia konca sektora. Zásuvný modul neobmedzuje dĺžku trvania sekvencie, táto sa deaktivuje jedine po dosiahnutí konca sektora. Manuál disku tiež hovorí, že každý prvý zapisovaný byte na sektore má mať nastavený najvyšší bit (tzv. *sync bit*) na hodnotu 1 a posledný byte (137-my byte rátaný od nuly) má mať hodnotu 0. Bolo to z dôvodu, aby hardvér vedel ľahšie identifikovať začiatok a koniec sektora. Zásuvný modul to však nevyžaduje.

čítanie: čítané údaje z tohto portu sú platné len vtedy, ak je vybraté diskové zariadenie (zápisom na port `08h`) a ak je čítaco/zapisovacia hlava umiestnená na povrchu disku (nastavením bitu D_2 na porte `09h`). Vracia číslo sektora, na ktorom sa nachádza hlava.

D_0 *Sector True* — ak je tu hodnota 0, pozícia na diskete v rámci sektora (*offset*) je 0. Podľa manuálu je tento bit nastavený iba $30\mu s$. Zásuvný modul však nastavenie tejto hodnoty neobmedzuje na čas. Je prakticky nastavená vždy a to z dôvodu, že pri načítaní pozície sektora sa tento stále mení a vždy dochádza k znovunastaveniu offsetu na počiatočnú hodnotu 0. Sektor sa teda pri čítaní inkrementuje modulo 32. Emuluje sa tým rotácia disku, ktorá v reálnom zariadení existovala. Práve rotácia disku spôsobuje zmenu sektora v čase — tak za určitý čas po niekoľkých čítaniach čísla sektora sa môže vrátiť iná hodnota. Keďže bol disk oveľa pomalší ako CPU, zápis/čítanie dát stihlo prebehnúť v poriadku.

D_1(LSB) až D_5(MSB) číslo sektora, počíta sa od 0.

D_6, D_7 nepoužité bity, sú nastavené na hodnotu 1.

Port `0Ah`

zápis: Zapísaním bytu na tento port sa tento bajt zapíše na aktuálnu pozíciu na diskete vo vybratom diskovom zariadení. Aby mohlo dôjsť k zápisu dát, musí byť aktivovaná zapisovacia sekvencia (zápis bitu D_7 na port `09h`). Predtým, ako program niečo zapíše na disk, je potrebné skontrolovať bit D_0 čítaním z portu `08h`.

čítanie: Na to, aby mohli byť údaje z diskety čítané, je potrebné nastaviť bit D_2 zápisom na port `09h`. Ak je nastavený bit D_7 čítaním z portu `08h` (*New read data available*), potom z portu `0Ah` môžme prečítať platný byte z diskety na aktuálnej pozícii.

11.5.1 Príklad

Nasleduje programová implementácia príkladu toho, ako zapisovať/čítať údaje z presnej pozície na diskete. Program využíva 3 procedúry na nastavenie pozície na diskete (`ltrack` pre

nastavenie stopy, `lsector` pre nastavenie sektora a `loffset` pre nastavenie pozície v rámci sektora), a dve ďalšie na čítanie (`read`) a zápis (`write`). Príklad zapíše ASCII hodnotu 65 (veľké `A`) na pozíciu: stopa 1, sektor 18, offset 20, ktorú načíta do operačnej pamäte na adresu $200h$.

```
; testovací program pre zápis/čítanie

disk0  equ 0    ; číslo disku
track  equ 1    ; číslo stopy
sector equ 18   ; číslo sektora
offset equ 20   ; pozícia v rámci sektora
data   equ 'A'  ; dáta na zápis/čítanie

dcx sp          ; nastaví register zásobníka
                ; na hodnotu 0FFFFh

mvi a, disk0    ; vyberie disk
out 08h

call ltrack     ; nastavenie stopy

call we         ; zapnutie sekvencie write enable
call lsector    ; nastavenie sektora
call loffset    ; nastavenie offsetu
call write      ; zápis dát

call lsector    ; znovunastavenie sektora (aby
                ; sa vynuloval offset)
call loffset    ; nastavenie offsetu
call read       ; čítanie dát

lxi h, readdata ; adresa, kde sa načítané dáta
                ; uložia
mov m, a        ; presun dát na nastavenú adresu

hlt             ; koniec

ltrack0:        ; procedúra nastaví stopu 0
in 08h          ; zistenie stavu disku
ani 1000000b    ; stopa 0 ?
rz              ; áno, návrat
mvi a, 1000b    ; head unload
out 09h
call movetrk    ; čakanie, kým sa dá pohnút hlavou
mvi a, 10b      ; step out, dekrementácia stopy
```

```
out 08h
jmp ltrack0

ltrack:          ; procedúra nastaví danú stopu
call ltrack0     ; najprv načíta stopu 0
mvi b, track+1   ; b = track + 1
stepin:          ; stepin: {
dcr b            ;   b--;
rz               ;   if (b == 0) return;
call movetrk     ;   čakanie, kým sa dá pohnúť hlavou
mvi a, 1         ;   step in, inkrementácia stopy
out 09h
jmp stepin       ;   goto stepin;
                 ; }

movetrk:         ; čakanie, kým sa dá pohnúť hlavou
in 08h           ; zistenie stavu disku
ani 10b          ; môžme hýbať hlavou ?
jnz movetrk      ; nie, čakaj...
ret              ; áno, návrat

lsector:         ; nastaví požadovaný sektor
mvi a, 100b      ; head load
out 09h
waits:
in 09h           ; zisti číslo sektora
ani 3Fh          ; zmaž nepoužité bity
rrc              ; rotácia doprava
cpi sector       ; je to žiadaný sektor ?
jnz waits        ; nie, skúšaj ďalej
ret              ; áno, návrat

loffset:         ; nastaví požadovaný offset
mvi b, offset+1  ; b = offset + 1
stepoff:         ; stepoff: {
dcr b            ;   b--;
rz               ;   if (b == 0) return;
call read        ;   čítaj dáta a tým sa
                 ;   inkrementuje offset
jmp stepoff      ;   goto stepoff;
                 ; }

read:            ; čítanie dát z diskety
in 08h           ; zisti stav disku
```

```
ani 100b        ; hlava musí byť nastavená
                ; na povrchu disku
rnz             ; ak nie je, návrat
waitr:
in 08h          ; zisti stav disku
ani 10000000b   ; New read data available ?
                ; sú dáta pripravené na čítanie?
jnz waitr       ; nie, čakaj...
in 0Ah          ; áno, čítaj dáta
ret             ; návrat

we:             ; zapnutie sekvencie write enable
mvi a, 10000000b ; write enable
out 09h
ret

write:          ; zápis dát na disketu
in 08h          ; zisti stav disku
ani 100b        ; hlava musí byť nastavená
                ; na povrchu disku
rnz             ; ak nie, návrat
waitw:
in 08h          ; zisti stav disku
ani 1           ; enter new write data ?
                ; môžem zapisovať ?
jnz waitw       ; nie, čakaj...
mvi a, data     ; áno, zápis
out 0Ah
ret

org 200h
readdata: db 0
```

Kapitola 12

Pseudo-zariadenie SIMH

Pseudo-zariadenie *SIMH* je vymyslené zariadenie, ktoré zabezpečuje komunikáciu medzi emulovaným počítačom a emulátorom. Bolo vytvorené z dôvodu umožnenia behu niektorých systémov, ktorých obrazy boli vytvorené (alebo modifikované) pre beh na emulátore SIMH [6]. Zariadenie je implementované v emulátore počítača Altair v systéme SIMH a do verzie pre emulátor ***emuStudio*** bolo *portované* len s obmedzenou funkcionalitou. Zásuvný modul má názov *SIMH pseudo device*, súbor `SIMH-pseudo.jar`, verzia `SIMH003`.

POZNÁMKA:
Manipulácia s touto časťou zdrojového kódu emulátora SIMH je možná, vďaka jeho licencii

Ako príklad by sa dal uviesť operačný systém CPM v.3, ktorý potrebuje bankovanú operačnú pamäť a toto zariadenie umožňuje selekciu pamäťových bánk. Pôvodné zariadenie v emulátore SIMH poskytuje množstvo funkcionality, ako napr.:

- zobrazenie a zisťovanie aktuálneho času
- zistenie verzie SIMH pseudo-zariadenia
- reset SIMH pseudo-zariadenia
- meranie času behu ľubovoľnej časti programu použitím časovačov
- pripájanie a odpájanie kazetových (PTP, PTR) zariadení počas behu emulácie
- zisťovanie a selekcia bánk operačnej pamäte, zistenie adresy spoločnej pamäte pre všetky banky
- zmena procesora počas behu emulácie (na výber sú 8080 a Z80)
- pravidelné prerušenia časovača (pri prerušení sa vykoná inštrukcia `CALL` na nastavenú adresu)
- pozastavenie emulácie na určitú dobu

- prenos dát (súborov) medzi emulovaným systémom a emulátorom

Zásuvný modul poskytuje len obmedzenú funkcionalitu:

- zistenie verzie SIMH pseudo-zariadenia
- reset SIMH pseudo-zariadenia
- zisťovanie a selekcia bánk operačnej pamäte, zistenie adresy spoločnej pamäte pre všetky banky

Tieto funkcie stačia na to, aby mohla väčšina systémov pre Altair (modifikovaných pre SIMH) bežať. Dôvod, prečo je taká snaha o beh systémov *modifikovaných pre SIMH*, je jednoduchý: nie sú k dispozícii originály.

12.1 Programovanie zariadenia

Programy bežiace v emulovanom prostredí komunikujú so zariadením cez port `0FEh`. Posielanie a prijímanie správ medzi CPU a zariadením sa riadi nasledujúcimi princípmi:

1. Príkazy (`<cmd>`), ktoré nepotrebujú parametre a nevracajú výsledky sa posielajú takto:

   ```
   ld  a,<cmd>
   out (0feh),a
   ```

   Špeciálnym prípadom je príkaz `reset`, ktorý potrebuje byť odoslaný 128 krát, aby sa zaistilo, že vnútorný stav zariadenia je správne resetovaný.

2. Príkazy (`<cmd>`) s parametrami (`<p1>`, `<p2>`, ...), ale ktoré nevracajú výsledok, sa posielajú takto:

   ```
   ld  a,<cmd>
   out (0feh),a
   ld  a,<p1>
   out (0feh),a
   ld  a,<p2>
   out (0feh),a
   ...
   ```

   Volajúci program musí poslať všetky parametre ako samostatné byty. V opačnom prípade sa bude pseudo-zariadenie nachádzať v nedefinovanom (nestabilnom) stave.

3. Pre príkazy (`<cmd>`), ktoré nepotrebujú parametre a vracajú výsledok, platí nasledujúci postup:

```
ld  a,<cmd>
out (0feh),a
in  a,(0feh)    ; <A> obsahuje prvý byte výsledku
in  a,(0feh)    ; <A> obsahuje druhý byte výsledku
...
```

Volajúci program musí „vybrať“ všetky byty výsledku. V opačnom prípade sa bude pseudo-zariadenie nachádzať v nedefinovanom (nestabilnom) stave.

4. Príkazy, ktoré potrebujú parametre, a vracajú výsledok v súčasnej verzii zásuvného modulu neexistujú

Nasledujúca tabuľka uvádza zoznam dostupných príkazov a ich popis:

Č.	**Význam**	**Parametre**	**Výsledok**
6	Zistí verziu SIMH zariadenia	žiadne	8 bytov reťazca `SIMH003`
11	Vráti aktívnu banku pamäte	žiadne	1 byte aktívna banka pamäte
12	Nastaví novú aktívnu banku pamäte	1 byte, číslo banky	žiadny
13	Zistí adresu spoločnej pamäte pre všetky banky	žiadne	2 byty adresa spoločnej pamäte
14	Reset SIMH pseudo-zariadenia	žiadne	žiadny
18	Zistí, či systém podporuje bankovanie pamäte	žiadne	1 byte, s významom: 0 - nepodporuje sa bankovanie 1 - podporuje sa bankovanie

12.1.1 Príklad

Tento príklad predstavuje ukážku použitia bankovania pamäte. Predpokladáme, že operačná pamäť je rozdelená aspň do dvoch bánk, veľkosti celého adresného priestoru. Ďalej predpokladáme, že aktívna je prvá banka (poradové číslo 0) a že spoločná pamäť pre všetky banky je `0C000h`. Príklad je napísaný pre procesor Z80.

```
org 0C000h      ; sme v spoločnej pamäti

ld c, 0FEh      ; číslo portu do C

ld a, 18        ; zistenie podpory bankovania
out (C), a
in a, (C)       ; návratová hodnota
jp z, exit      ; ak je to 0, bankovanie sa
                ; nepodporuje a teda skok na koniec

ld hl, 200h     ; adresa, kde uložíme testovaciu hodnotu
```

```
    ld (hl), 5      ; na adresu 200h sa uloží hodnota 5

    ld a, 12        ; zmena aktívnej banky
    out (C), a
    ld a, 1         ; parameter = číslo novej aktívnej banky
    out (C), a

    ld a, (hl)      ; do a <- hodnota z adresy 200h
                    ; a bude obsahovať 0
                    ; (nie uloženú hodnotu 5)

    ld a, 12        ; opäť zmena aktívnej banky
    out (C), a
    ld a, 0         ; parameter = vrátime sa na pôvodnú banku
    out (C), a

    ld a, (hl)      ; do a <- hodnota z adresy 200h
                    ; a bude obsahovať uloženú hodnotu 5
exit:
    halt
```

Ak by sa program nenachádzal v spoločnej pamäti, zmena banky pamäte by spôsobila stratu programu (pretože na tých istých adresách, kde sa nachádzal program v prvej banke, by sa zmenou banky tieto hodnoty prepísali).

Kapitola 13

Abstraktná páska pre abstraktné stroje

Táto kapitola popisuje zásuvný modul s názvom *Abstract tape*, súbor `AbstractTape.jar`, verzia `1.0b1`.

Abstraktné pásky sú používané v abstraktných strojoch, ako sú napr. RAM alebo Turingov stroj. Ide o nekonečnú a sekvenčnú postupnosť buniek, ktorých hodnoty sú symboly. Tieto symboly môžu byť špecifikované v akejsi abecede, ale môj zásuvný modul pásky chápe ako symbol ľubovoľný reťazec znakov, ľubovoľnej dĺžky. „Nekonečnosť" pásky je implementovaná ako dynamicky a podľa potreby sa rozširujúce pole reťazcov.

Páska bola naprogramovaná tak, aby jej význam a účel mohol byť daný samotným strojom, teda mysleným „CPU". To zahŕňa aj zmenu názvu pásky, preto v konfigurácii abstraktného stroja pravdepodobne nenájdete názov *Abstract tape*, ako je tu uvedený. Ďalšie možnosti, ktoré páska abstraktným strojom ponúka, je možnosť nastavenia ohraničenosti z jednej strany (zľava) a umožnenie/zakázanie editovať bunky pásky používateľom. Z týchto dôvodov je páska sama o sebe univerzálna a môže byť použitá pri tvorbe konfigurácie ľubovoľného abstraktného stroja.

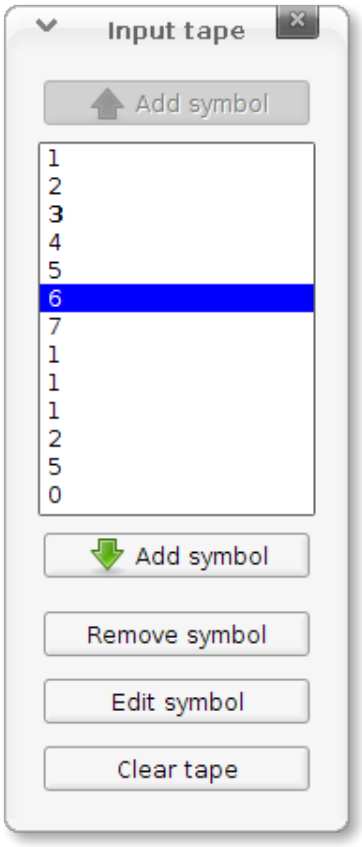
Programovanie pásky v zdrojovom kóde je tiež polemické, záleží od typu stroja. Napríklad v konfigurácii stroja RAM jazyk kompilátora podporuje pseudoinštrukciu `<input>`, ktorou môžeme naplniť vstupnú pásku stroja.

Páska je štandardne neohraničená, prázdna a so zapnutou možnosťou editovania jej buniek používateľom. Keď sa páska pripojí do daného stroja, tento jej priradí význam a môže aj zmeniť jej štandardné nastavenia.

13.1 Práca s páskou

Príklad GUI okna vstupnej pásky stroja RAM je zobrazený na Obr. 13.1. Okno obsahuje niekoľko tlačidiel. Tlačidlami *Add symbol* pridáme symbol (teda ľubovoľný reťazec znakov) do pásky zľava (horné tlačidlo), alebo sprava (dolné tlačidlo). Ak je páska ohraničená zľava, tak symboly môžme pridávať len sprava (nebude povolené kliknutie na horné tlačidlo, ako to na obrázku môžme vidieť).

Tlačidlom *Remove symbol* zmažeme vybraný symbol na páske. Výber symbola realizujeme



Obr. 13.1: *Okno vstupnej pásky stroja RAM*

kliknutím na neho v zozname symbolov. Vybraný symbol je zobrazený túčným písmom (na Obr. 13.1 je vybraný symbol 3). Tlačidlom *Edit symbol* vybraný symbol môžme zmeniť (prepísať), a tlačidlom *Clear tape* zmažeme obsah pásky.

Časť III

Virtuálne architektúry

Kapitola 14

Počítač MITS Altair

Počítač Altair 8800, pomenovaný podľa planéty v jednom z prvých dielov seriálu Star Trek, bol počítač, ktorý si mohol záujemca o elektroniku postaviť za len 397 dolárov. S procesorom Intel 8080 a 256 bajtmi pamäte bez obrazovky či klávesnice to bolo podľa dnešných štandardov úplné nič. Jeho autor, Ed Roberts, pomenoval svoj vynález „personal computer“ (osobný počítač). V dnešnej dobe sa termínom PC označuje prakticky každý počítač, ktorý môžete odniesť v rukách. Obrázok 14.1 ukazuje počítač Altair 8800 spolu s terminálom a disketovou mechanikou.

Obr. 14.1: *Počítač MITS Altair 8800, spolu s terminálom LSI ADM-3A a disketovou mechanikou*

Altair 8800 je jedným z najstarších komerčne dostupných osobných počítačov vôbec. Ed Roberts (zakladateľ a prezident spoločnosti MITS) predával tieto stroje poštou priamo z továrne.

Rôzni nadšenci pochopili silu Altairu a začali pre neho vytvárať programy a hardware. Pre týchto priekopníkov predstavoval Altair slobodu — akési uvoľnenie sa od dávkových úloh pre

mainframové systémy, spravované elitou. Na fenoméne počítača, ktorý môžete mať doma na kuchynskom stole, zarobili závratné bohatstvo dvaja zbehlí vysokoškolskí študenti — v roku 1975, Paul Allen a Bill Gates (vtedy študent na Harvarde), napísali orezanú verziu jazyka BASIC, čo ich priamo posunulo k založeniu spoločnosti Microsoft.

Objavila sa rada rôzneho hardwaru s rôznymi rozdielmi a softwaroví nadšenci radostne vytvárali nové programy pre nové systémy. Táto kapitola uvádza návody, ako spustiť obrazy softvéru určeného pre počítač Altair. Väčšina obrazov dodávaných k emulátoru ***emuStudio*** sú prevzaté z emulátora SIMH [6]. Niektoré boli modifikované pre beh na emulátore SIMH, čím trochu strácajú na originalite svojou štruktúrou, ale rozhodne nie funkcionalitou.

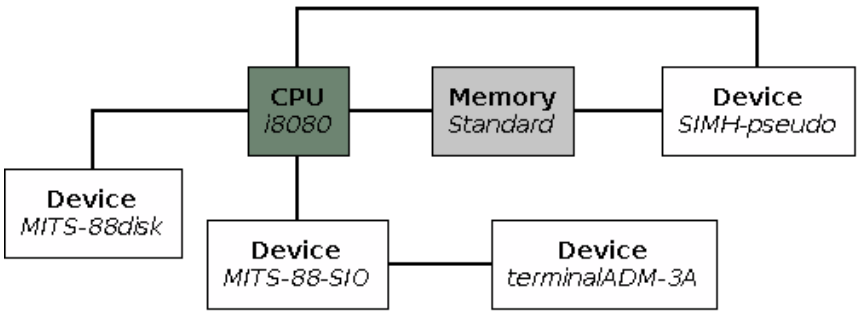
Základná konfigurácia počítača MITS Altair 8800 bola [10]:

Procesor	Intel 8080 alebo 8080a
Rýchlosť	2 MHz
RAM	od 256 bytov, do 64 KB
ROM	voliteľná. Obyčajne pamäte typu EPROM značky Intel 1702 s veľkosťou 256 bytov na každej, na ktorých boli uložené rôzne bootloader-y.
Pevná pamäť	voliteľne: papierové pásky, kazetové pásky, 5.25" alebo 8" diskety.
Rozšírenia	najprv 16 slotov, neskoršie matičné dosky mali 18 slotov.
Zbernica	S-100
Video	žiadne
I/O	voliteľne sériová alebo paralelná karta
Možnosti OS	MITS DOS, CP/M, Altair Disk BASIC

Neskôr firma MITS vyvinula ďalšie verzie tohto počítača: *Altair 8800a*, *Altair 8800b* a nakoniec *Altair 680b*. Všetky tieto počítače mali podobné konfigurácie, takže ich nebudem uvádzať.

14.1 Schéma pre emulátor

Na Obr. 14.2 je možné vidieť abstraktnú schému, ktorá je použitá pre emulovanie tohto počítača. Niektorý softvér (ako bude uvedené ďalej) vyžaduje pre svoj beh procesor Z80 (napr. operačný systém CPM v3). V takom prípade stačí zmeniť procesor abstraktnej schémy na Z80. Väčšina programov bežiacich na 8080 bude bežať aj na procesore Z80.



Obr. 14.2: *Abstraktná schéma počítača MITS Altair 8800 použiteľná v emulátore*

14.2 Zavádzanie softvéru z obrazov diskiet

Keďže je emulátor navrhnutý so zreteľom na realitu, je možné na ňom spustiť aj pôvodné verzie operačných systémov a iných programov určených pre počítač Altair 8800. Autori emulátora *simh* [6] na svojej stránke uverejňujú množstvo obrazov niektorých starších operačných systémov a programov, určených nielen pre Altair.

Nie všetky obrazy pre Altair boli odskúšané. Veľa softvéru potrebuje pre svoju funkčnosť inú konfiguráciu Altairu, ako je implementovaná v mojom emulátore. Ide o iné zariadenia, ktoré zatiaľ nie sú implementované.

Odskúšané a plne funkčné sú obrazy operačných systémov CP/M v2.2 a v3, Altair DOS v1.0 a programovacieho jazyka BASIC.

14.2.1 Operačný systém CP/M v2.2

V období rozkvetu počítača Altair 8800 sa vyvinulo množstvo operačných systémov, programov a programovacích jazykov. Jeden z najznámejších je iste operačný systém CP/M.

CP/M (Control Program for Microcomputers) je operačný systém pôvodne navrhnutý Garym Kildall-om z firmy Digital Research, Inc. Najskôr išlo o jednoúlohový a jednopoužívateľský operačný systém, ktorý nepotreboval viac ako 64 KB pamäte, neskoršie verzie doplnili viacpoužívateľské varianty, a boli portované na 16-bitové procesory.

Kombinácia operačného systému CP/M a počítačov so zbernicou S-100 (8-bitové počítače podobné Altair-u) bola veľkým „priemyselným štandardom“, široko rozšíreným v 70-tych až do polovice 80-tych rokov minulého storočia. Tým, že tento operačný systém odbremeňoval používateľa od potreby veľkého programovania pre nainštalovanie aplikácie na počítač, CP/M rapídne zvýšil dopyt po hardvéri, aj po softvéri.

Obraz s týmto operačným systémom obsahujúcim aj iné doplnkové programy, je v súbore `altcpm.dsk`. Postup pre jeho spustenie je nasledovný (predpokladá sa, že je spustená platforma ***emuStudio***v konfigurácii Altair 8800):

1. Načítanie štandardného bootloadera, ktorého úlohou je načítať operačný systém z disku
   (a) v GUI operačnej pamäte (Obr. 5.1) kliknite na ikonu otvorenia (importu) obrazu, je to druhá ikona v poradí na Obr. 5.2.
   (b) v otvorenom okne *Load an image* vyberte súbor s ROM obrazom bootloadera s názvom `boot.bin` a kliknite na tlačidlo *Load*
   (c) zobrazí sa dialógové okno žiadajúce adresu operačnej pamäte, na ktorú sa má obraz načítať. Zadajte adresu `0xFF00`[1].
2. Namontovanie obrazu operačného systému
   (a) v GUI diskového radiča 88-DISK (Obr. 11.3), v paneli *Configuration* overte nastavenie disku č. 0 (*Drive 0 (A)*) a kliknite na tlačidlo *Browse*

[1] Tvar zápisu čísla *0xčíslo* určuje, že ide o hexadecimálne číslo. Okrem tohto zápisu existuje aj osmičkový zápis (tvar *0číslo*) a klasický dekadický zápis

(b) v otvorenom okne *Open an image* vyberte súbor s obrazom diskety, pre operačný systém CP/M je to `altcpm.dsk` a kliknite na tlačidlo *Open*

(c) kliknite na tlačidlo *Mount*

3. Skok na adresu bootloadera

   (a) v GUI emulátora, okno debuggera (Obr. 4.2) kliknite na ikonu skoku na adresu (na Obr. 4.3 tretia ikona sprava)

   (b) v zobrazenom dialógovom okne *Jump* napíšte adresu `0xFF00`, čo je adresa načítaného bootloadera, a kliknite na tlačidlo *OK*

4. Nastavenie parametrov CPU — nepovinný krok.

   (a) V stavovom okne CPU (Obr. 4.5) nastavte požadovanú frekvenciu, odporúča sa 2000 kHz (pôvodná frekvencia CPU na počítači Altair)

5. Zobrazenie terminálu

   (a) otvorte GUI terminálu (Obr. 9.2 vpravo)

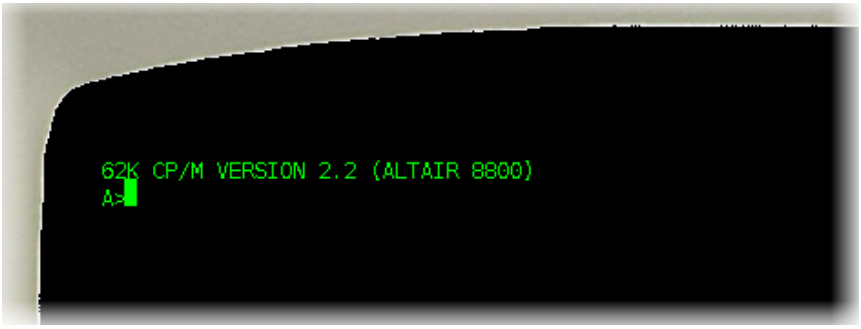
   (b) v okne terminálu kliknite na tlačidlo *Config*

   (c) v zobrazenom okne konfigurácie terminálu (Obr. 9.4) označte checkbox *Always on top*

   (d) zavrite okno konfigurácie

6. Spustenie emulácie — v okne emulátora kliknite na ikonu spustenia emulácie (piata ikona sprava na Obr. 4.3)

Po úspešnom absolvovaní všetkých týchto krokov by sa na termináli mala objaviť hláška o štarte operačného systému CP/M a potom výzva na zadanie príkazu (príkazový riadok), ako ukazuje Obr. 14.3.



Obr. 14.3: *Štart operačného systému CP/M*

Príkaz `dir` funguje, `ls` je lepšie `dir`. Podrobnejší popis tohto operačného systému a jeho príkazov môžte nájsť v [7].

14.2.2 Operačný systém CP/M v3

Postup spustenia operačného systému CP/M verzie 3 sa veľmi nelíši od postupu uvedeného v predchádzajúcej časti. Obraz systému má názov `cpm3.dsk` a je veľký asi 1,1 MB. Súčasná verzia zásuvného modulu disketovej jednotky (popísaný v časti 11) vie zvládnuť skoro ľubovoľne veľké súbory.

Pre svoj beh systém potrebuje mať podporu bankovania operačnej pamäte. Odporúčaný je počet 8 bánk, so spoločnou pamäťou od adresy $C000h$. Pre výpočet veľkosti pamäte a popis techniky bankovania nech čitateľ pozrie časť 5.3.

Pre samotný beh operačného systému postačí aj procesor 8080, no väčšina programov na tomto obraze potrebuje procesor Z80. Preto sa odporúča zmeniť použiť konfiguráciu s procesorom Z80.

Ostatný postup spustenia systému sa nelíši od popísaného v časti 14.2.1. Snáď bolo by dobré ešte upozorniť čitateľa, že montovanie diskov je možné vykonávať aj počas behu emulácie. Pri odmontovávaní treba dávať pozor na to, aby bežiaci program disk nepoužíval. Súčasné diskové zariadenie umožňuje pripojenie až 16-tich diskov súčasne, preto ak má čitateľ k dispozícii aj iné obrazy diskov s programami spustiteľnými na tomto operačnom systéme, môže ich namontovať (pred spustením emulácie, ale aj počas jej behu). Disky v operačnom systéme sa sprístupnia napísaním písmena disku a dvojbodky v konzole, napr.: `B:` namontuje disk „B“, teda č. 1. Príkazom `dir` sa zobrazí jeho obsah.

14.2.3 Operačný systém Altair DOS v1.0

Tento operačný systém bol dlho sľubovaný už od roku 1975. Niektorí ľudia si ho v tejto dobe dopredu objednali, ale svetlo sveta uzrel až roku 1977, kedy to už skoro prestalo byť dôležité. Aj preto tento systém nie je až tak známy a nebol ani veľmi používaný. Okrem toho po krátkej „obchôdzke“ po systéme zistíte, že je o dosť slabší ako operačný systém CP/M.

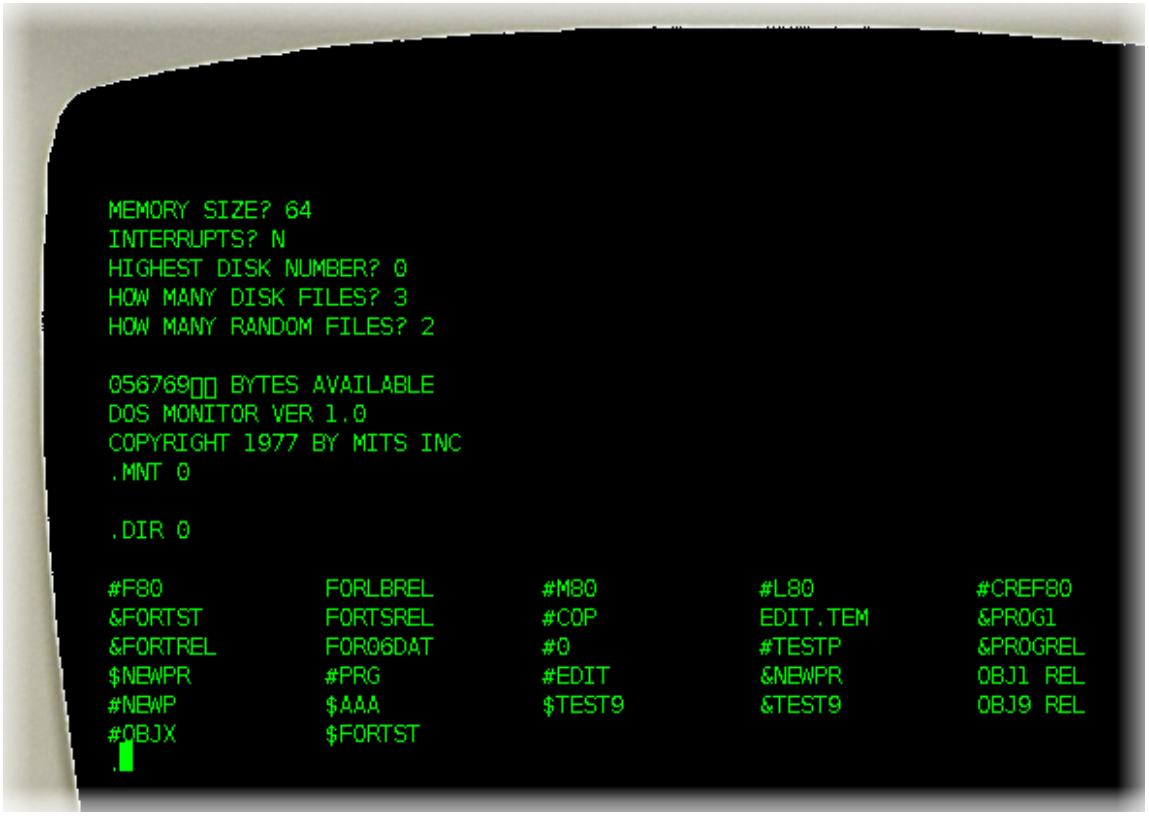
Tento systém sa spúšťa rovnakým postupom, ako bol popísaný v časti 14.2.1. Rozdiel je v tom, že pri montovaní obrazu diskety použijete súbor `altdos.dsk`. Bootloader je ten istý.
Ukážka spustenia systému je na Obr. 14.4.

Po spustení sa operačný systém opýta pár nasledujúcich otázok. Odpovedzte mu na tieto otázky podľa Obr. 14.4. Viac o tomto systéme sa môžte dočítať v manuáli [8].

14.2.4 Altair Basic v4.1

BASIC (*Beginner's All Purpose Symbolic Instruction Code*) je programovací jazyk vyvinutý na Dartmouth College v roku 1964 pod vedením J. Kemeny-ho a T. Kurtz-a. Najprv bol implementovaný pre mainfraimový počítač G.E.225 (vyrobený firmou General Electric). Pôvodná myšlienka bola, aby sa dal jazyk veľmi ľahko naučiť a tiež ľahko kompilovať. Neskôr sa vývojári tohto jazyka rozhodli, aby sa stal akýmsi medzikrokom pre študentov, ktorí sa chceli učiť výkonnejšie jazyky, ako napr. FORTRAN či ALGOL.

Bill Gates a Paul Allen však mali iný úmysel. V 70-tych rokoch minulého storočia, keď bol predstavený osobný počítač MITS Altair, Allen presvedčil Gatesa, aby mu naňho pomohol



Obr. 14.4: *Štart operačného systému Altair DOS*

portovať jazyk BASIC. Keďže firma MITS odpovedala so záujmom, začala sa tak budúcnosť jazyka BASIC aj na počítačoch PC.

Gates chodil na školu v Harvarde, Allen bol v tom čase zamestnancom firmy Honeywell. Allen a Gates predali firme MITS licenciu na ich BASIC. Táto verzia zaberala celkovo 4 KB pamäte vrátane kódu aj dát potrebných pre uchovanie interpretovaných zdrojových kódov.

Na mojom emulátore bol odskúšaný Altair Basic v4.1 (Disk Extended Version), ide o súbor obrazu diskety s názvom `mbasic.dsk`. Postup pre jeho zavedenie je rovnaký, ako je uvedený v časti 14.2.1.

Na Obr. 14.5 je zobrazená ukážka spustenia spomínanej verzie jazyka Basic. Po jeho spustení sa systém opýta na pár otázok, podobne ako systém Altair DOS. V tomto prípade sú však niektoré otázky a odpovede iné, postupujte podľa Obr. 14.5.

Manuál k tejto verzii jazyka sa mi nepodarilo zohnať, iba článok v časopise *Computer Notes* [9].

Obr. 14.5: *Štart jazyka Altair BASIC*

Kapitola 15

Abstraktný stroj RAM

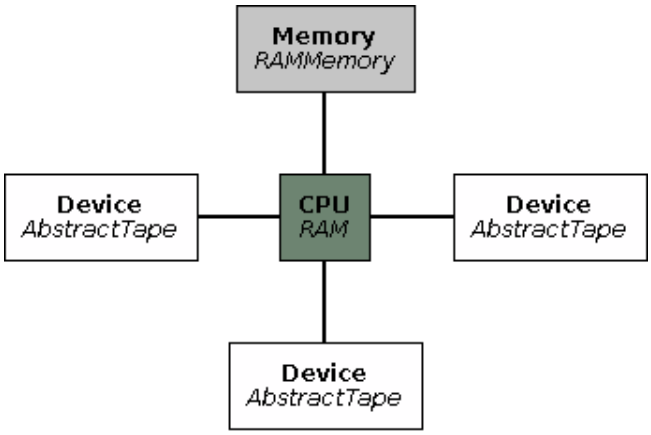
Abstraktné stroje sa používajú hlavne pre analýzu zložitosti programov. **RAM** (*Random Access Machine*) je abstraktný stroj, ktorý používa štyri pásky - vstupnú, výstupnú, pamäť registrov (dátová pamäť) a pamäť programu.

Skutočnosť, že má oddelenú pamäť dát od pamäti programu naznačuje, že stroj nie je reprezentantom Von-Neumannovej architektúry, ale má bližšie k tzv. Hardvardskej architektúre. Aj keď je emulátor stavaný na prácu s počítačmi reprezentovanými Von-Neumannovou architektúrou, úspešne sa podarilo vytvoriť emulátor aj tohto stroja.

Ekvivalent RAM stroja k univerzálnemu Turingovmu stroju (ktorý má dáta a program na jednej páske spolu), sa nazýva **RASP** (*Random Access Stored Program machine*). Tento už je príkladom Von-Neumannovej architektúry a má veľmi blízko k reálnym počítačom.

15.1 Schéma pre emulátor

Na Obr. 15.1 je možné vidieť abstraktnú schému, ktorá je použitá pre emulovanie tohto stroja.



Obr. 15.1: *Abstraktná schéma stroja RAM*

Literatúra

[1] MITS, Inc.: Altair 8080 Operators Manual, 1975
http://www.classiccmp.org/dunfield/altair/d/88opman.pdf

[2] Intel Corp.: 8080 Assembly Language Programming Manual, 1975
http://www.classiccmp.org/dunfield/r/8080asm.pdf

[3] Lear Siegler, Inc. (LSI): ADM-3A Operators Manual, 1979
http://www.classiccmp.org/dunfield/altair/d/adm3a.pdf

[4] MITS, Inc.: Serial I/O Board documentation, 1975
http://www.classiccmp.org/dunfield/s100c/mits/88sio_1.pdf

[5] MITS, Inc.: Disk Operators Manual, 1975
http://www.altair32.com/pdf/88dsk\%20manual\%20v2.pdf

[6] The Computer History Simulation Project
http://simh.trailing-edge.com/

[7] Digital Research: CP/M Operating System Manual
http://public.planetmirror.com/pub/cpm/manuals/cpm22-m.pdf

[8] MITS, Inc.: Altair Disk Operating System Documentation, 1977
http://www.classiccmp.org/dunfield/altair/d/altdos.pdf

[9] Altair Basic: Up and Running, Computer Notes, Vol 1, 1975
http://www.startupgallery.org/gallery/notesViewer.php?ii=75_4

[10] The Vintage Computer My Collection of Vintage Machines
http://www.vintage-computer.com/altair8800.shtml

[11] HUDÁK, Š.: Strojovo orientované jazyky *FEI, Košice*, 2003, 218 strán, ISBN 80-969071-3-1