

Spätný prekladač (disassembler) pre MIPS architektúru

KRYŠTOF KISS

Fakulta informatiky a informačných technológií, Slovenská Technická Univerzita

Abstrakt:

Cieľom práce je vytvoriť spätný prekladač pre MIPS architektúru, ktorý dokáže kategorizovať jednotlivé typy inštrukcií a následne podľa toho dekodovať použité registre, funkcie a dáta. V prípade reverzného inžinierstva sú tieto informácie dôležité pre programátora, napr. pri odstraňovaní chýb v programe alebo pre optimalizáciu programu. Práca sa ďalej zaoberá klasifikáciou architektúry. V práci taktiež opisujeme jednotlivé komponenty architektúry, formáty inštrukcií alebo aj inštrukčný cyklus.

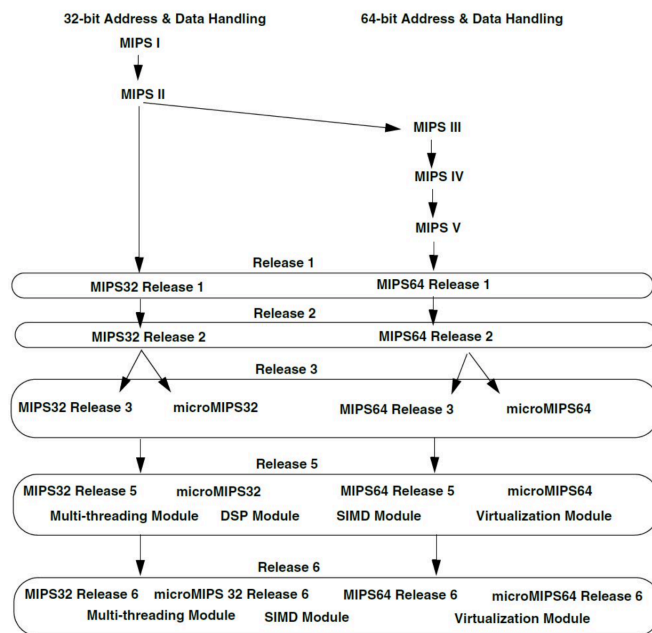
Kľúčové slová: spätný prekladač MIPS, MIPS architektúra, RISC procesor

1 ÚVOD

MIPS celým názvom: Microprocessor without Interlocked Pipe Stages. Hlavným cieľom návrhu tejto architektúry bol vysoký výkon pri vykonávaní skompilovaného kódu. Základnou filozofiou MIPSu bolo predstaviť inštrukčnú sadu, ktorá je riadená kompilátorom mikroprogramu. Preto nie je potrebné žiadne alebo minimálne dekodovanie, inštrukcie úzko zodpovedajú mikroinštrukciám. Procesorové fázy sú zreťazené ale neposkytujú interlock hardvér, preto musí byť táto funkcionality implementovaná softvérovo [1].

1.1 História

Prvú verziu MIPS architektúry navrhla spoločnosť MIPS Computer Systems pre svoj mikroprocesor R2000. Rokmi sa architektúra ďalej rozvíjala. Na **Obrázku 1** vidno rozvoj architektúry MIPS. Spočiatku architektúra pracovala v 32-bitovom režime, od verzie MIPS III sa preorientovala na 64-bitový režim. Od Releaseu 1, sa súbežne uvoľňovala 32-bitová aj 64-bitová architektúra.



Obrázok 1: Vývoj MIPS architektúry [2].

1.2 Použitie v priemysle

MIPS procesory mali a majú priemyselné využitie, použili sa v zariadeniach ako sú napríklad: SGI Indigo (MIPS R3000), Sony PlayStation (MIPS R3000), Nintendo 64 (MIPS R4300i), NEC Cenju-4 (MIPS R10000) alebo dokonca aj v automobile značky Tesla Model S (MIPS I-class) [3].

2 KLASIFIKÁCIA ARCHITEKTÚRY

Architektúru vieme klasifikovať na základe niekoľkých atribútov, ktorými sú napríklad práca s operandami, štruktúra pamäte alebo rozsiahlosť inštrukčnej sady.

2.1 Klasifikácia podľa operandov

Nasledovné štyri klasifikácie patria medzi hlavné klasifikácie architektúry podľa operandov.

2.1.1 Memory-to-memory

Architektúry tohto typu (VAX, PDP) umožňujú prístup do pamäte viac ako jednému operandu. Na jednu inštrukciu môžu prístupovať do pamäte až tri operandy, pričom to môžu byť registre alebo hlavná pamäť [4].

2.1.2 Register-memory

Táto architektúra (x86) umožňuje prístup do pamäte iba jednému operandu. Ďalšie operandy musia pristupovať k registrom procesora. Väčšina procesorov s pamäťou Register-memory povoľuje na jednu inštrukciu najviac dva operandy.

V porovnaní s Memory-to-memory architektúrou sú tieto procesory lacnejšie. Jednotlivé inštrukcie sa vykonávajú rýchlejšie, kvôli menšiemu počtu prístupov do pamäte. Na druhú stranu potrebujú vykonať viacero inštrukcií na dokončenie rovnakej úlohy [4].

2.1.3 Load-store

Load-store architektúra (MIPS, ARM) nedovoľuje väčšine inštrukcií pristúpiť do pamäte. Do pamäte majú prístup len inštrukcie LOAD a STORE, ktoré presúvajú dáta medzi registrami a hlavnou pamäťou. Všetky ostatné inštrukcie používajú na čítanie a na zapisovanie dát registre.

Väčšina inštrukcií sa vykonáva veľmi rýchlo, pretože pracujú s registrami, ktoré sú priamo v procesore. Avšak, táto architektúra potrebuje vykonať najväčší počet inštrukcií na dokončenie danej úlohy, pretože potrebujú načítať operandy z pamäte, vykonať operácie nad dátami a uložiť výsledky do pamäte [4].

2.1.4 Architektúra s akumulátorom

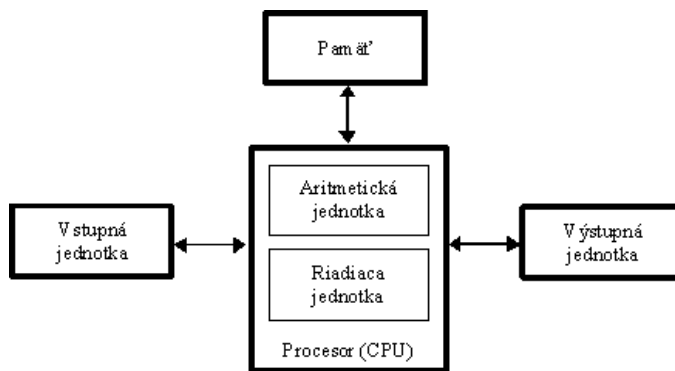
Architektúra s akumulátorom (8051) je jedno operandová architektúra a dovoľuje iba jeden operand na inštrukciu. V prípade potreby použitia dvoch operandov existuje špeciálny register: akumulátor, ktorý je implicitný [4].

2.2 Klasifikácia podľa štruktúry pamäte

Z pohľadu štruktúry pamäte existujú dve hlavné rozdelenia.

2.2.1 Von-Neumannova architektúra

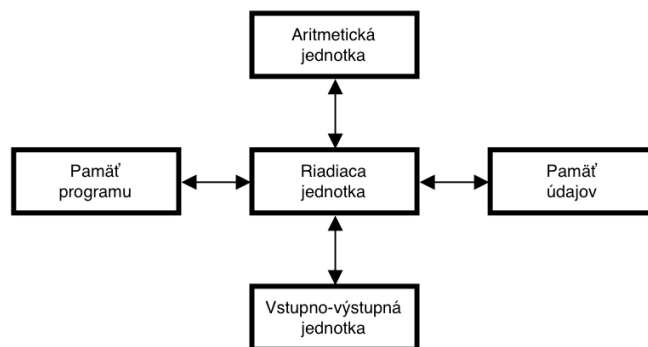
Von-Neumannova architektúra má len jednu pamäť, ktorá obsahuje inštrukcie programu a zároveň aj dáta. Je pomenovaná podľa Johna Von Neumanna, maďarsko-amerického matematika. Väčšina dnešných počítačov používa túto architektúru [4].



Obrázok 2: Von-Neumannova architektúra [5].

2.2.2 Harvardská architektúra

V harvardskej architektúre sú inštrukcie programu a dáta uložené v dvoch oddelených pamäťových priestoroch. Využíva sa najmä v mikropočítačoch, kde program (firmware) je uložený napríklad vo Flash pamäti, takže v nej ostane aj po dobu, keď je počítač vypnutý, dáta sa nachádzajú v hlavnej pamäti [4].



Obrázok 3: Harvardská architektúra.

2.3 Klasifikácia podľa veľkosti inštrukčnej sady

Architektúry delíme podľa veľkosti inštrukčnej sady do dvoch kategórií.

2.3.1 RISC (Reduced Instruction Set Computers)

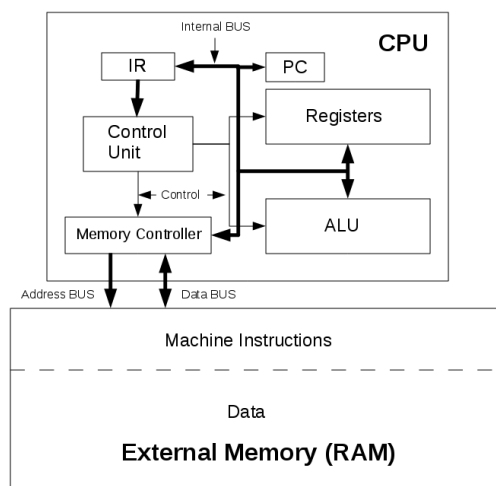
Počítače typu RISC majú oveľa menší počet inštrukcií oproti CISC počítačom. Počet inštrukcií sa pohybuje v ráde desiatok. Operácie sa vykonávajú nad dátami v registroch, ktoré sú typické tým, že ich je väčšie množstvo. Zvyčajne sa používajú v architektúrach s organizáciou load/store, znamená že s hlavnou pamäťou pracujú len inštrukcie load a store. Väčšina inštrukcií sa vykoná v jednom alebo v malom počte inštrukčných cyklov [4].

2.3.2 CISC (Complex Instruction Set Computers)

Počítače typu CISC majú spravidla väčšiu inštrukčnú sadu oproti RISC počítačom, tento počet sa pohybuje v rádoch stoviek. Používajú sa v architektúrach s organizáciou memory-to-memory alebo register-memory. Inštrukcie potrebujú viacero inštrukčných cyklov na ich vykonanie, čiastočne kvôli schopnosti získať prístup do pamäti [4].

3 ARCHITEKTÚRA MIPS

V predošlej kapitole sme rozdelili architektúry podľa niekoľkých atribútov. V tejto kapitole sa budeme zaoberať konkrétne s MIPS architektúrou, ktorá z pohľadu klasifikácie operandov sa zaraďuje medzi Load-store architektúru. Z pohľadu štruktúry pamäte sa zaraďuje do Harvardskej architektúry a z pohľadu veľkosti inštrukčnej sady sa zaraďuje do kategórie RISC. Architektúra MIPS obsahuje tridsaťdva univerzálnych registrov. Tieto registre majú veľkosť 32-bitov.



Obrázok 4: Vnútna štruktúra MIPS architektúry [4].

3.1 Pamäť

Väčšina dnešných počítačov používa bajtovú adresovateľnú pamäť, znamená že každá pamäťová adresa obsahuje 8 bitov dát. Teda 32-bitové slovo je uložené na štyroch pamäťových adresách.

Pri ukladaní hodnoty na viacero adries si musíme zvoliť poradie bitov, existujú 2 typy systémov.

- Big-endian systémy ukladajú najvýznamnejší bit na prvú pozíciu.
- Little-endian systémy ukladajú najmenej významný bit na prvú pozíciu.

MIPS architektúra s ktorou budeme pracovať pri spätnom preklade ukladá 32-bitov na jednu adresu, to sa rovná 4 bajty.

3.2 Dátové registre

Veľa procesorov zvyčajne obsahuje 8, 16 alebo 32 registrov na ukladanie dát. Keďže sú registre súčasťou procesora, prístup ku nim je veľmi rýchly v porovnaní napríklad s prístupom do pamäte RAM. Keď je to možné,

programátori sa snažia využiť tento typ pamäte pre optimalizáciu rýchlosti. V nasledovnej tabuľke je rozpisáných 32 registrov architektúry MIPS.

Číslo registra	Názor registra	Použitie
\$0	\$zero	Konštanta 0
\$1	\$at	Rezervované pre pseudo inštrukcie
\$2, \$3	\$v0, \$v1	Návratové hodnoty z funkcií
\$4 - \$7	\$a0 - \$a3	Argumenty do funkcií
\$8 - \$15	\$t0 - \$t7	Dočasné dáta (nezachované pre podprogramy)
\$16 - \$23	\$s0 - \$s7	Dáta (zachované pre podprogramy)
\$24, \$25	\$t8, \$t9	Dočasné dáta (nezachované pre podprogramy)
\$26, \$27	\$k0, \$k1	Rezervované pre jadro (nepoužívať!!!)
\$28	\$gp	Globálny ukazovateľ
\$29	\$sp	Zásobníkový ukazovateľ
\$30	\$fp	Rámcový ukazovateľ
\$31	\$ra	Návratová adresa

Tabuľka 1: Zoznam všeobecne použiteľných registrov [4].

3.3 Počítadlo (PC)

Počítadlo ukazuje na nasledovnú adresu na ktorej sa nachádza inštrukcia, ktorá sa má vykonať, v MIPS architektúre sa toto počítadlo inkrementuje o hodnotu 4, pretože ako sme spomenuli vyššie, MIPS používa 4 bajtové adresy.

3.4 Register inštrukcií (IR)

Register inštrukcií je špeciálny register, určený na držanie aktuálne spracováanej inštrukcie, zatiaľ čo sa inštrukcia nachádza v tomto registri sa dekoduje a následne vykoná.

3.5 Riadiaca jednotka (CU)

Na základe operačného kódu a operandov inštrukcie v registri inštrukcií vyšle riadiace signály do ostatných častí procesora na vykonanie danej inštrukcie.

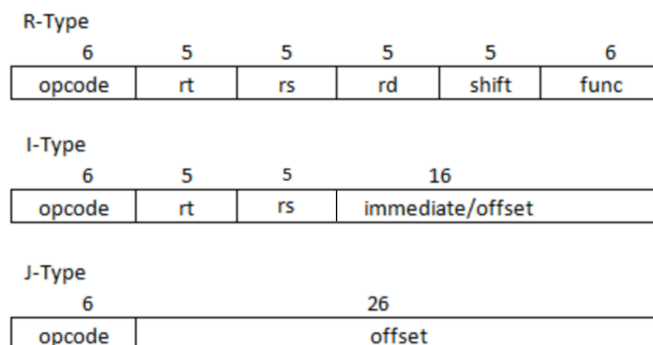
3.6 Aritmeticko logická jednotka (ALU)

Je kombinačný obvod schopný vypočítať rôzne aritmetické a logické funkcie. Ktorými sú napríklad: sčítanie, odčítanie, násobenie, delenie, logický and, logický or, zero test [6].

4 INŠTRUKCIE

MIPS procesory používajú pevnú dĺžku inštrukcie. Všetky inštrukcie majú rovnakú veľkosť 32-bitov. Inštrukcie s pevnou dĺžkou majú výhodu jednoduchšieho dekódovania, čo v praxi znamená väčšiu rýchlosť.

Inštrukcie majú variabilný formát, konkrétne sú rozdelené do troch kategórií, ktorými sú registrové inštrukcie, skokové inštrukcie a inštrukcie s priamym operandom. Na nasledovnom obrázku vidno štruktúru jednotlivých formátov.



Obrázok 5: Formáty inštrukcií architektúry MIPS.

Všetky 3 formáty začínajú s polom bitov vyjadrujúci opcode inštrukcie. Na základe tohto opcode (6 bitov) vieme kategorizovať typ inštrukcie.

4.1 R-Formát

R-formát inštrukcie je definovaný opcodeom: 000000, za opcodeom nasledujú 3 operandy, ktorými sú registre. Dva zdrojové a jeden cieľový register. Ako sme už spomenuli vyššie, MIPS používa 32 všeobecne použiteľných registrov, práve preto tieto registre sú reprezentované 5-bitmi. Ďalším polom inštrukcie je pole shift (5 bitov), používa sa pri posunových inštrukciách a vyjadruje hodnotu o koľko bitov sa má posunúť hodnota v zdrojovom registri doprava alebo doľava. Posledným polom je func (6 bitov), všetky inštrukcie typu R majú zakódovanú funkciu v posledných šiestich bitoch. Na základe tejto funkcie sa určí operácia, ktorá sa má vykonať.

4.2 J-Formát

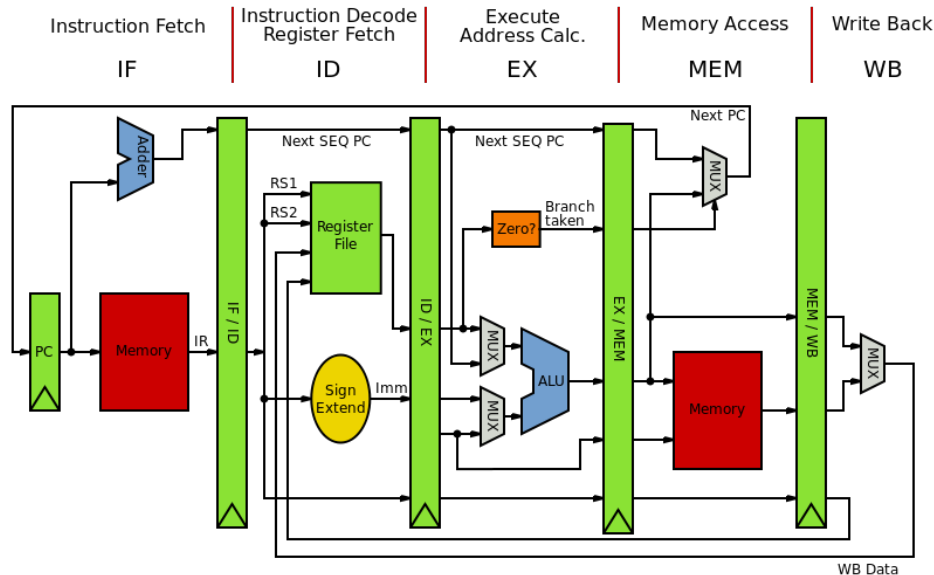
J-formát inštrukcie používa opcode: 00001x, kde x reprezentuje 0 alebo 1. Tento formát inštrukcií vykonáva nepodmienečný skok, kde zvyšných 26 bitov reprezentuje offset.

4.3 I-Formát

Ak 6 najvýznamnejších bitov (opcode) sa nezhoduje so žiadnymi z predošlých formátov tak sa jedná o inštrukciu typu I. Zároveň opcode určuje operáciu, ktorú má inštrukcia vykonať. Inštrukcia používa jeden zdrojový a jeden cieľový register. Posledných 16 bitov slúži na bezprostredný operand, ktorý je zadaný priamo v kóde programátorom [4, 7].

5 INŠTRUKČNÝ CYKLUS

Inštrukcia v MIPS architektúre sa spracováva v piatich krokoch: načítanie inštrukcie z pamäte, dekodovanie inštrukcie, vykonanie inštrukcie, zápis do pamäte a zápis do registrov.



Obrázok 6: Päť kroková schéma inštrukčného cyklu [8].

V prvom kroku sa prečíta inštrukčný kód z adresy uloženej v PC registri. V druhom kroku sa určí opkód/kód funkcie, na základe ktorých sa určia registre v ktorých sa nachádzajú operandy a s ktorými sa bude pracovať. V treťom kroku sa vykoná funkcia inštrukcie. Vo štvrtom kroku sa pristupuje do pamäte, inkrementuje sa počítadlo. V poslednom kroku sa výsledok zapíše do registrov. Zaujímavosťou je že jednotlivé fázy sa môžu vykonávať naraz v jednom hodinovom cykle. S týmto prichádzajú aj hazardy, ktoré v tejto práci nebudeme rozoberať [1, 9].

6 EXISTUJÚCE RIEŠENIA

Na internete sme našli množstvo projektov, ktoré robia spätný prekladač pre MIPS architektúru. Vybrali sme si 2, ktoré v tejto časti práce opíšeme. Oba projekty sú online aplikácie dostupné na adresách: [Mipsdis](#) a [MIPS Converter](#).

6.1 Mipsdis

Mipsdis je nástroj bežiaci v prehliadači. Skladá sa z dvoch zobrazovacích častí. V prvej časti sa nachádza vstup definovaný adresou a inštrukciou. V druhej časti sa nachádza symbolická inštrukcia. Ukážka programu sa nachádza na nasledovnom obrázku:

80001000: 3c048000	lui \$4,0x8000
80001004: 34844000	ori \$4,\$4,0x4000
80001008: 3c05b400	lui \$5,0xb400
8000100c: 34a503f8	ori \$5,\$5,0x3f8
80001010: 80860000	lb \$6,0(\$4)
80001014: 10c00005	beq \$6,\$0,0x8000102c
80001018: 00000000	sll \$0,\$0,0x0
8000101c: a0a60000	sb \$6,0(\$5)
80001020: 24840001	addiu \$4,\$4,1
80001024: 08000404	j 0x80001010
80001028: 00000000	sll \$0,\$0,0x0
8000102c: 3c05bfbf	lui \$5,0xbfbf
80001030: 34a50004	ori \$5,\$5,0x4
80001034: 3c060000	lui \$6,0x0
80001038: 34c6002a	ori \$6,\$6,0x2a
8000103c: a0a60000	sb \$6,0(\$5)

Disassemble

Obrázok 7: Ukážka nástroja Mipsdis.

Z obrázku vidieť, že nástroj nepoužíva pomenovanie registrov, musíme si ho dodatočne preložiť podľa čísla registra. Nástroj je jednoduchý na používanie.

6.2 MIPS Converter

MIPS Converter funguje aj ako assembler aj ako disassembler. Toto riešenie je navrhnuté skôr pre edukačné účely. Pri spätnom preklade nástroj nielenže dekoduje inštrukciu ale aj vypíše inštrukciu v binárnej forme, graficky vykreslí formát inštrukcie s využitými bitmi a vysvetlí načo sa daná inštrukcia používa. Nevýhodou tohto riešenia je, že dokážeme disassemblovať len po jednej inštrukcii, pri väčšom programe je dosť otravné písať inštrukcie po jednom.

MIPS Converter

Instruction ⇒ Hex

ex. add t1 t2 t3, addi t1 t2 0xffff, j 0x02ffff

Convert

Hex ⇒ Instruction

ex. 0x014B4820

Convert

Obrázok 8: MIPS Converter.

Ukážka MIPS Convertera

Na vstup sme zadali nasledovnú inštrukciu v hexadecimalnom tvare: 3c048000. Výstup nástroja je na nasledovnom obrázku:

LUI \$a0 0x8000

Binary: 00111100000001001000000000000000

Hex: 0x3c048000

31	26 25	21 20	16 15	0
LUI	0	\$a0	immediate	
001111	00000	00100	1000000000000000	
6	5	5	16	

LUI

Load Upper Immediate

Format: LUI rt, immediate [I-type] MIPS Architecture Extension: MIPS I

31	26 25	21 20	16 15	0
LUI	0	rt	immediate	
001111	00000			
6	5	5	16	

Purpose:

To load a constant into the upper half of a word.

Description:

rt <- immediate || 016.

The 16-bit immediate is shifted left 16 bits and concatenated with 16 bits of low-order zeros. The 32-bit result is sign-extended and placed into GPR rt.

Restrictions:

None.

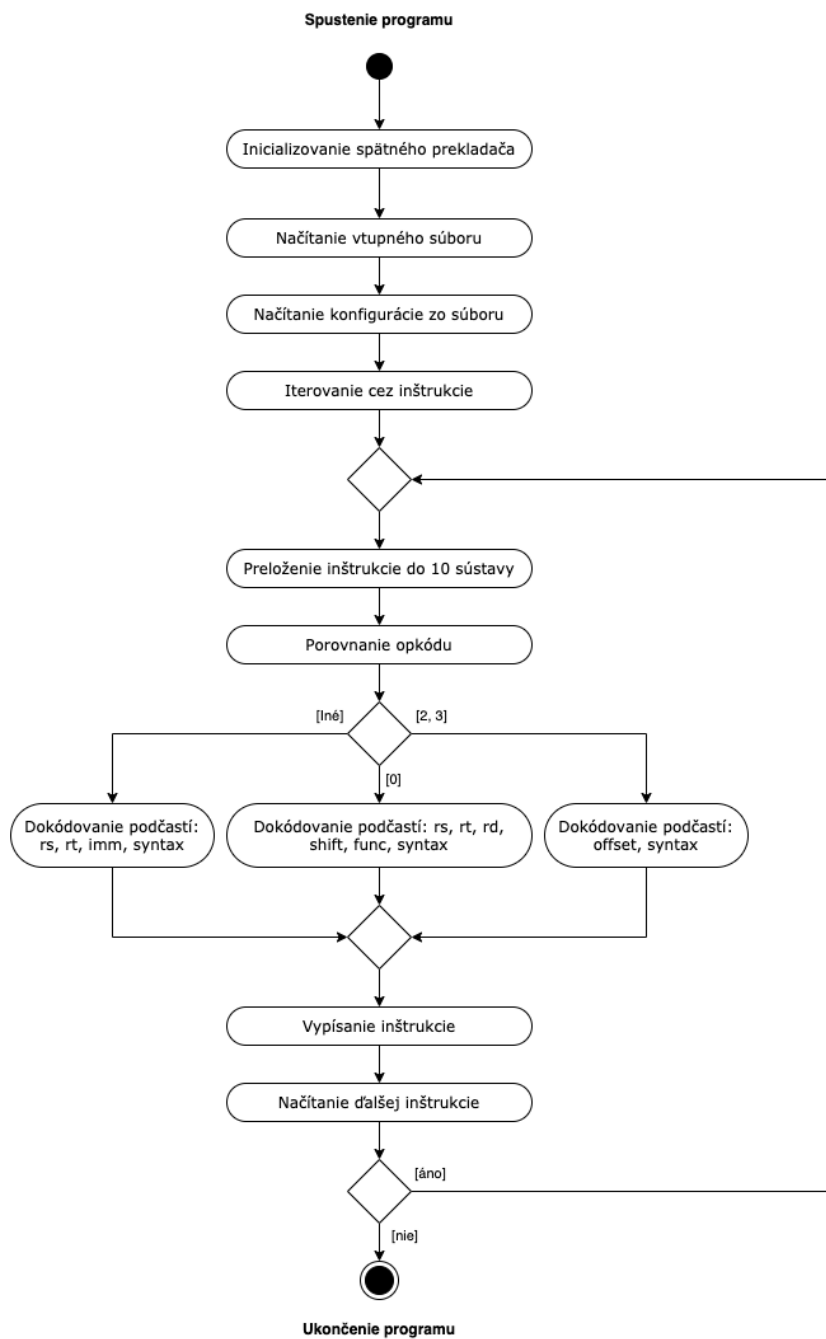
Operation:

GPR[rt] <- sign_extend(immediate || 016)

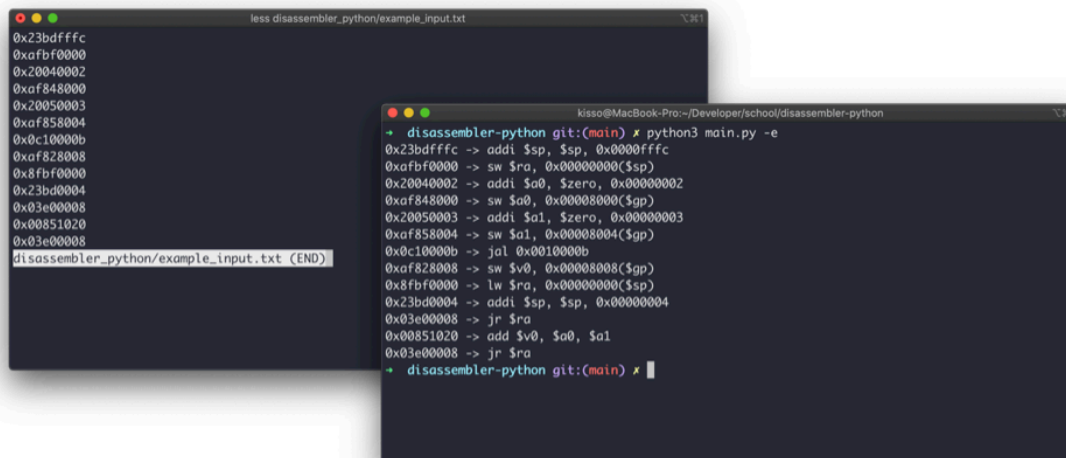
Obrázok 9: Výstup z nástroja MIPS Converter.

7 NAŠA IMPLEMENTÁCIA

Nástroj sme implementovali v jazyku Python ako konzolovú aplikáciu. Cieľom bolo vytvoriť offline spätný prekladač, ktorý sa dá spustiť bez ohľadu nato či používateľ má pripojenie k internetu alebo nie. Disassembler podporuje 49 inštrukcií. Vstupom aplikácie je textový súbor s požadovanými inštrukciami na spätné preloženie. Výstupom je zoznam inštrukcií v jazyku symbolických inštrukcií. V prípade nemožnosti dekódovania inštrukcie, aplikácia vypíše používateľovi "Unsupported instruction!". Viac o zdrojovom kóde sa nachádza v technickej dokumentácii na adrese: <https://github.com/kisso/disassembler-python>.



Obrázok 10: Aktivita diagram aplikácie



```
less disassembler_python/example_input.txt
0x23bdfffc
0xafbf0000
0x20040002
0xaf848000
0x20050003
0xaf858004
0x0c10000b
0xaf828008
0x8fbf0000
0x23bd0004
0x03e00008
0x00851020
0x03e00008
disassembler_python/example_input.txt (END)

kisso@MacBook-Pro:~/Developer/school/disassembler-python
+ disassembler-python git:(main) x python3 main.py -e
0x23bdfffc -> addi $sp, $sp, 0x0000fffc
0xafbf0000 -> sw $ra, 0x00000000($sp)
0x20040002 -> addi $a0, $zero, 0x00000002
0xaf848000 -> sw $a0, 0x00008000($gp)
0x20050003 -> addi $a1, $zero, 0x00000003
0xaf858004 -> sw $a1, 0x00008004($gp)
0x0c10000b -> jal 0x0010000b
0xaf828008 -> sw $v0, 0x00008008($gp)
0x8fbf0000 -> lw $ra, 0x00000000($sp)
0x23bd0004 -> addi $sp, $sp, 0x00000004
0x03e00008 -> jr $ra
0x00851020 -> add $v0, $a0, $a1
0x03e00008 -> jr $ra
+ disassembler-python git:(main) x
```

Obrázok 11: Ukážka vstupu/výstupu nášho riešenia.

7.1 Testovanie

Na overenie nášho riešenia sme použili nástroj Mars MIPS Simulator [10]. Je to simulátor fungovania MIPS architektúry. Programuje sa v ňom v jazyku symbolických inštrukcií. V tomto programe sme si vytvorili bezúčelový program do ktorého sme zapísali všetky inštrukcie ktoré má naše riešenie podporovať. Výstupom takéhoto programu boli zakódované inštrukcie, tieto inštrukcie sme následne použili ako vstup pre náš nástroj a porovnali sme či sa výstup nášho riešenia zhoduje so vstupom do programu MARS. Ďalej sme použili aj vyššie spomenuté nástroje pre spätný preklad, ktoré sme tiež porovnali s výstupom našej aplikácie.

Výsledkom bola jednoducho povedaná zhoda. Všetky použité nástroje dávali rovnaký výsledok ako naše riešenie. Je to kvôli tomu, že používame množinu bežných inštrukcií, ktoré táto architektúra poskytuje. Formát inštrukcií sa rokmi nemení tak nie je dôvod pre rozdielne výsledky.

8 ZHODNOTENIE

Práca sa zaoberá architektúrou MIPS, ktorú sme analyzovali z viacerých uhlov. V práci sme spomenuli dve existujúce riešenia spätných prekladačov, ktoré sme vyskúšali. Použili sme ich ako inšpiráciu pre naše riešenie. Aplikáciu na spätné prekladanie sa nám podarilo vytvoriť. Táto aplikácia má samozrejme obmedzenia a nie je plnohodnotná. Jedno obmedzenie je veľkosť inštrukčnej sady, nepodporujeme všetky dostupné inštrukcie ale len množinu tých najpoužívanejších. Ďalším obmedzením je spôsob vstupu dát na preloženie. V praxi by sme mali ako vstup použiť skompilovaný súbor, ktorý treba najskôr dekodovať a vyparsovať z neho potrebné dáta. Tento proces je náročný a vyžaduje dodatočnú analýzu problematiky štruktúry týchto súborov. Pre jednoduchosť sme použili ako vstup obyčajný textový súbor, v ktorom sa už nachádzajú len konkrétne dáta na preloženie. Výsledkom testovania je funkčný spätný prekladač pre architektúru MIPS s vyššie uvedenými obmedzeniami. Dodatočné informácie k podpore inštrukcií sa nachádzajú v technickej dokumentácii [11].

BIBLIOGRAFIA

- [1] John Hennessy, Norman Jouppi, Steven Przybylski, Christopher Rowen, Thomas Gross, Forest Baskett, and John Gill. 1982. MIPS: A microprocessor architecture. In Proceedings of the 15th annual workshop on Microprogramming (MICRO 15). IEEE Press, 17–22.
- [2] Stephen Barrett. MIPS Strikes Back: 64-bit Warrior I6400 Arrive. [Online] 2. September 2014, [Dátum: 2. November 2020]. <https://www.anandtech.com/show/8457/mips-strikes-back-64bit-warrior-i6400-architecture-arrives/2>
- [3] Alex Voica. Five most iconic devices to use MIPS CPUs. [Online] 8. Jún 2016, [Dátum: 2. November 2020]. <https://www.mips.com/blog/five-most-iconic-devices-to-use-mips-cpus/>
- [4] Jason W. Bacon. Computer Science 315 Lecture Notes. [Online] 2010, [Dátum: 2. November 2020]. <http://www.cs.uwm.edu/classes/cs315/Bacon/Lecture/HTML/ch05.html>
- [5] Obrázky architektúr. [Online] [Dátum: 2. November 2020]. <http://aplo.fiiit.stuba.sk/aps/frames/generuj.php?id=12>
- [6] Prof. Grishman. Lecture 9 - MIPS instruction set; MIPS ALU. [Online] 1999, [Dátum: 2. November 2020]. <https://cs.nyu.edu/courses/spring99/V22.0436-001/lecture9.html>
- [7] MIPS Instruction Reference. [Online] 10. September 1998, [Dátum: 2. November 2020]. <http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html>
- [8] Obrázok schémy päť krokového inštrukčného cyklu. [Online] [https://commons.wikimedia.org/wiki/File:MIPS_Architecture_\(Pipelined\).svg](https://commons.wikimedia.org/wiki/File:MIPS_Architecture_(Pipelined).svg)
- [9] Nathaniel Pinckney, Thomas Barr, Michael Dayringer, Matthew McKnett, Nan Jiang, Carl Nygaard, David Money Harris, Joel Stanley, and Braden Phillips. 2008. A MIPS R2000 implementation. In Proceedings of the 45th annual Design Automation Conference (DAC '08). Association for Computing Machinery, New York, NY, USA, 102–107.
- [10] Mars MIPS Simulator. [Online] <http://courses.missouristate.edu/kenvollmar/mars/>
- [11] Kryštof Kiss, Spätný prekladač (disassembler) pre MIPS architektúru, November 2020, Technická dokumentácia: <https://github.com/kisso/disassembler-python>