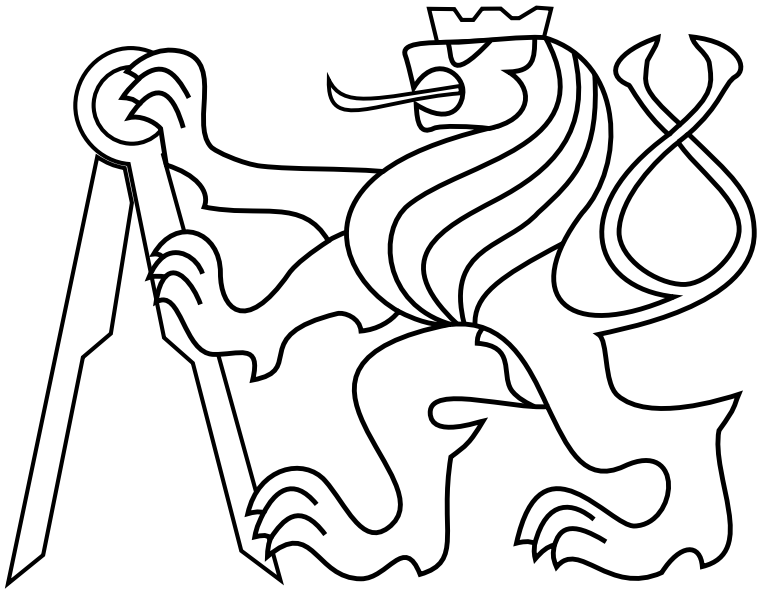
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ

Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE



David Hývl

Simulátor MIPS procesoru

Katedra řídící techniky

Vedoucí práce: **Ing. Michal Štěpanovský, Ph.D.**

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne..... .....

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **David Hývl**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Simulátor MIPS procesoru**

Pokyny pro vypracování:

1. Vypracujte řešení o volně dostupných simulátorech procesoru podporujícího instrukční sadu MIPS.
2 Identifikujte výhody a nevýhody jednotlivých simulátorů.
3. Navrhněte a realizujte simulátor pro potřeby předmětu A0B36APO Architektury počítačů.
4. Jednotlivé požadavky a postupy konzultujte s vedoucím práce.

Seznam odborné literatury:

[1] Hennesy, J. L., Patterson, D. A.: Computer Architecture: A Quantitative Approach, Third Edition, San Francisco, Morgan Kaufmann Publishers, Inc., 2002

Vedoucí: Ing. Michal Štepanovský, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

Prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 4. 11. 2015

Poděkování

Nejprve bych chtěl poděkovat vedoucímu práce Ing. Michalu Štěpanovskému, Ph.D., za jeho vedení a pomoc při vypracování této práce. Dále bych chtěl také poděkovat své rodině a přátelům za jejich podporu během mých studií.

Abstract

The aim of this thesis is to design and implement a MIPS processor simulator including graphical user interface for needs of A0B36APO subject. This simulator makes easier understanding for students of the work of this processing unit and it is also a visual aid during their studies. The simulator allows students to design their own program in assembler code, which can be further loaded into the visualization of the program where the code is simulated. Another aim of this work is to make a comparison of current MIPS simulators and identify their advantages and disadvantages. The functionality of the proposed simulator was validated on reference assembler file.

Keywords: assembler, MIPS, MIPS simulator, graphical user interface

Abstrakt

Prvním cílem této práce je navrhnout a realizovat simulátor MIPS procesoru včetně grafického uživatelského rozhraní pro potřeby předmětu A0B36APO. Tento simulátor usnadní studentům pochopení činnosti tohoto procesoru a je názornou pomůckou během jejich studia. Simulátor studentům umožní navrhnout vlastní program v assembler kódu, který je možno dále nahrát do vizualizační části programu, kde je tento kód simulován. Dalším cílem této práce je zpracovat porovnání současných MIPS simulátorů a identifikovat jejich výhody a nevýhody. Funkčnost navrženého simulátoru byla ověřena na referenčním assemblerovském souboru.

Klíčová slova: assembler, MIPS, MIPS simulátor, grafické uživatelské prostředí

Obsah

Seznam obrázků

1 Úvod

Procesor MIPS je zástupcem procesorů typu RISC. Jeho zkratka pochází z anglického překladu Microprocessor without Interlocked Pipeline Stages, což můžeme česky přeložit jako procesor bez automaticky organizované pipeline. Tento typ procesor se rozšířil zejména díky svému vysokému výkonu v oblasti zpracování trojrozměrného obrazu. V dnešní době došlo k poklesu využívaní tohoto typu procesoru, ale stále ho můžeme nalézt ve spotřební elektronice, či síťových směrovačích i v aplikacích zpracování obrazu [6].

Pro studijní účely byl tento procesor, a jeho architektura, zvolen díky jeho vlastnostem. A to hlavně díky shodné délce všech instrukcí a vykonávání každé instrukce v jednom cyklu běhu.

Hlavním cílem této práce je navrhnout a realizovat MIPS simulátor včetně grafického rozhraní pro výukové účely v předmětu A0B36APO [3]. V tomto simulátoru je možné vytvořit vlastní soubor v assembleru, či editovat stávající soubor. Tento soubor lze pak načíst v simulačním okně programu a simulovat činnost procesoru. Zároveň je v tomto okně možné vizuálně sledovat vykonávání instrukce při průchodu jednotlivými komponentami procesoru i sledovat plnění instrukční a datové paměti a jednotlivých registrů. Pro studijní účely byl omezen počet podporovaných instrukcí z instrukčního setu MIPS architektury jen na nejdůležitější.

Posledním cílem této práce je vypracování rešeršní části, ve které porovnám několik stávajících MIPS simulátorů procesoru a zaměřím se na jejich vlastnosti, výhody a nevýhody.

2 Návrh procesoru

Navržený procesor vychází z konceptu uvedeného v přednáškách předmětu A0B36APO [3]. Návrh byl po konzultaci s vedoucím práce proveden s ohledem na využití jako studijní pomůcka, došlo tedy k redukci instrukčního setu, a ke zjednodušení koncepce celého procesoru.

2.1 Instrukční set

Všechny instrukce procesoru MIPS mají stejnou délku, a to konkrétně 32 bitů. Můžeme je rozdělit do třech kategorií, na instrukce typu R, I a J. Před vykonáním příslušné operace je třeba instrukci v assembler zápisu přeložit do strojového kódu. Po přeložení je instrukce nahrána do instrukční paměti procesoru a následně vykonána.

2.1.1 Instrukce typu R

Tento typ instrukce se skládá z operačního kódu, prvního a druhého zdrojového registru, dále z cílového registru, posunu v logických operacích a funkčního operandu. Všechny tyto instrukce mají operační kód roven nule, tedy jednotlivé instrukce se rozeznávají na základě funkčního znaku (viz Tabulka 1).

Instrukce	Operační kód	Funkční znak
Add	000000	100000
Sub	000000	100010
And	000000	100100
Or	000000	100101
Slt	000000	101010
Sll	000000	000000

Tabulka 1: Podporované R instrukce

2.1.2 Instrukce typu I

Instrukce typu I, neboli instrukce s přímým operandem se skládají z operačního kódu, zdrojového a cílového registru a z konstanty. Instrukce jsou rozpoznávány na základě operačního kódu (viz Tabulka 2).

Instrukce	Operační kód
Addi	001000
Lui	001111
Ori	001101
Lw	100011
Sw	101011
Beq	000100
Bne	000101

Tabulka 2: Podporované I instrukce

2.1.3 Instrukce typu J

Obecně se instrukce typu J, tedy skokové instrukce, skládají z operačního kódu a adresy skoku. V tomto simulátoru není podporována žádná skoková instrukce, a to z důvodu jednoduchosti návrhu logiky procesoru. Skoky v programu však lze provádět pomocí instrukcí Beq a Bne, které však díky svému formátu spadají do výše uvedených instrukcí typu I.

2.1.4 Pseoudoinstrukce

Pseudoinstrukcí je myšlena v tomto případě instrukce la (load address). Tato instrukce je nazývána jako pseudoinstrukce, jelikož jí nelze vykonat v jednom cyklu procesoru. Konkrétně je třeba pro provedení rozložit tuto instrukci na dvě, a to na Lui a Ori.

2.2 Komponenty

Procesor byl navržen z následujících elementárních komponent. Jejich princip a funkce bude popsána níže.

2.2.1 Multiplexor

Multiplexor slouží k přepínání jednotlivých vstupů. K výběru dochází prostřednictvím nastavení řídícího signálu, poté je na výstup této komponenty přesměrován příslušný vstup.

2.2.2 Znaménkové rozšíření

Komponenta pro znaménkové rozšíření převezme svůj vstupní argument, což je přímý operand I instrukcí. Jelikož je toto číslo pouze šestnácti bitové, je třeba ho znaménkově

rozšířit na třiceti dvou bitové číslo, které je dále používáno ve výpočtech.

2.2.3 Bitový posun

Bitový posun doleva o dvě místa slouží při výpočtu koncové adresy skoku pro instrukce Beq a Bne. Přímý operand těchto instrukcí značí počet instrukcí, o které je třeba skočit. Jelikož adresy instrukční paměti jsou slovně zarovnány a jsou třiceti dvou bitové, je třeba při skoku o jednu instrukci zvýšit, případně snížit, adresu o čtyři. Posun zajišťuje právě tato komponenta, která převezme přímý operand a bitovým posunem vlevo o dvě místa, vynásobí toto číslo čtyřmi.

2.2.4 And a Or

And a or jsou využitá logická jednobitová hradla. Slouží pro nastavení řídícího signálu multiplexoru při skokových instrukcích, na základě vstupních parametrů těchto komponent.

2.2.5 Aritmeticko logická jednotka

Tato komponenta slouží, jak již její název napovídá, pro zpracování aritmetických a logických operací. Tato jednotka má dva vstupní a jeden výstupní parametr, dále řídící vstup a výstup pro detekci shodnosti vstupů. Na základě řídícího vstupu se nastaví příslušná aritmetická nebo logická operace, provedou se příslušné výpočty a jsou nastaveny výstupní parametry. Řídící vstup je tří bitové číslo, jehož hodnota určuje příslušnou aritmetickou či logickou operaci (viz Tabulka 3).

Operace	Řídící vstup
And	0x0
Or	0x1
Add	0x2
Sll	0x3
Lui	0x4
Sub	0x6
Slt	0x7

Tabulka 3: Řídící vstupy ALU

2.2.6 Programový čítač

Programový čítač je speciální druh registru procesoru, sloužící k adresování instrukce strojového kódu v instrukční paměti. Na jeho výstupu se nachází adresa vykonávané instrukce a na jeho vstupu je adresována instrukce následující. Tato komponenta se chová tedy jako posuvný registr a slouží k určování adresy vykonávané instrukce.

K přenosu adresy dochází při náběžné hraně hodin, tedy komponenta je tímto signálem řízena. V případě navrženého procesoru je výstup programového čítače při inicializaci nastaven vždy na adresu 0xFFF80000, což je adresa umístění první instrukce v instrukční paměti.

2.2.7 Instrukční paměť

Po provedení překladu assemblerovského souboru jsou v paměti této komponenty nahrány instrukce ve strojovém kódu, v pořadí, ve kterém se vykonávají. Vstupním parametrem je adresa instrukce, která se má v současném hodinovém cyklu vykonat a výstupem je tedy strojový kód instrukce, který je dále procesorem zpracován.

2.2.8 Paměť registrů

V paměti registrů jsou uchovávány jednotlivé pracovní hodnoty. K dispozici je 32 registrů, z nichž každý je třiceti dvou bitový. Do každého z nich je v tomto případě možné zapisovat i číst hodnotu, s výjimkou registru nula, ve kterém je trvale uložena nulová hodnota.

Vstupními parametry této komponenty jsou adresy dvou zdrojových registrů, adresa a data cílového registru. Na výstupu se nachází data z adresovaných zdrojových registrů a celý tento proces je řízen dvěma signály. Konkrétně to jsou hodinový signál a řídící parametr nastavovaný řadičem. Oba tyto kontrolní vstupy řídí čtení a zápis do příslušných registrů. V případě, že má signál vyslaný z řadiče hodnotu logické 1, dochází při následující náběžné hraně hodinového cyklu k zápisu dat do cílového registru. Nezávisle na hodnotě vyslané řadičem, dochází při stavu hodinového signálu high (tedy v logické 1) ke čtení dat z adresovaných vstupních registrů. Konkrétní adresy se získávají ze strojového kódu vykonávané instrukce, a to vyčleněním odpovídajících bitů, což závisí na typu instrukce.

2.2.9 Datová paměť

Datová paměť slouží k uchovávání dlouhodobějších informací mimo registry. V tomto modelu má datová paměť vyčleněné místo od bázové adresy 0xFFF40000, přičemž jed-

notlivé adresy jsou slovně zarovnány. Vstupním parametrem je adresa požadovaných dat, případně jejich zápisu. Výstupem jsou adresovaná data. Celá tato komponenta je řízena opět dvěma signály, a to hodinami a řídícím vstupem. Pokud je řídící vstup nastaven na logickou hodnotu 1 a hodinový signál se nachází ve stavu náběžné hrany, dochází v dalším hodinovém cyklu k zápisu nastavených dat na požadovanou adresu. Pokud se hodiny nachází v logické 1, dochází podobně jako u paměti registrů k přenosu adresovaných dat na výstup.

Tuto paměť lze naplnit daty buď pomocí inicializace pole a jeho hodnot v assembler kódu, kde se paměť plní při jeho překladu od počáteční adresy, případně díky instrukcím lw (load word), která čte hodnotu z paměti, nebo sw (store word), která do paměti hodnoty ukládá.

2.2.10 Řadič

Řadič je zodpovědný za nastavování všech příslušných řídících vodičů. O příslušném nastavení je rozhodováno na základě vstupních parametrů, kterými jsou operační kód (Opcode) a funkční znak (Funct) vykonávané instrukce, získané z jejího strojového kódu na příslušných bitech. Pro každou instrukci je sada výstupů řadiče unikátní a díky tomu je docíleno správného nastavení všech komponent pro požadovanou instrukci (viz Tabulky 4 a 5).

Označení	Význam
RegWrite (RW)	Zápis hodnoty do adresovaného registru v paměti registrů
RegDest (RD)	Výběr adresy cílového registru v paměti registrů
AluSrc (AS)	Výběr prvního vstupu aritmeticko - logické jednotky
SllSrc (SS)	Výběr druhého vstupu aritmeticko - logické jednotky
AluCtrl (AC)	Řídící vodič pro nastavení operace aritmeticko - logické jednotky
BranchBne (BBne)	Detekce skokové instrukce Beq
BranchBeq (BBeq)	Detekce skokové instrukce Bne
MemWrite (MW)	Zápis hodnoty na příslušnou adresu do datové paměti
MemToReg (MR)	Výběr dat pro zápis do cílového registru paměti registrů

Tabulka 4: Význam řídících signálů

Instrukce	RW	RD	AS	SS	AC	BBne	BBeq	MW	MR
Add	1	1	0	0	0x2	0	0	0	0
Sub	1	1	0	0	0x6	0	0	0	0
And	1	1	0	0	0x0	0	0	0	0
Or	1	1	0	0	0x1	0	0	0	0
Slt	1	1	0	0	0x7	0	0	0	0
Sll	1	0	1	1	0x3	0	0	0	0
Lw	1	0	1	0	0x2	0	0	0	1
Sw	0	0	1	0	0x2	0	0	1	0
Addi	1	0	1	0	0x2	0	0	0	0
Lui	1	0	1	0	0x4	0	0	0	0
Ori	1	0	1	0	0x1	0	0	0	0
Beq	0	0	0	0	0x6	0	1	0	0
Bne	0	0	0	0	0x6	1	0	0	0

Tabulka 5: Hodnoty řídících signálů

3 Výpočetní část programu

Výpočetní jádro bylo programováno v jazyce C, později kompilováno C++ kompilátorem. Důvodem bylo grafické uživatelské rozhraní vytvořené v jazyce C++, tedy výpočetní jádro bylo po vytvoření spojeno s touto grafickou částí. Jádro bylo navrženo co nejvíce univerzální aby bylo možné jej v budoucnu rozšiřovat a upravovat, a vycházelo z konceptu potřebného pro výukové účely předmětu A0B36APO [3]. Jednotlivé soubory a jejich funkce (viz Tabulka 6) budou popsány níže.

Soubor	Funkce
components.h	Definice jednotlivých struktur pro komponenty procesoru
functions.h	Hlavičky vyhodnocovacích funkcí komponent
init.h	Inicializační funkce komponent
instruction_set.h	Makra hodnot operačních kódů a funkčních znaků instrukcí
instructions.h	Makra pro nastavení hodnot hodin a řídícího signálu ALU
mips.h	Hlavičky funkcí pro obsluhu procesoru a definice struktur parseru
functions.cpp	Definice vyhodnocovacích funkcí komponent
init.cpp	Definice inicializačních funkcí komponent
mips.cpp	Parser a výkonný kód procesoru

Tabulka 6: Soubory jádra procesoru

3.1 Components

V tomto souboru se vyskytují deklarace struktur všech použitých komponent (viz Tabulka 7). Celý procesor je navržen na základě propagace informací mezi těmito komponentami. Nejdůležitějším prvkem je struktura WIRE, která symbolizuje sběrnici a uchovává v sobě informace o svojí šířce, nastavených datech, stavu sběrnice a svůj název. Touto strukturou jsou dále spojovány ostatní komponenty. K propagaci dochází způsobem, že určitá komponenta zastoupena svojí strukturou nastaví svému výstupnímu vodiči do datové proměnné vypočtenou hodnotu. Tím, že je tato sběrnice použita jako vstupní parametr pro jinou komponentu, je možné z ní číst aktuálně nastavená data.

Další komponenty popsané v předchozí kapitole jsou taktéž deklarovány jako struktury, kde jejich vstupy, výstupy a řídící signály jsou typu WIRE, a dále společně obsahují taktéž proměnnou určující jejich stav a název. Struktura symbolizující programový čítač ještě obsahuje informaci o počáteční adrese instrukční paměti. Instrukční, datová a paměť registrů obsahují ukazatele na příslušné bloky paměti.

Struktura	Komponenta
WIRE	Sběrnice
MUX	Multiplexor
SIGN	Znaménkové rozšíření
SHIFT	Bitový posun
ALU	Aritmeticko - logická jednotka
AND	Logické hradlo and
OR	Logické hradlo or
PC	Programový čítač
IMEM	Instrukční paměť
DMEM	Datová paměť
REGFILE	Paměť registrů
CUNIT	Řadič

Tabulka 7: Struktury komponent procesoru

3.2 Instruction_set a instructions

Tyto hlavičkové soubory slouží k definování symbolických konstant. Jedná se konkrétně o konstanty operačních kódů a funkčních znaků jednotlivých instrukcí, dále příslušné hodnoty řídícího vodiče aritmeticko logické jednotky, který rozhoduje o prováděné operaci a definici stavů hodinového signálu. Tyto konstanty jsou dále využívány při vykonávání efektivního kódu procesoru.

3.3 Init

V této části se nacházejí inicializační funkce všech struktur komponent. Každá tato funkce má podobnou strukturu. Výstupním parametrem všech funkcí je ukazatel na inicializovanou strukturu, přičemž inicializace se provede výkonem této funkce. Jako první vstupní proměnná je ukazatel na vlastní komponentu, která byla předem deklarována, dále následují všechny potřebné vstupní a výstupní sběrnice typu WIRE, prvotní stav komponenty a její název. U paměťových prvků je předáván ještě ukazatel na příslušný blok paměti. V definici samotné inicializační funkce se provede alokace paměti pro předanou komponentu, případně alokace pamětí pro instrukční, datovou a paměť registrů a přiřadí se jednotlivé parametry funkce k příslušným parametrům inicializované struktury. Po tomto přiřazení se předá ukazatel na alokovanou komponentu do návratové hodnoty.

3.4 Functions

Zde se nachází vyhodnocovací funkce všech komponent procesoru. Jelikož je každá struktura spojena pomocí sběrnice, stačí pro získání všech potřebných údajů pro vyhodnocení předat jako vstupní parametr pouze ukazatel na strukturu, která se má vyhodnotit. Při vyhodnocování se přistoupí k jednotlivým vstupům, na základě řídících signálů se provede požadovaná operace a komponentě se nastaví vypočtená data na výstupní sběrnici. Při volání těchto funkcí tedy dochází k automatické propagaci informace po sběrnici z výstupu zpracovávané komponenty na vstupy následujících, které si tyto hodnoty převezmou při volání vlastní vyhodnocovací funkce. Důležité evaluační funkce budou popsány níže.

3.4.1 Znaménkové rozšíření

Pokud je detekována řadičem instrukce typu I, tedy instrukce s přímým operandem, je třeba toto číslo dále zpracovat. Nachází se na posledních 16 bitech instrukce, proto je třeba toto číslo vyčlenit a dále rozšířit na 32 bitové, kvůli dalšímu zpracování.

K vyčlenění potřebných bitů se používá princip masky a bitového součinu. Jelikož potřebujeme pouze posledních šestnáct bitů, zvolíme masku v hexadecimálním zápisu 0x0000FFFF. Nad tímto číslem a strojovým kódem instrukce poté provedeme bitový součin, neboli operaci and. Nyní je třeba toto číslo znaménkově rozšířit, což znamená překopírovat šestnáctý bit do ostatních vyšších pozic. Znovu tedy použijeme masku, tentokrát o hodnotě 0x00008000, a bitový součin s vypočtenou hodnotou. Tím získáme hodnotu šestnáctého bitu a tu překopírujeme na vyšší pozice. Výsledná hodnota se poté nastaví na výstupní sběrnici této komponenty.

3.4.2 Bitový posun

Bitový posun je třeba provést o dvě místa vlevo, čímž dojde k vynásobení vstupního čísla čtyřmi. Při této bitové operaci se po posunu nastaví nejnižší bity na nulu, a na nejvyšších bitech dojde ke ztrátě přesnosti. Jelikož v našem případě je vstupním parametrem číslo jdoucí po sběrnici ze znaménkového rozšíření, máme jistotu, že horní dva zahozené bity mají pouze hodnotu kopie znaménkového bitu. Posunuté číslo se poté nastaví na výstupní sběrnici.

3.4.3 Aritmeticko logická jednotka

O operaci vykonávané ALU jednotkou se rozhoduje na základě řídícího signálu z řadiče. Operace je vybrána pomocí switch-case příkazu. Pokud hodnota řídícího signálu neodpovídá žádné operaci, je tato skutečnost detekována a vypsána do konzole. Při zpracovávání

všech operací je kromě výstupní hodnoty signálu nastavována ještě sběrnice pro detekci rovnosti vstupních hodnot. Tato hodnota se využívá při operaci odčítání, tedy pokud po odečtení vstupních hodnot zjistí ALU jednotka nulový výsledek, nastaví tuto sběrnici do logické 1. Tato hodnota se poté používá při skokových instrukcích Beq a Bne, kde je třeba pro skok testovat rovnost parametrů.

3.4.4 Programový čítač

Programový čítač slouží jako oddělovací registr pro adresu stávající instrukce a následující. Při svém vyhodnocování sleduje hodinový signál, jakožto svůj řídící vstup, a při stavu v náběžné hraně klopí vstup na výstup, tedy na výstupní sběrnici se nastaví adresa instrukce z vstupní sběrnice.

3.4.5 Instrukční paměť

Instrukční paměť slouží pro vystavení strojového kódu instrukce na základě adresy na její vstupní sběrnici. Bázová adresa je stanovena na 0xFFFF8000. Od této adresy jsou ukládány instrukce po překladu parserem. Počet možných instrukcí k uložení je omezen na 1024, tedy velikost instrukční paměti je 4096 B.

Jelikož jsou adresy slovně zarovnány, adresace do paměti probíhá následujícím způsobem. Poslední dva bity adresy slouží jako offset, je tedy nutné adresu posunout o dvě místa bitovým posunem vpravo. Jelikož maximální počet instrukcí je 1024, což odpovídá číslu 2^{10}, je třeba na adresování použít následujících deset bitů, čehož docílíme opět maskováním pomocí masky 0x000003FF, a bitovým součinem s posunutou adresou. Bázová adresa je tedy nastavena fixně na výše uvedenou hodnotu a je možné jí v kódu programu změnit pomocí změny hodnoty jejího makra. Vypočtená adresa je tedy indexem instrukce v poli instrukční paměti procesoru, a následně je vystaven strojový kód z této adresy na výstupní sběrnici.

3.4.6 Datová paměť

Adresování datové paměti je shodné jako adresovaní instrukční paměti, tedy pomocí dvoubitového offsetu na konci adresy, přepočtu indexu pomocí masky a bitového součinu následujících deseti bitů. Tím se získá index to pole dat, a je možné uložit případně vyčíst příslušná data. Evaluační funkce dále sleduje řídící signály, kterými jsou hodinový cyklus a řízení zápisu nastavené na příslušnou sběrnici řadičem. Tedy pokud je hodinový signál ve stavu logické 1 a řídící vodič nastaven pro čtení dat, jsou data vystavena na výstupní sběrnici. Pokud je řídící sběrnice nastavena na zápis dat do paměti, jsou v následujícím

hodinovém cyklu při stavu náběžné hrany data zapsána na příslušný index do paměťového pole.

3.4.7 Paměť registrů

Při vyhodnocování je v první řadě nutné vyčlenit hodnoty vstupních registrů. Ty se nachází na pozicích bitů [25 : 21] a [20 : 16], je tedy třeba provést bitový součin s maskou, která je společná a má hodnotu 0x0000001F, a posunutou hodnotou instrukce. V prvním případě je třeba provést bitový posun vpravo o 21 bitů, v druhém případě o 16 bitů. Získané hodnoty mají význam indexu do paměti registrů a jsou ošetřeny na nepodporované hodnoty. Těmito hodnotami jsou záporná čísla registrů a index větší než 31. Pokud nastane tento výjimečný stav, je vypsána příslušné chybové hlášení do konzole.

Zápis a čtení do paměti podléhá shodným řídícím signálům jako datová paměť, tedy řídící sběrnici řadiče a stavu hodin. Pokud je hodinový signál ve stavu HIGH a řídící signál nastaven pro čtení, vystaví se na výstupní sběrnici čtená data příslušná adresovaným registrům. Pokud je nastaven mód zápisu, při následující náběžné hraně hodin jsou zapsána data do adresovaného cílového registru.

3.4.8 Řadič

Vyhodnocovací funkce řadiče je závislá na operačním kódu a funkčním znaku příchozí instrukce po vstupní sběrnici. Je tedy v první řadě nutné tato data vyčlenit. Funkční znak se nachází na posledních šesti bitech instrukce, tedy pro jeho získání stačí využít logický součin mezi instrukcí a maskou o hodnotě 0x0000003F. Operační kód se vyskytuje na prvních šesti bitech, tím pádem je třeba nejprve instrukci posunout bitově o dvacetšest míst vpravo a poté provést shodný bitový součin jako pro získání funkčního znaku. Celá výkonná struktura řadiče je navržena pomocí switch-case struktury, na základě vyčleněných parametrů. Pokud se tedy vyskytnou operační kódy a funkční znaky známých instrukcí, jsou na výstupní sběrnici nastaveny příslušné hodnoty řídících signálů. Pokud se vyskytnou neznámé hodnoty, je do konzole vypsána příslušné chybové hlášení.

3.5 Mips

V tomto úseku se nachází funkce pro inicializaci a běh kompletního procesoru, a všechny potřebné nástroje pro překlad assembler souboru do strojového kódu, tedy parseru. Dále jsou zde deklarovány funkce pro předávání potřebných parametrů procesoru grafickému uživatelskému rozhraní, tedy zde dochází k jejich vzájemnému propojení. A na počátku tohoto souboru jsou deklarovány všechny potřebné proměnné pro komponenty, a pro zpracování překladu.

3.5.1 Cyklus procesoru

První důležitou funkcí je inicializační funkce všech komponent procesoru deklarovaných na počátku souboru. Návratovou hodnotou této funkce je testováno, zda při inicializaci komponent nedošlo k výjimečné události. Pokud alokace proběhla bezchybně, návratová hodnota je rovná nule, v opačném případě funkce vrací mínus jedna. S touto částí úzce souvisí funkce pro uvolnění alokované paměti všech komponent, včetně jejich vnitřních alokovaných dat.

Nejdůležitější je pak funkce pro vykonání jednoho hodinového cyklu procesoru, tedy jedné funkce. Výkon je rozdělen do třech částí podle stavu hodinového signálu. Pokud je hodinový signál ve stavu náběžné hrany, dochází, pokud jsou nastaveny příslušné řídící vodiče, k zápisu dat do pamětí a k propagaci adresy nové instrukce v programovém čítači. Při hodinovém cyklu v logické 1 dochází k obsluze všech komponent v daném pořadí. Pořadí je pevně dáno kvůli nastavování řídících signálů a správné propagaci informace procesorem. Hodinový signál v logické 0 slouží k zobrazení hodnot vypočtených komponentami procesoru. Návratovou hodnotou této funkce je opět informace o správnosti běhu, tedy pokud funkce vrátí hodnotu nula, proběhl cyklus v pořádku. Pokud se na výstupu funkce objeví mínus jedna, došlo při běhu k chybě.

3.5.2 Parser

Parser slouží k překladu assembler souboru do strojového kódu, který se poté nahrává do instrukční paměti procesoru. Dále se pomocí něho plní datová paměť, pokud je v programu napsána příslušná sekce pro její plnění. Navržený parser funguje na principu třech průchodů souborem. Během prvního průchodu dochází k uložení symbolů registrů, plnění datové paměti. Pokud proběhne inicializační průchod bez chybového výskytu a parser detekuje konec souboru díky návěští end, spouští se druhý průchod.

Během tohoto kroku se detekují jednotlivé instrukce a dochází k jejich překladu do strojového kódu. Během této části se využívají definované symboly registrů nalezené v minulém kroku, a také práce s alokovanými poli. Přeložené instrukce se ukládají do pole struktur typu INSTR, kde je uchována jejich přeložená hodnota a jejich hash kód. Dále jsou uchovávány původní texty instrukcí, kvůli zobrazování informací v grafickém rozhraní. Hash kód instrukce je v tomto případě číslo řádku, na kterém se tato instrukce nachází. Tento údaj je využit při chybových hlášeních, kdy se uživateli zobrazí informace chybné řádky, a dále při hledání návěští skokových instrukcí. Pokud tato část opět proběhne bezchybně, spustí se třetí část parseru, která má za úkol vyhledat následující instrukci po hledaném návěští skokové instrukce. Pokud příslušnou instrukci nalezne, je díky zvolenému hash kódu vypočten počet instrukcí, o kolik se má posunout programový čítač, a tento údaj je uložen do strojového kódu příslušné instrukce. V případě bezchybného průchodu i touto částí, jsou hodnoty instrukcí překopírovány do paměti komponenty představující instrukční

paměť. Pomocné proměnné jsou vynulovány pro případný další překlad a alokovaná paměť pomocných polí a struktur je uvolněna.

4 Grafické uživatelské rozhraní

V této části bude popsán princip a funkčnost grafického uživatelského rozhraní, které bylo vyvinuto pomocí QT frameworku [2]. Hlavním cílem bylo navrhnout uživatelsky přívětivé rozhraní, které bude podporovat tvorbu vlastního assembler souboru v editoru, a dále simulování těchto souborů pomocí modelu MIPS procesoru a zobrazení všech důležitých simulačních hodnot. Všechny ikony a obrázky byly navrženy pomocí programu Gimp2 [8].

4.1 QT framework

QT je multiplatformní aplikace a uživatelské rozhraní pro vývojáře používající C++ nebo QML. QT a jeho podpůrné nástroje jsou vyvinuty jako open souce projekt, QT tedy může být využit v rámci open source licencí (GPL v3 a LGPL v2. 1). Pro vývoj simulátoru byl použit QT Creator, který podporuje multiplatformní QT integrované vývojové prostředí.

4.2 Hlavní okno

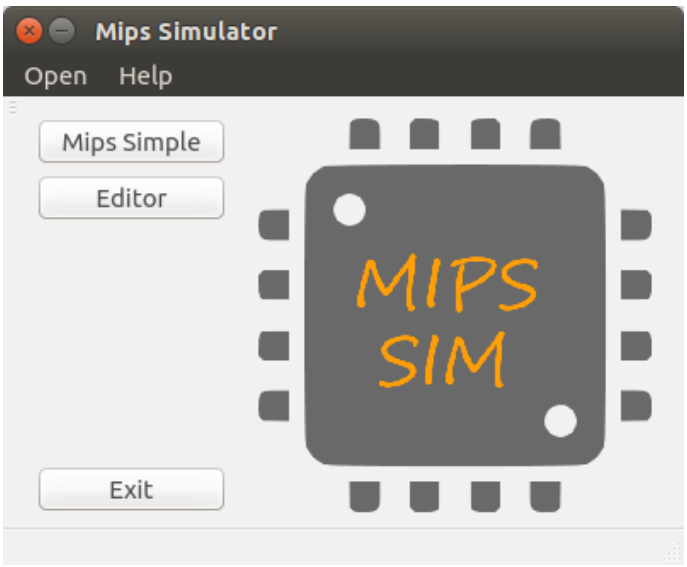
Hlavní okno (viz Obrázek 1) slouží jako rozcestník pro požadovanou práci. Nalezneme zde nápovědu pro práci s programem, dále informace o QT vývojovém prostředí, to vše pod záložkou Help. V rozbalovacím menu Open se nachází shodná tlačítka jako na ploše, a to konkrétně spuštění editoru assembler souboru pod tlačítkem Editor, otevření samotného simulátoru pomocí Mips Simple a pro ukončení aplikace tlačítko Exit. Při otevření nového okna se hlavní okno deaktivuje a otevře se požadovaný nástroj. Po ukončení práce a zavření nového okna se tato úvodní obrazovka znovu aktivuje. K úplnému ukončení programu tedy dojde při zavření této obrazovky.

4.3 Editor

Okno editoru (viz Obrázek 2) slouží pro návrh či editaci programu v assembler kódu. Při otevření tohoto okna se zobrazí prázdná textová plocha, a je možné začít vytvářet program. Okno poskytuje další funkce popsané níže.

4.3.1 Nový soubor

Nový soubor je možné začít tvořit v centrální textové oblasti, ihned po prvním spuštění okna editoru. Dále je možné využít tlačítka v liště nástrojů, případně v rozbalovacím menu File. Pokud došlo ke změně v textové oblasti, pak je při snaze vytvořit nový soubor či



Obrázek 1: Hlavní okno simulátoru

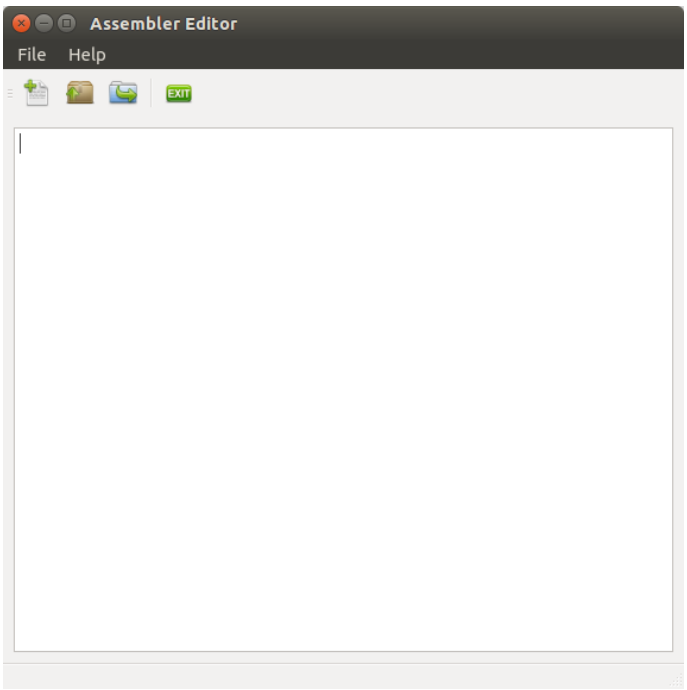
ukončit práci s editorem uživatel varován dialogovým oknem (viz Obrázek 3), že aktuální program není uložen a je mu tato možnost nabídnuta, včetně stornování všech změn.

4.3.2 Ukládání souboru

Navržený, případně editovaný soubor, lze uložit několika způsoby. Jedním způsobem je tlačítko Save v rozbalovacím menu File, další možností je příslušná ikona v liště nástrojů a posledním způsobem je varovné okno (viz Obrázek 3), otevírané při zavírání editoru. Pokud se jedná o existující soubor, uložení proběhne právě do tohoto souboru. Jestliže se jedná o nový program, při uložení se otevře okno výběru místa uložení nového souboru (viz Obrázek 4). Ten je pak v dané lokaci vytvořen, a je do něj uložen text z okna editoru. Ukládání je omezeno na soubory typu assembler (*.s), jelikož simulátor pracuje pouze s nimi.

4.3.3 Načtení souboru

Načtení existujícího assembler souboru je možné provést buď pomocí stlačení ikony v liště nástrojů, nebo pomocí tlačítka v rozbalovacím menu File. Po provedení se otevře okno souborového systému (viz Obrázek 4). V tomto okně je zobrazování opět omezeno pouze na assembler soubory (*.s), které se po vyhledání a potvrzení nahrají do okna editoru a jejich text se nahraje do centrální textové oblasti (viz Obrázek 5), kde je ho možné editovat a po dokončení práce uložit změny.



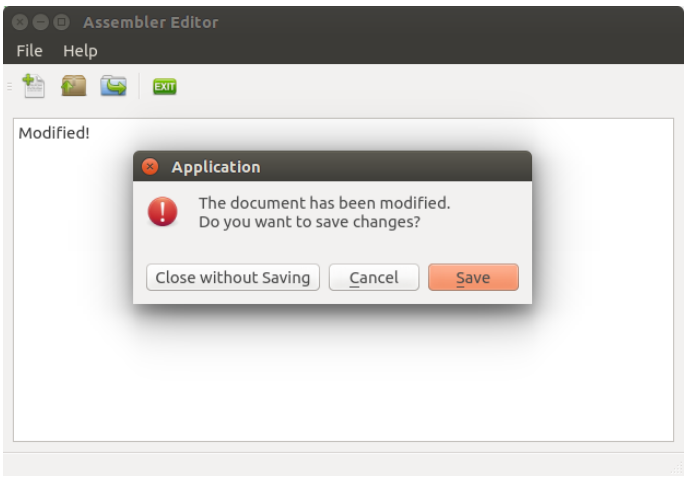
Obrázek 2: Okno editoru

4.4 Okno procesoru

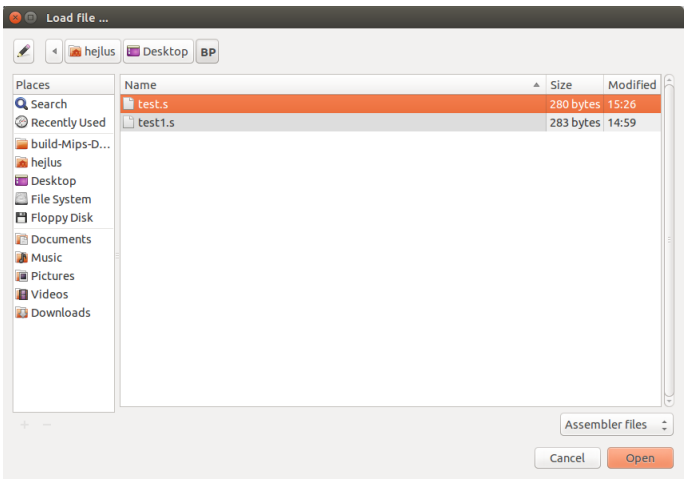
Toto okno je nejdůležitějším v celém simulátoru, zde dochází k interakci s výpočetní částí, a k samotné simulaci činnosti procesoru (viz Obrázek 6). Okno je členěno na několik sekcí. Centrální sekce obsahuje graficky znázorněný model MIPS procesoru, jsou zde zobrazeny všechny výše uvedené komponenty. V obdélníkových oblastech se zobrazují data příslušná vyznačené sběrnici. Po stranách se nalézají dokovací okna, představující paměti jednotlivých paměťových komponent. Tato okna lze minimalizovat a lze je také odepnout do popředí centrálního okna. Ovládání zobrazování postranních oken se nachází v ovládacím panelu pod příslušnými tlačítky, a také v rozbalovacím menu View.

4.4.1 Načtení souboru

Po spuštění okna simulátoru je aktivní pouze ikona pro načtení souboru. Po stlačení se otevře souborový dialog pro vyhledání assembler souboru v souborovém systému. Tento soubor nemusí být pro spuštění nutně vytvořen nebo upraven v editoru tohoto softwaru. Po zvolení se načte cesta k vybranému souboru a uloží se do vnitřní proměnné. Dále se aktivuje v ovládacím panelu ikona pro překlad assembler programu do strojového kódu.



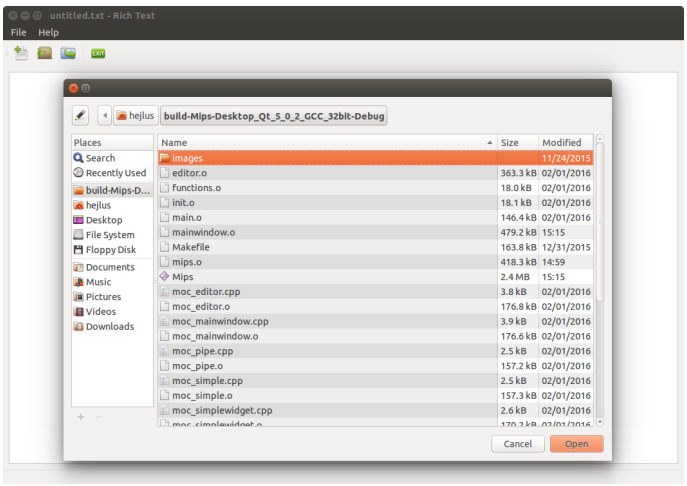
Obrázek 3: Varování modifikace souboru



Obrázek 4: Dialog pro ukládání a načítání assembler souborů

4.4.2 Překlad

Překlad souboru proběhne po zmáčknutí tlačítka Build, buď v ovládacím panelu, případně v záložce Run. Po stisknutí se volají příslušné funkce parseru ze souboru mips.cpp, které mají za následek překlad programu do strojového kódu. Parser je detailněji popsán v příslušné kapitole. Pokud se během překladu vyskytne chyba, procesor je uveden do původního stavu a je třeba znovu nahrát opravený soubor a znovu se pokusit o překlad. Chybová hlášení jsou vytvořena formou vyskakovacích oken, přičemž je zde vypsána příslušná chybová hláška, a v případě překladu instrukcí i řádka chybové instrukce. Pokud dojde k bezchybnému překladu, aktivují se funkce pro simulaci činnosti procesoru, a dojde k naplnění dokovacích oken příslušnými daty.

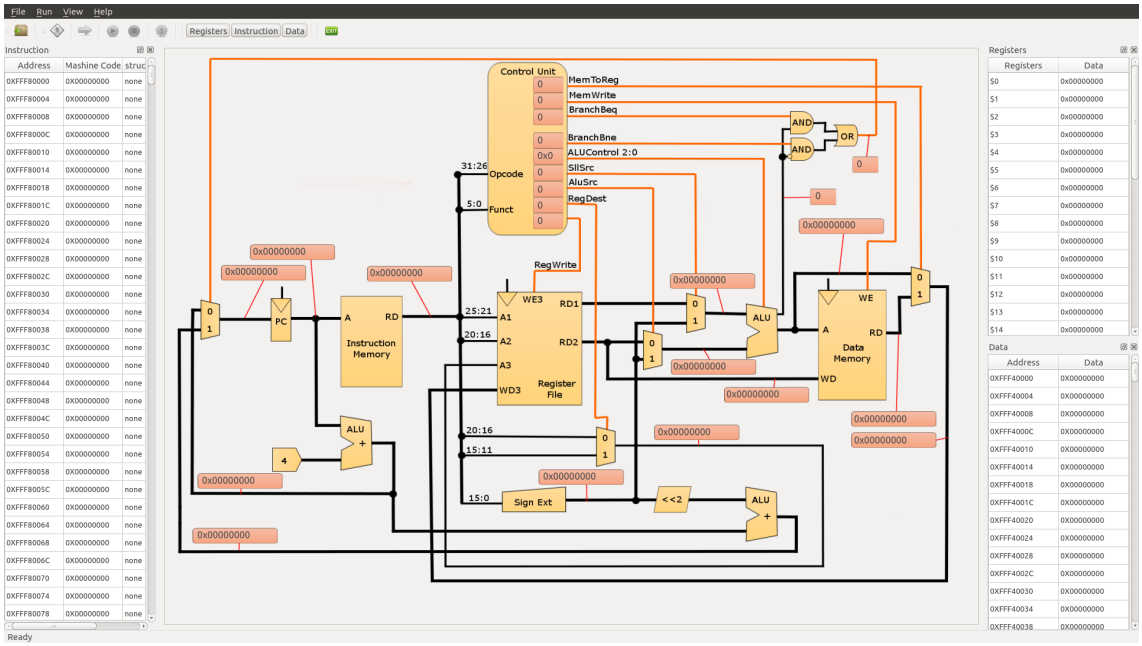


Obrázek 5: Načtený assembler soubor v editoru

4.4.3 Simulace

Simulaci je možné provádět po úspěšném překladu. V tomto modelu je možné simulovat dvěma způsoby, prvním je krokování instrukce po instrukci, druhým je spuštění kontinuálního vykonávání instrukcí z instrukční paměti. Vykonávaná instrukce je vyznačena v dokovacím okně v příslušné řádce tabulky. Při každém kroku jsou aktualizovány, jak popisky jednotlivých sběrnic v centrálním okně, tak hodnoty v datové paměti a paměti registrů. Pokud je spuštěno kontinuální vykonávání programu, je možné ho kdykoliv zastavit pomocí tlačítka Stop. Pokud je třeba, lze procesor s nahraným souborem resetovat, tím se nastaví hodnota programového čítače na bázovou adresu, a dojde k obnově původních hodnot v pamětích.

Jelikož je tento simulátor navržen pro studijní účely, má svá omezení. Jedním z nich je velikost pamětí, a to jak instrukční, tak datové. Instrukční paměť je omezena na 1024 instrukcí, z nichž každá má 32 bitů. Pokud dojde programovým čítačem k překročení maximální adresy, objeví se příslušné varovné hlášení, a dojde k resetu procesoru. Datová paměť má stejná datová omezení jako instrukční. Pokud v datové paměti bude překročena maximální adresa, dojde díky mapování k zápisu dat opět od počáteční adresy .



Obrázek 6: Okno simulátoru procesoru

5 Existující simulátory MIPS

V současné době existují desítky simulátorů procesoru MIPS [7], které jsou využívány jak k výukovým účelům, tak při vývoji reálných aplikací. Tato architektura se stále využívá díky své názornosti, jednodušší struktuře instrukcí, a také díky své názornosti při vykonávání instrukcí. Většina z nich byla navržena v jazyce C/C++, případně Java, a liší se jak velikosí instrukčních setů, tak přímo porporou různých typů procesoru. Dalším markantním rozdílem je zpracování grafického uživatelského rozhraní, a také podpora editace assembler souboru přímo v programu simulátoru, či možnost simulace nejenom zjednodušené architektury procesoru, ale také zřetězený procesor.Vybrané a nejčastěji používané simulátory jsou popsány níže.

5.1 MARS

Simulátor MARS [9] (MIPS Assembler and Runtime Simulator) byl navržen Dr. Kenem Vollmarem jež je profesorem na státní universitě v Missouri. Vyvinut byl jako alternativa a rozšíření simulátoru SPIM, pro studijní účely, a je základem populární učebnice Computer Organization and Design[4]. Funkce MARS simulátoru byly implementovány s ohledem na potřeby studentů a profesorů, pracujících s tímto programem. Tento simulátor podporuje nejen simulování assembler souborů, ale má také implementován jednoduchý textový editor pro úpravu těchto programů. Také obsahuje debugger pro lepší orientaci při simulování. Při simulování lze tedy nastavovat breakpointy na instrukce v programu.

Program je omezen na velikosti pamětí, a to jak instrukční a datové, tak i zásobníku. Limit je stanoven na 4MB pro každou paměť a jejich adresy začínají na příslušných bázových adresách. Implementace MARS nepodporuje zřetězené vykonávání instrukcí s pipeline modem, tedy není zde zavedeno pozastavení procesoru, přeposílání dat (forwarding), ani vyprazdňování pipeline.

5.1.1 Parametry

Program i jeho GUI je implementováno v jazyce JAVA, pro práci je tedy třeba stáhnout soubor typu .jar, a mít nainstalované Java JDK pro jeho spuštění. MARS podporuje většinu instrukcí a pseudoinstrukcí z MIPS-32 instrukčního setu, a dále sedmnáct systémových volání pro práci se vstupně-výstupními zařízeními. Podporované instrukce jsou uvedeny v publikaci Computer Organization and Design Third Edition v tabulkách 3.24, str. 226 a 3.25, str. 227, a také v okně nápovědy přímo v aktuální verzi programu MARS simulátoru.

5.1.2 Nástroje

V rolovacích menu File a Edit nalezneme nástroje pro práci s editorem. Nachází se zde klasické ikony pro vytvoření nového či, načtení existujícího souboru, ukládání, tisk i zavření aktivního nebo všech podoken souborů. Dále klasické nástroje pro editaci textu, kterými jsou vrácení změny zpět a vpřed, cut, copy a paste, a také je zde implementována možnost vyhledávání konkrétního textu.

Menu Run obsahuje funkce pro simulování běhu procesoru. Nástroje jsou deaktivovány až na první položku, která provádí kompilaci souboru. Pokud kompilace proběhne bezchybně jsou ostatní nástroje aktivovány pro použití. Význam a funkce těchto nástrojů je popsána v kapitole 5.1.5.

Záložka Settings obsahuje kromě nastavení pro editor, výjimky a paměť, také zaškrtávací pole pro zobrazení určitých položek v simulačním a editorovém okně, jako například návěští v textové či datové části assembler souboru, zobrazovaní adres nebo dat v decimálním či hexadecimálním tvaru, případně zpožděné vykonávání skoků a členění pseudoinstrukcí do jednotlivých instrukcí.

V menu Tools se nachází velké množství rozšiřujících nástrojů. Hlavní nevýhodou je otevírání každého nástroje ve vlastním okně. Pokud bude využito více těchto nástrojů, stává se grafické uživatelské rozhraní nepřehledným. Každý nástroj se připojuje ke zkompilovanému souboru nahranému v simulátoru pomocí tlačítka Connect to MIPS. Po stisknutí lze vykonávat simulaci, nástroj je propojen se současným programem a vykonává příslušnou funkci. Všechny nástroje se stručným popisem jsou uvedeny v Tabulce 8.

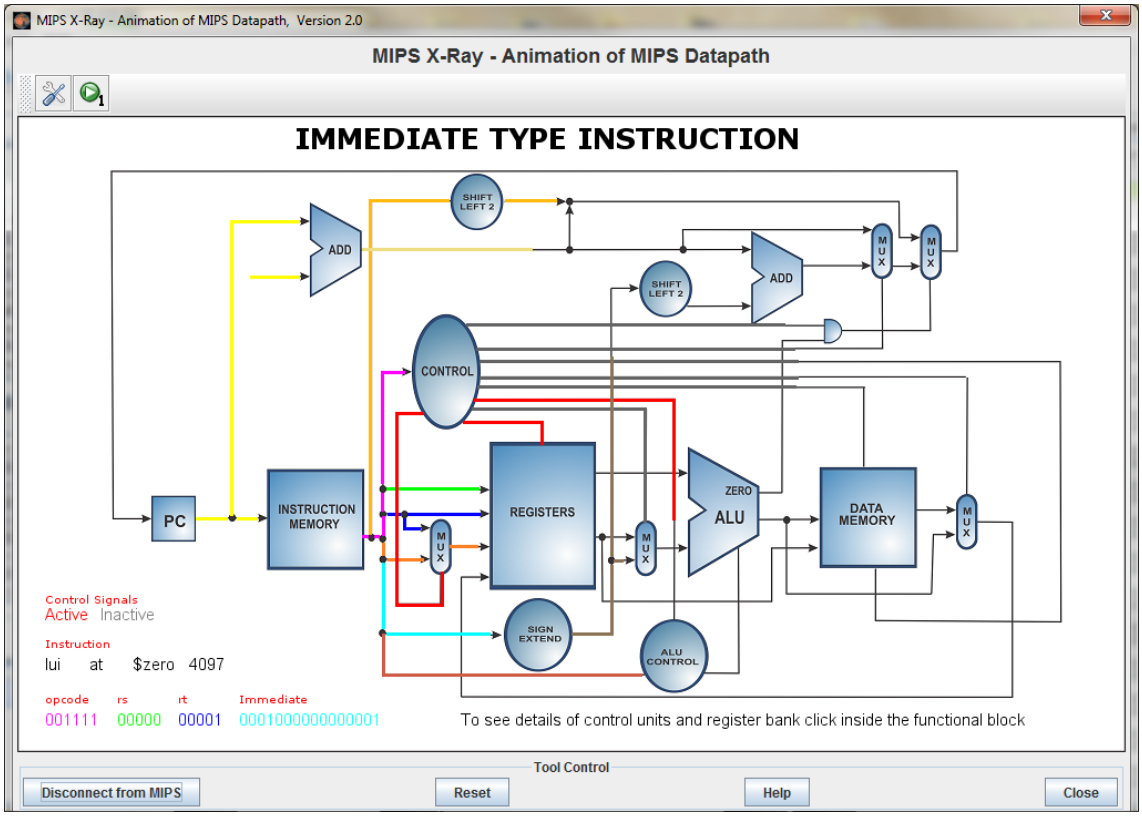
Výhodou je, že se v těchto nástrojích nachází možnost zavedení prediktoru skokových instrukcí, dále kešování datové paměti, a jako nejnázornější nástroj je MIPS X - Ray (viz Obrázek 7). V této funkci lze simulovat jednotlivé instrukce a graficky pozorovat průchod instrukce procesorem. Instrukce, se svými parametry a strojovým kódem, je uvedena v levém dolním rohu okna. Po začátku vykonávání se začne vykreslovat příslušnými barvami propagace instrukce přes jednotlivé komponenty. Dále lze také v řadiči a paměti registrů zobrazit kombinační obvody, a stejným způsobem zobrazit propagaci informace přes tyto komponenty. Tento nástroj je velmi názorný, a je to užitečná pomůcka při studiu.

5.1.3 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní je navrženo jako jedno hlavní okno s dvěma přepínatelnými záložkami pro okno editoru a simulační okno. V horní části se nachází ovládací panel s ikonami pro obsluhu běhu simulace i pro editaci. Dále zde nalezneme panel rolovacích menu, jehož funkce jsou popsány v kapitole 5.1.2. V hlavním okně je dále zobrazena tabulka hodnot registrů, na pravé straně okna a v dolní části je implementována výstupní konzole pro hlášení vstupně - výstupních zařízení a samotného simulátoru. Celé okno lze maximalizovat,

Název	Význam
BHT Simulator	Prediktor skokových instrukcí
Bitmap Display	Vykreslování na výstupní zařízení obrazovky
Data Cache Simulator	Cachování datové paměti
Digital Lab Sim	Simulátor digitálních sedmi - segmentových displejů
Floating Point Representation	Převodník čísel v plovoucí desetinné čárce
Instruction Counter	Čítač instrukcí
Instruction Statistics	Statistiky vykonaných instrukcí
Introduction to Tools	Nástroj pro implementaci vlastních funkcí
Keyboard and Display Simulator	Simulátor práce s klávesnicí a displejem
Memory Reference Visualisation	Vizualizace využití paměti
MIPS X-Ray	Grafické zobrazení vykonávání instrukce v procesoru
Screen Magnifier	Snímání pracovní plochy simulátoru

Tabulka 8: Nástroje



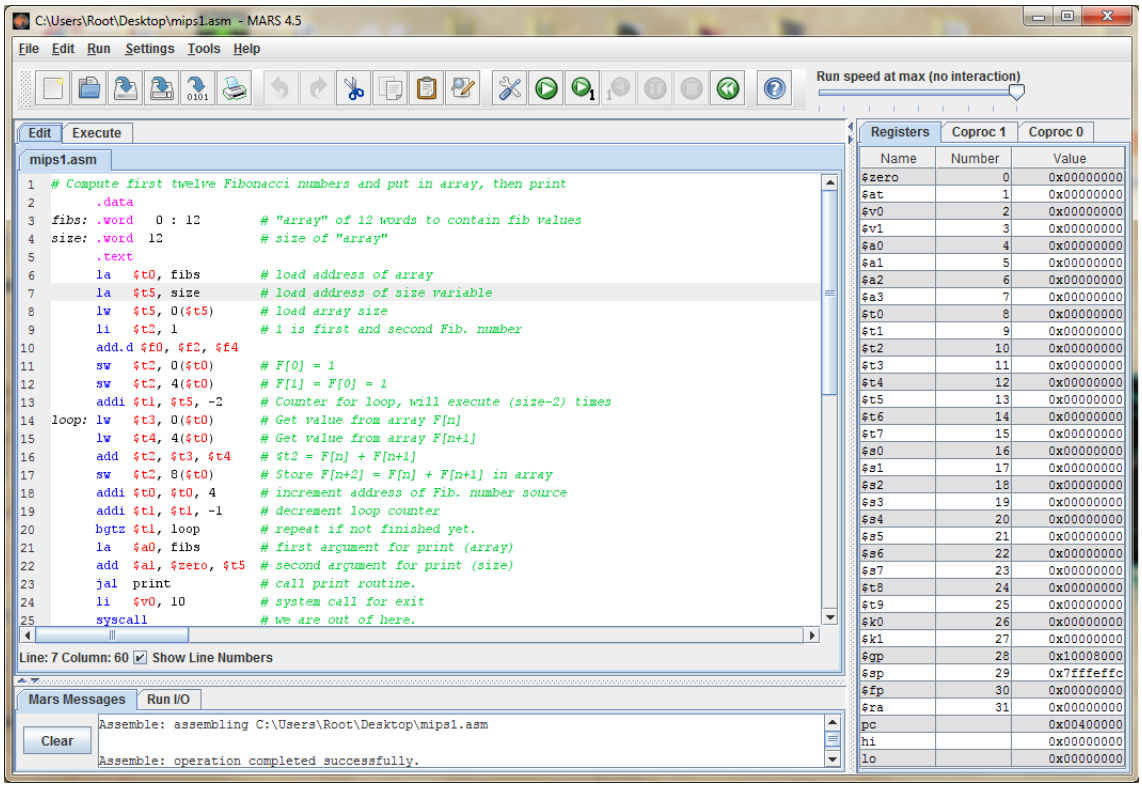
Obrázek 7: MIPS X - Ray

jednotlivým podoknům lze měnit velikost v rámci hlavního okna, avšak jejich minimální velikosti jsou omezeny. Možnost vyčlenění záložek do nového okna v tomto simulátoru není implementována.

5.1.4 Editor

Pod záložkou Edit v hlavním okně simulátoru MARS se nalézá textový editor assembler souborů. Lze zde vytvářet nové soubory, ale také editovat již navržené. Dále je zde umožněna editace či vytváření více souborů ve stejném čase, tyto soubory se v okně otevírají pod novými záložkami. Editor má implementováno zbarvování textu dle jeho významu, tedy direktivy se barví růžově, komentáře zeleně, registry červeně a instrukce jsou zbarveny modře. Ostatní text je veden klasickou černou barvou. Komentáře jsou podporovány, a lze je zapisovat za řídící znak #.

Další výhodou tohoto editoru je inteligentní doplňování a nápověda při psaní textu, a to jak instrukcí, tak direktiv. Významně tato funkce usnadňuje a zrychluje práci. Také je zde možné, na základě zaškrtávacího, pole zobrazovat lištu s čísly řádků, což je dobrá pomůcka při debuggování případných chyb v souboru. V okně pro nastavení editoru lze upravovat styly a velikost textu, podbarvování částí kódu, zobrazování nápovědy instrukcí a direktiv, a také počet mezer při psaní tabulátoru, rychlost blikání kurzoru a podbarvování označeného řádku při editaci.



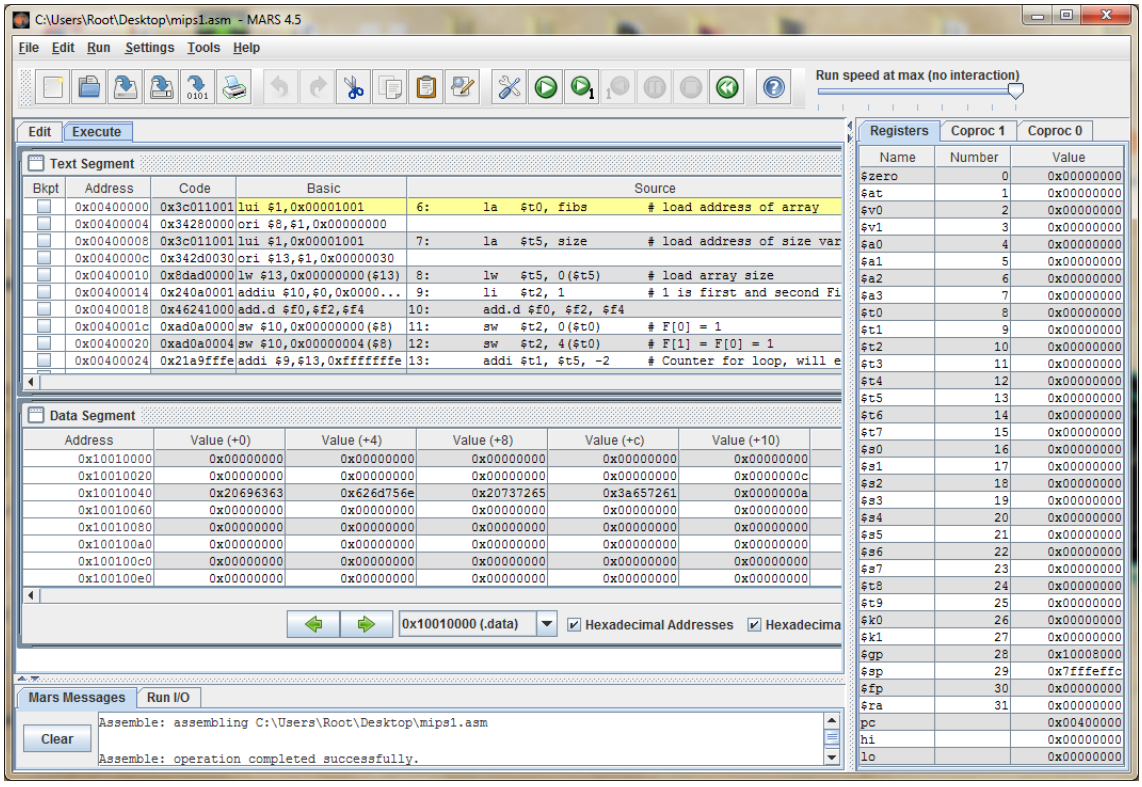
Obrázek 8: Editor simulátoru MARS

5.1.5 Simulační okno a simulace

Simulační okno je členěno na tabulku datové paměti, instrukční paměti a volitelně i názvů textových a datových návěští a jejich adres. V tabulce instrukční paměti je uvedena adresa instrukce, hexadecimální zápis a převedený a zdrojový text instrukce, včetně komentářů. Ve všech tabulkách lze přepínat decimální a hexadecimální zápis adres i dat, a také je možné aktivovat překlad dat do ASCII znaků.

Pokud byla bezchybně provedena kompilace souboru, je automaticky naplněna datová a instrukční paměť a registry. Dále jsou aktivovány nástroje pro spuštění simulace. Simulování, kromě běžných nástrojů jako jsou vykonání jedné instrukce, sekvenční vykonávání celého programu a zastavení, obsahuje ještě navíc možnost nastavení rychlosti vykonávání instrukcí pomocí posuvného jezdce v liště nástrojů. Rychlost je určena v rozmezí 0.05 až 30 instrukcí za sekundu, přičemž lze nastavit maximální rychlost, tedy bez iterace instrukcí.

Další výhodou je možnost zpětného krokování, které vrátí všechny změněné hodnoty na původní, tedy před vykonáním instrukce. Vykonávaná instrukce a změny hodnot, jak v datové paměti, tak v registrech jsou vyznačovány podbarvením pozadí příslušné buňky tabulky. Výstupy a výsledky programu, či systémová volání jsou vypisovány do konzole pod simulačním oknem.

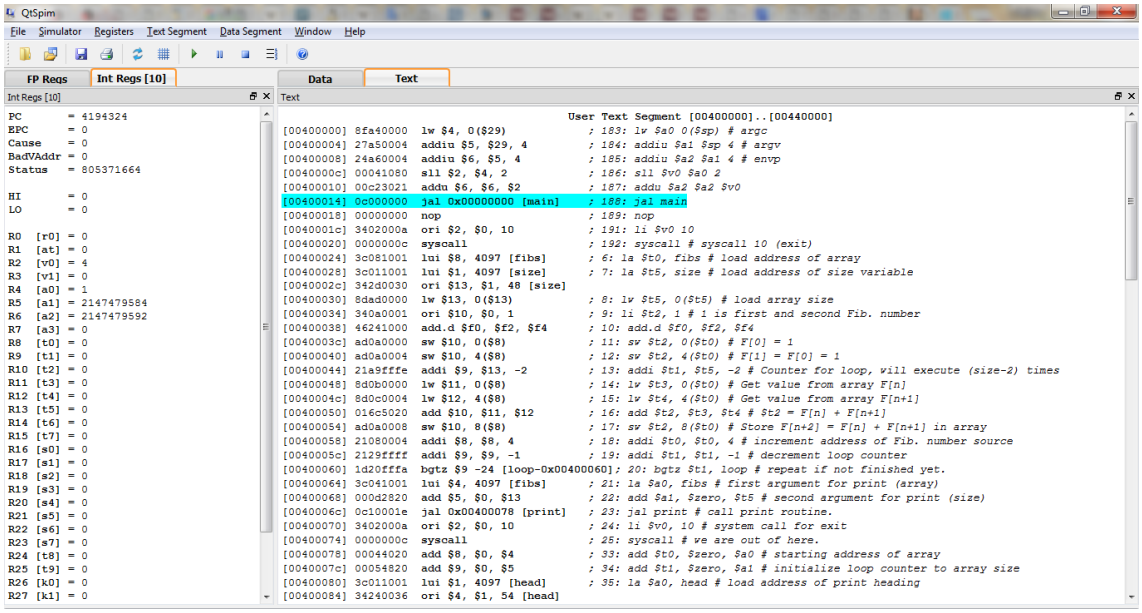


Obrázek 9: Simulační okno MARS simulátoru

5.2 SPIM

Simulátor SPIM [5] (pouze MIPS psané pozpátku) je emulátorem MIPS procesorů řady R2000 a R3000 podporující téměř úplnou sadu MIPS-32 instukcí . Jako vstup podporuje pouze assembler soubory, které překládá do strojového kódu, se kterým je dále pracováno. Program byl nejdříve implementován v jazyce C, nejnovější verze, která se nazývá QtSPIM je psána v jazyce C++ a je využito knihoven QT Frameworku pro grafické uživatelské rozhraní. SPIM byl navržen pro studijní i vývojářské potřeby, a je taktéž základem učebnice Computer Organization and Design [4].

Tento simulátor nemá zabudovaný textový editor, je tedy nutné vytvářet a upravovat soubory pomocí jiného nástroje a výsledek importovat. Po importu probíhá automatický překlad souboru do strojového kódu, včetně ošetření případných chyb. Pokud je překlad neúspěšný, je třeba soubor opravit a nahrát znovu. Tento program má čistě textovou formu, neobsahuje žádné grafické znázornění simulovaného procesoru, či průběhu vykonávání instrukce.



Obrázek 10: Simulátor SPIM

5.2.1 Grafické uživatelské rozhraní

Po spuštění nainstalovaného simulátoru jsou otevřena dvě okna, jedním je simulační okno a druhým je konzole. Simulační okno je členěno na podokno pro registry, kde se nachází vypsané hodnoty registrů, jak v celých číslech, tak v plovoucí řádové čárce. Ve druhém podokně se nachází výpis hodnot datové a instrukční paměti. Tato okna jsou

tvořena pomocí doků, lze tedy měnit jejich velikost, a také je vyčlenit do popředí jako samostatné okno. Nástroje pro ovládání simulace a práce se soubory jsou umístěny v horní liště nástrojů. Další ovládací prvky jsou umístěny v příslušných rolovacích menu.

5.2.2 Nástroje

V rolovacím menu File najdeme nástroje pro práci s assembler souborem, tedy načtení souboru, záznam o otevřených souborech, reinicializace a znovunačtení souboru, možnost uložení výsledku simulace, možnost tisku a samozřejmě tlačítko pro ukončení programu. Nejdůležitější z těchto nástrojů jsou umístěny také v panelu nástrojů. Další částí je menu Simulator, ve kterém se nachází ovládací prvky pro simulaci činnosti procesoru, tyto prvky jsou popsány v kapitole 5.2.3. Další menu, Registers, Text Segment, Data Segment a Window slouží k přepínání a ovládání jednotlivých podoken simulátoru, za pomoci příslušných zaškrtávacích polí. Za zmínku stojí možnost přepínání mezi binárním, decimálním a hexadecimálním výpisem registrů, zobrazení komentářů a dalších dat v datovém a instrukčním segmentu, a také možnost aktivace a deaktivace jednotlivých částí, s možností obnovení původního rozložení oken.

5.2.3 Simulace

Po nahrání a úspěšném překladu souboru se aktivují ikony pro simulaci. Simulovat lze jednak pomocí nástroje Step, který vykoná pouze jednu instrukci, případně pomocí nástroje Run, který bude postupně vykonávat instrukce do doby stisknutí tlačítka pro Stop, nebo dokud nedojde na konec simulovaného programu. Během simulace jsou zvýrazňovány příslušné vykonávané instrukce, a také aktualizovány hodnoty registrů a datové paměti, pokud došlo k jejich změně. Dalšími nástroji využitelnými při simulaci je vyčištění registrů, které inicializuje všechny registry na původní hodnoty a reinicializace simulátoru, která uvede simulovaný procesor do stavu těsně po nahrání assembler souboru.

5.3 MipsIt

MipsIt [1] je simulátor MIPS procesoru používaný pro výuku v předmětu A0B36APO [3]. Tento simulátor byl vyvinut na univerzitě v Lundu jako studijní pomůcka, a taktéž opírá se o publikaci Computer Organization and Design [4]. Podporuje standartní instrukční sadu MIPS-32, která je uvedena právě v této publikaci. Tento simulátor obsahuje vývojové prostředí pro psaní vlastních programů, podporu pro nahrání programu do cílového hardwaru a sérii simulátorů. Je členěn do několika provázaných spustitelných souborů (viz Tabulka 9), které se po spuštění otevírají ve vlastních oknech. Simulační programy jsou závislé na okně vývojového prostředí, ze kterého je třeba pro simulaci nahrát přeložený

kód do pamětí procesoru.

Název	Funkce
Mips.exe	Simulátor nezřetězeného procesoru
MipsIt.exe	Vývojové prostředí
MipsPipeS.exe	Simulátor zjednodušeného zřetězeného procesoru
MipsPipeXL.exe	Simulátor úplného zřetězeného procesoru

Tabulka 9: Funkce souborů MipsIt simulátoru

5.3.1 Editor

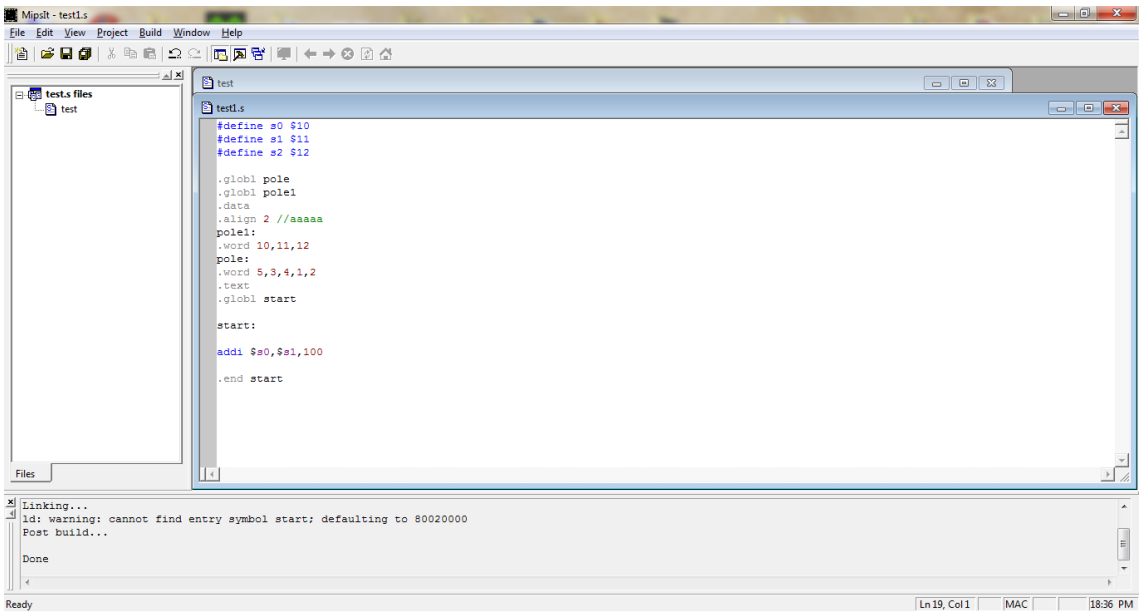
Editor je implementován v souboru MipsIt. Podporuje tvorbu programů, jak v jazyce C, tak v assembleru, jednak jako samostatné soubory, ale také je zde implementovaná správa projektů. Hlavní okno je členěné na tři části (viz Obrázek 11), vlevo nalezneme podokno se stromovou strukturou souborů projektů, v dolní části se nachází výstupní konzole a v centrální části je plocha pro editaci souborů. Soubory se otevírají ve vlastních oknech, která jsou v základu přidána do vyčleněného okna kaskádně. Jednotlivá okna lze maximalizovat v rámci vyčleněného prostoru, avšak pokud je třeba přepnout na jiné okno, je třeba v rozevíracím menu přepnout zpět na kaskádní zobrazení, editace více souborů se pak stává nepřehlednou.

Editor má implementováno zvýraznění jednotlivých částí kódu, a to konkrétně definice maker a názvy instrukcí jsou psány modře, direktivy šedě, návěští černě, číselné hodnoty červeně a definované proměnné registrů fialově. Dále je zde podporována možnost psaní komentářů za řídícím znakem //, ty jsou poté zbarveny zeleně. Editor podporuje standartní nástroje pro práci se soubory a projekty, kterými jsou například založení nových souborů a projektů, jejich ukládání a zavírání, dále také tisk souborů a uchovává naposledy použité projekty a soubory. Dále nástroje pro úpravu textu, včetně vyhledávání a nahrazování textového úseku jiným, v celém programu.

Pro nahrání je třeba napsaný kód nejprve přeložit, a poté zkompilovat, pokud proběhne vše v pořádku je nutné program nahrát do příslušného simulátoru, kde je poté možné program simulovat.

5.3.2 Simulace

Simulaci je možné provádět na nezřetězeném procesoru, případně na zjednodušeném nebo úplném zřetězeném procesoru. Pro každou možnost je třeba spustit příslušný soubor (viz Tabulka 9), a je nutné mít pro simulaci nahraný přeložený program v paměti procesoru.

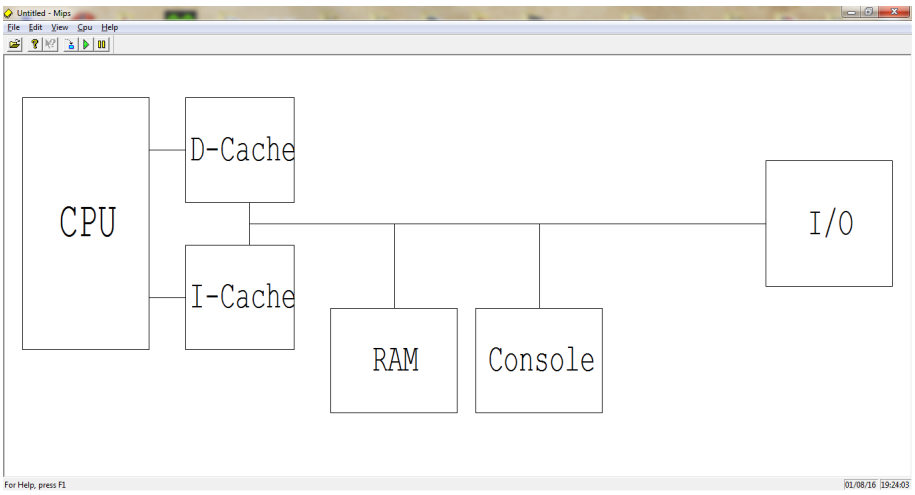


Obrázek 11: Editor Mips

Úvodní okna všech simulátorů jsou shodná (viz Obrázek 12), a obsahují stejné aktivní a ovládací prvky.

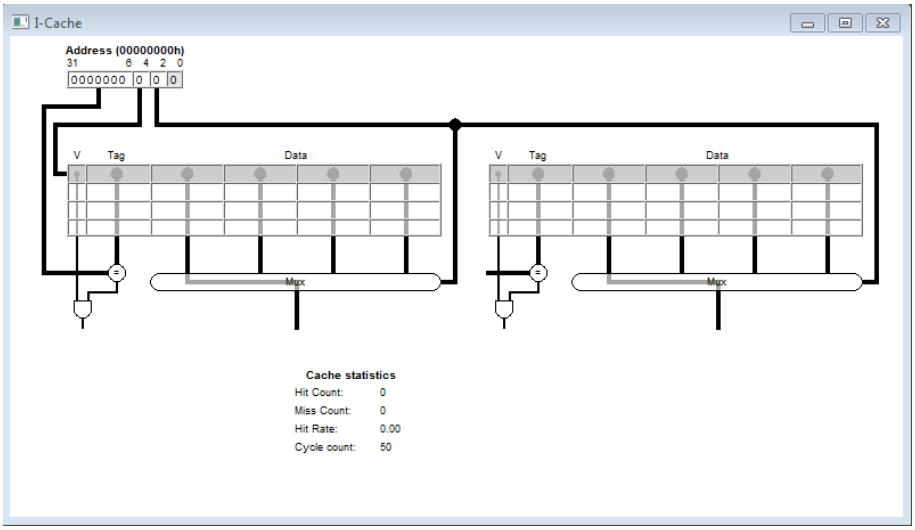
Simulátor podporuje nastavení pro cacheování instrukční a datové paměti, nastavovat lze celkovou velikost i velikost bloku, dále počet bloků v setu a algoritmus nahrazování prvků v paměti. U datové paměti lze ještě nastavit algoritmus zápisu změněných dat do paměti vyšší úrovně. Po nastavení příslušných cache pamětí se upraví po rozkliknutí bloky reprezentující tyto paměti (viz Obrázky 13 a 14). V bloku RAM (viz Obrázek 15), nalezneme zobrazení paměti, přičemž jsou zde uvedeny adresy v hexadecimálním zápisu, obsah paměťové buňky a textový zápis. Vykonávaná instrukce je v tabulce vyznačena modrým podbarvením, a při vykonávání zde lze nastavovat breakpointy. V bloku I/O je reprezentována vstupně výstupní sběrnice, vstupní sběrnici nahrazují graficky zpracované spínače a výstupní LED diody (viz Obrázek 16). Simulovat nahraný program lze pomocí funkcí pro vykonání jedné instrukce (Step), případně pomocí Run, kdy se začnou vykonávat sekvenčně všechny instrukce nahrané v instrukční paměti. Tuto funkci zastavíme pomocí nástroje Stop.

Jednotlivé programy se liší v bloku CPU. V simulátoru zjednodušeného procesoru se nachází tabulka s hodnotami jednotlivých registrů, čítače instrukcí a dalších hodnot. Není zde graficky zobrazena podoba simulovaného procesoru. V programu pro simulaci zjednodušeného a úplného procesoru se zřetězením je graficky zpracováno pozadí okna (viz Obrázky 17 a 18). Jsou zde umístěna textová pole, do kterých jsou vypisovány hodnoty příslušných sběrnic při vykonávání jednotlivých částí instrukce. Zároveň jsou na přepínačích znázorněny aktivované cesty na základě kontrolního signálu. Blok řadiče není uveden u

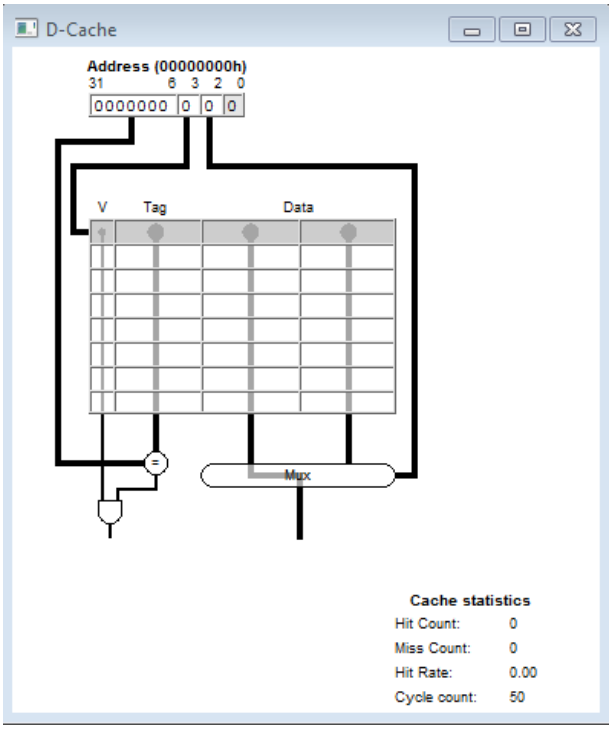


Obrázek 12: Simulační okno

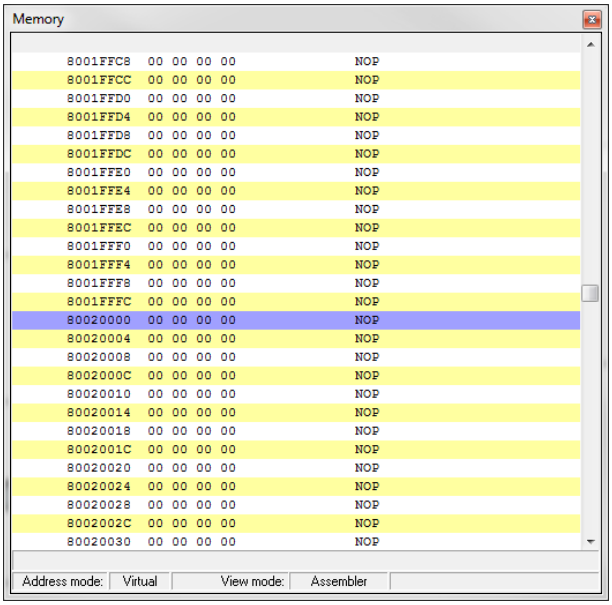
zjednodušeného procesoru, a to hlavně z důvodu přehlednosti schématu.



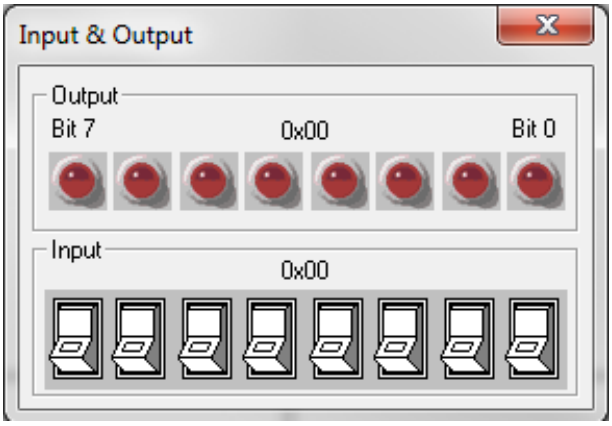
Obrázek 13: Datová paměť



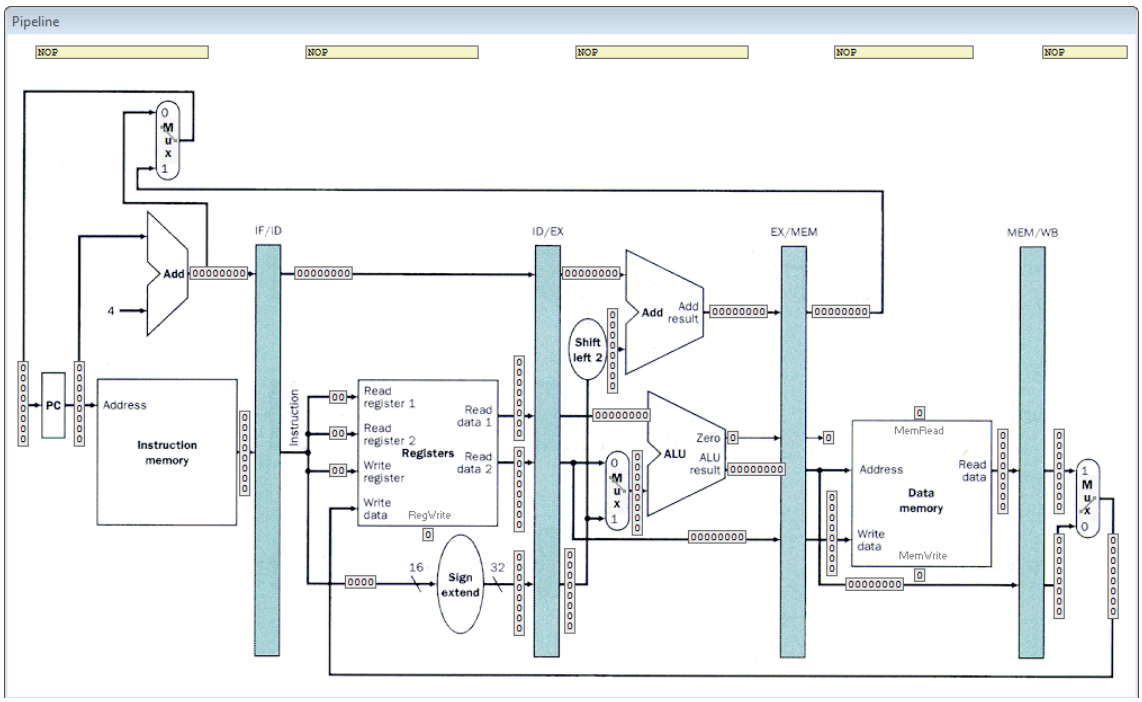
Obrázek 14: Instrukční paměť



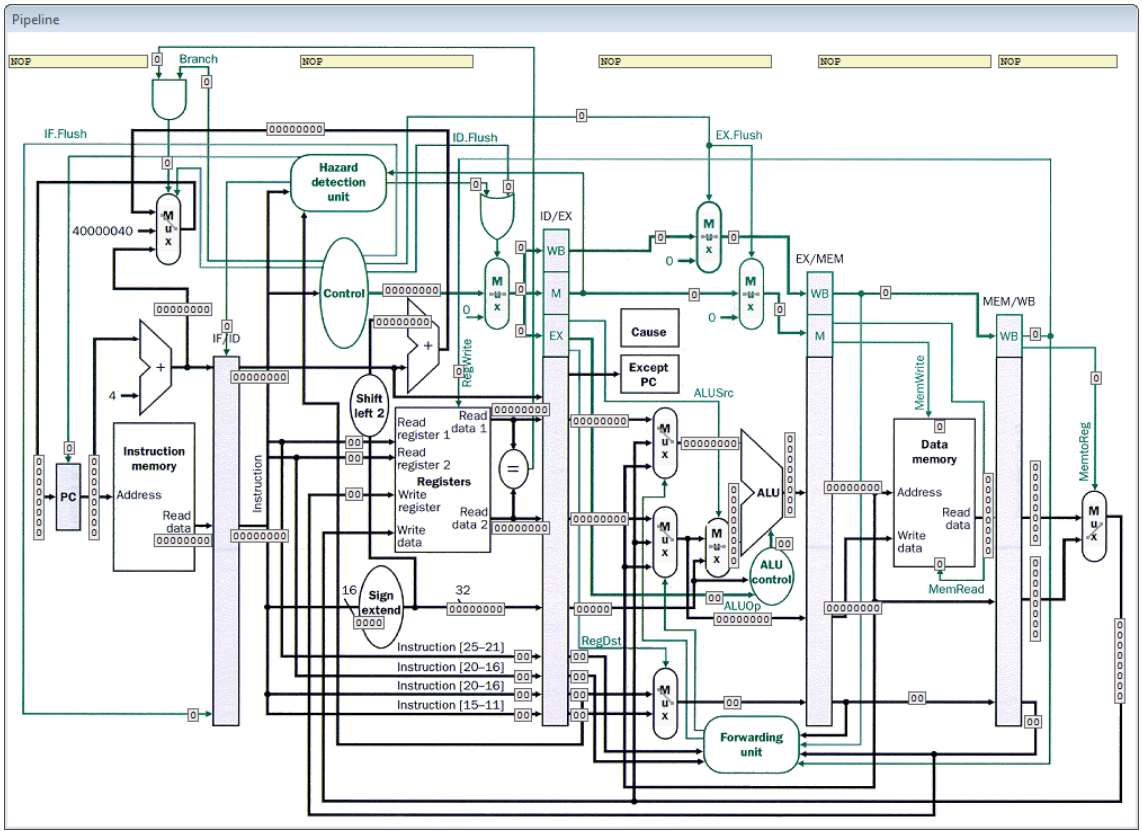
Obrázek 15: Paměť RAM



Obrázek 16: Vstupně výstupní zařízení



Obrázek 17: CPU blok zjednodušeného zřetězeného procesoru



Obrázek 18: CPU blok úplného zřetězeného procesoru

6 Shrnutí

Hlavním cílem této práce bylo navrhnout a realizovat simulátor MIPS procesoru, včetně grafického uživatelského rozhraní. Dalším cílem bylo nalézt volně dostupné a používané simulátory těchto procesorů a nalézt jejich výhody a nevýhody. Tento simulátor se má stát pomůckou studentů při výuce předmětu A0B36APO. Navrhl jsem tedy grafické rozhraní a výpočetní část, které podporují jak simulování vlastních programů v assembleru, tak návrh vlastního kódu, který je možné do simulátoru nahrát.

Grafické rozhraní bylo navrženo a implementováno pomocí QT frameworku a QT Creatoru. Program obsahuje hlavní okno, ze kterého je možné přejít do okna editoru a simulátoru. V editoru je možné upravovat existující soubory assembleru, případně vytvářet vlastní. V okně simulátoru je možné tyto programy simulovat, sledovat průchod instrukcí procesorem a jejich výsledky.

Funkčnost simulátoru byla otestována na referenčním programu, který je přiložen v příloze na CD.

Při zpracování a porovnání jednotlivých simulátorů jsem zjistil, že většina pracuje pouze s textovým výstupem. Pro studijní účely jsou lépe využitelné simulátory s grafickou reprezentací procesoru jako například MipsIt a MARS, které jsou díky tomu názornější a lze snadněji pochopit chování procesoru a jeho částí při zpracování instrukcí. Další výhodou je implementace editoru souborů a možnost simulování zřetězeného procesoru.

Během zpracování této práce jsem zjistil, že pochopení a správná implementace simulátoru je náročná a vyžaduje mnoho času a pozornosti. Avšak simulace zvláště ve vhodném simulačním prostředí je poté velmi dobrou pomůckou při studiu principu a funkcionality každého problému.

7 Literatura

[1] M. Brorsson. Mipsit - a simulation and development environment using animation for computer architecture education. https://www.ncsu.edu/wcae/ISCA2002/submissions/brorsson.pdf, 2016.

[2] The QT company. https://www.gimp.org/about/, 2016.

[3] České vysoké učení technické. https://cw.fel.cvut.cz/wiki/courses/a0b36apo/start, 2016.

[4] Patterson D. A. Hennesy, J. L. *Computer Architecture: A Quantitative Approach, Third Edition*. Morgen Kaufmann Publishers, Inc., San Francisco, 2002.

[5] James R. Larus. http://http://spimsimulator.sourceforge.net/, 2016.

[6] Imagination Technologies Limited. https://imgtec.com/about/, 2016.

[7] MediaWiki. http://courses.missouristate.edu/kenvollmar/mars/index.htm, 2016.

[8] The GIMP Team. https://www.qt.io/qt-framework/, 2016.

[9] Sanders P. Vollmar, K. Mars: An education-oriented mips assembly language simulator. http://www.facom.ufms.br/ ricardo/Courses/CompArchII/Tools/Mars/fp288-vollmar.pdf, 2016.

Příloha A Obsah CD

V Tabulce 10 jsou uvedeny názvy složek a popis jejich obsahu na přiloženém CD.

Název složky	**Popis**
práce	Bakalářská práce v pdf formátu
práce_zdroj	Zdrojové kódy práce v latexu
GUI	C++ zdrojové kódy programu
grafika	Obrázky užité v bakalářské práci

Tabulka 10: Obsah CD

Příloha B Seznam zkratek

V Tabulce 11 je uveden seznam zkratek použitý v této práci.

Zkratka	**Význam**
MIPS	Microprocessor without Interlocked Pipeline Stages
GUI	Graphical User Interface
MARS	MIPS Assembler and Runtime Simulator
SPIM	Název MIPS simulátoru psaný pozpátku

Tabulka 11: Seznam zkratek