»Mladi za napredek Maribora 2024«
41. srečanje

EMULATOR MIKROPROCESORJEV MOTOROLA M68XX

Raziskovalno področje: Računalništvo in informatika

Inovacijski predlog

Avtor: Vladimir Januš
Mentor: Dušan Fugina

Šola: Srednja elektro-računalniška šola Maribor

Število točk: 163 / 170

Maribor 2024

KAZALO VSEBINE

Povzetek

To je inovacijski predlog, ki prikazuje razvoj in delovanje emulatorja za mikroprocesorje Motorola M68XX. Ta emulator omogoča uporabnikom programiranje in uporabljanje mikroprocesorjev M6800 in M6803. Zaradi preproste arhitekture emuliranega procesorja vedoželjnim omogoča edinstveno priložnost za učenje osnovnega zbirnega jezika. Po primerjavi s podobnimi izdelki sem zaključil, da moj emulator predstavlja veliko bolj tehnološko izpopolnjeno različico emulatorja M6800. Inovacija je v napredku, saj ima manj napak, je hitrejši, robustnejši in vsebuje veliko več funkcij kot njegovi konkurenti.

1 UVOD

Veliko programerjev ima eno temeljno težavo. Ta je, da ne vedo, kaj se pri višjih programskih jezikih dogaja pod pokrovom.

Najenostavnejša rešitev za to težavo je učenje nižjega programskega jezika in arhitekture računalnika. Nova težava se pa pojavi v dejstvu, da so današnji računalniki mnogo preveč kompleksni za začetnika. Učenje in razumevanje zbirnih jezikov je zahtevno zaradi nizke stopnje abstrakcije, potrebe po poznavanju strojne opreme, omejitve zbirnih ukazov, specifičnosti posameznega računalniškega sistema (arhitektura, CPE, RAM …) in ročnega upravljanja pomnilnika, kar vse prispeva k strmi učni krivulji in omejeni podpori orodij.

Dosti boljša alternativa je, da se programer uči te koncepte na enostavnejšem procesorju. Motorolin mikroprocesor M6800 vključuje vse osnovne komponente modernih procesorjev, kot so podpora sklada, indeksnih registrov in podobnega. Njegova preprosta struktura omogoča lažje razumevanje delovanja procesorja v primerjavi z novejšimi procesorji, ki imajo več registrov in ostalih kompleksnejših komponent. Znanje, ki ga uporabnik pridobi z izkušnjami na takem procesorju, pa se z lahkoto prenese na modernejše procesorje in višje programske jezike.

Pri programiranju mikroprocesorjev je pomembno tudi znanje elektrotehnike, saj je tak čip zelo občutljiva zadeva. Priprave za uporabo mikroprocesorja so zamudne in nevarne. Da bi se temu izognili, lahko uporabimo emulator. Emulator je računalniški program, ki omogoča, da programska oprema teče tudi v drugih okoljih in operacijskih sistemih, ne samo v tistih, za katerega je bila ustvarjena. Emulator na računalnik zgolj namestimo in tako imamo celotno okolje že pripravljeno.

Zaradi tega je namen inovacijskega predloga razvoj emulatorja mikroprocesorja M6800. Emulator tega enostavnega in sposobnega mikroprocesorja bo omogočil učenje programiranja zbirnega jezika, saj poznavanje katerega koli zbirnega jezika omogoči globlje razumevanje delovanja računalnikov, prevajalcev in delovanja višjih programskih jezikov. V pomoč bo tudi tistim, ki želijo delati s M6800, a tega procesorja v živo nimajo. Emulator se lahko uporabi tudi kot razhroščevalnik za prave M6800-procesorje.

Nekaj emulatorjev za M6800 že obstaja. A so ti tako zastareli in neučinkoviti, da je razvoj novega res potreben. Nov emulator bo inovacija tudi zaradi vseh novih funkcij in mnogo višjega nivoja kvalitete.

Najprej sem si zastavil kriterije dobrega emulatorja, nato sem naredil načrt, si izbral programsko ogrodje in se lotil razvijanja.

2 METODE DELA

2.1 Načrt inovacijskega predloga

Vsaka inovacija se začne z idejo. Če nekdo drug te ideje še ni uresničil, se postavlja vprašanje, kako jo uresničiti na najboljši in najučinkovitejši način. Proces moje inovacije se lahko razdeli na sledeče korake: oblikovanje ideje in ciljev (poglavje 3), postavitev kriterijev kvalitetnega izdelka (poglavje 3), pregled stanja tehnike (poglavje 3), oblikovanje načrta (poglavje 4), izvedba ter razvoj inovacije (poglavje 5) in potrditev napredka(poglavje 6).

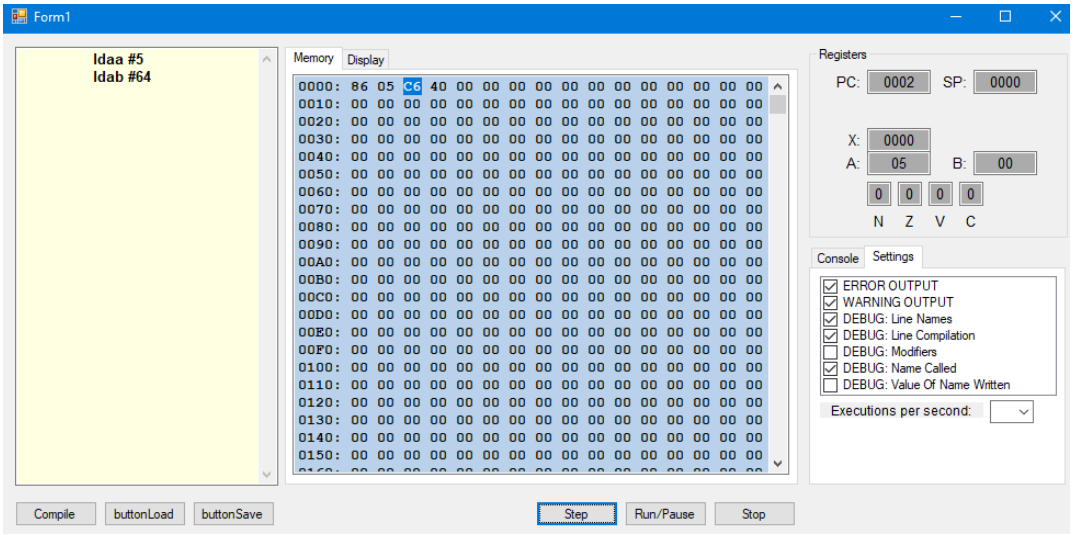
2.2 Iskanje primernega programskega ogrodja

Programsko ogrodje je program, ki ima funkcije, namenjene razvijanju programov, kot so: urejevalnik izvorne kode, prevajalnik in razhroščevalnik. Znana programska ogrodja za razvoj namiznih aplikacij so: WinForms, Qt, JavaFX, WPF, Xamarin in Swift.

Delal sem na projektih, ki so prevladujoče uporabljali programski jezik c#. Zato sem iskal ogrodja, ki uporabljajo C# kot primarni programski jezik.

Program sem začel razvijati v ogrodju WinForms z jezikom c#. Naredil sem nekaj okenc, ki prikazujejo vrednosti registrov, polje za vnos izvorne kode in polje za prikaz spominskih celic. Dodal sem tudi nekaj nastavitev in konzolo, v kateri se bodo izpisala sporočila, namenjena uporabniku.

Napisal sem nekaj funkcij, ki uporabniku omogočajo urejanje kode in funkcijo, ki pretvori izvorno kodo v strojni jezik. Dodal sem tudi funkcijo, ki izvršuje ukaz. Ta poskusna različica emulatorja je podpirala le nekaj osnovnih ukazov, kot je "LDA", in te samo pri vsebovanem naslavljanju.



Slika 1: Poskusna različica emulatorja v ogrodju WinForms (vir: lasten vir)

To različico in njeno ogrodje sem kmalu zapustil, saj se je izkazalo, da je WinForms zelo počasen pri spreminjanju vrednosti v elementih, ki prikazujejo registre in vrednosti pomnilnika.

Projekt sem kmalu obudil v novem ogrodju, za katerega sem predpostavljal, da bo zapolnil pomanjkljivosti ogrodja WinForms. Imenuje se WPF ali "Windows Presentation Foundation". Žal se je tudi to ogrodje izkazalo za moje potrebe prezapleteno in prepočasno. Datoteke in posnetke tega projekta sem izgubil.

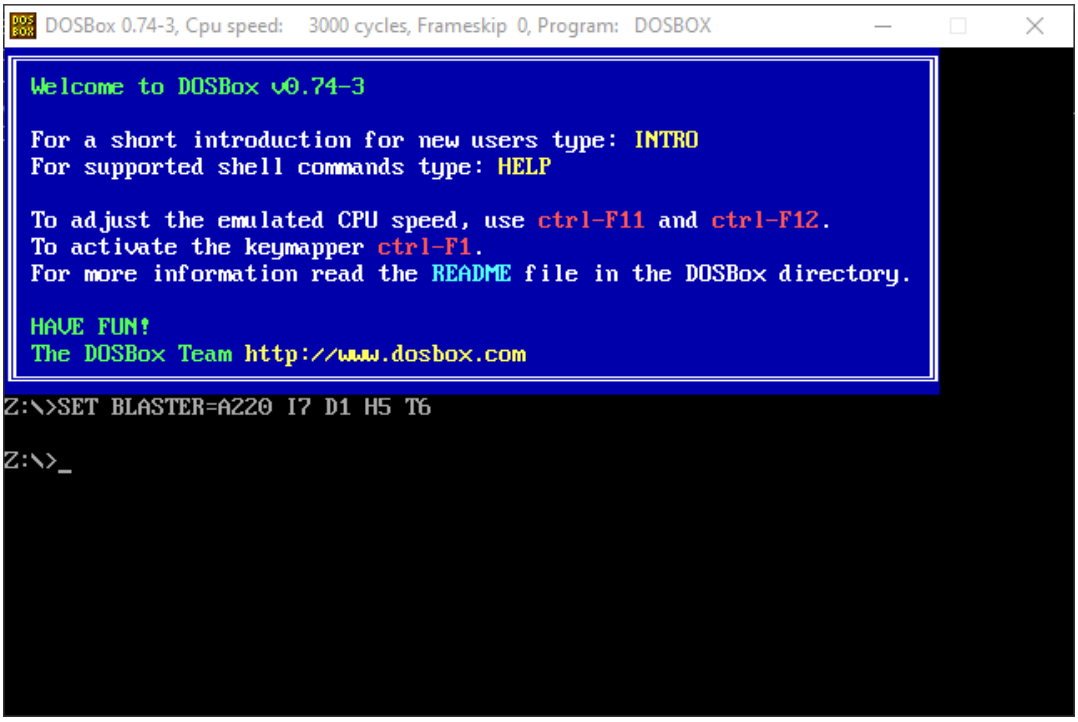
Po nekaj dneh brskanja po spletu sem naletel na programsko ogrodje Qt. Na prvi pogled me je zmotilo dejstvo, da imam opravka s programskim jezikom C++. Odločen priti do cilja sem kljub temu preizkusil zmogljivosti enostavne aplikacije, ki hitro spreminja vsebino okenca. Kljub predsodkom sem bil nepričakovano in prijetno presenečen nad hitrostjo delovanja novega ogrodja. Urejevalnik Qt Creator mi je bil všeč. Hitro sem usvojil osnove in se sproti naučil programirati v C++.

3 MIKROPROCESORJI IN EMULATORJI

3.1 Emulator

Emulator je računalniška oprema, ki omogoča, da en računalniški sistem (ta služi kot gostitelj) oponaša delovanje drugega (gost). Emulirajo se lahko računalniške igre, igralne konzole, operacijski sistemi, mikroprocesorji in podobno.

Njihov namen je digitalizacija naprav in uporaba zastarelih ter nepodpiranih programov ali naprav. Sledi primer emulatorja, kjer je gost operacijski sistem MS-DOS. Gostitelj pa je Microsoft Windows.



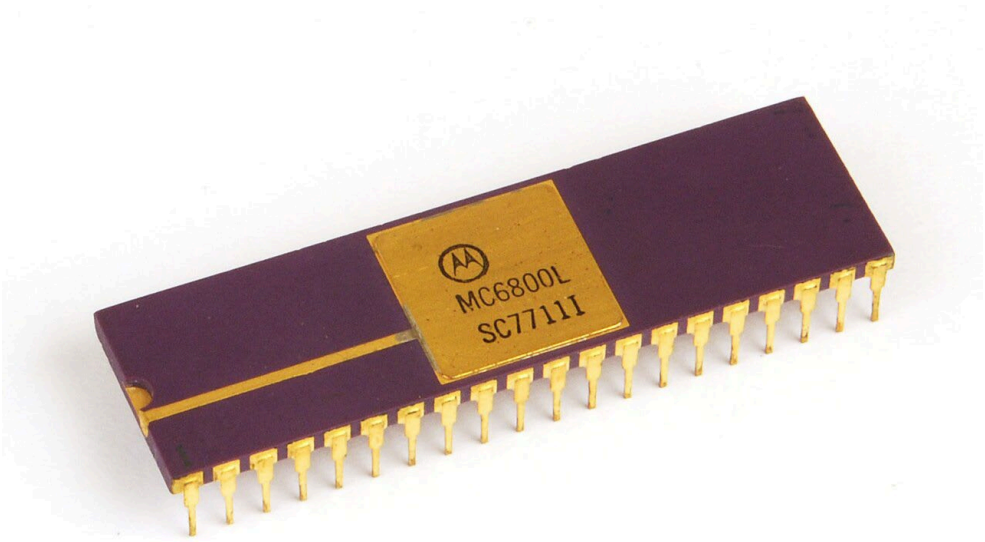
Slika 2: Primer emuliranega operacijskega sistema MS-DOS (vir: lasten vir)

Ustvaril bom emulator, ki bo delal na operacijskem sistemu Windows. Emuliran pa bo Motorolin mikroprocesor M6800.

3.2 Mikroprocesor

Mikroprocesor je računalniški procesor, pri katerem sta logika in krmiljenje obdelave podatkov vključena v eno samo integrirano vezje.

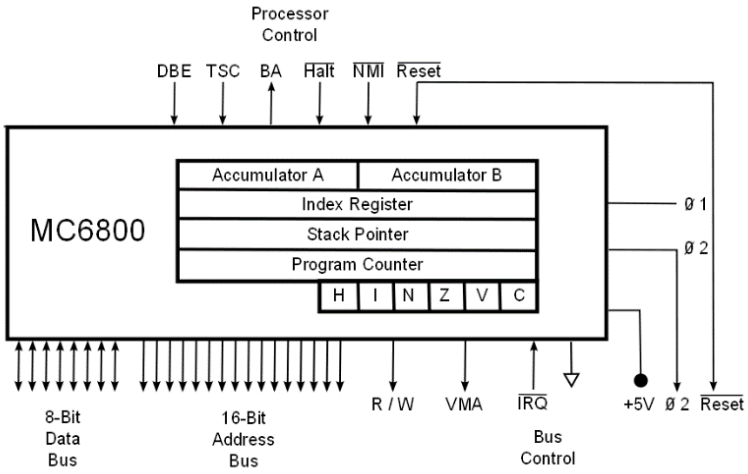
3.2.1 Kaj je Motorola M6800?



Slika 3: Mikroprocesor Motorola M6800 (vir: https://en.wikipedia.org/wiki/Motorola_6800)

M6800 je 8-bitni mikroprocesor iz družine "M6800 Microcomputer System" (z vzdevkom M68XX). Motorola ga je zasnovala in prvič izdelala leta 1974.

Procesor ima dva 8-bitna akumulatorja, imenovana akumulator A in akumulator B. Procesor ima tudi tri 16-bitne registre, ti so: indeksni register, programski števec in kazalec sklada. Procesor ima tudi en 8-bitni statusni register, ta register vsebuje zastavice, ki podajajo nekaj informacij o izvedbi prejšnjega ukaza. Zadnja dva bita tega registra sta neuporabljena in vedno imata vrednost 1. Zastavice pa so razvrščene od najbolj utežene (peti bit) do najmanj utežene (ničti bit): zastavica pol-prenosa (H), zastavica prekinitvene maske (I), zastavica negativnega predznačenega števila (N), zastavica nič (Z), zastavica prekoračitve (V) in zastavica prenosa (C).



Slika 4: Prikaz registrov in nožic M6800 (vir: https://en.wikipedia.org/wiki/Motorola_6800)

Ta procesor podpira šest načinov naslavljanja. Ti so: vsebovano, takojšnje, neposredno, razširjeno, indeksno in relativno. Način naslavljanja je način, ki opisuje kako računalnik pride do operanda, nad katerim se izvede ukaz.

Ta mikroprocesor se je uporabljal v igralnih konzolah, na osebnih računalnikih in drugih napravah, in sicer: osebnih računalnikih Sphere 1 in Tektronix 4051, igralni konzoli APF MP1000, namiznem kalkulatorju HP 9815A in mnogih drugih napravah.

Slika 5: Reklama za osebni računalnik Tektronix 4051

(vir: https://en.wikipedia.org/wiki/Motorola_6800)

3.2.2 Zakaj emulator mikroprocesorja M6800?

V primerjavi z ostalimi je ta procesor po arhitekturi med preprostejšimi, saj ima manj registrov in drugih kompleksnih enot, kot je predpomnilnik. Zato ima velik potencial za učenje programiranja katerega koli nižjega nivojskega programskega jezika. Primeren je tudi za začetnike in tiste, ki jim svet nižjih programskih jezikov ni poznan.

Motivacijo za razvoj novega emulatorja mi je dala tudi izkušnja s tem procesorjem v šoli. Tam smo uporabljali zastarel in neučinkovit emulator. To me je navdihnilo, da razvijem boljšo rešitev in omogočim boljšo izkušnjo pri delu s tem procesorjem.

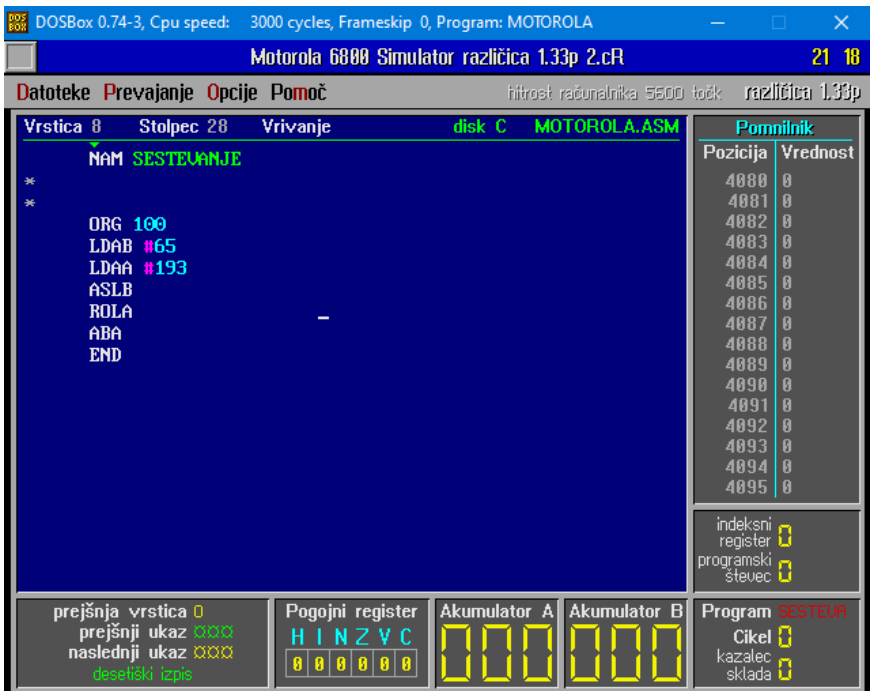
3.3 Kriteriji dobrega emulatorja

Da bi primerjal obstoječe emulatorje in kasneje tudi potrdil napredek z mojim emulatorjem, sem si zastavil kriterije dobrega emulatorja. Ti kriteriji so lastnosti in funkcije, ki jih tak emulator mora dosegati.

1. Emulator mora vsebovati brezhiben zbirnik (ang. assembler). Zbirnik mora poznati vsak ukaz, ki ga podpira mikroprocesor M6800. Prepoznati mora vse oblike naslavljanja. Če je v kakem primeru možnih več oblik, mora uporabiti najučinkovitejšo. To se lahko pojavi v primeru, ko ukaz podpira neposredno in razširjeno naslavljanje ter njegov operand ne presega 255.
2. Uporabniški vmesnik mora biti učinkovit. Prikaz spomina in drugih notranjih elementov procesorja mora omogočati hiter in razumljiv dostop do podatkov.
3. Ročno urejanje spomina in disassembler. Da bi se uporabnik lažje naučil, kako deluje zbirnik, mora imeti na voljo orodja za ročno vpisovanje strojne kode. Ker je ročno pisanje strojne kode zahtevno, bom uporabniku podal tudi orodje za razgradnjo ukazov (pretvorba strojne kode v izvorno kodo?. S tem lahko preveri in razhrošči svojo strojno kodo.
4. Originalnemu procesorju popolnoma enako izvajanje. Vsak ukaz se mora obnašati enako kot pri avtentičnem procesorju. Delovati morajo tudi prekinitve in ciklično izvajanje. To vključuje tudi hitrost izvajanja, ki je v originalnem mikroprocesorju 1 Mhz.
5. Orodja za razhroščevanje. Emulator mora vsebovati vsa potrebna orodja za razhroščevanje.
6. Uporaba vhodnih in izhodnih naprav. Možnost interakcije z emuliranim mikroprocesorjem na bolj dinamičen način omogoča zanimivejšo izkušnjo, saj bi bila namesto statičnega algoritma, ki izvaja eno funkcijo, možna komunikacija s perifernimi napravami. To omogoča ustvarjanje bolj dinamičnih in interaktivnih programov, ki odražajo resnično uporabo mikroprocesorja.
7. Nastavitve. Da bi se uporabnik v mojem emulatorju počutil bolj domače, bodo na voljo raznovrstne nastavitve, da si vsak uporabnik prilagodi delovanje emulatorja po svojih željah.
8. Vgrajene informacije in navodila za uporabo emulatorja in samega mikroprocesorja.
9. Učinkovito in predvidljivo delovanje. Emulator mora delovati zanesljivo in nedvoumno. Odzivati se mora tudi na uporabnikove želje, kot uporabnik to pričakuje.

3.4 Motorola 6800 Simulator različica 1.33p 2.cR

Njegovo ime je "Motorola 6800 Simulator različica 1.33p 2.cR", ustvaril pa ga je Darko Kropf leta 1995.



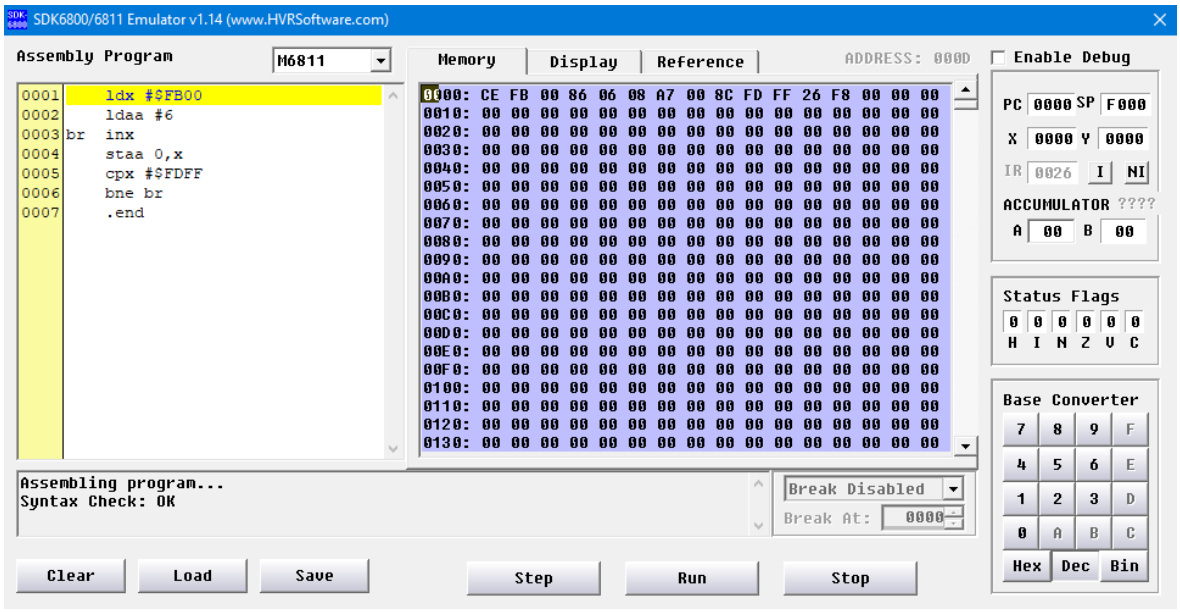
Slika 6: Uporabniški vmesnik emulatorja "Motorola 6800 Simulator različica 1.33p 2.cR" (vir: lasten vir)

Ta emulator ima mnoge pomanjkljivosti. Razvit je bil za operativni sistem MS-DOS. Zato je za njegovo delovanje potreben emulator MS-DOS-a, kot je DosBox.

1. Zbirnik ne podpira nekaj ukazov, kot so SWI in WAI. Relativno naslavljanje je možno le s simboli in ne s številom opisanim relativnim naslovom. Če ukaz kliče simbol, bo emulator vedno uporabljal razširjeno naslavljanje in ne neposredno. To je spominsko neučinkovito, saj porabi več pomnilnika kot je potrebno.
2. Uporabniški vmesnik je neodziven. Prikazi vrednosti spomina in registrov med posodabljanjem utripajo in so neberljivi. Premikati se po spominu je počasno.
3. Emulator podpira ročno urejanje spominov, ampak je zaganjanje programa z ročno napisano strojno kodo nerazumljivo in težavno. Na voljo ni orodij za razgradnjo (ang. disassembly) ukazov.
4. Procesor ne podpira nekaterih ukazov (npr. SWI, WAI ...). Ne podpira cikličnega izvajanja ali prekinitev. Največja teoretična hitrost izvajanja je 1000 ukazov na sekundo, a te hitrosti sam nisem nikoli dosegel.
5. Emulator ne podpira nobenih orodij za razhroščevanje, kar močno oteži pisanje programov.

6. Procesor ne podpira nobenih vhodno-izhodnih enot.
7. Emulator podpira nekaj nastavitev, in sicer: šestnajstiški prikaz, premikajoči pomnilnik, izpis registrov med hitrim izvajanjem, čiščenje pomnilnika, barvno osvetljevanje in hitrost izvajanja.
8. Glede informacij je na voljo primitiven opis vsakega ukaza.
9. Emulator je počasen in nekatere njegove funkcije in orodja delujejo nepredvidljivo.

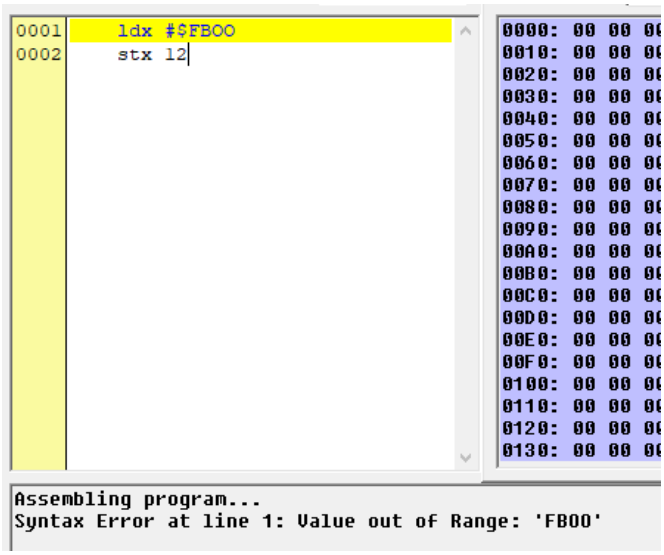
3.5 SDK6800/6811 Emulator v1.14 (www.HVRSoftware.com)



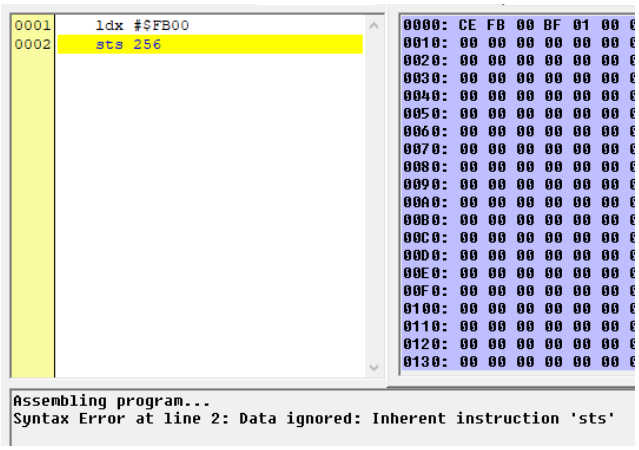
Slika 7: Uporabniški vmesnik emulatorja "SDK6800/6811 Emulator v1.14" (vir: lasten vir)

Ta emulator predstavlja velik napredek v primerjavi s prej omenjenim, saj tudi naravno deluje na OS Windows 10. Tudi ta ne dosega večine kriterijev.

1. Zbirnik ima mnoge napake. Nekateri ukazi so napačno implementirani in emulator zamenjuje načine naslavljanja. Npr. ukaz STS lahko uporablja neposredno, indeksirano in razširjeno naslavljanje. Emulator pa napačno prepozna ukaz in dovoli samo vsebovano naslavljanje. Kar pri tem ukazu niti ni mogoče in je nesmiselno. Tudi takojšnje naslavljanje pri ukazu LDX ne deluje. Emulator misli, da je število nad 256 za ta ukaz preveliko. To pa ne drži, saj ima takojšnji ukaz LDX operand velik 2 bajta.



Slika 8: SDK6800/6811 Emulator: Primer napake v ukazu LDX (vir: lasten vir)



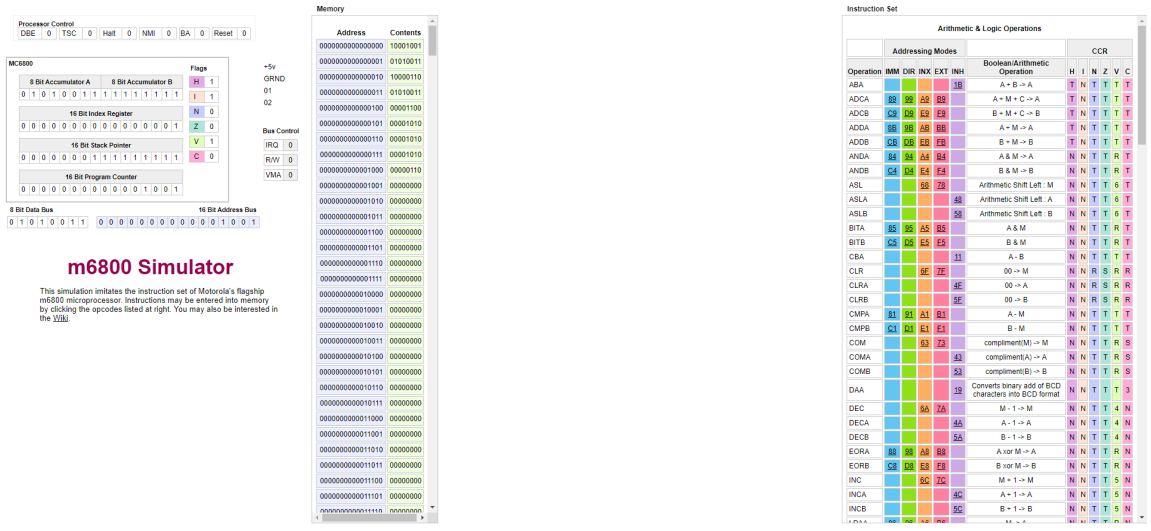
Slika 9: SDK6800/6811 Emulator: Primer napake v ukazu STS (vir: lasten vir)

2. Tudi emulatorjev uporabniški vmesnik je slabo razvit. Na polju za prikaz spomina ni omogočeno skrolanje. Po samodejnem izvajanju se vsebine registrov ponastavijo, zato se rezultati programov izgubijo. Pri samodejnem izvajanju prikaz spomina utripa in je zato nepregleden.
3. Emulator ne podpira ročnega pisanja v spomin ali razstavljanja.
4. Emulator podpira vse ukaze M6800, ampak ne podpira cikličnega izvajanja. Ne podpira niti prekinitve. Največja hitrost izvajanja je okoli 1500 ukazov na sekundo, a to je le v primeru, da emulator ne posodablja prikaza registrov in spomina. Hitrosti se ne da poljubno nastaviti in po testiranju sem ugotovil, da je hitrost naključna in neenakomerna.
5. Nekaj razhroščevalnih orodij je na razpolago, ampak ne delujejo pravilno. Emulator ponuja pretvornik številskih sistemov, ki ga je nadležno uporabljati. Število, ki ga uporabnik želi pretvoriti, se vpiše s tipkovnico na zaslonu, kar je dosti počasnejše kot s tipkanjem na fizični tipkovnici. Potek pretvorbe je tudi precej nejasen in enostavno

se je zmotiti. Če je bila pred pretvorbo spremenjena koda, se bo po poskusu pretvorbe števil koda ponastavila na prejšnje stanje. To je grozna napaka, saj lahko vodi v veliko izgubo napredka v pisanju kode. Na voljo je tudi sistem ustavitve samodejnega izvajanja. Ta sistem ima v primerjavi z mojim emulatorjem manj možnosti.

6. Omogočen je vhodno-izhodni element emulatorja v obliki zaslona, ki sprejema tipke na tipkovnici in lokacijo miške. Zaslon se občasno ponastavi in ne dela do naslednjega izvajanja.
7. Emulator nima nobenih nastavitev, razen izbiro mikroprocesorja. To predstavlja uporabniku težavo, saj si ne more prilagoditi delovanja emulatorja glede na svoje navade in potrebe.
8. Vključen je zavihek, ki ima osnovni, nejasen in včasih tudi dvoumen opis M6800 ukazov.
9. Emulator je zelo nestabilen in deluje neenakomerno. Velikokrat je njegovo delovanje nepredvidljivo, zaradi neznanih vzrokov se samodejno izvajanje ustavi in preneha delovati.

3.6 Spletni M6800-simulator



Slika 10: Uporabniški vmesnik spletnega simulatorja M6800

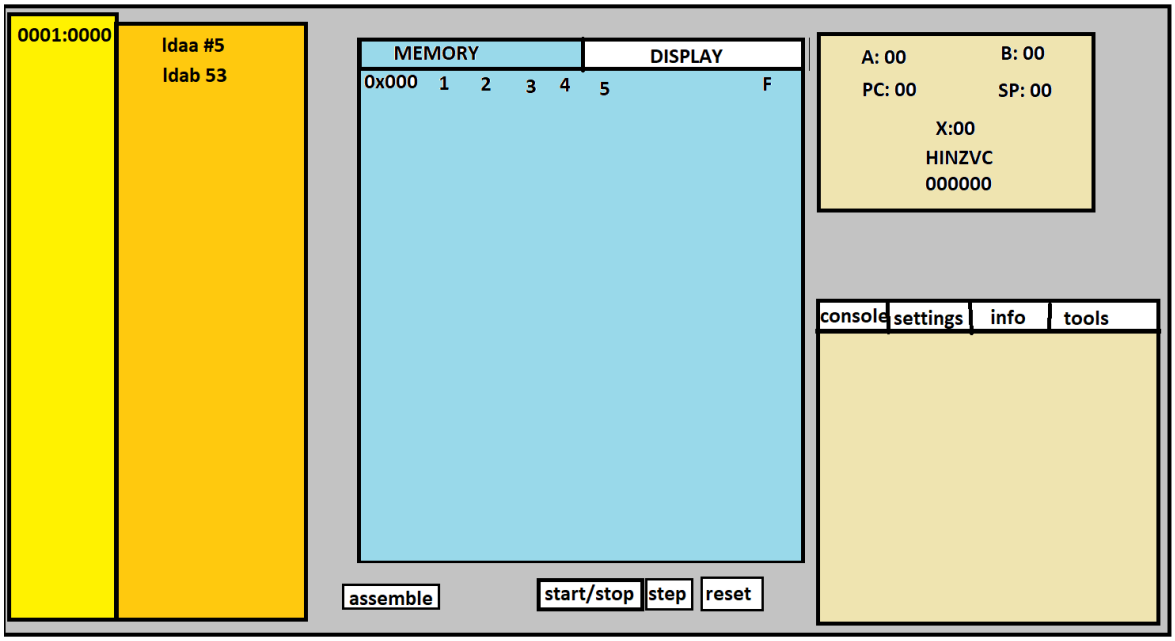
(vir: mypluribus.github.io/m6800/)

Ta simulator je dosegljiv na razvijalčevi spletni strani. Nima zbirnika, saj deluje tako, da uporabnik s klikom na operacijsko kodo vstavi ukaz v spomin in ga takoj izvrši. Nima niti samodejnega izvajanja ali katerih koli drugih orodij, zato ga je težko umestiti med kriterije. Ima samo spis vseh ukazov, prikaz registrov in prikaz spomina, ki je omejen na 255 naslovov.

4 NAČRT IN PRIPRAVE

4.1 Načrt

Projekt sem začel z grobo skico izgleda emulatorja. Odločil sem se, da bo uporabniški vmesnik podoben tistemu iz emulatorja "SDK6800/6811 Emulator" (vir: http://www.hvrsoftware.com/6800emu.htm).



Slika 11: Skica emulatorja v programu Microsoft Paint (vir: lasten vir)

Smiselno je, da so vedno vidna najpomembnejša polja:

- polje za prikaz spomina,
- polja za prikaz registrov,
- polje za vpis kode in
- gumbi, ki nadzirajo osnovne funkcije emulatorja.

Ob kriterijih, ki sem si jih zastavil, bom v emulator vključil sledeče funkcionalnosti:

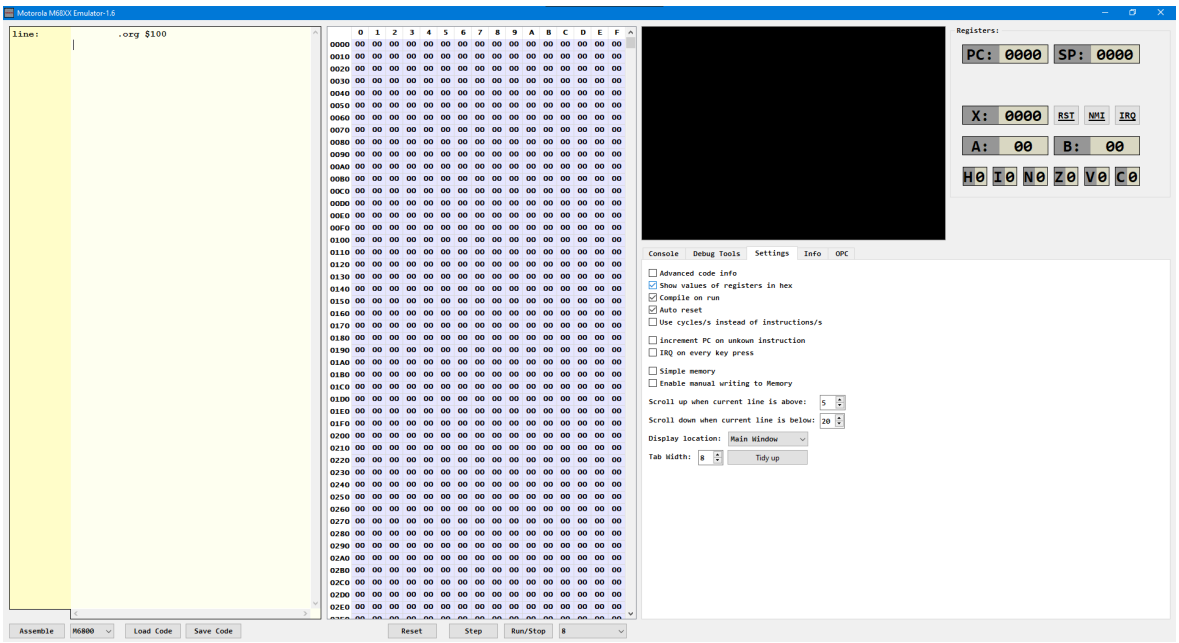
- Funkcija, ki prikaže informacije o ukazu. Ta bo na voljo v meniju, ko uporabnik klikne z desnim miškinim gumbom na ukaz v polju za vpis kode. Na voljo bo tudi v tabeli nabora ukazov.
- Orodja za označevanje ukazov, pomnilniških lokacij. Ta orodja bodo uporabljena, ko zbirnik označuje sintaktično napako, ko uporabnik želi označiti določen naslov za boljšo preglednost in ob označbi ukaza, na katerega trenutno kaže vrednost programskega števca.

- Konzola za prikaz napak, opozoril in sporočil.
- Dva načina prikazovanja spomina. Eden bo prikazoval manj pomnilniških celic. Ta bo enostavnejši za branje in uporaben za učitelje, saj bo v takem primeru emulator najverjetneje projiciran in bo veliko število majhnih celic neberljivo.
- Shranjevanje izvorne kode in pomnilnika v zunanjo datoteko.
- Nalaganje programov in pomnilnika iz zunanje datoteke.
- Ogrodje za implementacijo ostalih procesorjev iz družine M68XX. Emulator bo tako razvit, da bo dodajanje novega mikroprocesorja zelo enostavno. Za preizkus tega ogrodja bom v emulator dodal tudi M6803.
- Funkcija, ki prikaže strojno kodo, v katero se je pretvoril posamezni ukaz.
- Nastavitev, ki uporabniku omogoča, da izbere, kje se bo prikazal zaslon. Izbiral bo med glavnim oknom in zunanjim oknom.
- Nastavitev širine znaka "tab". Ob njej bo tudi gumb, ki poravna vse zamike ukazov na izbrano vrednost.

Emulator bom razvijal po načelu "UCD" (User-centered design). To je tehnika razvijanja aplikacij, pri kateri je najpomembnejša uporabnikova izkušnja s programom. Vsaka dodana funkcionalnost bo pazljivo razvita z namenom, da se bo obnašala tako, kot uporabnik pričakuje. Na voljo bodo tudi vse funkcije, ki bi jih uporabnik potreboval pri delu z emulatorjem mikroprocesorja. En primer je pretvornik številskih sistemov.

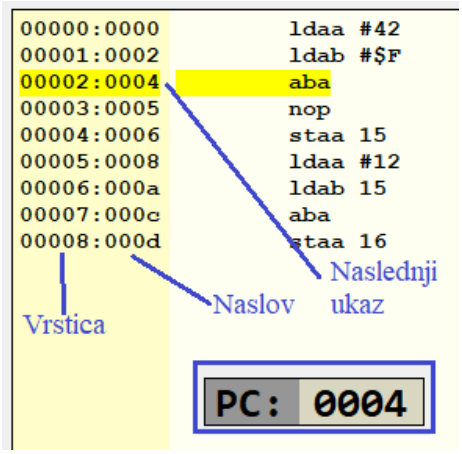
5 RAZVOJ

5.1 Opis uporabniškega vmesnika



Slika 12: Uporabniški vmesnik (vir: lasten vir)

5.1.1 Števec vrstic

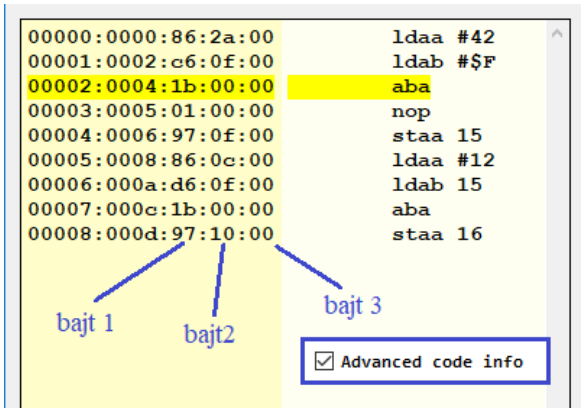


Slika 13: Element uporabniškega vmesnika: Števec vrstic (vir: lasten vir)

Na levi strani glavnega okna je polje, namenjeno štetju vrstic izvorne kode. Privzeto prikazuje dva podatka, ločena z dvopičjem. Število na levi je število vrstice, v kateri je zapisan ukaz v polju za vpis zbirne kode. Če je program trenutno preveden, bo na desni strani dvopičja šestnajstiško število, ki uporabniku sporoča pomnilniški naslov, kjer je shranjen prvi bajt prevedenega ukaza. Ukaz, na katerega trenutno kaže programski števec, bo obarvan

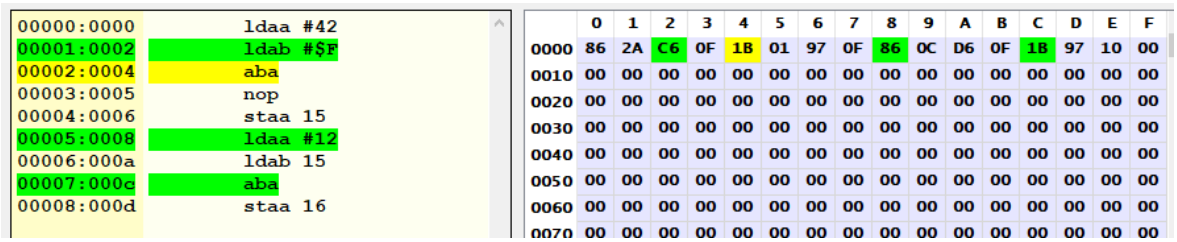
rumeno. V primeru, prikazanem na priloženi sliki 13, je to druga vrstica, saj ima PC (programski števec) vrednost štiri in ukaz na pomnilniški lokaciji 4 je ABA.

Če je vklopljena nastavitev "napredne informacije kode", bodo ob pomnilniški lokaciji izpisani vsi bajti, v katere se ukaz na ustrezni vrstici prevede, ločeni z dvopičjem. Prvi bajt je operacijska koda ukaza, drugi in tretji pa sta operanda ukaza (število operandov je odvisno od ukaza in načina naslavljanja).



Slika 14: Element uporabniškega vmesnika: Števec vrstic – napredne informacije kode (vir: lasten vir)

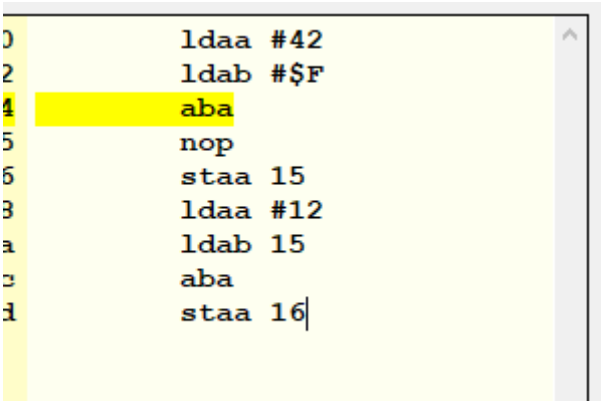
Emulator ponuja sistem označevanja ukazov. Če je koda pretvorjena, bo klik na vrstico tega polja obarval število vrstice, ukaz in naslov prvega bajta tega ukaza v spominu z zeleno barvo. Uporabnik lahko oznako izbriše s ponovnim klikom na označeno vrstico. Vse oznake se izbrišejo ob desnem kliku miške in ob spremembi izvorne kode. Te oznake pomagajo pri preglednosti spomina in razhroščevanju.



Slika 15: Element uporabniškega vmesnika: Števec vrstic – primer oznak (vir: lasten vir)

5.1.2 Polje za vpis kode

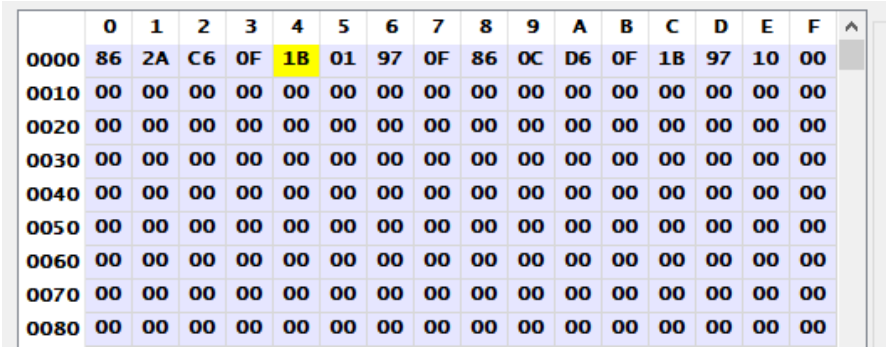
Ob številcu vrstic je polje za vpis kode. Vpis v polje je mogoč, ko je emulator nastavljen na način pisanja izvorne kode. Desni klik na to polje bo v meniju ob drugih podal tudi možnost "informacije ukaza", ki bo prikazala opis in vse ostale informacije o ukazu, nad katerim je kurzor.



Slika 16: Element uporabniškega vmesnika: Polje za vpis kode (vir: lasten vir)

5.1.3 Polje za prikaz spomina

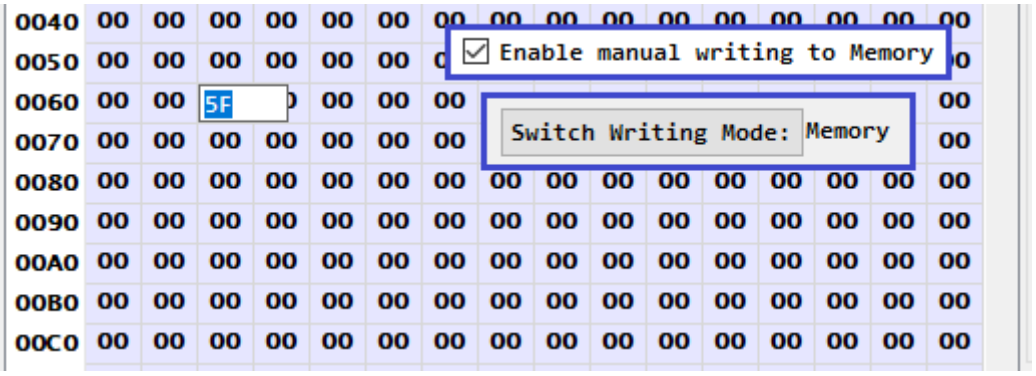
Polje za prikaz spomina, ki je na sredini glavnega okna v podobi tabele, prikazuje vse pomnilniške celice in vrednosti, ki jih hranijo. Privzeto so celice prikazane v šestnajstiškem številskem sistemu, možna pa je nastavitev na desetiški številski sistem.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	86	2A	C6	0F	1B	01	97	0F	86	0C	D6	0F	1B	97	10	00
0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

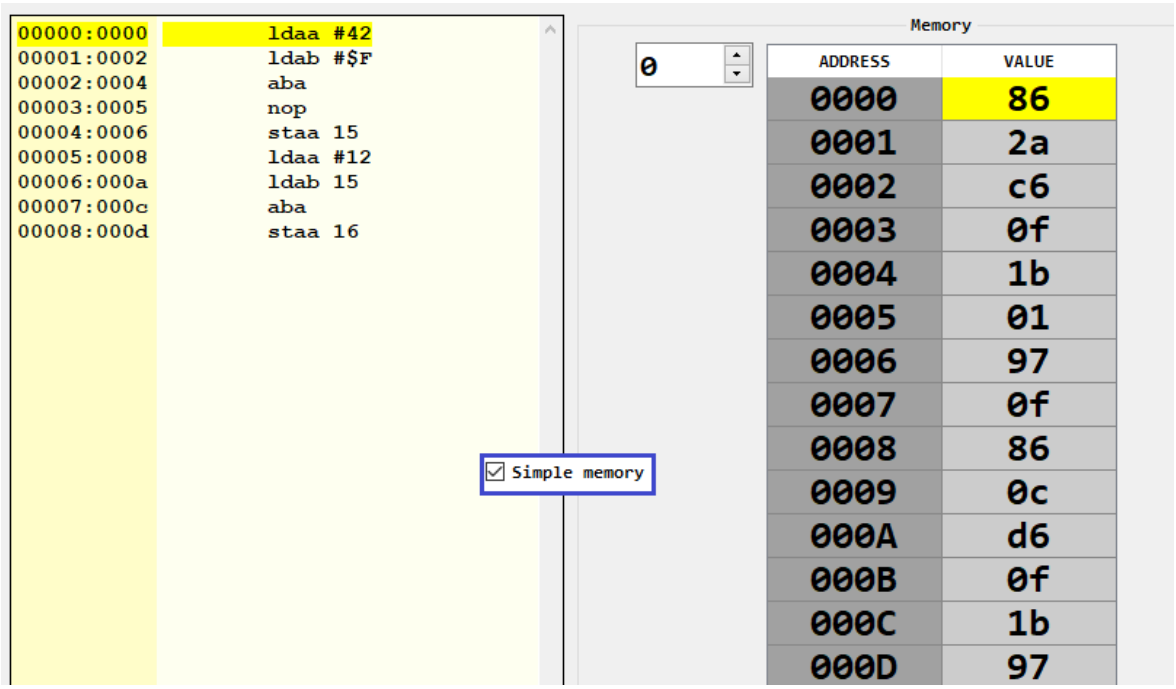
Slika 17: Element uporabniškega vmesnika: Prikaz spomina (vir: lasten vir)

Emulator ima nastavitev, ki nastavi emulator na način pisanja v pomnilnik (način razgradnje). Takrat lahko uporabnik ročno spreminja vrednost spominskih celic. Če je celica med izvajanjem spremenjena, bodo te spremembe ob kliku na gumb "ponastavi" zanemarjene. Če pa emulator trenutno ne izvaja ukazov, pa se bo stanje spomina ob zadnji spremembi zabeležilo kot privzeto. Posledično bo gumb za ponastavljanje emulatorja vrnil takratno vsebino spomina.



Slika 18: Element uporabniškega vmesnika: Prikaz spomina – sprememba celice pri načinu pisanja v spomin (vir: lasten vir)

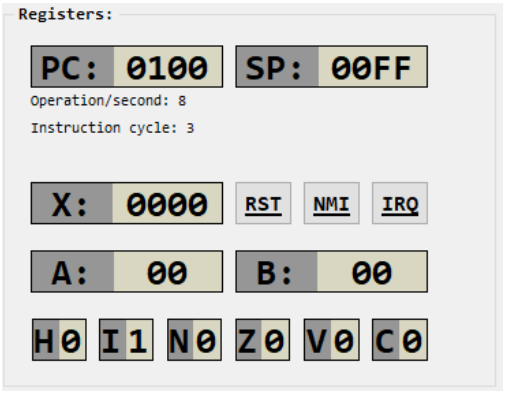
Obstaja enostavnejši prikaz spomina, ki je uporaben, ko je pomembno pozorno spremljanje sprememb nekaj celic ali pa če je za uporabnika privzeti prikaz spomina preveč zapleten. Inspiracijo za ta prikaz sem dobil pri emulatorju "Motorola 6800 Simulator", vendar v tistem primeru tak prikaz deluje bolj kot omejitev, saj je pregled nad razpršenimi variablami skoraj nemogoč. V levem stolpcu je pomnilniški naslov, v desnem pa vrednost spomina na tem naslovu. Prikazani naslovi se spreminjajo s števcem, ki je na levi strani tabele.



Slika 19: Element uporabniškega vmesnika: Prikaz spomina – enostaven spomin (vir: lasten vir)

5.1.4 Registri

Desno zgoraj je polje, ki prikazuje vsebine registrov mikroprocesorja, hitrost trenutnega izvajanja, trenutni cikel ukaza in kontrole prekinitev. Vrednosti, shranjene v registrih, so privzeto prikazane v šestnajstiškem številskem sistemu.

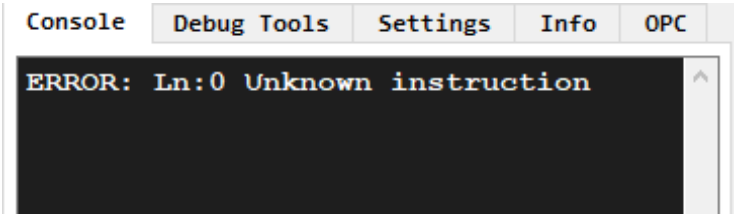


Slika 20: Element uporabniškega vmesnika: Prikaz registrov (vir: lasten vir)

5.1.5 Polje z zavihki

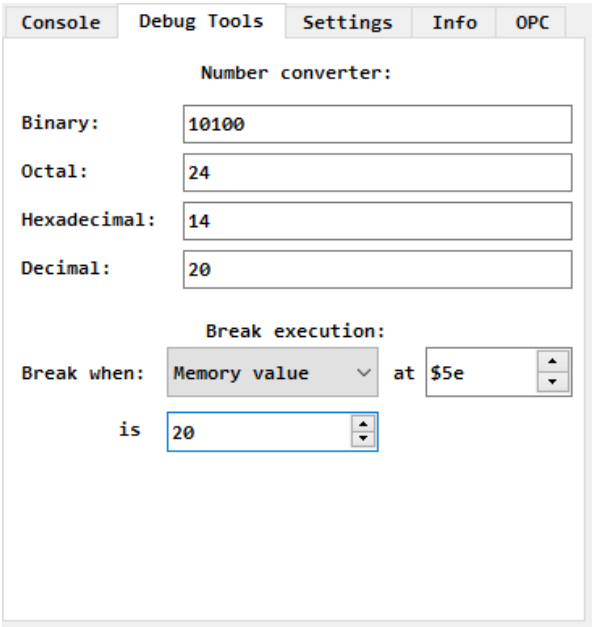
Desno spodaj je polje za izbiro zavihkov.

1. Prvi zavihek je okno konzole ki prikazuje napake, opozorila in informacije, ki jih emulator posreduje uporabniku.



Slika 21: Element uporabniškega vmesnika: Konzola (vir: lasten vir)

2. Drugi vsebuje orodja za odpravljanje napak kode (razhroščevanje). Vsebuje prevajalnik številskih sistemov in sistem prelomnih točk, ki uporabniku omogočajo samodejno ustavitev izvajanja ukazov ob določenem dogodku ali pogoju. Na primer, program se lahko ustavi, če bo kateri register vseboval zaželeno vrednost. Lahko se ustavi na določeni vrstici/ukazu ali pa, ko neka celica v spominu vsebuje zaželeno vrednost. Možne nastavitve so: preverjanje vrednosti registra (PC, SP, X, A, B), preverjanje vrednosti zastavice (H, I, N, Z, V, C), preverjanje vrednosti spominske celice in preverjanje trenutne vrstice izvajanja.



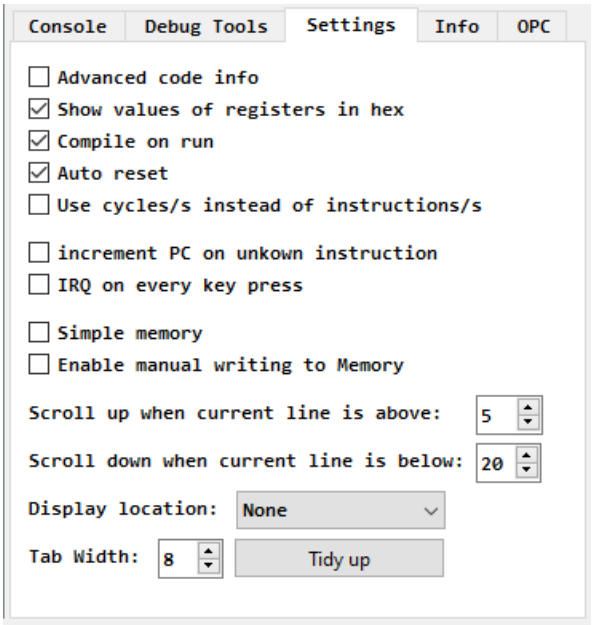
Slika 22: Element uporabniškega vmesnika: Razhroščevalna orodja (vir: lasten vir)

3. Tretji vsebuje večino nastavitev emulatorja. Možne nastavitve so:
    1. Vklop nastavitve "napredne informacije kode" bo v števcu vrstic izpisal vse bajte, v katere se ukaz na ustrezni vrstici pretvori, ločene z dvopičjem.

2. Nastavitev, ki določa, v kakšnem številskem sistemu bodo napisane vrednosti registrov in spominskih celic.
3. Nastavitev, ki določa, ali se bo zbirnik pred izvajanjem ukazov samodejno zagnal.
4. Nastavitev, ki določa, ali se bo emulator samodejno ponastavil na stanje pred zadnjim izvrševanjem.
5. Nastavitev, s katero uporabnik določi način izvajanja procesorja. Če je vklopljena, bo procesor ukaze izvajal ciklično. To pomeni, da bo upošteval število ciklov, ki jih posamezni ukaz potrebuje. Alternativa je, da izvaja določeno število ukazov na sekundo in cikle zanemari.
6. Nastavitev, ki emulatorju pove, da se v primeru neznanega ukaza ne ustavi, temveč poveča programski števec za 1 in nadaljuje z izvajanem. Ko ta nastavitev ni označena, bo emulator ob neznanem ukazu ustavil samodejno izvajanje.
7. Če je nastavitev "IRQ ob vsakem pritisku gumba" vključena, bo vsak vnos s tipkovnice na zaslon klical IRQ-prekinitev. Če pa je ta nastavitev izklopljena, pa bo IRQ klican po pritisku gumba samo, če je procesor v stanju čakanja na odziv na WAI-ukaz.
8. Nastavitev, ki določa, ali bo uporabniku prikazan standardni prikaz spomina, ki prikazuje 16 pomnilniških naslovov in njihovih vrednosti v eni vrstici, ali enostavnejši prikaz, ki prikazuje 20 poljubnih zaporednih pomnilniških naslovov in vrednosti, ki jih vsebujejo.
9. Nastavitev, ki omogoča nastavljanje delovanja emulatorja med načinom pisanja izvorne kode in načinom pisanja strojne kode. To posledično tudi določa, ali bo emulator izvorno kodo sestavljal ali strojne ukaze razgradil.

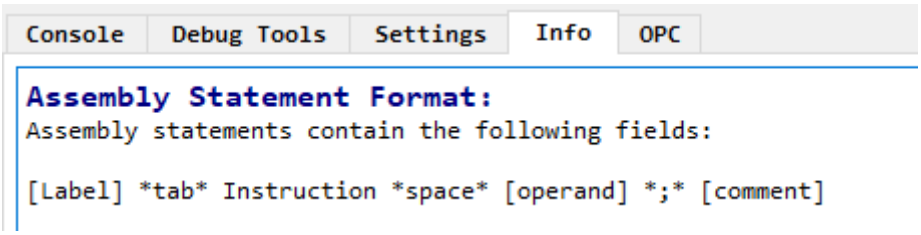
   Ko je nastavitev vklopljena, se na spodnjem delu glavnega okna prikaže gumb, ki prikazuje trenutni način delovanja emulatorja ter ponuja uporabniku, da ga zamenja.
10. Nastavitvi za samodejno premikanje polja za vpis kode na ukaz, ki se trenutno izvršuje.
11. Nastavitev, ki določa, kje bo prikazan zaslon. Če je nastavitev nastavljena na "glavno okno", bo zaslon prikazan na glavnem oknu med poljem za spomin in poljem registrov, ko bo ta imel zadosti prostora oziroma, ko bo glavno okno dovolj široko. Če je nastavitev nastavljena na "zunanje okno", bo zaslon prikazan zunaj glavnega okna v obliki okna za dialog.

12. Nastavitev, ki določa, kako širok je "tab" znak. Privzeto ima širino osmih presledkov. Ob tej nastavitvi je gumb, ki počisti uporabnikovo kodo. To naredi tako, da poravna vse zamike besedila in zamenja presledke s potrebnim številom "tab" znakov.



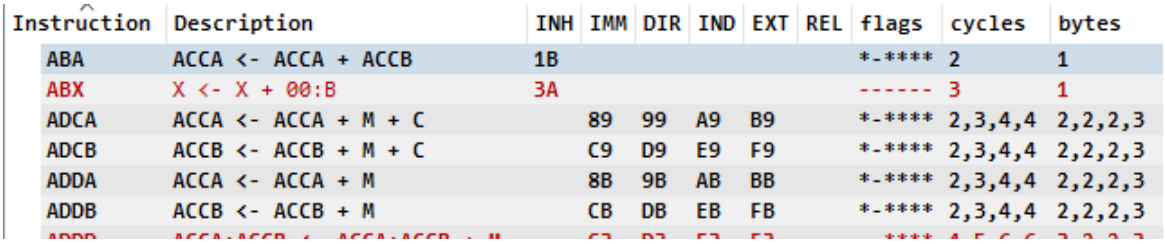
Slika 23: Element uporabniškega vmesnika: Nastavitve (vir: lasten vir)

4. Četrti zavihek vsebuje opis in navodila za uporabo emulatorja.



Slika 24: Element uporabniškega vmesnika: Informacije (vir: lasten vir)

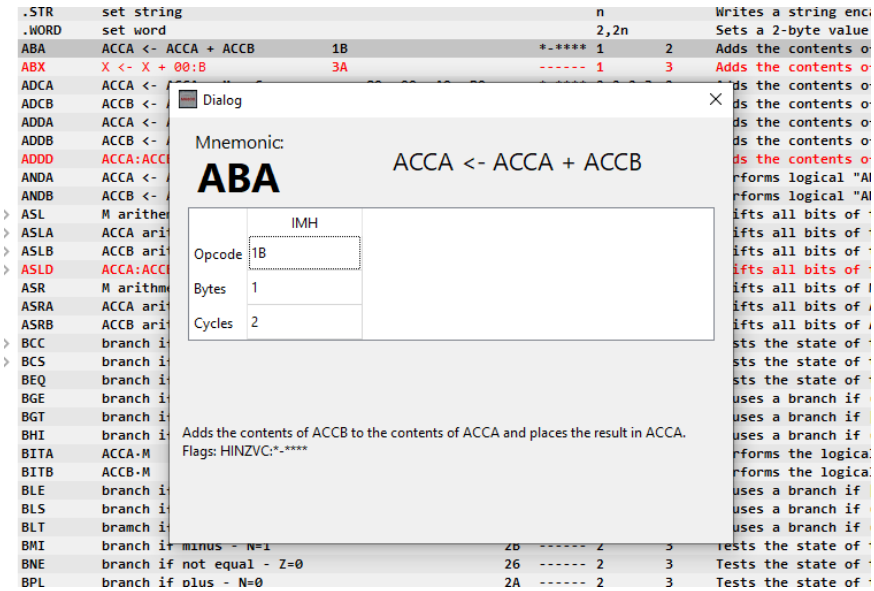
5. Peti vsebuje tabelo z vsemi ukazi mikroprocesorjev Motorola M6800 in Motorola M6803. Ukazi, operacijske kode, število ciklov, velikost in opisi zadnje omenjenega so obarvani rdeče.



Instruction	Description	INH	IMM	DIR	IND	EXT	REL	flags	cycles	bytes
ABA	ACCA <- ACCA + ACCB	1B						*-****	2	1
ABX	X <- X + 00:B	3A						------	3	1
ADCA	ACCA <- ACCA + M + C		89	99	A9	B9		*-****	2,3,4,4	2,2,2,3
ADCB	ACCB <- ACCB + M + C		C9	D9	E9	F9		*-****	2,3,4,4	2,2,2,3
ADDA	ACCA <- ACCA + M		8B	9B	AB	BB		*-****	2,3,4,4	2,2,2,3
ADDB	ACCB <- ACCB + M		CB	DB	EB	FB		*-****	2,3,4,4	2,2,2,3

Slika 25: Element uporabniškega vmesnika: Zavihek ukazov (vir: lasten vir)

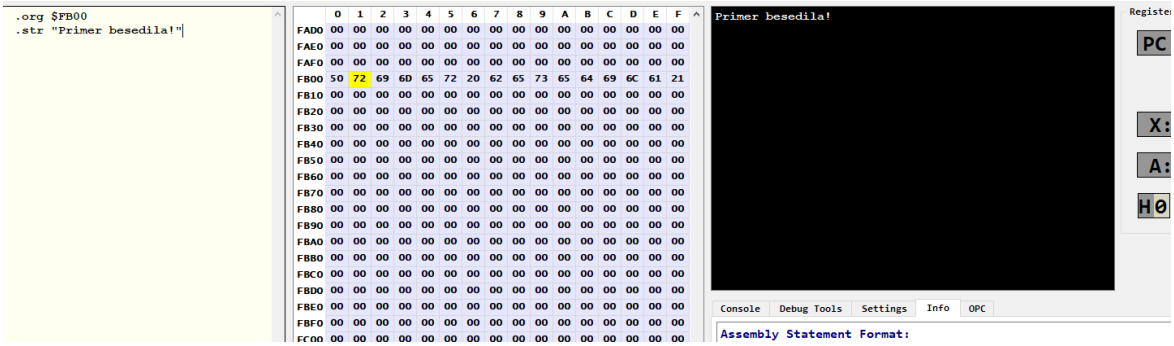
Klik na ukaz bo odprl okence, ki bolj pregledno prikazuje opis in vse ostale informacije ustreznega ukaza.



Slika 26: Okno informacij ukaza ABA (vir: lasten vir)

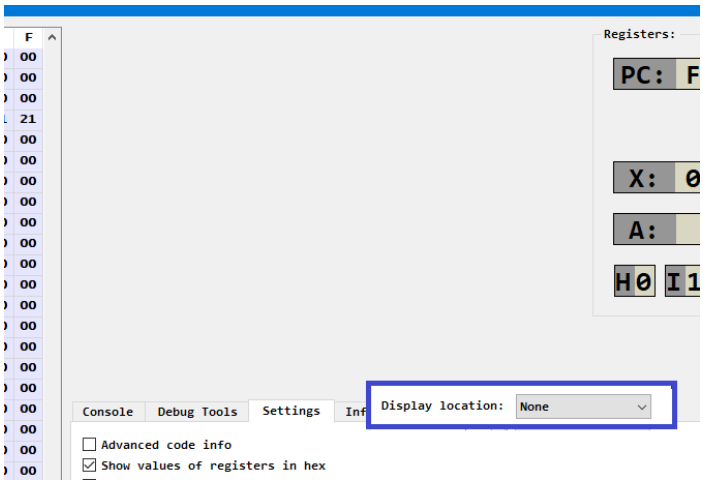
5.1.6 Zaslon

Zaslon omogoča zunanjo komunikacijo z emuliranim mikroprocesorjem. Deluje kot vhodna in izhodna naprava. Privzeto je onemogočen, če je programsko okno zadosti široko (če je širina okna večja ali enaka 1785 slikovnih pik), se bo pojavil med poljem za prikaz spomina in poljem registrov.

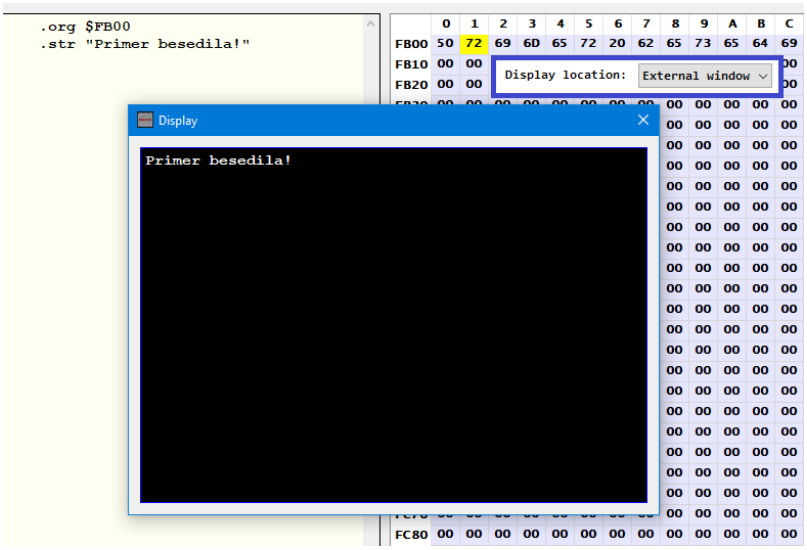


Slika 27: Prikaz zaslona na glavnem oknu (vir: lasten vir)

V nastavitvah lahko uporabnik spremeni lokacijo zaslona. Zaslon je lahko prikazan na glavnem oknu, zunanjem oknu, lahko pa je izklopljen.



Slika 28: Primer izklopljenega zaslona (vir: lasten vir)



Slika 29: Primer zaslona, prikazanega na zunanjem oknu (vir: lasten vir)

Za spreminjanje zaslona je zadolžen medpomnilnik. Ta se nahaja v pomnilniku od naslova 0xFB00 do naslova 0xFF38. Ta zaslon je znakovni zaslon z 20 vrsticami in 54 stolpci. Vsak naslov v medpomnilniku je povezan z enim znakom na zaslonu. Vrednosti teh pomnilniških celic se z ASCII-tabelo pretvorijo v znake, ki se izpišejo na ustreznem polju na zaslonu.

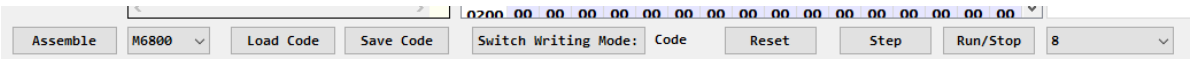
Ko je zaslon fokusiran, bo prejemal tudi vhodne enote, ki so tipke na tipkovnici, tipke na miški in lokacija miške. Te vrednosti se posodobijo pred izvedbo vsakega ukaza.

Zadnja tipka, ki je bila pritisnjena, se bo zapisala v medpomnilnik na lokaciji 0xFFF0.

Stanje miškinih tipk se bo zapisovalo na lokaciji 0xFFF1. Ob pritisku na levo tipko, se bo tam shranila vrednost 1. Pritisk desne je vreden 2, pritisk srednje tipke pa 3. Izpust levega gumba bo zapisal vrednost 4, desnega 5 in srednjega 6.

Vrstica znaka, na katerega kaže miška, bo zapisana na lokaciji 0xFFF2, stolpec znaka pa na 0xFFF3.

5.1.7 Gumbi in kontrole



Slika 30: Element uporabniškega vmesnika: Gumbi in kontrole (vir: lasten vir)

Na spodnji strani glavnega okna so gumbi in izbirni meniji za olajšano in uporabniku prijaznejšo uporabo emulatorja.

Ti so:

1. Gumb "sestavi/razgradi" v načinu pisanja *pisanje izvorne kode* bo ukaze, ki so trenutno napisani v polju za pisanje ukazov, pretvoril v strojno kodo. Če pa je način pisanja nastavljen na *pisanje v pomnilnik*, bo ta gumb razgradil strojno kodo, zapisano v spominu, in jo zapisal v polje za ukaze.
2. Meni za izbiro različice Motorola procesorjev. Trenutno emulator podpira dve različici: Motorola M6800 in Motorola M6803, ampak je zasnovan tako, da omogoča nadaljnjo razširitev in dodajanje novih procesorjev. Emulator se bo ravnal po izbranemu mikroprocesorju.
3. Gumb za nalaganje bo glede na izbiro načina pisanja iz zunanje datoteke naložil strojno kodo ali pa vrednosti pomnilnika.
4. Gumb za shranjevanje bo glede na izbiro načina pisanja shranil izvorno kodo ali vsebino pomnilnika v datoteko.
5. Gumb za zamenjavo načina pisanja. Privzeto je ta gumb nedosegljiv, lahko se pa prikaže z nastavitvijo "dovoli zamenjavo načina pisanja". Desno od tega gumba je izpisan trenutni način pisanja.
6. Gumb za ponastavitev ponastavi emulator na stanje, ki je bilo pred izvajanjem.
7. Gumb "korak" izvede en ukaz ali odziv na prekinitev s procesorjem, izbranim v meniju za izbiro različice procesorja.
8. Gumb "zaženi/ustavi" v primeru, da procesor trenutno ne izvaja ukazov, zažene samodejno izvajanje ukazov. V primeru, da procesor trenutno izvaja ukaze, pa ga ta gumb začasno ustavi. Uporabnik lahko izvajanje vedno nadaljuje z vnovičnim pritiskom na ta gumb. Gumb deluje v skladu z menijem hitrosti in nastavitvijo načina izvajanja (upoštevanje ciklov). Procesor bo samodejno prenehal izvajanje, kadar naleti na strojni kod 0x00. Če programski števec (PC) trenutno vsebuje pomnilniško lokacijo, na kateri se nahaja vrednost 0x00, in če je nastavitev "samodejna ponastavitev" vklopljena, se bo emulator ponastavil.

9. Meni za nastavitev hitrosti izvajanja. Prikazuje število ukazov ali ciklov na sekundo. Možne hitrosti so potence števila dva od 1 do 1048576. Najvišja možna hitrost je odvisna od zmogljivosti uporabnikovega računalnika.

5.2 Opis notranjih komponent emulatorja

5.2.1 Zbirnik

Zbirnik je programska oprema, ki prevaja ukaze, operacije, načine naslavljanja in operande določenega zbirnega jezika v njihove številske vrednosti (strojno kodo). Kot mnogi drugi zbirniki uporablja in prepozna simbolične reference. To je način poimenovanja določenih pomnilniških lokacij ali ukazov, katerih uporaba nadomesti ročne izračune in posodobitve pomnilniških naslovov po spremembah programa ter olajša uporabo spremenljivk. Moj zbirnik za M68XX je "One-pass assembler", kar pomeni, da gre skozi kodo samo enkrat in rezervira prostor za simbole, ki še niso bili definirani. Na koncu pa zbirnik preide skozi rezervirane pomnilniške naslove in jih nadomesti z vrednostjo simbolov.

Moj zbirnik deluje tako, da gre v zanki skozi vsako vrstico kode. Za vsako vrstico si naredi kopijo, ki jo nato manipulira, krajša in si jo pripravlja, da jo lahko razume. Ta proces ima v glavnem tri dele. Na začetku odstrani vse komentarje (komentarji se začnejo s podpičjem).

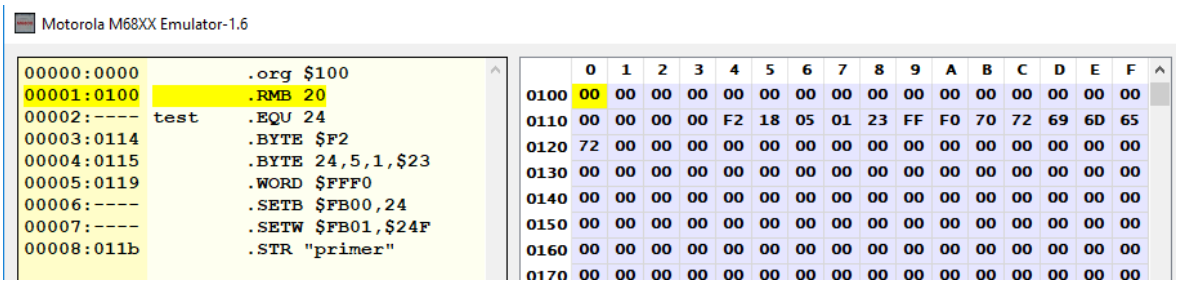
Nato pride v prvi del, ki izvleče simbol ukaza, če je podan. Njegov obstoj preveri tako, da preveri, ali je prvi znak vrstice črka. Nato s pomočjo variable "charNum" pregleda vrstico in zabeleži, kdaj se pojavi znak, ki ni črka, število ali podčrtaj. Potem preveri še nekaj samoumevnih stvari, na primer, ali je vrstica prazna, ali so v simbolu nedovoljeni znaki in podobno.

V drugem delu se vrstica razdeli na ukaz in operand ter se razveljavi, če ta nista pravilno podana. Podobno kot prej "charNum" gre skozi vrstico in gleda, kdaj se skupek črk začne in konča. Operandu preveri tudi validnost.

V tretjem oziroma zadnjem delu pa identificira in prevede ukaz. Na podlagi operanda ugotovi, katero naslavljanje je primerno in ali je operand za tak ukaz primeren. Če je operand že definiran simbol, bo v lokacijo operanda vstavljena njegova vrednost. V primeru, da operand še ni definiran, pa bo ukaz zapisan z največjim primernim naslavljanjem in prostor operanda bo ostal prazen. Zbirnik ima tri sezname, ki beležijo nedefinirane simbole in lokacije, v katere sodijo. Nato, če se vsaka vrstica prevede brez težav, bo emulator pregledal vse sezname in zapolnil manjkajoča mesta z vrednostmi, ki jih ti simboli vsebujejo. Na koncu se preveri, ali je "šlo vse kot po maslu", in emulator si zabeleži, da je zadnja pretvorba ukazov bila uspešna, in to sporoči tudi uporabniku. V primeru sintaktične napake bo pretvorba neuspešna, pomnilnik bo ponastavljen in vrstica, kjer se je napaka pojavila, bo označena; če je težava v specifičnem znaku vrstice, bo označen tudi ta znak. Opis napake pa bo zapisan v konzoli.

V zbirnik sem vključil tudi nekaj zbirnih direktiv. To so navodila, ki jih zbirniku podamo, da nadzorujemo ali prilagajamo proces pretvarjanja izvorne kode v strojni jezik. Moj emulator podpira naslednje:

1. .ORG: Določi izhodiščni ali začetni naslov programa ali odseka. Sledeči ukazi se bodo po pretvorbi zapisali na podan naslov. Ta se zapiše tudi v prekinitveni vektor (RST) za ponastavljanje procesorja.
2. .RMB: Rezervira spomin oziroma preskoči pomnilniške lokacije in jih pusti prazne.
3. .EQU: Ta ukaz definira konstanto. Simbolu, pod katerim je klican .EQU, je prirejena vrednost, ki jo določi uporabnik. V primeru "test .EQU 24" bo simbol "test" vseboval število 24.
4. .BYTE: V naslednjo pomnilniško lokacijo se zapiše en ali več zaporednih bajtov.
5. .WORD: V naslednjo pomnilniško lokacijo se zapiše ena ali več besed (2 bajta).
6. .SETB: V podan naslov shrani podan bajt. Primer: ".SETB $FB00,24".
7. .SETW: V podan naslov shrani besedo.
8. .STR: V naslednjo pomnilniško lokacijo shrani en ali več nizov.



Slika 31: Primer uporabe zbirnih direktiv (vir: lasten vir)

5.2.2 Izvajanje ukazov

V tem poglavju je opis delovanja najpomembnejšega dela mojega emulatorja. To je izvajanje ukazov.

5.2.2.1 Funkcija executeInstruction

To je glavna funkcija za izvajanje ukazov. Ko je klicana, si rezervira nekaj začasnih variabel, s katerimi si pomaga pri izvajanju posameznega ukaza. Nato naloži operacijsko kodo ukaza, na katerega trenutno kaže programski števec. Z uporabo C++ "switch" stavka emulator ve, kako naj izvede ukaz. Tu je nekaj primerov izvedbe:

1. Operacijska koda je 0x1B ali 27: To je operacijska koda za ukaz ABA. V prvo začasno neznanko se shrani vsota akumulatorjev A in B. Nato procesor primerja razlike med ACCA, ACCB in vsoto ter primerno posodobi zastavice. Na koncu se vsota shrani v ACCA in se vrednost PC poveča.
2. Operacijska koda 0: Če je bila funkcija klicana s samodejnim izvajanjem, se bo samodejno izvajanje ustavilo, drugače se bo vrednost PC povečal za ena.

3. Operacijske koda, ki jo izbrana različica procesorja ne podpira: Če je šesta nastavitev vklopljena, se bo vrednost PC povečala za ena. Če pa je ta nastavitev izklopljena in je funkcija bila klicana s samodejnim izvajanjem, se bo samodejno izvajanje ustavilo.

5.2.2.2 Izvedba posameznega ukaza

Z gumbom "korak" bo emulator izvedel en ukaz ali odziv na prekinitev. Najprej se izklopi samodejno izvajanje. Program se bo samodejno sestavil, če je nastavitev "sestavi ob izvajanju" vklopljena in program še ni sestavljen. Po potrjenem stanju sestavljenosti se bo izvršil en korak procesorja. Potek izvajanja koraka je podoben izvajanju funkcije executeInstructionTick, prikazane v diagramu slike 34. Edina razlika je, da se ta del programa konča pred preverjanjem prelomnih točk. Če prekinitev ni bila klicana, bo klicana funkcija za izvršitev ukaza, na katerega kaže PC (poglavje 5.2.2.1.). Po izvršitvi ukaza je klicana posodobitev trenutnih elementov UI.

5.2.2.3 Samodejno izvajanje ukazov

Samodejno izvajanje se zažene in ustavi z gumbom "zaženi/ustavi". Glavna spremenljivka, ki določa stanje samodejnega izvajanja, se imenuje "running". Ko ima running vrednost 1 ali "true", procesor samodejno izvaja ukaze. Ko pa ima running vrednost 0 ali "false", pa je samodejno izvajanje ustavljeno. Ko uporabnik pritisne gumb "zaženi/ustavi" in je vrednost "running" enaka 0, se bo program samodejno sestavil, če je nastavitev "sestavi ob izvajanju" vklopljena in program še ni sestavljen. Nato bo emulator preveril, ali je hkrati operacijska koda naslednjega ukaza enaka 0 in je vklopljena nastavitev "samodejno ponastavljanje". Takrat bo se emulator ponastavil. Potem bo klicana funkcija startExecution.

5.2.2.3.1 Funkcija startExecution

Ta funkcija najprej nastavi spremenljivko running na 1. Sočasno požene časovnik iz razreda "QTimer", ki po intervalih izvršuje funkcijo "updateIfReady". Nato prikaže UI elemente, ki uporabniku sporočajo, da emulator trenutno samodejno izvaja ukaze. Nazadnje zažene novo nit, ki je namenjena neprestanemu izvrševanju ukazov. To sem dosegel z razredoma "QFutureWatcher" in "QtConcurrent". Nova nit je potrebna, saj je neprestano izvrševanje ukazov v višjih hitrostih zelo težavno za procesor uporabnikovega računalnika in bi povzročilo veliko težav, kot so neodzivni uporabniški vmesnik, počasno posodabljanje zaslona in podobno.

Izvajanje ukazov poteka v zanki. Ko je čas primeren, se koraki (odvisno od nastavitev je korak cikel ali ukaz) izvajajo serijsko. To pomeni, da se v eni seriji izvede nekaj korakov zaporedoma. V prvih različicah emulatorja so se koraki izvajali zaporedoma s konstantnim intervalom. Ko sem pa želel dodati večje hitrosti, sem ugotovil, da je zaradi prestopov med iteracijami v zanki in nenehnega merjenja časa na nanosekunde natančno izvajanje korakov s konstantnim intervalom neizvedljivo. Na teh operacijah se porabi preveč časa in tudi zmogljivejši gostiteljski procesorji niso sposobni izvajanja na tak način. Mojo rešitev s serijskim izvajanjem podpirajo tudi dejstva, da večina zaslonov podpira le 60 posodobitev na sekundo in bi bila višja meja za povprečnega uporabnika okoli 240 posodobitev na sekundo. Zato je posodabljanje uporabniškega vmesnika več kot 240-krat na sekundo nepotrebno in za povprečen procesor prezahtevno. Tako sem določil, da je maksimalno število posodobitev zaslona 256 posodobitev na sekundo. To pomeni, da je do hitrosti 256 korakov na sekundo serijsko izvajanje nepotrebno. Pri hitrostih od 512 korakov na sekundo naprej pa se postopno zvišuje število korakov, ki se izvajajo v serijah. V tabeli 1 so prikazane vse možne hitrosti izvajanja, koliko korakov je v vsaki seriji, kolikšen bi bil časovni zamik med posameznimi koraki in časovni zamik med serijami.

Zanka deluje tako, da si vedno zabeleži čas prejšnjega izvajanja koraka. Ta spremenljivka se shrani na začetku izvajanja serije korakov. Takrat se spremenljivki prišteje tudi zmnožek števila serijsko klicanih zaporednih korakov in čakalnega časa enega koraka. Tako dobimo spremenljivko, ki vsebuje čas naslednjega izvajanja. Tako zanka nenehno preverja, ali je trenutni čas večji od te spremenljivke, in če je, potem izvede naslednjo serijo ukazov ali ciklov. Izvedba koraka se precej razlikuje v načinu izvajanja po ciklih in ukazih. To delovanje je prikazano v diagramih na slikah 32, 33 in 34.

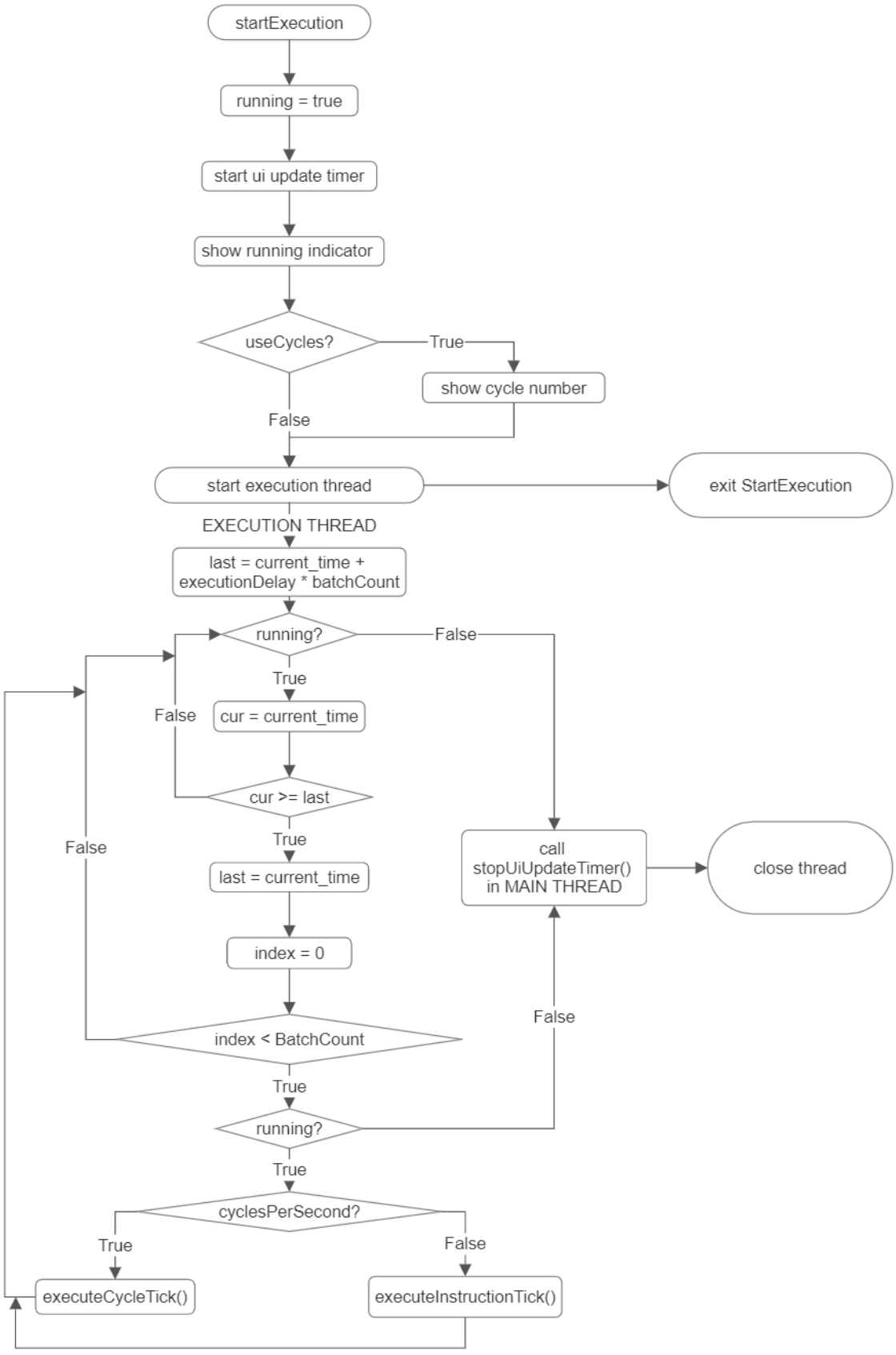
Po izvedbi koraka ali odziva na prekinitev se preverijo stanja razhroščevalne prekinitve. Te uporabnik nastavi v zavihku za razhroščevanje. Če stanje procesorja ustreza danemu pogoju, bo spremenljivka running nastavljena na 0, kar bo povzročilo, da se bo samodejno izvajanje ustavilo.

Tabela 1: Časi serijskega izvajanja (vir: lasten vir)

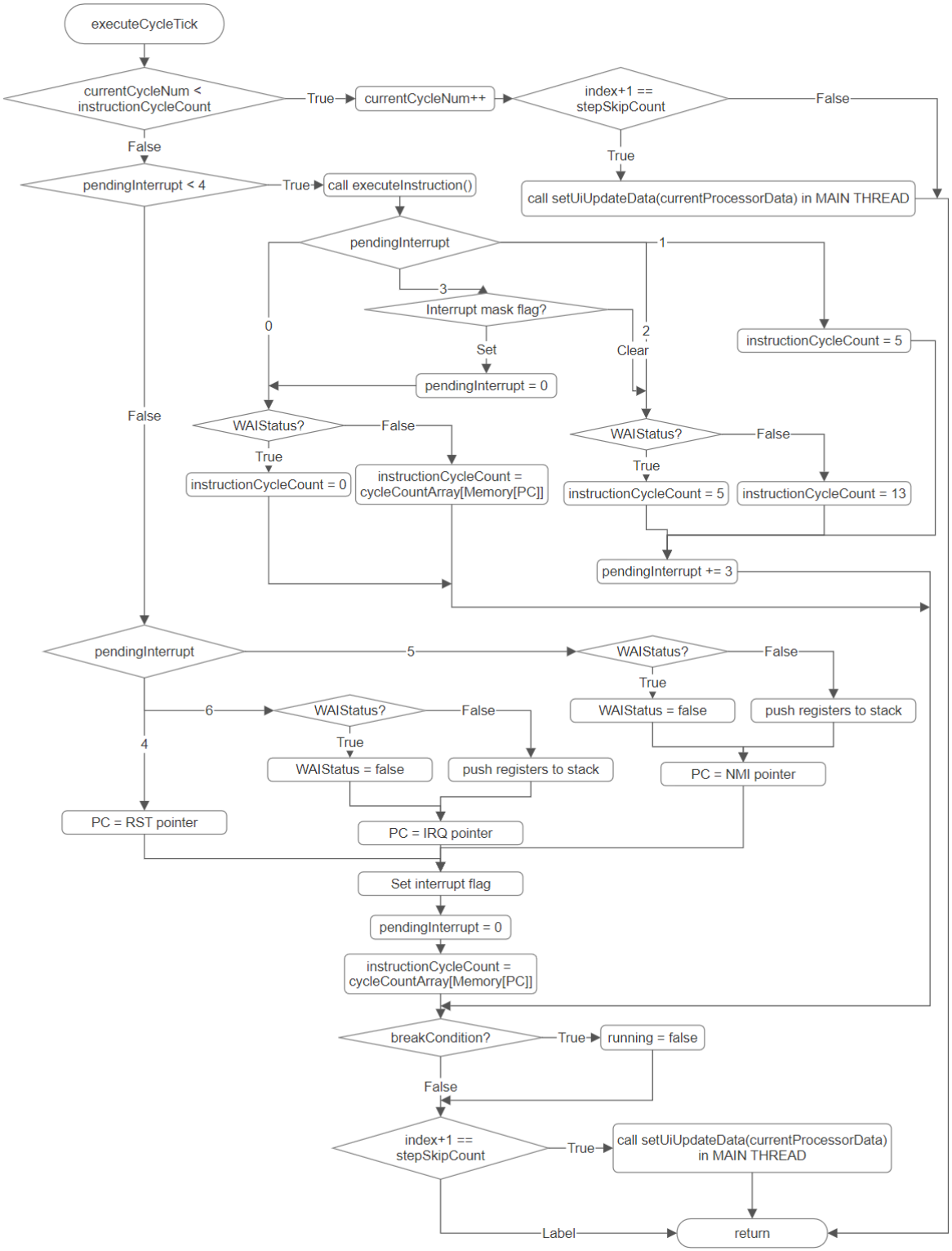
Cikli na sekundo	Cikli v seriji	Časovni zamik med cikli	časovni zamik med serijami
1	1	1000000000ns	1000000000ns
2	1	500000000ns	500000000ns
4	1	250000000ns	250000000ns
8	1	125000000ns	125000000ns
16	1	62500000ns	62500000ns
32	1	31250000ns	31250000ns
64	1	15625000ns	15625000ns
128	1	7812500ns	7812500ns
256	1	3906250ns	3906250ns
512	2	1953125ns	3906250ns
1024	4	976563ns	3906252ns
2048	8	488282ns	3906256ns
4096	16	244141ns	3906256ns
8192	32	122071ns	3906272ns
16384	64	61036ns	3906304ns
32768	128	30518ns	3906304ns
65536	256	15259ns	3906304ns
131072	512	7630ns	3906560ns
262144	1024	3815ns	3906560ns
524288	2048	1908ns	3907584ns
1048576	4096	954ns	3907584ns

Po zadnjem izvršenem koraku v seriji je iz izvrševalne niti poslano trenutno stanje procesorja v glavno nit. Ob naslednjem utripu časovnika (QTimer) se bo v glavni niti posodobil uporabniški vmesnik na zadnje stanje, ki ga je glavna nit prejela iz izvajalne niti.

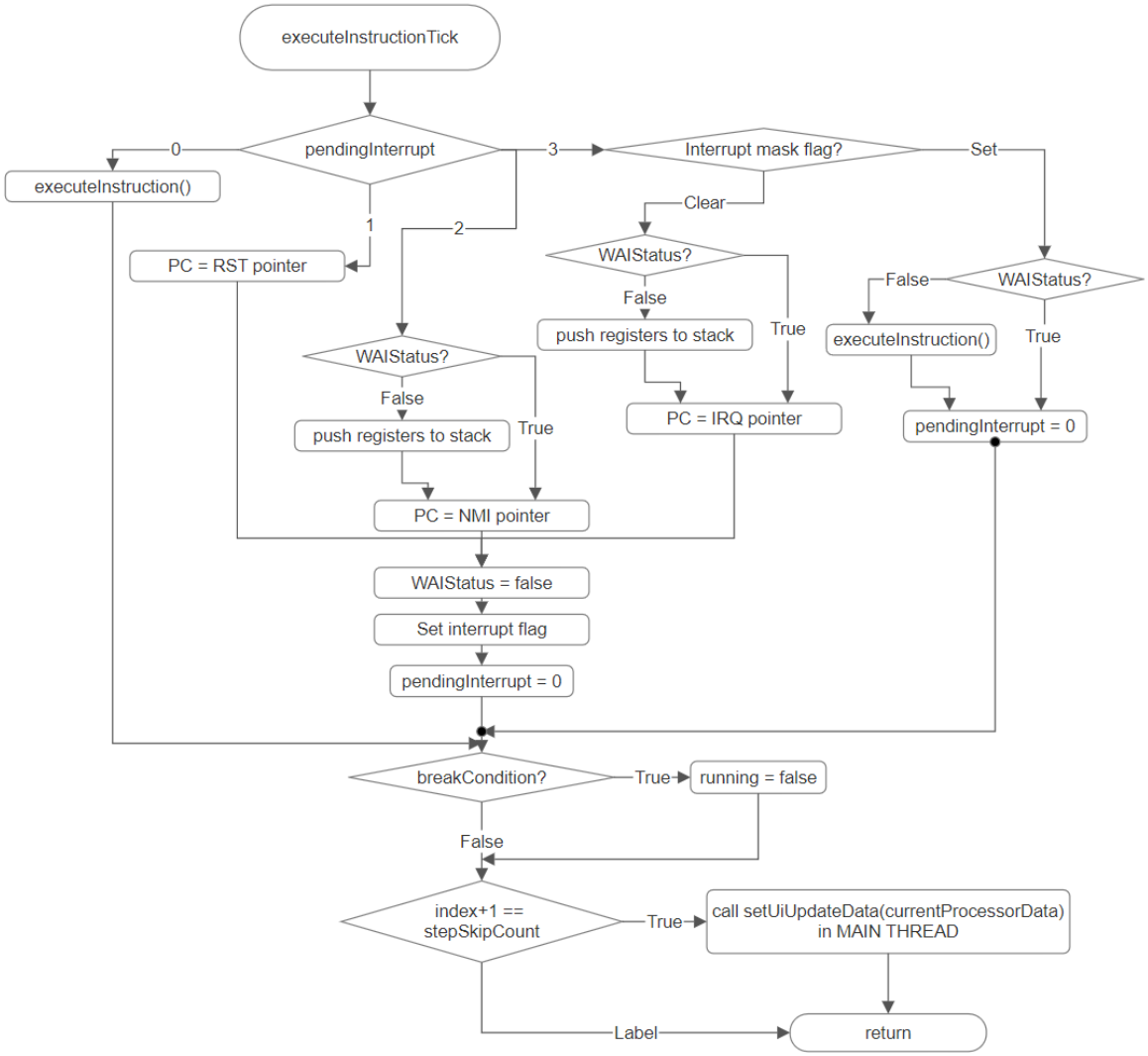
Spodaj so podani diagrami poteka, ki podrobneje predstavljajo delovanje funkcije "startExecution".



Slika 32: Diagram poteka: Funkcija StartExecution (vir: lasten vir)



Slika 33: Diagram poteka: Funkcija StartExecution: executeCycleTick (vir: lasten vir)



Slika 34: Diagram poteka: Funkcija startExecution: executeInstructionTick (vir: lasten vir)

5.2.2.3.2 Funkcija stopExecution

Ta funkcija je odgovorna za ustavljanje samodejnega izvajanja. Klicana je na zahtevo uporabnika. Vse, kar ta funkcija naredi, je, da spremeni vrednost spremenljivke running na 0.

Po končanem ciklu ali med čakanjem na izvajanje se bo zanka v funkciji startExecution končala in klicala funkcijo "stopUiUpdateTimer" v glavni niti.

5.2.2.3.3 Funkcija stopUiUpdateTimer

Ta funkcija ustavi časovnik uiUpdateTimer. Nato skrije elemente UI, ki uporabniku sporočajo, da je samodejno izvajanje v teku. Na koncu pa posodobi vse elemente na trenutno stanje procesorja.

5.2.2.3.4 Posodabljanje UI med samodejnim izvajanjem

V izvajalni zanki je na koncu vsake serije v glavni niti klicana funkcija setUiUpdateData. Ta funkcija shrani trenutno stanje procesorja v C++ strukturo podatkov (struct).

Utripi časovnika uiUpdateTimer so povezani s funkcijo updateIfReady. Ta funkcija ob vsakem utripu preveri, ali je bilo od zadnjega utripa posodobljeno stanje procesorja. V takem primeru pa posodobi vse elemente UI na prejeto stanje procesorja.

5.2.3 Disassembler

V emulator sem vključil tudi primitiven razstavljavec. To je program, ki strojno kodo prevede oz. razstavi v zbirne ukaze. Ko je emulator v načinu pisanja *v spomin*, bo namesto gumba "sestavi" prikazan gumb "razgradi". Pred razstavljanjem bo uporabnik določil, kje se program začne. Vrednosti pred podanim naslovom bo program prepisal z ".BYTE" ukazom. Nato bo razstavljavec v zanki šel skozi vse pomnilniške naslove, ki nimajo ničelne vrednosti. Če dissasembler naleti na neznan ali ne podpiran ukaz, bo uporabnik vprašan, kje naj se razstavljanje nadaljuje. Tudi tu bodo manjkajoči podatki zapisani z ukazom ".BYTE".

5.3 Testiranje

Med razvojem sem nenehno zaganjal kodo in pregledoval ter popravljal njeno delovanje.

Na koncu sem v vsaki funkcionalnosti posebej iskal in odpravljal napake, ki bi program sesule ali povzročile nepričakovano delovanje emulatorja. Pri poljih za vhode sem vstavljal mejne vrednosti in poskušal naleteti na nenačrtovan scenarij, ki bi sesul program.

Zbirnik sem tako preizkusil, da sem napisal program z vsemi ukazi. Nato sem to kodo prevedel in s pomočjo priročnika preveril pretvorbo vsakega ukaza.

Izvajanje ukazov sem preveril tako, da sem izvedel vsak ukaz posebej. V operande sem vstavljal mejne vrednosti (prim. 0, 127, 128, 255, 256, 65535). Pri vsakem ukazu sem pravilno izvedbo potrdil s priročnikom "M6800 Programming reference manual".

Ostale funkcionalnosti sem podobno testiral tako, da sem iskal nepredvidene načine uporabe.

6 RAZPRAVA

6.1 Evalvacija lastnega emulatorja

Menim, da moj emulator dosega in v nekaterih točkah presega vse prej zastavljene kriterije. Uspešno sem vključil tudi vse želene funkcije.

1. Moj emulator učinkovito sestavlja ukaze za M6800 in za M6803. Ročno sem preizkusil pretvorbo vsakega ukaza, vključujoč z vsako možno vrsto naslavljanja.
2. Prikaz spomina je pregleden in prikaže več kot 512 spominskih celic naenkrat. Vsebine registrov so pregledno prikazane v za to namenjenem polju. Vsak register je kratko in jasno označen. V tem polju so prikazani tudi podatki delovanja procesorja, kot sta hitrost izvajanja in trenutni cikel izvajanja ukaza. Na voljo je tudi števec vrstic, ki uporabniku prikaže naslov, na katerem se ukaz nahaja.
3. Emulator omogoča ročno pisanje v pomnilnik. Na voljo je tudi razstavljavec.
4. V emulator sem vključil vse ukaze in vrste prekinitev, ki jih je imel originalen procesor. Omogoča tudi ciklično izvajanje. Najvišja možna hitrost je 1 Mhz.
5. Dodana so orodja za: sistem pogojne ustavitve izvajanja, pretvornik številskih sistemov, "konzolno" okno, sistem označevanja ukazov in pomnilniških celic za boljšo preglednost.
6. Emulator ponuja tudi uporabo vhodnih in izhodnih naprav. Ima zaslon, ki se posodablja s pomočjo medpomnilnika. Ta zaslon tudi sprejema lokacijo miškinega kazalca, miškine gumbe in tipke na tipkovnici.
7. Uporabniku so ponujene tudi razne nastavitve za osebno prilagoditev emulatorja.
8. Emulator vsebuje tudi podroben opis vsakega ukaza v svojem zavihku. Ima tudi zavihek s splošnimi informacijami o delovanju M6800 in navodila za uporabo emulatorja.
9. Med ustvarjanjem vsake funkcionalnosti emulatorja sem se vedno spraševal, kakšen odziv na dejanja bi pričakoval uporabnik, ki še nikoli ni uporabljal emulatorja. Program nima težav z nepredvidenim nedelovanjem, zaostajanjem (ang. lag) in neodzivnostjo.

Na podlagi opisanega lahko sklenem, da je opazen velik skok v napredku v primerjavi z ostalimi emulatorji.

6.2 Težave z razvojem

Pri nobeni komponenti emulatorja nisem imel zadosti velike težave, da bi nad njo obupal in jo opustil. Sem pa bil blizu, ko sem poskušal doseči hitrost izvajanja 1 Mhz. Na začetku je emulator podpiral le hitrost 1000 ukazov na sekundo. Takrat sem izvajal funkcijo executeInstruction s časovnikom QTimer. Večkrat sem vso kodo, povezano z izvajanjem, zavrgel in jo napisal od začetka. Rešitev sem po kakšnem mesecu garanja našel v serijskem izvajanju in večnitnosti.

6.3 Čas in stroški razvoja

Program sem razvijal štiri mesece. Programiral sem med prostim časom povprečno dve uri na dan. Največ časa mi je vzel razvoj zbirnika in sistem, ki časovno enakomerno izvaja ukaze.

Vsa orodja, ki sem jih uporabil, so odprtokodna in brezplačna za nekomercialno uporabo, zato denarnih stroškov ni bilo.

7 DRUŽBENA ODGOVORNOST

Pri umeščanju moje inovacije glede na trajnost je najpomembnejši vidik, da emulator deluje v digitalni obliki, zato ni potreben nakup dodatnih komponent. Uporabnik ga zgolj namesti na svoj računalnik. Ni mu treba kupiti mikroprocesorja niti ni potrebna namestitev dodatnih virov napajanja, zato dodatno ne obremenjuje okolja in ne ustvarja odvečnih elektronskih odpadkov. Emulator je hitrejši, učinkovitejši in popolneje emulira originalen procesor od obstoječih verzij, v primerjavi z njimi je tudi energijsko varčnejši. Znanja željnim posameznikom pa omogoča lažje učenje arhitekture osnovnega mikroprocesorja in njegovega zbirnega jezika.

8 ZAKLJUČEK

Projekt sem uspešno zaključil. Dosegel sem tudi vse cilje, ki sem si jih zastavil. S tem projektom sem se tudi sam naučil veliko o delovanju mikroprocesorjev. Naučil sem se tudi programirati v programskem jeziku C++.

Menim, da bo emulator uporabno orodje vsem, ki se bodo želeli učiti zbirnega programskega jezika. Mislim tudi, da bo uporaben učiteljem kot praktični primer arhitekture in delovanja mikroprocesorja.

9 VIRI

Priročniki:

1. M6800 Microprocessor Applications Manual (MOTOROLA INC., 1975)
2. M6800 Programming reference manual (MOTOROLA INC., 1976)

Elektronski viri:

3. "Assembly language". https://en.wikipedia.org/wiki/Assembly_language#Assembler (pridobljeno 9. 12. 2023)
4. "Emulator". https://sl.wikipedia.org/wiki/Emulator (pridobljeno 10. 1. 2024)
5. "Emulator". https://en.wikipedia.org/wiki/Emulator (pridobljeno 10. 1. 2024)
6. "Microprocessor". https://en.wikipedia.org/wiki/Microprocessor (pridobljeno 25. 1. 2024)
7. "Motorola 6800". https://en.wikipedia.org/wiki/Motorola_6800 (pridobljeno 25. 1. 2024)
8. "HyperVision Research". http://www.hvrsoftware.com/ (pridobljeno 10. 1. 2024)
9. "m6800 Simulator". https://mypluribus.github.io/m6800/ (pridobljeno 10. 1. 2024)