**»Mladi za napredek Maribora 2023«**

**40. srečanje**

Emulator MikroprocesorjEV Motorola M680X

Raziskovalno področje: informatika.

inovacijski predlog

KAZALO VSEBINE

[1 Uvod 1](#_Toc157272784)

[2 Načrt in priprave 2](#_Toc157272785)

[2.1 Osebna zgodovina progrmairanja. 2](#_Toc157272786)

[2.2 Kaj je Motorolin M6800 mikroprocesor? 2](#_Toc157272787)

[2.3 Zakaj emulator mikroprocesorja M6800? 2](#_Toc157272788)

[2.4 Načrt 3](#_Toc157272789)

[2.5 Iskanje primernega programskega okolja 5](#_Toc157272790)

[3 RAZVOJ 6](#_Toc157272791)

[3.1 Razvoj prve različice emulatorja. 6](#_Toc157272792)

[3.2 Nadaljnji razvoj 8](#_Toc157272793)

[4 Končni izdelek 10](#_Toc157272794)

[4.1 Uporabniški vmesnik 10](#_Toc157272795)

[4.1.1 Števec vrstic 10](#_Toc157272796)

[4.1.2 Polje za vpis assembly kode 11](#_Toc157272797)

[4.1.3 Polje za prikaz spomina 12](#_Toc157272798)

[4.1.4 Registri 14](#_Toc157272799)

[4.1.5 Stran z večmi zavihki 14](#_Toc157272800)

[4.1.6 Gumbi in kontrole 18](#_Toc157272801)

[4.2 Notranje komponente emulatorja. 19](#_Toc157272802)

[4.2.1 Assembler(Sestavljavec) 19](#_Toc157272803)

[4.2.2 Izvajanje ukazov 20](#_Toc157272804)

[4.2.2.1 Funkcija executeInstruction 20](#_Toc157272805)

[4.2.2.2 Izvedba posameznega ukaza 20](#_Toc157272806)

[4.2.2.3 Samodejno izvajanje ukazov 21](#_Toc157272807)

[4.2.2.3.1 Funkcija startExecution 21](#_Toc157272808)

[4.2.2.3.2 Funckija stopExecution 25](#_Toc157272809)

[4.2.2.3.3 Funkcija stopUiUpdateTimer 25](#_Toc157272810)

[4.2.2.3.4 Posodabljanje UI med samodejnim izvajanjem 25](#_Toc157272811)

[4.2.3 Razstavljavec 26](#_Toc157272812)

[5 Primerjava in potrditev napredka 27](#_Toc157272813)

[5.1 Evalvacija lastnega emulatorja 27](#_Toc157272814)

[5.2 Motorola 6800 Simulator različica 1.33p 2.cR 27](#_Toc157272815)

[5.3 SDK6800/6811 Emulator v1.14 (www.HVRSoftware.com) 28](#_Toc157272816)

[5.4 Ostale alternative 30](#_Toc157272817)

[7 ZAKLJUČEK 31](#_Toc157272818)

[8 VIRI 31](#_Toc157272819)

**Povzetek**

To je inovacijski predlog, ki opisuje razvoj in delovanje emulatorja za mikroprocesorje Motorola M68XX. Ta emulator omogoča uporabnikom programiranje in uporabljanje mikroprocesorjev M6800 in M6803. Po primerjavi z podobnimi izdelki sem zaključil da moj emulator predstavlja veliko bolj tehnološko izpopolnjeno različico emulatorja M6800. Ima manj napak, je hitrejši, robustnejši in vsebuje veliko več funkcij kot njegovi konkurenti.

# Uvod

Veliko programerjev ima eno temeljno težavo. Ta je, da se jim ne sanja 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, omejene izraznosti, platformne specifičnosti ter 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 enostavnem, a robustnem procesorju. Motorolin mikroprocesor M6800 vključuje vse osnove modernih procesorjev, kot so podpora sklada, indeksni registri in podobno. Njegova preprosta struktura omogoča lažje razumevanje delovanja procesorja v primerjavi z novejšimi procesorji, ki imajo več registrskih in kompleksnejših komponent. Znanje, ki ga uporabnik pridobi z izkušnjami na takem procesorju pa se z lahkoto prenese na modernejše procesorje in višje programske jezike.

Zaradi tega je namen inovacijskega predloga razvoj Emulatorja mikroprocesorja M6800, ki bo programerjem, katerim to znanje manjka, ponudil robustno okolje za učenje. V pomoč bo tudi tistim, kateri želijo delati s M6800, a tega procesorja v živo nimajo. Emulator se tudi lahko uporabi kot razhroščevalnik za prave M6800 procesorje.

Inovacijski predlog opisuje razvoj emulatorja za mikroprocesor 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.

# Načrt in priprave

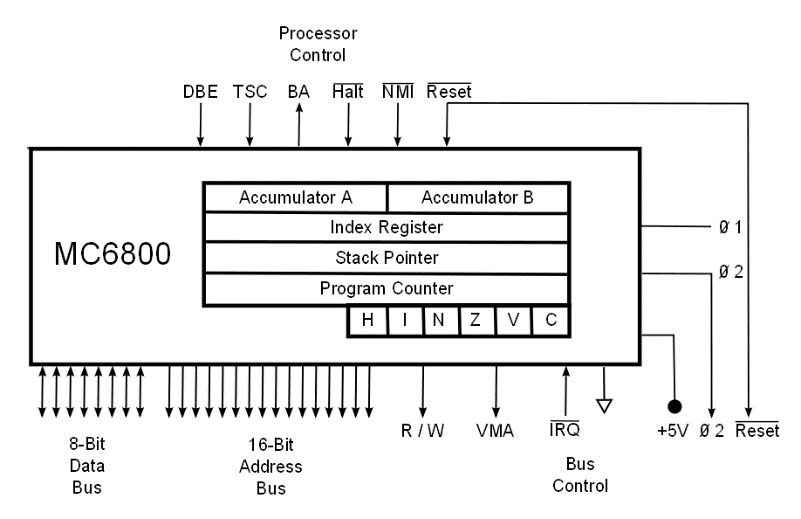
## Osebna zgodovina progrmairanja.

Pred začetkom tega projekta sem kot programer imel izkušnje z programskimi jeziki C#, C in python. C# sem se naučil s programskim orodjem Unity. Tam sem naredil nekaj prototipov videoiger. S pythonom sem delal nekaj orodij za avtomatizacijo vsakdanjih opravil. Imel sem izkušnje tudi z mikroprocesorjema Arduino in Raspberry Pi.

## Kaj je Motorola M6800?

M6800 je 8-bitni mikroprocesor iz družine »M6800 Microcomputer System«(katere vzdevek je M68XX). Motorola ga je zasnovala in prvič izdelala leta 1974.

Procseor 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 i izvedbi prejšnjega ukaza. Najbolj utežena bita tega registra sta neuporabljena in sta vedno imata vrednost 1. Zastavice pa so po vrsti od najbolj utežene do najmanj utežene: zastavica pol-prenosa(H), zastavica prekinitvene maske(I), zastavica negativnega predznačenega števila(N), zastavica nič(Z), zastavica prekoračitve(V), zastavica prenosa(C).



Slika 1: Prikaz registrov in nožic M6800

Ta procesor podpira 6 načinov naslavljanja. Te so: vsebovano, takojšnje, neposredno, razširjeno, indeksno in relativno. Način naslavljanja je način, kako naj računalnik pride do operanda, nad katerim naj se izvede ukaz.

## Zakaj emulator mikroprocesorja M6800?

V primerjavi z ostalimi je ta procesor po arhitekturi med preprostejšimi. Zato ima velik potencial za učenje programiranja katerega koli nižje nivojskega programskega jezika. Primeren je tudi začetnikom in tistim katerim svet nižjih programskih jezikov ni poznan. Osebno mi je motivacijo za razvoj novega emulatorja dala tudi izkušnja z tem emulatorjem v šoli. Tam smo uporabljali zastarel in neučinkovit emulator. Ta izkušnja me je navdihnila, da razvijem boljšo rešitev in omogočim boljšo izkušnjo pri delu s tem procesorjem.

## Načrt

Projekt sem začel z grobo skico izgleda emulatorja. Odločil sem se, da bo uporabniški vmesnik podoben emulatorju »SDK6800/6811 Emulator« opisanem v poglavju 5.3. Smiselno je tudi, da so vedno vidna najpomembnejša polja, kot so polje za prikaz spomina ter registrov, polje za vpis kode in gumbe, ki nadzirajo osnovne funkcije emulatorja.

Ker obstaja že nekaj emulatorjev za M6800, sem si postavil kriterije katere moj emulator mora doseči ali presegati. S temi bi potrdil, da je moj emulator boljši od ostalih na vsaki ravni.

1. Emulator mora vsebovati brezhiben sestavljavec. Sestavl javec rabi poznat vsak ukaz, ki ga podpira mikroprocesor M6800. Prepoznati mora vse oblike naslavljanja. Če je v kakem primeru možnih več, mora uporabiti najučinkovitejšo. To se lahko pojavi pri primeru, kjer ukaz podpira neposredno in razširjeno naslavljanje in njegov operand ne presega 255.
2. Učinkovit prikaz spomina in drugih notranjih elementov procesorja, ki omogočajo hiter in razumljiv dostop do podatkov.
3. Ročno urejanje spomina in razstavljavec. Da bi se uporabnik lažje naučil kako deluje sestavljanje ukazov, sem si zadal nalogo, da omogočim uporabniku da ročno piše strojno kodo. Ker je ročno pisanje strojne kode zahtevno, bom uporabniku podal tudi orodje za razstavljanje ukazov. S tem lahko preveri in razhrošči svojo strojno kodo.
4. Originalnemu procesoru popolnoma enako izvajanje. Vsak ukaz se mora obnašati enako kot pri avtentičnem procesoru. Delovati morajo tudi prekinitve in ciklično izvajanje. To vključuje tudi hitrost izvajanja, ki je v originalnem mikroprocesorju 1Mhz.
5. Orodja za razhroščevanje. Emulator rabi vsebovati vsa potrebna orodja za razhroščevanje.
6. Uporaba vhodnih in izhodnih naprav. Možnost interakcije s emuliranem mikroprocesorjem na bolj dinamičen način, omogoča zanimivejšo izkušnjo, saj namesto statičnega algoritma, ki izvaja eno funkcijo, bi bila možna komunikacijoa s perifernimi napravami. To omogoča ustvarjanje bolj dinamičnih in interaktivnih programov, ki odražajo resnično uporabo procesorja.
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

## Iskanje primernega programskega okolja

Programsko okolje je program, ki ima implementirane funkcije kot so urejevalnik izvorne kode, prevajalnik, povezovalnik, razhroščevalnik in razna orodja za preverjanje in izboljšave delovanja programa. Nekaj znanih programskih okolij za razvoj namiznih aplikacij so WinForms, Qt, JavaFX, WPF, Xamarin, Swift.

Osebno sem delal na projektih, ki so prevladujoče uporabljali programski jezik c#. Zaradi tega sem iskal okolja, ki uporabljajo c# kot primarni programski jezik.

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

Napisal sem nekaj funkcij, ki uporabniku omogočajo urejanje kode in funkcijo ki prevede/sestavi ukaze v strojno kodo. Dodal sem tudi funkcijo ki izvršuje ukaz. Ta poskusna različica emulatorja je podpirala le nekaj osnovnih ukazov kot so »LDA« in tele samo pri vsebovanem naslavljanju.

A screenshot of a computer

Description automatically generated

Slika 2: Poskusna različica emulatorja v okolju WinForms

Slika 3: Uporabniški vmesnik emulatorja razvit v WinFormsu

To različico in njeno okolje sem kmalu zapustil saj se je izkazalo da je WinForms zelo počasen pri spreminjanju vrednosti v elementih kot so registri in spomin.

Projekt sem kmalu obudil v novem okolju za katerega sem predpostavljal da bom zapolnil pomanjkljivosti WinFormsa. Imenuje se WPF ali »Windows Presentation Foundation«. Žal se je tudi to okolje izkazalo za moje potrebe prekompleksno in prepočasno. Datoteke povezane z tem projektom sem izgubil.

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

# Razvoj

## Razvoj prve različice emulatorja.

Odločil sem se da bom emulator razvil modularno in to tako, da bom začel z uporabniškim vmesnikom. Nato bom razvil sestavljavec. Potem se bom lotil izvajanja ukazov in drugih komponent emulatorja. Pravilnost sestavljanja sem preverjal s pomočjo dokumenta »M6800 Programming reference manual«, ki je bil izdan leta 1976. Največ težav pri prevajanju sem imel z nazivi. Ukaz ali vrstica v sestavljanju z M6800 lahko ima naziv(label). Nek drug ukaz lahko kliče ta naziv, ki vrne neko pomnilniško lokacijo ali vrednost. Težava se pojavi ko nek ukaz kliče naziv ki še ni bil definiran. Primer:

A screenshot of a computer

Description automatically generated

Slika 4: Primer ukaza, ki kliče nedefiniran naziv

Ukaz »JMP« ali skok ima za operand »label«. Ko se ta koda prevaja bo v strojni kodi začasno bil operand prazen. Ko se pa enkrat vsi ukazi zapišejo in se njihovi nazivi shranijo, pa prevajalnik znova gre skozi vse ukaze ki so klicali nazive in v prazna mesta operandov zapiše ustrezno vrednost.

Slab mesec dni sem porabil na vplejavi izvajanja ukazov. Rabil sem prebrat veliko dokumentacije in skozi čas se je sistem izvajanja mnogo krat spremenil. Še sploh pri posodabljanju UI(uporabniški vmesnik) elementov. V prvih različicah je vsak ukaz javil emulatorju kateri UI elementi rabijo posodobitev. Pri implementaciji časovno enakomernega samodejnega izvajanja ukazov sem imel veliko težav zaradi neizkušenosti z takim projektom. Začasno sem ta del emulatorja rešil z Qt c++ razredom QTimer. QTimer razred omogoča uporabo programskega časovnika, kateri se lahko z »connect« funkcijo poveže z neko drugo funkcijo, katero je zaželeno izvajati na določenem intervalu. Težava se pojavi pri omejitevi QTimera, da je najmanjši možen interval 1ms. To pomeni, da bo teoretično najhitrejša hitrost 1000 ukazov na sekundo.

V prevajalnik sem vključil tudi nekaj sestavljalnih direktiv. To so navodila, ki jih sestavljalcu podamo, da nadzorujemo ali prilagajamo proces sestavljanja. Moj emulator podpira naslednje:

1. .ORG: Določi izhodiščni ali začetni naslov programa ali odseka. To se zgodi tako da sestavljalec sestavi sledeče ukaze na določenem pomnilniškem naslovu. V prekinitveni vektor za ponastavljanje procesorja vstavi podan naslov.
2. .RMB: Rezervira spomin oziroma preskoči pomnilniške lokacije med sestavljanjem.
3. .EQU: Nazivu pod katerim je klican .EQU je prirejena vrednost katero določi uporabnik. V primeru: »test .EQU 24« bo naziv test vseboval število 24.
4. .BYTE: V trenutnem naslovu sestavljanja se shrani en ali več zaporednih bajtov. Primer:

A white background with black text

Description automatically generated

Slika 5: Primer uporabe ukaza ».BYTE«

1. .WORD: V trenutni naslov sestavljanja se shrani ena ali več besed(2 bajta).
2. .SETB: V podan naslov shrani podan bajt. Primer: ».SETB $FB00,24«.
3. .SETW: V podan naslov shrani besedo.
4. .STR: Shrani niz ali več nizov v trenutni naslov. ».STR "primer1","primer2"«

Dodal sem tudi konzolo ki kaže napake, opozorila in podobno ter polje, ki označuje vrstice v polju za vpis kode. Ustvaril sem tudi nekaj osnovnih nastavitev.

Nato sem se lotil programiranja delovanja procesorja M6800. Dodal sem vse registre in okenca ki jih prikazujejo ter gumbe, ki upravljajo s procesorjem. Dodal sem tudi gumbe, ki kodo shranjujejo in nalagajo.

Ko sem dodal še možnost direktnega urejanja spomina sem zraven dodal še sistem označevanja ukazov, ki označijo napake, naslednji ukaz, zabeležene ukaze ter lokacijo urejanja spomina.

A screenshot of a computer

Description automatically generatedPo veliko optimizacije in prilagajanja uporabniškega vmesnika sem dodal tudi ukaze ter sestavljavec za procesor M6803. Začel sem tudi implementirat zaslon, ki je viden ko je okno dovolj široko.

Slika 6: Ena izmed prvih delujočih različic emulatorja

## Nadaljnji razvoj

Nadaljnji razvoj je imel velik poudarek na optimizaciji ter odpravljanju napak. Dodal sem tudi enostavnejšo različico prikaza spomina, ki se vključi v nastavitvah.

A screenshot of a computer

Description automatically generated

Največji dodatek je bil uvedba razstavljanja. To omogoča uporabniku da iz strojne kode razstavi ukaze v assembly jezik. Ta funkcija je dostopna ko je način pisanja nastavljen na spomin(direktno pisanje v spomin). Ko se disassembler zažene bo uporabnik vprašan kje se program začne, to omogoča uporabo prvih pomnilniških naslovov za spremenljivke. Podatke pred tem bo disassembler zapisal z ukazom ».BYTE«. 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«.

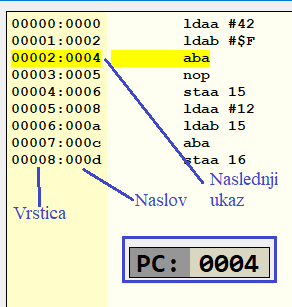
Še en omembe vreden dodatek je uvedba možnosti da procesor deluje po hitrosti »ciklov« na sekundo. Tako deluje tudi pravi procesor. Vsak ukaz zahteva nekaj ciklov procesorja. Privzeta nastavitev je še vedno ukazi na sekundo, saj je tako začetnikom enostavnejše upravljati z hitrostjo programa. Ta funkcija se je skozi čas veliko spreminjala saj sem imel težave z enakomernostjo izvajanja in maksimalno hitrostjo.

Ko sem začel implementirati izvajanje ukazov v emulator, sem si dal ambiciozen cilj da bi emulator lahko izvajal s enako hitrostjo kot avtentičen M6800 mikroprocesor, to je frekvenca 1 MHz ciklov ali en milijon ciklov na sekundo. Proti koncu razvoja sem to staro ogrodje uravnavanja hitrosti zavrgel in znova začel načrtovati sistem za urejanje hitrsoti. V noven sistemu sem vplejlal uporabo večnitnosti. Definiral sem da se na dodatni niti dogaja vse, kar je povezano s samodejnim izvajanjem ukazov. Posodobitve uporabniškega vmesnika in vse ostalo pa se bo dogajalo na glavni niti programa. Če je hitrost izvajanja dvakrat večja kot hitrost posodabljanja zaslona(privzeto 256 posodobitev na sekund) se bodo izvajale zaporedne serije ukazov ali ciklov. Ta sistem deluje tako, da je časovno in navidezno neopazno, da se ukazi v ozadju ne izvajajo en za drugim z stalnim intervalom. Po vsakem zadnjem ciklu ali ukazu v paketu se v glavno nit pošlje zahteva za posodobitev zaslona. Delovanje nove različice te funkcije je podrobnejše opisano v poglavju 5.2.2.3.

# Končni izdelek

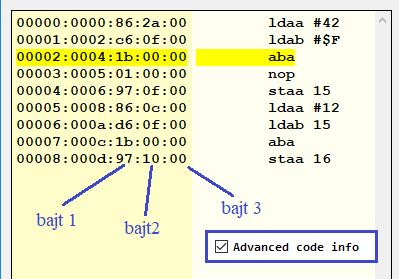
## Uporabniški vmesnik

### Števec vrstic

Na levi strani glavnega okna je polje namenjeno štetju vrstic assembly programa. Privzeto prikazuje dva podatka ločena s dvopičjem. Levo število je število vrstice kjer je napisan ukaz v polju za vpis assmebly kode. Če je program trenutno sestavljen, bo na desni šestnajstiško število, ki uporabniku sporoča pomnilniški naslov, kjer je shranjen prvi bajt sestavljenega ukaza. Ukaz na katerega trenutno kaže programski števec bo obarvan rumeno. V primeru prikazanem na priloženi sliki 2, je to vrstica dve saj ima PC(programski števec) vrednost štiri in ukaz na pomnilniški lokaciji 4 je ABA.

Slika 7: Element uporabniškega vmesnika: Števec vrstic

V primeru da je vklopljena nastavitev »napredne informacije kode« pa bodo ob pomnilniški lokaciji bili izpisani vsi bajti v katere se ukaz na ustrezni vrstici sestavi 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 8: Element uporabniškega vmesnika: Števec vrstic - napredne informacije kode

Ko je koda sestavljena, 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 z ponovnim klikom na označeno vrstico. Vse oznake se izbrišejo ob desnem kliku miške in ob spremembi kode.

A screenshot of a computer

Description automatically generated

Slika 9: Element uporabniškega vmesnika: Števec vrstic - primer oznak

### Polje za vpis assembly kode

A screenshot of a computer

Description automatically generated Ob števcu vrstic je polje za vpis assembly kode. /////Vpis v polje je mogoč, ko je emulator nastavljen na način vpisa kode. Desni klik na to polje bo ob navadnem meniju uporabniku podal še eno možnost »informacije ukaza«, ki bo prikazala opis ter vse ostale informacije o ukazu nad katerim je kursor v okencu za dialog.

### Polje za prikaz spomina

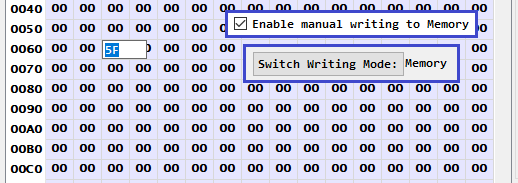
Polje za prikaz spomina, ki je na sredini glavnega okna v podobi tabele prikazuje vse pomnilniške celice ter vrednosti ki jih hranijo. Privzeto so celice prikazane v šestnajstiškem številskem sistemu možna pa je nastavitev na desetiški številski sistem.

A screenshot of a computer

Description automatically generated

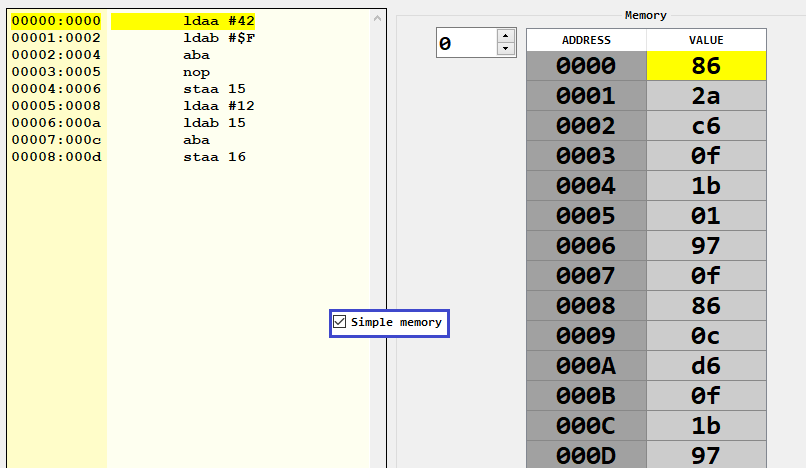
Slika 10: Element uporabniškega vmesnika: Prikaz spomina

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



Slika 11: Element uporabniškega vmesnika: Prikaz spomina – sprememba celice pri načinu pisanja v spomin

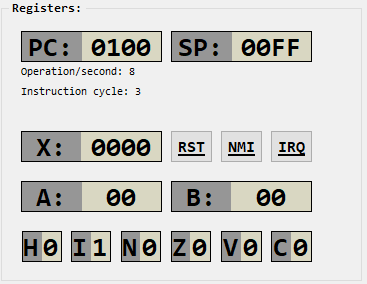
Obstaja enostavnejši prikaz spomina, ki je uporaben ko je pomembno pozorno spremljanje sprememb nekaj celic ali pa če je uporabniku privzeti prikaz spomina preveč zapleten. Inspiracijo za ta prikaz sem dobil iz prej omenjenega emulatorja »Motorola 6800 Simulator«. Le da v tistem primeru tak prikaz deluje 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 12: Element uporabniškega vmesnika: Prikaz spomina – enostaven spomin

### 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 sistemu.



Slika 13: Element uporabniškega vmesnika: Prikaz registrov

### Stran z večmi zavihki

Desno spodaj je stran za izbiro večih zavihkov.

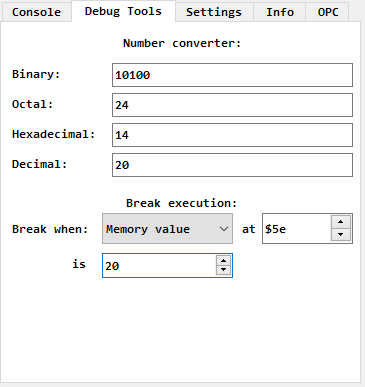
1. Prvi je okno konzole ki prikazuje napake, opozorila in informacije ki jih emulator posreduje uporabniku.

A screenshot of a computer

Description automatically generated

Slika 14: Element uporabniškega vmesnika: Konzola

1. Drugi vsebuje orodja za odpravljanje napak kode(razhroščevanje). Vsebuje prevajalnik številskih sistemov ter sistem prelomnih točk, ki uporabniku omogočajo samodejno ustavljanje izvajanja ukazov ob določenem dogodku ali pogoju. Na primer, program se lahko ustavi če po kateri register vseboval zaželeno vrednost, lahko se ustavi na določeni vrstici/ukazu ali pa, ko neka celica v spominu vsebuje zaželeno vrednost.



Slika 15: Element uporabniškega vmesnika: Razhroščevalna orodja

1. 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 sestavi ločene z dvopičjem.
   2. Nastavitev, ki določa v kakšnem številskem sistemu bodo napisane vrednosti registrov in spominskih celic.
   3. Nastavitev, ki vklopi samodejno sestavljanje v primeru, da je program zagnan po spremembi ukazov.
   4. Nastavitev, ki določa, če bo se 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, katere rabi posamezen ukaz. 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 bil klican po pritisku gumba samo če je procesor v stanju čakanja na odziv na WAI ukaz.
   8. Nastavitev, ki določa če bo uporabniku prikazan standardni prikaz spomina ki ima 16 pomnilniških naslovov in njihovih vrednosti v eni vrstici, ali enostavnejši prikaz ki prikazuje 20 poljubnih zaporednih pomnilniških naslovov in vrednosti katere vsebujejo.
   9. Nastavitev, ki omogoča nastavljanje delovanja emulatora med pisanjem kode ter neposredno v spomin, to posledično tudi določa če bo emulator kodo ali ukaze sestavljal ali razstavljal. Na spodnjem delu glavnega okna se pojavi gumb ki prikazuje trenutni način delovanja emulatorja ter ponuja uporabniku da ga zamenja. Privzeta nastavitev je da deluje na način sestavljanja, to je da uporabnik piše ukaze v zbirnem jeziku ter jih prevajalec sestavi v strojno kodo, druga nastavitev je način razstavljanja, takrat je uporabniku omogočeno spreminjanje strojne kode ter razstavljanje spomina, to je da se strojna koda v spominu prevede v assembly ukaze ki se bodo izpisali v polju za assembly program.
   10. Nastavitvi za samodejno premikanje polja za vpis kode na ukaz kateri 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 polju za spomin in polju 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 z potrebnim številom »tab« znakov.

A screenshot of a computer program

Description automatically generated

Slika 16: Element uporabniškega vmesnika: Nastavitve

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

A screenshot of a computer

Description automatically generated

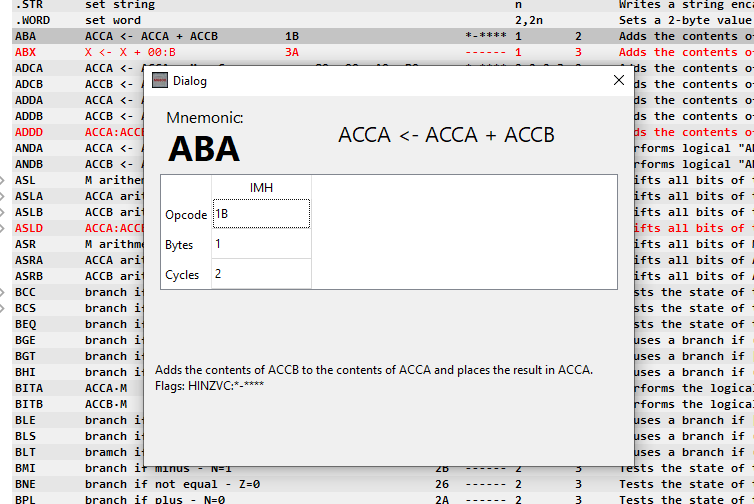
Slika 17: Element uporabniškega vmesnika: Informacije

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

A screenshot of a computer

Description automatically generatedKlik na ukaz bo odprl okence, ki bolj pregledno prikazuje opis in vse ostale informacije ustreznega ukaza

Slika 18: Element uporabniškega vmesnika: zavihek ukazov



Slika 19: Okno informacij ukaza ABA

### Gumbi in kontrole

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

Slika 20: Element uporabniškega vmesnika: Gumbi in kontrole

Ti so:

1. Gumb »Sestavi/Razstavi« v načinu pisanja »pisanje kode« bo ukaze, ki so trenutno napisani v polju za pisanje ukazov sestavil in prevedel v strojno kodo. Če pa je način pisanja nastavljen na pisanje v spomin, bo ta gumb razstavil strojno kodo zapisano v spominu ter 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 tako zasnovan da omogoča nadaljnjo razširitev ter dodajanje novih procesorjev. Emulator bo se ravnal, sestavljal razstavljal ter izvajal ukaze glede na izbran procesor.
3. Gumb za nalaganje, bo glede na izbiro načina pisanja iz zunanje datoteke naložil assembly kodo ali pa spomin.
4. Gumb za shranjevanje, bo glede na izbiro načina pisanja shranil assembly kodo ali pa trenutni spomin v datoteko.
5. Gumb za zamenjavo načina pisanja. Privzeto je ta gumb nedosegljiv, lahko se pa prikaže s nastavitvijo »dovoli zamenjavo načina pisanja«. Desno od tega gumba je izpisan trenutni način pisanja.
6. Gumb za ponastavitev ponastavi emulator na stanje kakšnega je bil po zadnjem sestavljanju ali pred zadnjim izvajanjem.
7. Gumb »korak« ki izvede en ukaz z 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 ter nastavitvijo za izbiro načina izvajanja. 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 bo se 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.

## Notranje komponente emulatorja.

### Assembler(Sestavljavec)

Assembler je programska oprema, ki prevaja ukaze, operacije, načine naslavljanja ter operande v njihove številske vrednosti(strojno kodo) določenega assembly jezika. Kot mnogi drugi assemblerji, uporablja in prepozna simbolične reference. To je način poimenovanja določenih pomnilniških lokacij ali ukazov, katere uporaba nadomesti ročne izračune in posodobitve pomnilniških naslovov po spremembah programa ter olajšajo uporabo spremenljivk. Moj assembler za M680X je »One-pass assembler«, to pomeni da gre skozi kodo samo enkrat in rezervira prostor za simbole, ki še niso bili definirani. Na koncu pa assembler preide skozi rezervirane pomnilniške naslove in jih nadomesti s vrednostjo nazivov.

Moj sestavljalec deluje tako da gre v zanki skozi vsako vrstico kode. Za vsako vrstico si naredi kopijo katero nato manipulira, krajša in si jo pripravlja da jo lahko razume. Ta proces ima v glavnem 3 dele. Na začetku odstrani vse komentarje(komentarji se začnejo s podpičjem) . Nato pride v prvi del, ki izvleče naziv ukaza, če je podan. Njegov obstoj preveri tako da preveri, če je prvi znak vrstice črka. Nato s pomočjo variable »charNum« preglada vrstico in zabeleži kdaj se pojavi znak, ki ni črka, število ali podčrtaj. Nato preveri še nekaj samoumevnih stvari, na primer, če je vrstica prazna, če so v nazivu 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 in zadnjem delu pa identificira in prevede ukaz. Na podlagi operanda ugotovi katero naslavljanje je primerno in če je operand za tak ukaz primeren. Če je operand že definiran naziv bo v lokacijo operanda vstavljena njegova vrednost. V primeru da pa operand še ni definiran pa bo ukaz zapisan z največjim primernim naslavljanjem in prostor operanda bo ostal prazen. Sestavljavec ima tri sezname, ki beležijo nedefinirane nazive in lokacije v katere sodijo. Nato, če se vsaka vrstica prevede brez težav bo emulator pregledal vse sezname in zapolnil manjkajoča mesta s vrednostmi, katere ti nazivi vsebujejo. Na koncu se preveri če je šlo vse po maslu in emulator si zabeleži da je zadnje sestavljanje bilo uspešno ter to sporoči tudi uporabniku. V primeru sintaktične napake bo sestavljanje neuspešno, pomnilnik bo ponastavljen in vrstica kjer se je napak pojavila bo označena, če je težava v specifičnem znaku vrstice bo označen tudi ta znak. Opis napake pa bo zapisan v konzoli.

### Izvajanje ukazov

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

#### 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. S 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 PC se inkrementira.
2. Operacijska koda 0: Če je funkcija bila klicana z samodejnim izvajanjem bo se samodejno izvajanje ustavilo, drugače bo se PC inkrementiral.
3. Operacijske kode ne podpira izbrana različica procesorja: Če je šesta nastavitev vklopljena bo se PC inkrementiral. Če pa je ta nastavitev izklopljena in je funkcija bila klicana z samodejnim izvajanjem bo se samodejno izvajanje ustavilo.

#### Izvedba posameznega ukaza

S gumbom »korak« bo emulator izvedel en ukaz. 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 bo se izvršil en korak procesorja. Korak ni en procesorski cikel, ampak je le en ukaz ali odziv na prekinitev. Potek izvajanja koraka je podoben izvajanju funkcije executeInstructionTick prikazane v diagramu slike 17. Edina razlika je, da se ta del programa konča pred preverjanjem prekinitvenih 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.

#### Samodejno izvajanje ukazov

Samodejno izvajanje se zažene in ustavi s 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, če je hkrati operacijska koda naslednjega ukaza enaka 0 in je vklopljena nastavitev »samodejno ponastavljanje«. Takrat bo se emulator ponastavil. Potem bo klicana funkcija startExecution.

##### 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. Na zadnje zažene novo nit, ki je namenjena neprestanemu izvrševanju ukazov. To sem dosegel s 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 z kosntantnim intervalom. Ko sem pa želel dodati večje hitrosti sem ugotovil, da zaradi prestopov med iteracijami v zanki in nenehnega merjenja časa na nanosekunde natančno je 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 z serijskim izvajanjem podpirajo tudi dejstva, da večina zaslonov podpira le 60 posodobitev na sekundo in višja meja za povprečnega uporabnika bi bila 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 spremeljivki 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 če je trenuten čas večji od te spremenljivke in če je temu res, izvede naslednjo serijo ukazov ali ciklov. Izveda koraka se precej razlikuje v načinu izvajanja po ciklih in po ukazih. To delovanje je prikazano v diagramih na Slikah 15, 16 in 17.

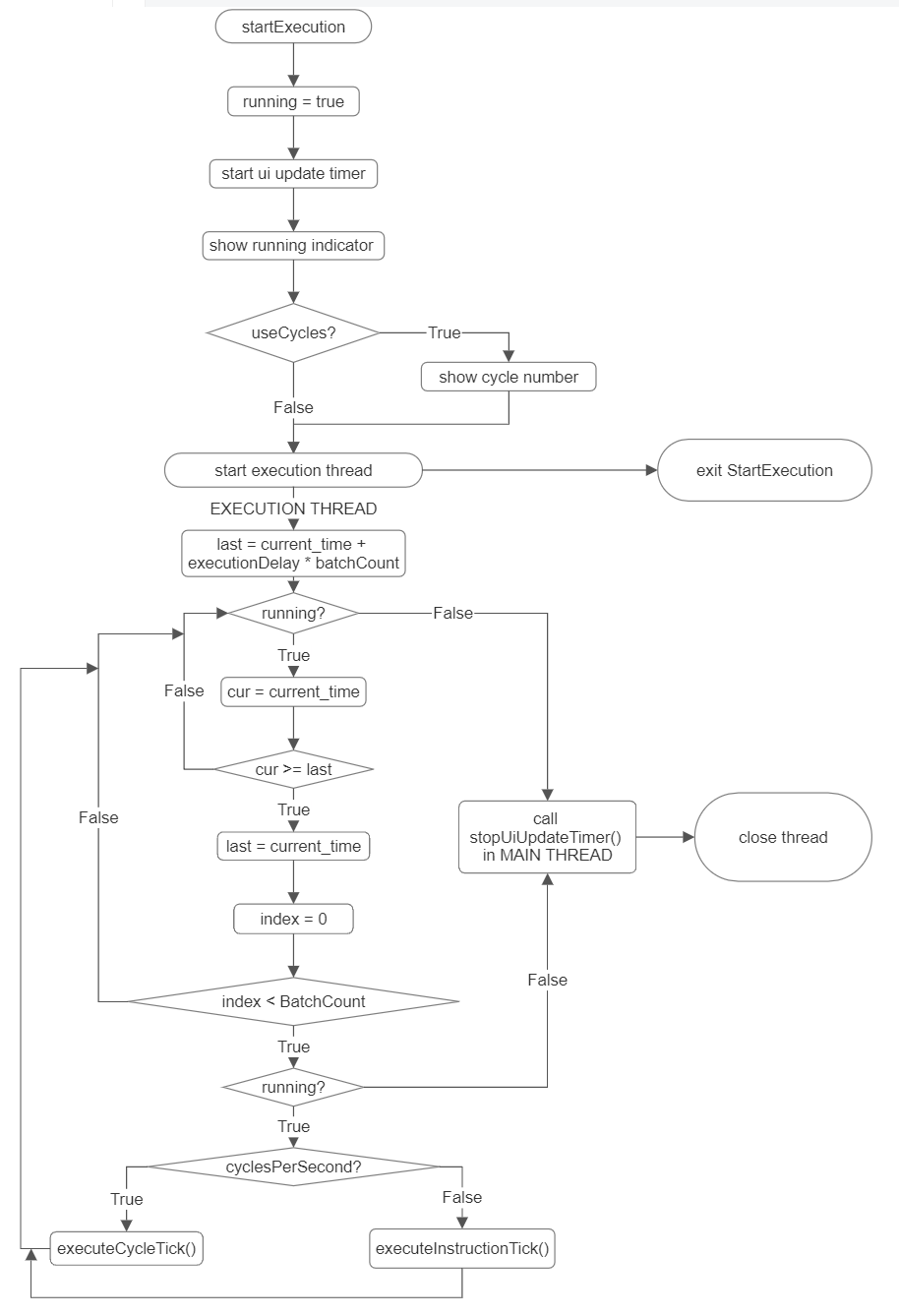
Po izvedbi koraka ali odziva na prekinitev se preverijo stanja razhroščevalne prekinitve. Te uporabnik nastavi v zavihku za razhroščevanje. Če stanje procesorja ustreza danemu pogoju, bo spremenljivka running bila nastavljena na 0, kar bo povzročilo da bo se samodejno izvajanje ustavilo.

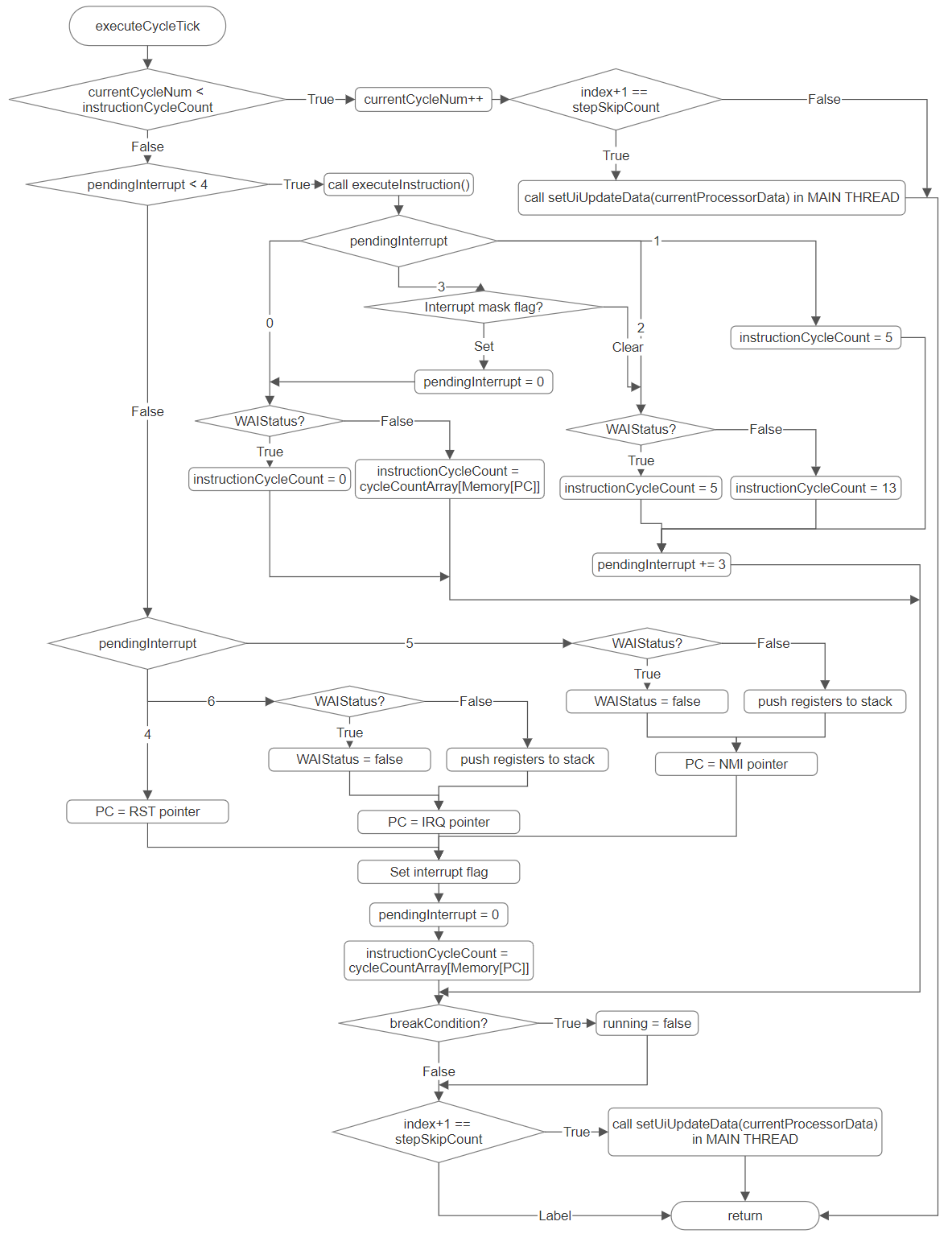
Tabela 1: Časi serijskega izvajanja

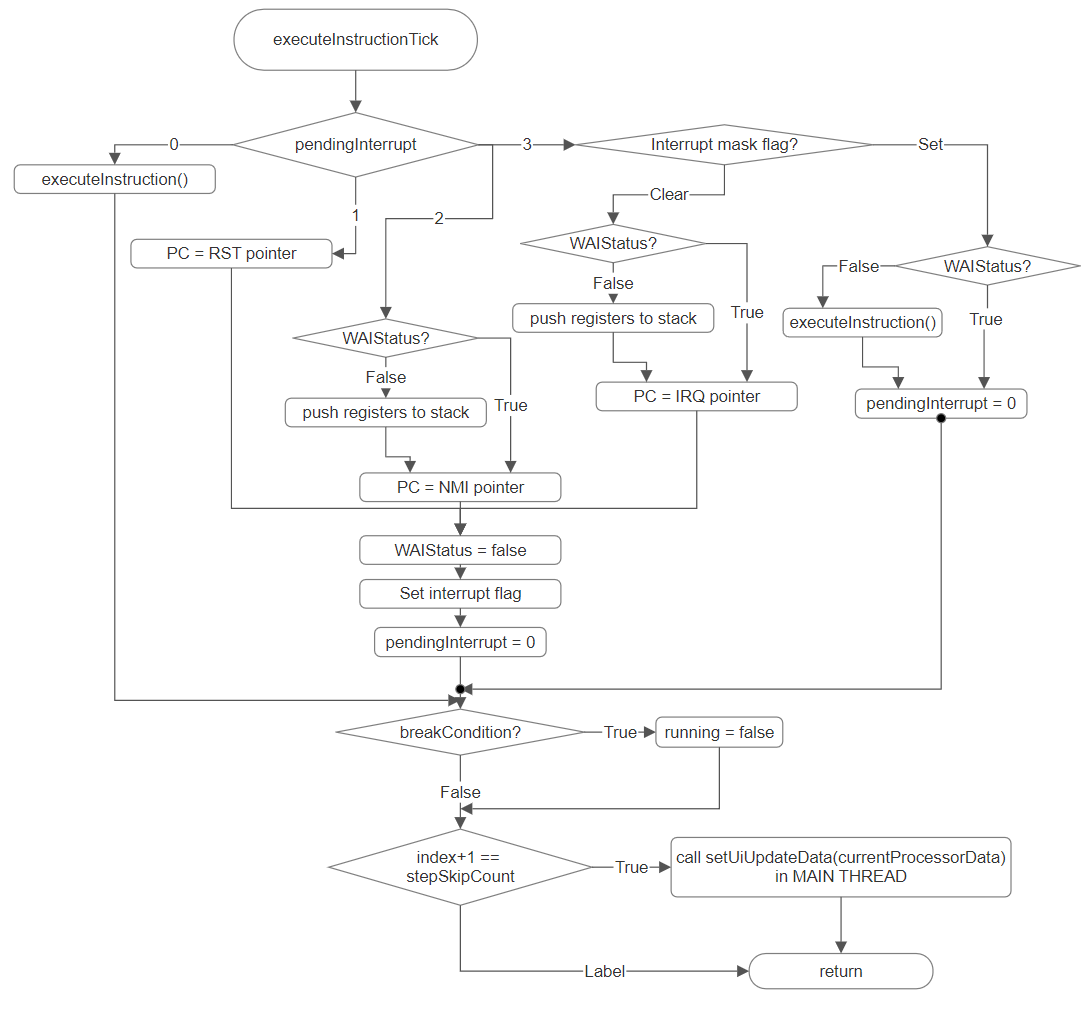
|  |  |  |  |
| --- | --- | --- | --- |
| Cikli na sekundo | Cikli v seriji | Časovni zamik med cikli | časovni zamik med serijami |
| 1 | 1 | 1000000000ns | 1000000000ns |
| 2 | 1 | 500000000ns | 500000000ns |
| 4 | 1 | 250000000ns | 250000000ns |
| 8 | 1 | 125000000ns | 125000000ns |
| 16 | 1 | 62500000ns | 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 v iz izvrševalne niti poslano trenutno stanje procesorja v glavno nit. Ob naslednjem utripu časovnika(QTimer) bo se v glavni niti posodobil uporabniški vmesnik na zadnje stanje, ki ga je glavna nit prejela iz izvajalne niti.

Spodaj so podani diagrami poteka, ki podrobnejše opisujejo delovanje funkcije »startExecution«.



Slika 21: Diagram poteka: funkcija StartExecution

Slika 22: Diagram poteka: funkcija StartExecution: executeCycleTick

Slika 23: Diagram poteka: funkcija startExecution: executeInstructionTick

##### Funckija 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 bo se zanka v funkciji startExecution končala in klicala funkcijo »stopUiUpdateTimer« v glavni niti.

##### 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.

##### 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, če je od zadnjega utripa bila posodobljeno stanje procesorja. V takem primeru pa posodobi vse elemente UI na prejeto stanje procesorja.

### Razstavljavec

V emulator sem vključil tudi primitiven razstavljavec. To je program, ki strojno kodo prevede oz. razstavi v assembly ukaze. Ko je emulator v načinu pisanja v spomin, bo namesto gumba »Sestavi« prikazan gumb »Razstavi«. 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«.

# Primerjava in potrditev napredka

## Evalvacija lastnega emulatorja

Menim, da moj emulator dosega in presega vse prej zastavljene kriterje.

1. Brezhiben sestavljavec. Moj emulator brezhibno in učinkovito sestavlja ukaze za M6800 in za M6803. Ročno sem preizkusil sestavljanje vsakega ukaza, vključujoč z vsako možno vrsto naslavljanja.
2. Prikaz spomina je pregleden in prikaže več kot 512 spominskih celic na enkrat. Vsebine registrov so pregledno prikazane v temu namenjenem polju. Vsak register je kratko in jasno označen. V tem polju pišejo tudi podatki delovanja procesorja kot je hitrost izvajanja in trenutni cikel izvajanja ukaza.
3. Emulator omogoča ročno pisanje v spomin. Na voljo je tudi razstavljavec.
4. V emulator sem vključil vsak ukaz in vsako vrsto prekinitev. Omogočeno je tudi ciklično izvajanje. Najvišja možna hitrost je 1Mhz.
5. Dostopna razhroščevalna orodja so: 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.

## Motorola 6800 Simulator različica 1.33p 2.cR

To je prvi emulator Motorolinega mikroprocesorja M6800, katerega sem uporabljal. Prvič mi ga je predstavil profesor računalništva in z njim smo delali pri pouku računalništva.

Njegovo ime je »Motorola 6800 Simulator različica 1.33p 2.cR«. Ustvaril ga pa je Darko Kropf leta 1995.

A computer screen shot of a computer

Description automatically generated

Slika 24: Uporabniški vmesnik emulatorja »Motorola 6800 Simulator različica 1.33p 2.cR«

## SDK6800/6811 Emulator v1.14 ([www.HVRSoftware.com](http://www.HVRSoftware.com))

Ta emulator sem našel sam, je velik napredek od prej omenjenega saj tudi naravno deluje na OS Windows 10. Ukazi se izvršujejo veliko hitreje. Ima tudi možnost da ne prikazuje vseh informacij o trenutnem izvajanju, ki pospeši izvrševanje. Ko prikazuje vse informacije izvaja približno 256 ukazov na sekundo. Ko pa ne prikazuje informacij(razen zaslona in polja za prikaz spomina) pa približno 1500 ukazov na sekundo.

* Ta emulator je tudi dokaj nepredvidljiv, saj pod neznanimi pogoji samodejno izvajanje ne deluje.
* Na koncu samodejnega izvajanja se tudi emulator samodejno ponastavi kar onemogoči pregled spremenljivk ali registrov po izvajanju nekega programa.
* Ta emulator tudi ne podpira izvrševanje po ciklih. In njegova hitrost je naključna in neenakomerna
* Nekateri ukazi ne delujejo pravilno, npr. ukaz STS ki lahko uporablja direktno indeksirano ter razširjeno naslavljanje je v tem programu označen kot ukaz.
* Emulator ne podpira prekinitve.

A screenshot of a computer

Description automatically generated

Slika 25: Uporabniški vmesnik emulatorja »SDK6800/6811 Emulator v1.14«

## Ostale alternative

Našel sem še nekaj emulatorov, ki jih nisem uspel niti zagnat. Našel sem pa enega ki deluje preko spletne strani. Ta nima assemblerja saj deluje tako, da uporabnik s klikom na operacijsko kodo vstavi ukaz v spomin in ga izvrši.

A screenshot of a computer

Description automatically generated

Slika 26: Uporabniški vmesnik spletnega emulatorja m6800 mypluribus.github.io/m6800/

# Družbena odgovornost

Pri umeščanju moje inovacije glede na trajnost je najpomembnejši vidik, da emulator deluje v digitalni obliki. Zaradi tega ni potreben nakup dodatnih komponent. Uporabnik ga zgolj namesti na svoj računalnik. Ne rabi kupiti mikroprocesorja, niti ni potrebna namestitev dodatnih izvirov napajanja. Zato dodatno ne obremenjuje okolja in ne ustvarja odvečnih elektronskih odpadkov. Emulator je hitrejši, učinkovitejši in popolneje emulira originalen procesor od obstoječih verzij. Zato je nenazadnje energijsko bolj varčen. Znatiželjnim posameznikom pa omogoča lažje učenje arhitekture osnovnega mikroprocesorja ter njegovega zbirnega jezika.

# ZAKLJUČEK

Viri besedila

<https://en.wikipedia.org/wiki/Assembly_language#Assembler>

# VIRI

*Motorola 6800 - Wikipedia*. (11. 17 2023). Pridobljeno iz Wikipedia The Free Encyclopedia: https://en.wikipedia.org/wiki/Motorola\_6800

*QT Group*. (15. 11 2023). Pridobljeno iz spletnega mesta podjetja QT inc.: https://www.qt.io/

1. *Master/slave(Technology)* 2022 [online]. Wikipedija, The Free Encyclopedia. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://en.wikipedia.org/wiki/Master/slave\_(technology)
2. *USB* 2022 [online]. Wikipedija, The Free Encyclopedia. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://en.wikipedia.org/wiki/USB
3. *Universal Serial Bus Specification* 2022 [online]. Universal Serial Bus Specification. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://fl.hw.cz/docs/usb/usb10doc.pdf
4. *USB Hardware* 2022 [online]. Wikipedija, The Free Encyclopedia. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://en.wikipedia.org/wiki/USB\_hardware
5. *USB 3.0* 2022 [online]. Wikipedija, The Free Encyclopedia. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://en.wikipedia.org/wiki/USB\_3.0
6. *USB-C* 2022 [online]. Wikipedija, The Free Encyclopedia. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://en.wikipedia.org/wiki/USB-C
7. *Star network* 2022 [online]. Wikipedija, The Free Encyclopedia. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://en.wikipedia.org/wiki/Star\_network
8. *USB(Communications)* 2022 [online]. Wikipedija, The Free Encyclopedia. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://en.wikipedia.org/wiki/USB\_(Communications)
9. *Univerzalno serijsko vodilo* 2022 [online]. Wikipedija, prosta enciklopedija. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://sl.wikipedia.org/wiki/Univerzalno\_serijsko\_vodilo
10. *What is Micro USB Pinout and Types* 2022 [online]. eTechnophiles. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://www.etechnophiles.com/micro-usb-pinout-features/
11. *Univerzalno serijsko vodilo* 2022 [online]. Wikipedija, prosta enciklopedija. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://sl.wikipedia.org/wiki/Univerzalno\_serijsko\_vodilo
12. *What is USB-C* 2022 [online]. electronicsnotes Pridobljeno 7. 3. 2022 na spletnem naslovu: https://www.electronics-notes.com/articles/connectivity/usb-universal-serial-bus/usb-c-connector.php
13. *USB 3.2 Specification Language Usage Guidelines from USB-IF* [online]. USB-IF Pridobljeno 7. 3. 2022 na spletnem naslovu: https://www.usb.org/sites/default/files/usb\_3\_2\_language\_product\_and\_packaging\_guidelines\_final.pdf

Viri slik

1. *USB* 2022 [online]. Wikipedija, The Free Encyclopedia. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://en.wikipedia.org/wiki/USB
2. *What is Micro USB Pinout and Types* 2022 [online]. eTechnophiles. Pridobljeno 7. 3. 2022 na spletnem naslovu: https://www.etechnophiles.com/micro-usb-pinout-features/
3. *USB(Communications)* 2022 [online]. Wikipedija, The Free Encyclopedia. Pridobljeno 7. 3. 2022 na spletnem naslovu: <https://en.wikipedia.org/wiki/USB_(Communications)>

https://upload.wikimedia.org/wikipedia/commons/1/1b/M6800\_Processor\_Diagram.png

[Slika 1: Uporabniški vmesnik emulatorja razvit v WinFormsu 6](#_Toc155733326)

[Slika 2: Element uporabniškega vmesnika: Števec vrstic 12](file:///D:\projects\QT\MotorolaEmulatorGithub\MotorolaEmulator\Dokumentacija,%20raziskovalna\raziskovalna%20dokumenti\Motorola%20Emulator.docx#_Toc155733327)

[Slika 3: Element uporabniškega vmesnika: Števec vrstic - napredne informacije kode 12](#_Toc155733328)

[Slika 4: Element uporabniškega vmesnika: Števec vrstic - primer oznak 13](#_Toc155733329)