Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 2, 842 19 Bratislava 4

**Patrik Sčensný**

**Architektúra počítačových systémov**

**Semestrálne zadanie**

Študijný program: Internetové technológie

Ročník: 1.

Cvičiaci: Ing. Dušan Bernát

Ak. rok: 2017/2018

Obsah

[Zadanie 3](#_Toc497775004)

[Úvod 3](#_Toc497775005)

[Analýza 3](#_Toc497775006)

[Hardvér 3](#_Toc497775007)

[MIPS architektúra 3](#_Toc497775008)

[ELF 3](#_Toc497775009)

[Objdump 3](#_Toc497775010)

[Disassembler 3](#_Toc497775011)

[Programovací jazyk 4](#_Toc497775012)

[Návrh riešenia 4](#_Toc497775013)

[Implementácia 4](#_Toc497775014)

[Zistenie inštrukcie 4](#_Toc497775015)

[Priebeh Programu 5](#_Toc497775016)

[Testovanie 6](#_Toc497775017)

[Záver 8](#_Toc497775018)

[Referencie 8](#_Toc497775019)

[Technická dokumentácia 8](#_Toc497775020)

# Zadanie

Vytvorte spätný prekladač (disassembler) pre zvolenú architektúru.

# Úvod

Spätný prekladač, už iba disassembler, som si vybral pre architektúru MIPS. Výsledky som porovnával s utilitou objdump.

# Analýza

V tejto časti sa budem venovať softvérovej a architektickej analýze na splnenie zadania.

## Hardvér

Zadanie bude implementované a testované na notebooku ACER Aspire V3-571g s procesorom i5-2300 8GB RAM.

## MIPS architektúra

Je reduced instruction set computer (RISC) architektúra inštrukcií (ISA) vyvinutá spoločnosťou MIPS Technologies (predtým MIPS Computer Systems). Skoré MIPS architektúry boli 32-bitové, 64-bitové verzie boli pridané neskôr. Jedna z vlastností MIPS architektúry je že všetky JSI príkazy sú dlhé 32-bitov.

## ELF

(Executable and Linkable Format) bol pôvodne vyvinutý a uverejnený systémovými laboratóriami UNIX  
(USL) ako súčasť Aplikačného binárneho rozhrania (ABI). Výbor pre štandardy rozhrania nástrojov  
(TIS) vybral vyvíjajúci sa štandard ELF ako súbor prenosného objektu, ktorý funguje na 32-bitovej technológii Intel Architektúry prostredia pre rôzne operačné systémy. Štandard ELF je určený na zefektívnenie vývoja softvéru tým, že poskytuje vývojárom rozhranie, ktoré sa rozprestiera vo viacerých prevádzkových prostrediach. Malo by to znížiť počet rôznych implementácií rozhrania, čím sa znižuje potreba prepisovania a rekompilácie kódu.

## Objdump

je program na zobrazovanie rôznych informácií o ELF súboroch. Napríklad, môže byť použitý ako disassembler na zobrazenie spustiteľného súboru vo forme zostavy. Je súčasťou GNU Binutils pre riadenie spustiteľných súborov a iných binárnych údajov

## Disassembler

je počítačový program, ktorý prekladá jazyk stroja do assemblerového jazyka - inverznú operáciu ako assembler. Disassembler sa líši od dekompilátora, ktorý sa zameriava skôr na jazyky na vysokej úrovni než na assembler. Výstup disassemblera, je často formátovaný pre ľudskú čitateľnosť skôr ako vhodnosť pre vstup do assembleru, čo je hlavne nástroj pre reverzné inžinierstvo.

## Programovací jazyk

Pri implementácií som rozmýšlal nad použitím jazyka C/C++, ale samotná utilita objdump je implementovaná v C. Preto som sa rozhodol použiť jazyk Python, ktorý je momentálne veľmi rozšírený a populárny.

# Návrh riešenia

Zadanie som sa rozhodol realizovať v programovacom jazyku Python. Na prvotné zisťovanie architektúry ELF súboru som použil utility: objdump, hexdump. Z pax-utils som použil dumpelf, ktorý mi najviac pomohol pri zistení vnútornej štruktúry súboru.

# Implementácia

Predtým ako môžeme prekladať bajty na inštrukcie musíme zistit kde sa tieto inštrukcie nachádzajú. V ELF súbore sa inštrukcie nachádzajú v .text sekcii. Túto sekciu nájdeme v hlavičkách sekcii, ale musíme zistit, či máme správnu hlavičku.

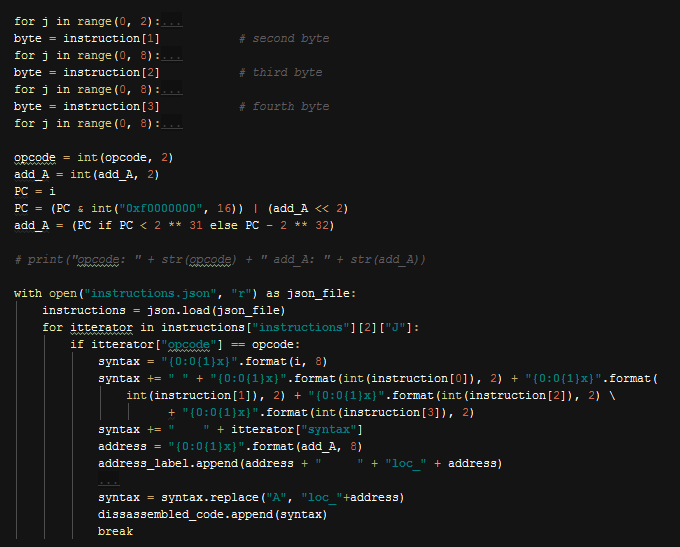
Musíme zistit mená hlavičiek, tie nájdeme v sekcii, .shstrtab, po jej nájdený nájdeme offset .text sekcie a jej dĺžku.

## Zistenie inštrukcie

Inštrukcie v MIPS môžeme rozložiť do 4 kategórií:

* R inštrukcia
  + opcode – 6 bitov(iba 0b000000)
  + source register – 5 bitov
  + target register – 5 bitov
  + destination register – 5 bitov
  + shamt(posunutie) immidiate – 5 bitov
  + funct – 6 bitov
* RI inštrukcia
  + opcode – 6 bitov (iba 0b000001)
  + source register – 5 bitov
  + regimm - 5 bitov
  + constant immidiate – 16 bitov
* J inštrukcia
  + opcode – 6 bitov (iba 0b000010 a 0b000011)
  + address – 26 bitov
* I inštrukcia
  + opcode – 6 bitov (všetky ostatné)
  + source register – 5 bitov
  + target register – 5 bitov
  + constant immidiate – 16 bitov

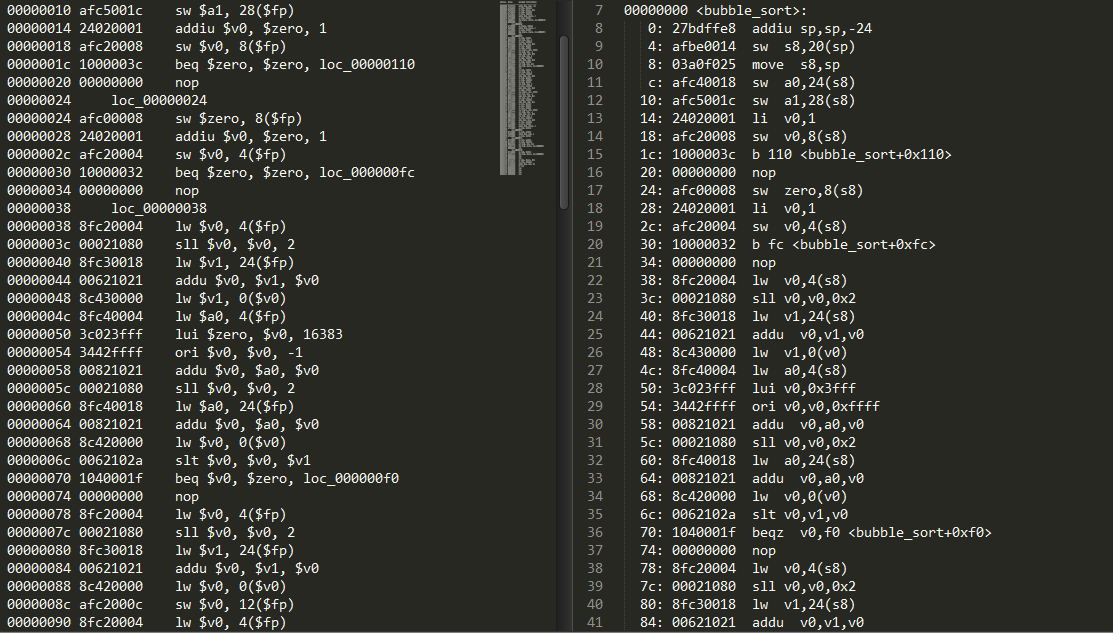
Inštrukcie načítavam so súboru JSON, vybral som ho preto lebo Python ma knižnicu ktorá dokáže narábať s JSON súbormi veľmi jednoducho.



Obr. 1 Ukážka program pre rozklad inštrukcie

### Priebeh Programu

Keď program spustíte zadáte 2 súbory jeden pre vstup a druhý kde sa má ukladať výstup. Program najprv zistí či vstupný súbor je vhodný pre jeho inštrukcie, ELF súbor to má zapísané. Následne zistí kde sa nachádzajú section headery a ich veľkosti, všetky sú rovnakej veľkosti. Program prezrie všetky headery, a potenciálne headery pre .text a .shstrtab si uloží. Potom program nájde ten správny .shtrtab header, a vďaka nemu nájde .text header. Potom program nájde začiatok .text a jeho veľkosť. Program načítava 4 bajty naraz a tie pomaly rozkladá na jednotlivé podľa typu inštrukcie. Pri inštrukcii typu branch alebo jump, si program zapamätá na ktorú adresu chcel program skočiť a na konci programu na tú adresu napíše návestie.



Obr. 3 vľavo disassembler, vpravo objdump

# Testovanie

Pri testovaní som porovnával programy ktoré, som často písal v Bakalárskom štúdiu.

Pri testovaní sa podarilo spustit program iba na prostredí Windows, budem sa snažiť opraviť kód aby bolo možné spustit na Unix-like systémoch.

* Quick sort
* Merge sort
* Insert sort
* Bubble sort
* A moje vlastné malé programy

V tabuľke Objdump je čas vykonávania objdump, disassembler je čas vykonávania mojej implemenácie v pythone.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Bubble | | Insert | | Merge | |
| Počet vykonaní merania | Objdump  [ms] | Disassembler  [ms] | Objdump  [ms] | Disassembler  [ms] | Objdump  [ms] | Disassembler  [ms] |
| 1 | 4 | 122 | 5 | 103 | 5 | 104 |
| 2 | 5 | 117 | 4 | 118 | 5 | 129 |
| 3 | 5 | 102 | 4 | 108 | 5 | 104 |
| 4 | 5 | 112 | 4 | 115 | 5 | 114 |
| Priemer | 4,75 | 113,25 | 4,25 | 111 | 5 | 112,75 |
|  | Quick | | Snippet 1 | | Snippet 2 | |
| Počet vykonaní merania | Objdump  [ms] | Disassembler  [ms] | Objdump  [ms] | Disassembler  [ms] | Objdump  [ms] | Disassembler  [ms] |
| 1 | 5 | 114 | 3 | 111 | 7 | 112 |
| 2 | 5 | 107 | 4 | 106 | 7 | 109 |
| 3 | 5 | 100 | 4 | 105 | 8 | 110 |
| 4 | 5 | 108 | 4 | 105 | 7 | 100 |
| Priemer | 5 | 107,25 | 3,75 | 106,75 | 7,25 | 107,75 |
|  | Snippet 3 | |  | |  | |
| Počet vykonaní merania | Objdump  [ms] | Disassembler  [ms] |  |  |  |  |
| 1 | 3 | 126 |  |  |  |  |
| 2 | 3 | 105 |  |  |  |  |
| 3 | 4 | 110 |  |  |  |  |
| 4 | 4 | 100 |  |  |  |  |
| Priemer | 3,5 | 110,25 |  |  |  |  |

Obr.3 grafické znázornenie priemerných časov jednotlivých kódov

# Záver

Pri testovaní som zistil že môj program je pomalší ako utilita objdump, čo sa dalo predpokladať lebo jazyk Python je interpretovaný jazyk oproti C ktoré je kompilovaný. Aj počas písania tohto programu som narazil na problémy ktoré by sa dali v C ľahko riešiť, ale naopak som ma aj problémy ktoré som ľahšie riešil v Pythone ako v C. Môj program je síce pomalý ale spĺňa zadanie, a v porovnaní so zdrojovým kódom objdump, je aj kratší. A vďaka tomu že všetky inštrukcie mám uložené v JSON-e je inštrukčná sada ľahko rošíriteľná bez zmeny zdrojového kódu.

# Referencie

www.skyfree.org/linux/references/ELF\_Format.pdf

# Technická dokumentácia

Program môžeme spusit pomocou príkazu.

“python disassembler.py -I vstupna\_binarka -o vystup\_programu”

Pri problémoch použite prepínač “-h”