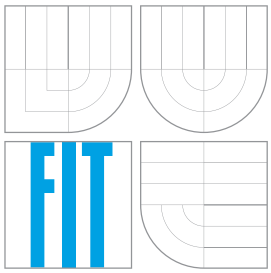
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# ZPĚTNÝ PŘEKLAD Z VYBRANÝCH FORMÁTŮ SPUSTITELNÝCH SOUBORŮ

DECOMPILATION FROM SELECTED OBJECT FILE FORMATS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL BANDZI

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETER MATULA

BRNO 2016

*Zadání bakalářské práce/17988/2015/xbandz00*

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav informačních systémů Akademický rok 2015/2016

# Zadání bakalářské práce

Řešitel: **Bandzi Michal**
Obor: Informační technologie
Téma: **Zpětný překlad z vybraných formátů spustitelných souborů**
**Decompilation from Selected Object File Formats**
Kategorie: Překladače

Pokyny:

1. Studujte problematiku zpětného inženýrství. Zaměřte se na zpětný překlad binárního kódu do vyšší formy reprezentace. Dále se seznamte se zpětným překladačem společnosti AVG Technologies.
2. Seznamte se s vybranými formáty spustitelných souborů (např. Mach-O, Intel HEX, SREC a další).
3. Navrhněte metody, které umožní zpětný překlad kódů uložených v těchto formátech do vyšší formy reprezentace.
4. Po konzultacích s vedoucím metody navržené v předchozím bodě implementujte.
5. Vytvořené řešení důkladně otestujte sadou minimálně dvaceti testů, zohledněte překladače a jejich nastavení. Zhodnoťte svou práci a diskutujte budoucí vývoj.

Literatura:

- Křoustek, J.: Rekonfigurovatelná analýza strojového kódu, disertační práce. Brno, VUT FIT, 2015.
- Matula, P.: Nástroje pro konverzi formátů spustitelných souborů, bakalářská práce. Brno, FIT VUT, 2011.
- Popis platformy LLVM [online]. 2015 [cit. 2015-09-23]. Dostupný z WWW: <www.llvm.org>.
- Levine, J. R.: Linkers and Loaders, Morgan Kaufmann Publishers, Inc., 1999.
- Další dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- První tři body zadání, částečně bod čtvrtý.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Matula Peter, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
*vedoucí ústavu*

# Abstrakt

Objektové súbory obsahujú strojový kód, ktorý môže byť vykonaný procesorom. Každý objektový súbor má formát, ktorý popisuje jeho štruktúru. Pre vykonanie spätného prekladu je nutné súbor spracovať a previesť dáta do vnútornej reprezentácie spätného prekladača. Táto práca pojednáva o návrhu a implementácii nových modulov pre podporu spracovania formátov, ktoré budú súčasťou Rekonfigurovateľného spätného prekladača. Cieľom práce je pridanie podpory pre formáty Intel HEX a Mach-O a nová implementácia už podporovaného formátu Portable Executable. Implementácia modulov pre Intel HEX a Mach-O bola úspešná a je možné použiť ich pre spätný preklad. Spracovanie formátu PE nedosahuje dostatočnej kvality kvôli chybám knižnice LLVM, na ktorej je implementácia založená. Upozornenie: toto je verejná skrátená verzia práce.



# Abstract

Object files contain machine code that can be executed by processor unit. Structure of an object file is defined by its file format. In order to decompile an object file, it is necessary to process and convert file data to internal representation of decompiler. This thesis discusses design and implementation of new modules for file format processing that will be part of the Retargetable Decompiler project. The goal of this work is to add support for Intel HEX and Mach-O file formats and new implementation of already supported Portable Executable file format. Implementation of modules for file formats Intel HEX and Mach-O was successful and it is possible to use them for reverse compilation. Processing of PE file format is not possible in sufficient quality due to errors in used LLVM library. Warning: this is public censored version of thesis.



# Kľúčové slová

objektové súbory, binárne súbory, univerzálne binárne súbory, reverzné inžinierstvo, spätný preklad, spätný prekladač, Intel HEX, Portable Executable, PE, Mach-O, Mach-O Universal Binary

# Keywords

object files, binary files, universal binaries, reverse engineering, decompilation, decompiler, Intel HEX, Portable Executable, PE, Mach-O, Mach-O Universal Binary


# Citácia

BANDZI, Michal. *Zpětný překlad z vybraných formátů spustitelných souborů.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Matula Peter.

# Zpětný překlad z vybraných formátů spustitelných souborů

## Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Petra Matulu. Uviedol som všetky literárne zdroje a publikácie, z ktorých som čerpal.

.....
Michal Bandzi
17. mája 2016


## Poďakovanie

Ďakujem svojmu vedúcemu Ing. Petrovi Matulovi za odborné vedenie, za poskytnuté rady a za čas, ktorý mi pri tvorbe práce venoval.

# Obsah


**1 Úvod** **3**

**2 Reverzné inžinierstvo** **5**
2.1 Nástroje reverzného inžinierstva . . . . . 5
2.2 Spätný prekladač . . . . . 6
2.2.1 Porovnanie vybraných spätných prekladačov . . . . . 6

**3 Rekonfigurovateľný spätný prekladač spoločnosti AVG** **9**
3.1 Štruktúra rekonfigurovateľného spätného prekladača . . . . . 9
3.1.1 Projekt LLVM . . . . . 9
3.1.2 Predspracovanie . . . . . 10
3.1.3 Front-end . . . . . 11
3.1.4 Middle-end . . . . . 11
3.1.5 Back-end . . . . . 12

**4 Knižnica fileformatl** **13**

**5 Formáty objektových súborov** **14**
5.1 Intel HEX . . . . . 15
5.1.1 Vytvorenie súboru s formátom Intel HEX . . . . . 15
5.1.2 Štruktúra formátu Intel HEX . . . . . 15
5.2 Portable Executable . . . . . 17
5.2.1 Štruktúra formátu Portable Executable . . . . . 17
5.3 Mach-O . . . . . 19
5.3.1 Štruktúra formátu Mach-O . . . . . 20
5.3.2 Mach-O Universal Binary . . . . . 22

**6 Spätný preklad súborov formátu Intel HEX** **23**

**7 Spätný preklad súborov formátu Portable Executable** **24**

**8 Spätný preklad súborov formátu Mach-O** **25**

**9 Záver** **26**
9.1 Budúci vývoj . . . . . 26

**Literatúra** **28**

**Prílohy** **30**
Zoznam príloh . . . . . 31

**A Obsah CD** **32**

# Kapitola 1

# Úvod

Spustiteľný súbor je súbor obsahujúci inštrukcie, ktoré budú po jeho spustení vykonané počítačom. Môže sa jednať buď o inštrukcie cieľovej architektúry uložené v binárnej forme, doplnené dodatočnými informáciami, alebo inštrukcie interpretovaného jazyka. Každý takýto súbor má určitý formát, ktorý popisuje jeho vnútornú štruktúru.

Binárne spustiteľné súbory vznikajú prekladom (anglicky compilation) zdrojového súboru, vytvoreného užívateľom-programátorom, pomocou prekladača (anglicky compiler) zvoleného programovacieho jazyka. Prekladač analyzuje vstupný zdrojový súbor a prevedie ho na inštrukcie cieľovej architektúry, pribalí k nim statické dáta a ďalšie potrebné informácie a uloží ich do výstupného súboru, ktorý je následne možné spustiť. Opačný proces, prevedenie spustiteľného súboru do vyššieho programovacieho jazyka, sa nazýva spätný preklad (anglicky decompilation). Nástroj, ktorý túto činnosť vykonáva, sa nazýva spätný alebo reverzný prekladač (anglicky decompiler). Jedná sa o významný nástroj softvérového reverzného inžinierstva.

Reverzné inžinierstvo je proces získavania nových vedomostí alebo dokumentácie o určitom objekte, ktorý bol vytvorený ľudskou činnosťou [7]. V prípade softvérového reverzného inžinierstva je takýmto objektom softvér alebo jeho časť. Historicky odbor vznikol v podstate spolu so softvérom samotným. Prvé spätné prekladače sa začali objavovať približne desať rokov po klasických prekladačoch [5]. Motívy pre spätný preklad a všeobecne reverzné inžinierstvo môžu byť rôzne, od zvyšovania bezpečnosti (hľadanie chýb alebo škodlivého softvéru) až po urýchlenie vývoja softvéru.

Počas historického vývoja informačných technológií vzniklo niekoľko desiatok rôznych formátov objektových súborov pre rôzne architektúry, operačné systémy a účely. Medzi najznámejšie a dnes najpoužívanejšie patrí formát ELF, typický pre operačné systémy rodiny Unix, formáty COFF a PE využívané operačnými systémami Windows, formát Mach-O, ktorý nájdeme v operačných systémoch rodiny Mac OS a mnohé ďalšie. Ak chceme spustiteľný súbor preložiť späť do vyššieho programovacieho jazyka, potrebujeme získať zdrojový kód a ďalšie potrebné informácie. Aby sa nám to podarilo, musíme poznať vnútornú štruktúru týchto súborov a byť schopný tieto informácie dodať vo forme, ktorej rozumie spätný prekladač.

Táto práca si dáva za cieľ rozšíriť *Rekonfigurovateľný spätný prekladač* (anglicky *Retargetable Decompiler*) vyvíjaný spoločnosťou *AVG Technologies CZ, s.r.o.* o podporu nových formátov nielen spustiteľných objektových súborov. Rekonfigurovateľný spätný prekladač je možné jednoducho rozšíriť naprogramovaním dodatočných modulov, ktoré majú za úkol analyzovať vstupný súbor a poskytnúť ďalším častiam spätného prekladača všetky potrebné informácie pre vykonanie prekladu.

Práca je členená do niekoľkých kapitol. V kapitole 2 sa nachádzajú všeobecné informácie o reverznom inžinierstve a spätnom preklade. V kapitole 3 je detailnejšie popísaný Rekonfigurovateľný spätný prekladač spoločnosti AVG. Kapitola 5 poskytuje pohľad na formáty jednotlivých objektových súborov. Kapitola 9 obsahuje zhrnutie tejto práce. Kapitoly 4, 6, 7 a 8 sú utajené.

# Kapitola 2

# Reverzné inžinierstvo

Podľa [7], z ktorej nasledujúca kapitola vychádza, je reverzné inžinierstvo proces získavania nových vedomostí alebo dokumentácie o určitom objekte, ktorý bol vytvorený človekom. V prípade softvérového reverzného inžinierstva je takýmto objektom softvér alebo jeho časť. Dôvody reverzného inžinierstva môžu byť rôzne, podľa [7] ich môžeme rozdeliť na dve hlavné skupiny: bezpečnosť a ďalší vývoj softvéru.

Z hľadiska bezpečnosti sa jedná predovšetkým o vyhľadávanie škodlivého alebo nebezpečného softvéru, predovšetkým počítačových vírusov a červov. Tento prístup sa využíva napr. v antivírusových programoch. Reverzné inžinierstvo je však možné zneužiť aj na opačnú činnosť – prehľadávanie softvéru a hľadanie bezpečnostných dier, na ktoré bude možné škodlivým softvérom zaútočiť. Ďalej môže byť reverzné inžinierstvo použité na odhalenie fungovania niektorých kryptografických algoritmov a algoritmov pre zabezpečenie digitálneho obsahu, predovšetkým tých, kde je kľúčové utajenie použitého algoritmu. Tretie využitie reverzného inžinierstva je kontrola proprietárneho softvéru, ktorého zdrojové kódy nie sú normálne dostupné užívateľom.

Z hľadiska vývoja softvéru môže byť reverzné inžinierstvo použité pre zaistenie interoperability nedostatočne dokumentovaného softwéru, nahliadnutie do konkurenčného projektu, prípadne pre kontrolu kvality a robustnosti softvéru, ktorého zdrojové kódy nie sú verejné.

## 2.1 Nástroje reverzného inžinierstva

Podľa [7] môžeme pri reverznom softvérovom inžinierstve použiť predovšetkým štyri hlavné druhy nástrojov:

- **Nástroje pre monitorovanie systému** – tieto nástroje zaznamenávajú informácie súvisiace s používaním operačného systému spusteným programom. Vytvárajú tak obraz o využití siete, prístupe k súborom, registrom a službám operačného systému.
- **Disassembler** – relatívne jednoduchý nástroj, ktorý prevedie binárne inštrukcie programu do zápisu formou textových operačných inštrukcií danej architektúry. Veľká väčšina týchto programov je schopná pracovať len s jednou architektúrou, existuje však aj málo disassemblerov, ktoré sú schopné zamerať sa na viacero architektúr.
- **Ladiace nástroje** – aj keď ich primárnym účelom nie je reverzné inžinierstvo, je možné ich na tieto účely použiť. Užívatelia môžu program krokovať a sledovať tak priebeh jeho činnosti inštrukciu po inštrukcii.

- **Spätný prekladač** – nástroj, ktorý je schopný previesť vstupné binárne súbory do programovacieho jazyka vyššej úrovne.

## 2.2 Spätný prekladač

Podľa [5], z ktorej táto podkapitola vychádza, je spätný prekladač program, ktorý číta program napísaný v strojovom kóde – zdrojovom jazyku – a prekladá ho na ekvivalentný program napísaný v jazyku vyššej úrovne. Vnútorná štruktúra spätného prekladača je veľmi podobná štruktúre klasických prekladačov. [5] hovorí o nasledovných fázach spätného prekladu:

- **Syntaktická analýza** – analyzátor prevedie bajty zdrojového programu na gramatické vety daného strojového jazyka. Najväčší problém tejto fázy je rozhodovanie, čo sú dáta a čo inštrukcie, táto fáza sa tak stáva závislá na cieľovej architektúre.
- **Sémantická analýza** – táto fáza vyhľadáva sémantický význam v skupinách inštrukcií. Fáza prebieha pomocou vyhľadávania typických idiómov a je rovnako závislá na architektúre.
- **Generovanie prechodného kódu** – je nutné pre ďalšiu analýzu. Tento kód by nemal byť náročný na jeho generáciu a zároveň by mal vhodne reprezentovať cieľovú architektúru.
- **Generovanie grafov toku riadenia programu** – je potrebné pre vytvorenie obrazu o vyšších riadiacich prvkoch programu a prípadnú elimináciu medziskokov.
- **Analýza toku dát** – v tejto fáze dochádza k eliminácii použitia dočasných registrov a príznakov, ktoré v jazykoch vyššej úrovne nie sú dostupné.
- **Analýza toku programu** – vytvára na základe grafov toku riadiace štruktúry vyššej úrovne (if-else vetvenie, cykly).
- **Generovanie kódu** – finálna fáza spätného prekladu, vygenerovanie kódu cieľového jazyka.

Podľa [5] môžeme tieto fázy rozdeliť do troch skupín:

- **Front-end** – tieto fázy sú závislé na vstupnej architektúre. Patrí sem syntaktická a sémantická analýza, generovanie prechodného kódu a generovanie grafu toku riadenia programu.
- **Univerzálny spätný prekladač** – fázy tejto skupinu nie sú závislé ani na architektúre vstupného strojového kódu ani na cieľovom jazyku. Patrí sem analýza toku dát a analýza toku riadenia programu.
- **Back-end** – táto fáza je závislá na cieľovom jazyku. Jedná sa o generovanie výsledného kódu daného jazka.

### 2.2.1 Porovnanie vybraných spätných prekladačov

Táto podkapitola sa zameriava na porovnanie najznámejších a najpoužívanejších spätných prekladačov predovšetkým z hľadiska podpory rôznych objektových formátov, architektúr a cieľových jazykov vyššej úrovne.

| Názov | Podporované architektúry | Podporované formáty |
|---|---|---|
| Hex-Rays | x86, x64, ARM32, ARM64 | PE, ELF a iné v rámci IDA |
| REC Studio 4 | x86, x64, MIPS, PPC, mc68k | COFF, ELF, PE, Mach-O |
| SmartDec | x86, x64 | PE, ELF a iné v rámci IDA |
| C4Decompiler | Intel 64, x86 | PE |
| Hopper | x86, x64, ARM32, ARM64 | ELF, PE, Mach-O |
| *RetDec* | x86, ARM32, MIPS32, PIC32, PPC32 | ELF, PE, COFF |

Tabuľka 2.1: Porovnanie najpopulárnejších spätných prekladačov.

#### Hex-Rays

Komerčný spätný prekladač Hex-Rays disponuje najväčšou podporou rôznych objektových formátov – zvláda všetky bežne vyskytujúce sa formáty ako PE, COFF, ELF, Mach-O a takmer 40 ďalších v rámci disassembleru IDA. Značnou nevýhodou je podpora architektúr limitovaná na architektúry x86, x64 a ARM. Výstupom prekladu je pseudo-kód založený na jazyku C [8].

#### REC Studio 4

Bezplatný closed-source spätný prekladač podporujúci architektúry x86, x64, MIPS, PowerPC a mc68k. Podporuje formáty PE, COFF, ELF a Mach-O. Podpora architektúry ARM je vo vývoji. Výstupom prekladu je pseudo-kód založený na jazyku C [3].

#### SmartDec

Spätný prekladač podporujúci architektúry x86 a x64. Samostatná verzia podporuje formáty ELF a PE, prekladač je však možné použiť spoločne s programom IDA, ktorý rozširuje podporu o desiatky ďalších formátov. Výstupom prekladu je kód v jazyku C s významnou podporou štruktúr jazyka C++ [16]. SmartDec bol neskôr rozšírený o podporu formátu Mach-O a architektúry ARM v rámci odvodeného projektu Snowman [17].

#### C4Decompiler

Jedná sa o komerčný spätný prekladač zameraný na operačné systémy rodiny Windows. Zaujímavosťou je podpora architektúry Intel Itanium (Intel 64). Jeho podpora objektových formátov je však obmedzená len na formát Windows PE, podpora formátu ELF je vo vývoji. Výstupom je kód v jazyku C [4].

#### Hopper

Komerčný dissasembler a spätný prekladač zameraný predovšetkým na operačné systémy Unix a OS X. Zvláda základné formáty binárnych súborov ELF, PE a Mach-O. Binárny kód prekladá do jazyka C s čiastočnou podporou pre jazyk Objective-C. Podporované architektúry sú x86, x64 a ARM [6].

#### Záver

Väčšina prekladačov sa obmedzuje na základné objektové formáty kľúčové pre jednotlivé operačné systémy. Jedná sa o PE (Windows), COFF, ELF (Unix) a Mach-O (OS X, iOS).

Rekonfigurovateľný spätný prekladač spoločnosti AVG, ktorý je podrobnejšie popísaný v kapitole 3, momentálne podporuje formáty COFF, PE a ELF. Zavedenie podpory pre formát Mach-O je kľúčové vzhľadom na relatívne vysoký podiel OS X a iOS na trhu operačných systémov [15]. Po pridaní podpory pre formáty Mach-O a Intel HEX sa rekonfigurovateľný spätný prekladač zaradí na prvé miesto v počte podporovaných objektových formátov bez závislosti od softvéru tretích strán, ako je to napr. v prípade programu Hex-Rays.

# Kapitola 3

# Rekonfigurovateľný spätný prekladač spoločnosti AVG

Rekonfigurovateľný spätný prekladač je nástroj vyvíjaný spoločnosťou *AVG Technologies CZ, s.r.o.* v spolupráci s *Fakultou informačních technologií VUT v Brne*. Podľa [2] sa jedná o spätný prekladač neviazaný ku žiadnej konkrétnej architektúre, operačnému systému alebo formátu binárnych súborov. Prekladač v dobe písania práce podporuje architektúry Intel x86, ARM + Thumb, MIPS, PIC32, a PowerPC (32-bit verzie), vstupné formáty ELF, PE a COFF a je schopný prekladať do jazyka C alebo upravenej formy jazyka Python. Nástroj je vývíjaný v jazyku C++.

## 3.1 Štruktúra rekonfigurovateľného spätného prekladača

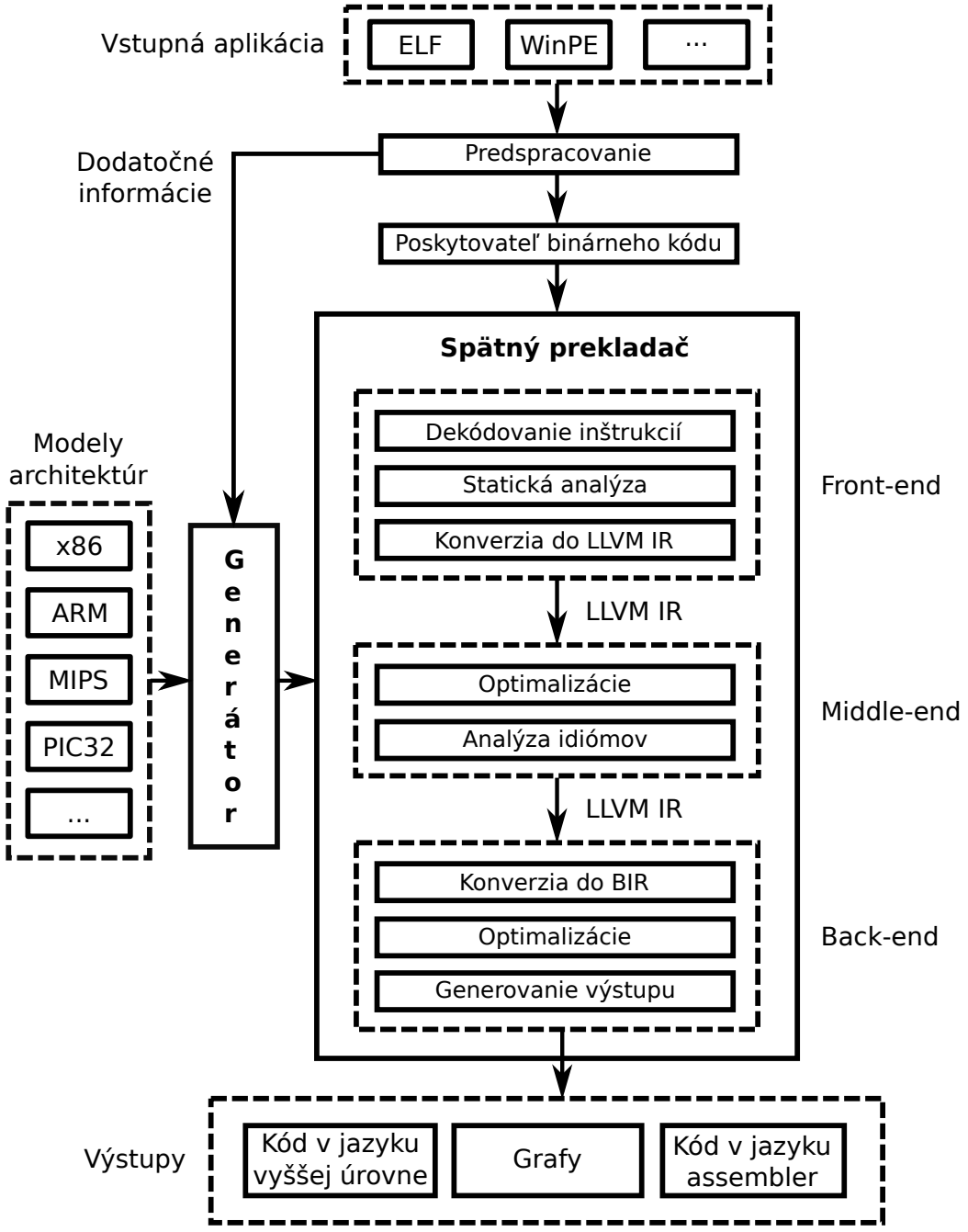
Táto podkapitola bola spracovaná na zákalade [10] a [18].

Štruktúra rekonfigurovateľného spätného prekladača je podobná štruktúre naznačenej v podkapitole 2.2. Nájdeme tu tri základné časti: front-end, middle-end a back-end. Vzhľadom nato, že zdrojový kód môže byť uložený vo forme rôznych objektových formátov, je nutné vstupný súbor pred samotným spätným prekladom najskôr predspracovať. Viaceré komponenty spätného prekladača spoločnosti AVG využívajú technológie LLVM.

### 3.1.1 Projekt LLVM

Projekt LLVM je kolekcia modulárnych a znovu-použiteľných technológií určená pre použitie v prekladačoch a ďalších nástrojoch [13].

Reprezentácia kódu LLVM IR (LLVM intermediate representation) poskytuje typovú bezpečnosť, operácie nízkej úrovne, flexibilitu a možnosť reprezentovať jazyky vyššej úrovne. LLVM kód môže byť použitý v troch rôznych formách a to ako reprezentácia v pamäti v rámci prekladača, bitový kód pre uloženie na disk a ako textová forma jazyka assembler určená pre interakciu s ľuďmi. Všetky tri reprezentácie sú si ekvivalentné. Projekt tak poskytuje efektívne prostriedky pre transformácie a analýzy vykonávané prekladačom a zároveň možnosť kód prirodzene ladiť a zobrazovať. Cieľom LLVM IR je poskytnúť odľahčenú, typovanú, rozšíriteľnú a univerzálnu reprezentáciu nízkej úrovne. Vzhľadom na prítomnosť typovej informácie je možné aplikovať veľké množstvo rozličných buď vstavaných alebo vlastných optimalizácií [12].




Obr. 3.1: Štruktúra rekonfigurovateľného spätného prekladača.

### 3.1.2 Predspracovanie

Predspracovanie (anglicky preprocessing) má za úlohu analyzovať vstupný zdrojový súbor a získať všetky potrebné dáta. Okrem kódu sa vo vstupnom súbore nachádzajú aj informácie o cieľovej architektúre, veľkosti kódu, bitovej šírke, prípadne o prekladači, o použitom jazyku a mnohé iné. Dáta sú v zdrojovom súbore v závislosti od jeho typu často rozdelené do viacerých sekcií a segmentov. Úlohou predspracovania je extrahovať tieto dáta a jednotlivé sekcie a segmenty previesť do vnútornej objektovej reprezentácie spätného prekladača.

Ďalším problémom, ktorý rieši predspracovanie, je prípadná kompresia a ochrana binárnych súborov, dáta je pred ďalším spracovaním nutné dekomprimovať. Ak spustiteľný súbor obsahuje informácie pre ladiace nástroje, tieto sú extrahované a pripadne použité pri samotnom spätnom preklade.

Základný prvok predspracovania tvorí knižnica *fileformatl*, ktorej úlohou je detekcia formátu a prevod vstupného súboru do vnútornej reprezentácie. Moduly pre prevod nových formátov, ktoré budú vytvorené v rámci tejto práce, budú umiestnené práve v tejto časti spätného prekladača. Knižnica je bližšie popísaná v kapitole 4.

### 3.1.3 Front-end

Úlohou tejto časti spätného prekladača je preložiť strojový kód závislý na vstupnej architektúre do vnútornej, od architektúry nezávislej, reprezentácie LLVM IR. V tejto časti ďalej dochádza k niekoľkým analýzam statického kódu ako napríklad oddelenie kódu od dát, dekódovanie inštrukcií, analýza toku riadenia programu a toku dát, prípadne rekonštrukcia niektorých konštrukcií vyššej úrovne.

Ladiace a symbolické informácie je možné použiť pre získanie informácií o moduloch, ktoré boli použité pri vzniku vstupného súboru, funkciách a lokálnych či globálnych premenných, prípadne o riadkovaní. Rozpoznávanie staticky linkovaných funkcií umožňuje tieto z procesu spätného prekladu vynechať. Pre užívateľa spätného prekladača väčšinou nie sú zaujímavé, keďže ich funkcionalita je známa. Ďalšími výhodami sú zvýšená prehľadnosť výsledného kódu a väčšia rýchlosť spätného prekladu. Takto upravený kód je pripravený pre dekódovanie inštrukcií, ktoré má za úlohu previesť inštrukcie strojového kódu do reprezentácie LLVM IR.

Analýza toku riadenia programu má za úlohu rozdelenie kódu do základných blokov a vytvorenie grafov toku riadenia programu, ktoré sú neskôr použité v ďalších fázach spätného prekladu. Ďalej sa analýza pokúsi vyhľadať a rozpoznať jednotlivé funkcie. Analýza toku dát rekonštruuje dátové typy, argumenty a návratové typy funkcií. Front-end ešte obsahuje niekoľko ďalších menej dôležitých analýz ako napr. analýza dátovej sekcie a analýza zásobníku.

Posledným krokom je generovanie LLVM IR v textovej forme. Najprv sa generujú deklarácie globálnych premenných a linkovaných funkcií, následne sa generuje IR kód rozdelený do funkcií, ktoré boli rozpoznané. Ako posledné sa odošlú pomocné dáta (počet a reálne mená funkcií a premenných) pre ďalšie časti spätného prekladača. Ďalším možným výstupom môže byť kód jazyka nízkej úrovne. Narozdiel od bežného disassembleru je možné výstup obohatiť o niektoré pomocné dáta spomenuté vyššie.

### 3.1.4 Middle-end

Táto časť je zodpovedná za optimalizáciu LLVM IR reprezentácie získanej ako výstup predošlej fázy. Motivácia pre tento krok je zjednodušenie kódu odstránením prebytočných inštrukcií a vytvorenie čo najvhodnejšieho kódu pre poslednú časť spätného prekladača. Využívaný je predovšetkým nástroj *opt*, ktorý je súčasťou LLVM frameworku. Nástroj *opt* obsahuje veľké množstvo vstavaných optimalizácií a zároveň umožňuje spúšťať vlastné optimalizácie.

Vstavané optimalizácie boli pôvodne navrhnuté pre optimalizáciu kódu pri klasickom preklade. Vzhľadom na rozdielne ciele spätného prekladu je nutné niektoré optimalizácie vynechať, napr. rozloženie štruktúr na obyčajné premenné by viedlo k zhoršeniu kvality výsledného kódu. Medzi najdôležitejšie vstavané optimalizácie pre spätný preklad patrí

eliminácia mŕtveho kódu, vyhľadávanie alternatívnych prístupov k premenným (tzv. aliasy), spojenie viacerých inštrukcií do menej jednoduchších inštrukcií, zmena poradia inštrukcií komutatívnych výrazov a iné.

Ďalšou dôležitou súčasťou je vyhľadávanie inštrukčných idiómov, sekvencie niekoľkých inštrukcií, reprezentujúce malú konštrukciu jazyka vyššej úrovne. Tieto idiómy vznikajú v rámci optimalizácie kódu pri klasickom preklade, kód sa však stáva ťažšie čitateľný. Typickým príkladom takéhoto idiómu je nahradenie inštrukcie pre uloženie nuly do registru inštrukciou logického exkluzívneho súčtu. Analýza inštrukčných idiómov sa snaží vrátiť tieto inštrukcie do pôvodného stavu. Nejedná sa o vstavanú optimalizáciu.

### 3.1.5 Back-end

Táto časť spätného prekladača konvertuje optimalizovaný prechodný kód na cieľový jazyk vyššej úrovne. Konverzia prebieha vo viacerých krokoch. V prvom kroku sa vstup v reprezentácii LLVM IR prevedie na reprezentáciu BIR (Back-end intermediate representation). BIR je interná reprezentácia LLVM IR používaná v zadnej časti prekldača a je možné ju udržať v pamäti bez akejkoľvek textovej reprezentácie. BIR ďalej umožňuje modelovať všetky konštrukcie jazyka LLVM IR a vytvárať konštrukcie jazykov vyššej úrovne (if-then-else, while, for atď.), ktoré LLVM IR nepodporuje. Ďalším vstupom sú ladiace informácie, ktoré sa neskôr použijú na premenovanie premenných.

Po získaní vstupov nasleduje dodatočná optimalizácia kódu v reprezentácii BIR. Medzi najdôležitejšie optimalizácie patrí zjednodušenie aritmetických výrazov, odstránenie prebytočných priradení, premenovanie premenných a konverzia numerických konštánt na konštanty symbolické.

Posledným krokom je generovanie kódu v jazyku vyššej úrovne, grafu toku riadenia programu a grafu volania funkcií. Spätný prekladač momentálne podporuje výstup formou dvoch jazykov, jazyka C alebo upraveného jazyka Python.

# Kapitola 4

# Knižnica fileformatl

Obsah tejto kapitoly je klasifikovaný ako utajený, viď. licenčné ujednanie.

# Kapitola 5

# Formáty objektových súborov

Formát objektového súboru popisuje, ako sú informácie nachádzajúce sa v objektových súboroch uložené a štruktúrované. Podľa [11] je možno informácie obsiahnuté v objektových súboroch rozdeliť do nasledujúcich sekcií:

- **Hlavičkové informácie** – poskytujú všeobecné informácie o súbore ako dátum vytvorenia, meno zdrojového súboru alebo veľkosť kódu a rôzne príznaky ako napr. cieľová architektúru alebo bajtové usporiadanie (anglicky endianness).
- **Objektový kód** – binárne inštrukcie a dáta vygenerované prekladačom.
- **Relokačné záznamy** – zoznam miest v objektovom kóde, ktoré musia byť upravené, ak linker zmení adresy objektového kódu.
- **Symboly** – globálne symboly definované v tomto module, symboly z ostatných modulov alebo symboly definované linkerom.
- **Ladiace informácie** – informácie určené pre ladiaci program, ako napr. číslovanie riadkov kódu, lokálne symboly a popis dátových štruktúr.

Nie všetky formáty obsahujú všetky vyššie spomenuté sekcie a niektoré formáty zase definujú svoje vlastné. Podľa [11] je možné rozdeliť objektové súbory na základe ich funkcie do nasledujúcich kategórií:

- **Spustiteľné súbory** – (anglicky executable files) obsahujú objektový kód, väčšinou bez symbolov a relokačných informácií. Objektový kód je buď jeden veľký segment alebo rozdelený do malej množiny segmentov, ktoré reflektujú danú architektúru. Tento typ súboru je možné spustiť ako samostatný program.
- **Linkovateľné súbory** – (anglicky linkable files) popri objektovom kóde obsahujú veľké množstvo informácii o symboloch a relokačných informácii potrebných pre proces linkovania. Objektový kód je rozdelený do veľkého množstva logických segmentov. Tieto súbory sú vstupom pre linker, ktorý vytvorí výsledný spustiteľný súbor.
- **Súbory ukladané do pamäte** – (anglicky loadable files) môžu byť uložené do pamäte spolu so spúšťaným programom, napr. knižnice. Tento typ súboru nie je možné spustiť samostatne.

Niektoré formáty je ďalej možné previesť na iný typ formátu, ktorý väčšinou slúži inému účelu, prípadne kóduje informácie rozdielnym spôsobom. Príkladom takýchto formátov je *Intel HEX* alebo *Motorola SREC*, ktoré ukladajú výsledné informácie formou ASCII textu. Ich výhodou je napr. možnosť ich použitia v spolupráci s nástrojmi, ktoré nepodporujú binárne data (textový edtior, posielanie pomocou SMTP).

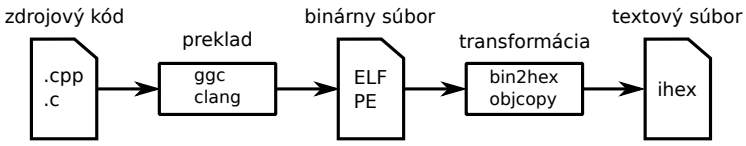
## 5.1 Intel HEX

Táto podkapitola bola spracovaná na základe [9].

Intel HEX je formát navrhnutý spoločnosťou Intel Corporation, ktorý ukladá binárne informácie formou ASCII textu – nejedná sa tak o binárny formát v úplnom zmysle slova. Využitie formátu Intel HEX nájdeme predovšetkým v nástrojoch pre programovanie mikrokontrolérov, zapisovanie informácií do ROM, PROM a EPROM pamätí, prípadne pre simulácie softvéru v rámci jednotlivých vývojových prostredí. Ďalší rozdiel oproti bežným binárnym súborom je, že formát neposkytuje informácie o tom, čo binárne informácie obsiahnuté v súbore predstavujú – nenájdeme tu informácie o bajtovom usporiadaní či cieľovej architektúre, tieto informácie musíme poznať alebo získať iným spôsobom. Chýbajú tiež segmenty a sekcie, ktoré nájdeme v objektových súboroch iných formátov.

### 5.1.1 Vytvorenie súboru s formátom Intel HEX

Intel HEX súbor nevzniká kompiláciu zdrojového súboru určitého jazyka, ale konverziou už kompilátorom vytvoreného binárneho súboru. Väčšina Unixových systémov poskytuje spolu s prekldačom nástroje na to určené, konktérne ide o programy *bin2hex* alebo *objcopy*. Postup ilustruje obrázok 5.1.




Obr. 5.1: Vznik súboru s formátom Intel HEX.

### 5.1.2 Štruktúra formátu Intel HEX

Formát Intel Hex ukladá binárne informácie ako hexadecimálne číslo v ASCII podobe (využíva znaky *'a'* až *'f'* alebo *'A'* až *'F'* a číslice *'0'* až *'9'*). Jeden bajt binárnej informácie je zakódovaný dvojicou ASCII znakov, výsledná reprezentácia dát tak zaberie dvakrát viac bajtov než pôvodná binárna reprezentácia. V poradí prvý znak jednej dvojice v súbore vždy predstavuje štyri najvýznamnejšie bity (anglicky most significant bits).

Súbor je rozdelený na záznamy, väčšinou každý uložený na práve jednom riadku (špecifikácia sa o riadkovaní nezmieňuje, väčšina nástrojov ho však pre lepšiu čitateľnosť používa). Začiatok nového záznamu je označený symbolom dvojbodky (ASCII #58), záznam ďalej obsahuje nasledujúce informácie:

- **Veľkosť dát** – je určená dvomi ASCII znakmi, maximálna veľkosť dátovej časti jedného záznamu je tak obmedzená na 255 bajtov binárnej informácie resp. 510 ASCII znakov. Prakticky je však kôli čitateľnosti toto číslo oveľa menšie.

- **Adresa** – je určená štyrmi ASCII znakmi, máme tak k dispozícií 16-bitovú adresu. Pre zápis 32-bitovej adresy potrebujeme špeciálny záznam, ktorý poskytne najvýznamnejších 16 bitov 32-bitovej adresy. Ak takýto záznam chýba, uvažuje sa nula. Adresa je vo formáte Intel HEX vždy uložená v bajtovom usporiadaní big endian.
- **Typ záznamu** – bajtová hodnota určená dvomi ASCII znamki identifikujúca typ záznamu. Môže nadobúdať hodnoty 0 až 5.
- **Dáta** – samotné dáta záznamu. Formát nepredpisuje žiadne usporiadanie, v prípade potreby tejto informácie je nutné použiť iný spôsob pre jej zistenie.
- **Kontrolný súčet** – bajtová hodnota slúžiaca pre overenie správnosti záznamu. Hodnotu vypočítame ako súčet jednotlivých bajtov záznamu s jeho následnou negáciou (prípadné pretečenie pri sčítaní neberieme do úvahy).

```
     adresa                    dáta
 :10 8564 00 60000000B6FEFFFF190000000410E08 85
 :10 8574 00 8502420D0555C50C0404000038000000 B6
 :10 83E0 00 D0C9E979FFFFFF90E973FFFFFF5589E5 E9
velkosť   typ                                kontrolný súčet
```

Obr. 5.2: Ukážka formátu Intel HEX.

Formát Intel HEX definuje nasledujúce typy záznamov:

- **Dátový záznam** – obsahuje dáta a spodných 16 bitov ich adresy.
- **Záznam pre koniec súboru** – označuje koniec súboru, vyskytuje sa práve raz ako posledný záznam v súbore. Dátová časť je prázdna a adresa je typicky 0.
- **Záznam pre 20-bitové adresovanie** – poskytuje 16-bitovú segmentovú adresu pre *real mode* adresovanie architektúry 80x86.
- **Záznam pre zápis obsahu registrov CS:IP** – špecifikuje obsah registrov CS (Code Segment) a IP (Instruction Pointer). Jedná sa o vstupný bod programu pri použití 20-bitového adresovania.
- **Záznam pre rozšírenie lineárnej adresy** – umožňuje 32-bitové adresovanie. Adresová časť záznamu sa ignoruje a dátová časť obsahuje najvýznamnejších 16 bitov adresy. Adresa nasledujúcich záznamov tak vznikne súčtom poskytnutej hodnoty posunutej o 16 bitov doľava a hodnoty uloženej v adresovej časti nasledujúcich záznamov. Táto hodnota je platná, pokiaľ nie je prepísaná novým záznamom pre rozšírenie adresy. Ak sa tento záznam v súbore nevyskytne pred dátovým záznamom, uvažuje sa nula.
- **Záznam pre zápis obsahu registru EIP** – špecifikuje 32-bitovú hodnotu, ktorá bude nahraná do registru EIP. Register EIP (Extended Instruction Pointer) v architektúre x86 uchováva adresu nasledujúcej inštrukcie, v prípade Intel HEX formátu sa jedná o prvú vykonanú inštrukciu – vstupný bod programu.

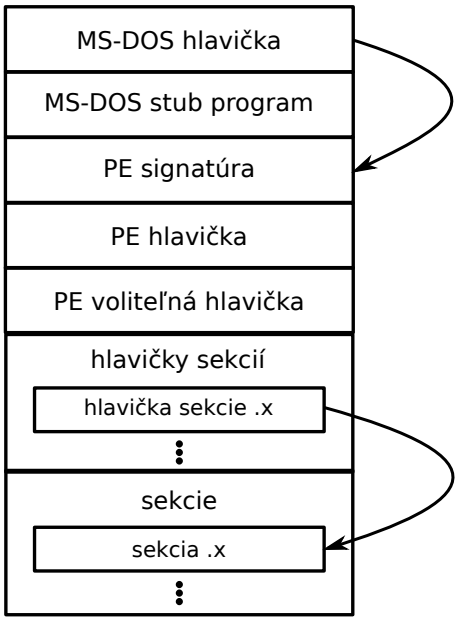
## 5.2 Portable Executable

Táto podkapitola bola spracovaná na základe [14].

Formát Portable Executable (PE) je formát pre spustiteľné súbory, linkovateľné súbory alebo dynamicky linkované knižnice (DLL). Formát bol vyvinutý spoločnosťou Microsoft a prvýkrát predstavený s operačným systémom Windows NT 3.1 a používa sa dodnes v tridsaťdva a šesťdesiatštyri bitových verziách OS Windows. Formát je založený na Unixovom formáte COFF a ako názov napovedá, formát je prenosný a teda nie je viazaný na žiadnu konkrétnu cieľovú architektúru.

### 5.2.1 Štruktúra formátu Portable Executable

Štruktúra formátu je podobná všeobecnej štruktúre spomenutej na začiatku kapitoly 5. Nájdeme tu hlavičku a dáta logicky rozdelené na viaceré časti – sekcie. Štruktúra formátu je zobrazená na obrázku 5.3.




Obr. 5.3: Štruktúra formátu Portable Executable. Prevzaté z [14], upravené.

#### MS-DOS stub

Prvá časť súboru pozostáva z MS-DOS hlavičky a MS-DOS stub programu, ktorý je schopný behu v reálnom móde operačného systému MS-DOS. Jeho účelom je vypísanie hlášky *„This program cannot be run in DOS mode“* alebo inej, pokiaľ bola špecifikovaná pri linkovaní súboru. Zabezpečuje tak kompatibilitu s OS MS-DOS. Lokácia 0x3C obsahuje posuv PE signatúry od začiatku súboru, umožňuje tak preskočiť MS-DOS stub program, ktorého dĺžka nie je pevná.

### PE signatúra

Signatúra jednoznačne identifikujúca formát súboru ako Portable Executable. Pozostáva zo štyroch bajtov obsahujúcich sekvenciu 'PE\0\0'.

### PE hlavička

PE hlavička sa nachádza na začiatku linkovateľného súboru alebo ihneď za PE signatúrou v prípade spustiteľného súboru. Informácie obshiahnuté v PE hlavičke popisuje tabuľka 5.1.

| Pozícia | Počet bajtov | Popis |
|---|---|---|
| 0 | 2 | Identifikátor cieľovej architektúry. |
| 2 | 2 | Počet sekcií. |
| 4 | 2 | Časové razítko. |
| 8 | 4 | Ukazovateľ na tabuľku symbolov. |
| 12 | 4 | Počet symbolov v tabuľke symbolov. |
| 16 | 2 | Veľkosť voliteľnej hlavičky |
| 18 | 2 | Charakteristika (príznaky) súboru. |

Tabuľka 5.1: Štruktúra PE hlavičky. Prevzaté z [14], upravené.

### Voliteľná hlavička

Hneď za PE hlavičkou sa nachádza voliteľná hlavička. Voliteľnosť závisí od typu súboru, v spustiteľných súboroch je voliteľná hlavička povinná, v linkovateľných súboroch táto hlavička nemá význam. Veľkosť voliteľnej hlavičky nie je fixná, jej veľkosť je určená poľom *SizeOfOptionalHeader* PE hlavičky súboru. Voliteľná hlavička má vlastnú signatúru, ktorá určuje, či sa jedná o 32-bitový (*PE32*) alebo 64-bitový (*PE32+*) súbor. Voliteľnú hlavičku môžeme rozdeliť na tri hlavné časti:

- **Štandardné polia** – tieto polia sú definované pre všetky implementácie COFF hlavičky. Polia popisuje tabuľka 5.2.
- **Polia špecifické pre OS rodiny Windows** – dodatočné polia pre podporu špecifických funkcií OS Windows.
- **Definície špeciálnych tabuliek** – tieto polia tvoria páry adresa-veľkosť, ktoré určujú pozíciu a veľkosť špeciálnych tabuliek, ktoré sa nachádzajú v súbore. Príkladom môže byť tabuľka exportovaných symbolov alebo tabuľka importovaných symbolov.

### Hlavičky sekcií

Hlavičky sekcií sa nachádzajú hneď za voliteľnou hlavičkou. Počet sekcií je daný poľom *NumberOfSections* PE hlavičky súboru. Jednotlivé hlavičky sú číslované (od 1) a ich poradie je dané procesom linkovania. Každá hlavička sekcie má veľkosť 40 bajtov a jej formát popisuje tabuľka 5.3.

| Pozícia | Počet bajtov | Popis |
|---|---|---|
| 0 | 2 | Signatúra voliteľnej hlavičky. |
| 2 | 1 | Verzia použitého linker-u (veľké číslo). |
| 3 | 1 | Verzia použitého linker-u (malé číslo). |
| 4 | 4 | Veľkosť sekcie (sekcií) obsahujúcej kód. |
| 8 | 4 | Veľkosť sekcie (sekcií) pre inicializované dáta. |
| 12 | 4 | Veľkosť sekcie (sekcií) pre ne-inicializované dáta. |
| 16 | 4 | Adresa vstupného bodu po načítaní do pamäti. |
| 20 | 4 | Adresa začiatku kódovej sekcie po načítaní do pamäti. |
| 24 | 4 | Adresa začiatku dátovej sekcie po načítaní do pamäti. |

Tabuľka 5.2: Štandardné polia voliteľnej hlavičky. Prevzaté z [14], upravené.

| Pozícia | Počet bajtov | Popis |
|---|---|---|
| 0 | 8 | Názov sekcie. |
| 8 | 4 | Veľkosť sekcie po načítaní do pamäte. |
| 12 | 4 | Virtuálna adresa po načítaní do pamäte. |
| 16 | 4 | Veľkosť sekcie (alebo inicializovaných dát) na disku. |
| 20 | 4 | Ukazovateľ na výskyt sekcie v súbore. |
| 24 | 4 | Ukazovateľ na výskyt relokačných záznamov v súbore. |
| 28 | 4 | Ukazovateľ na výskyt záznamov riadkovania pre ladenie. |
| 32 | 2 | Počet relokačných záznamov. |
| 34 | 2 | Počet záznamov riadkovania. |
| 36 | 4 | Príznaky sekcie. Predovšetkým R/W práva a typ dát. |

Tabuľka 5.3: Štruktúra hlavičky sekcie. Prevzaté z [14], upravené.

#### Sekcie

Inicializované dáta sekcií pozostávajú z blokov bajtov. V prípade, že je celá sekcia inicializovaná na nuly, tieto dáta nemusia byť zahrnuté v súbore. Veľkosť a pozícia konkrétnej sekcie v súbore sú uložené v hlavičke danej sekcie. Ak je veľkosť sekcie na disku menšia, než veľkosť sekcie po načítaní do pamäti, rozdiel je inicializovaný na nuly. V prípade spustiteľného súboru musí poradie sekcií zohľadňovať ich virtuálnu adresu a požiadavky na zarovnanie dané poľom *FileAlignment* voliteľnej hlavičky.

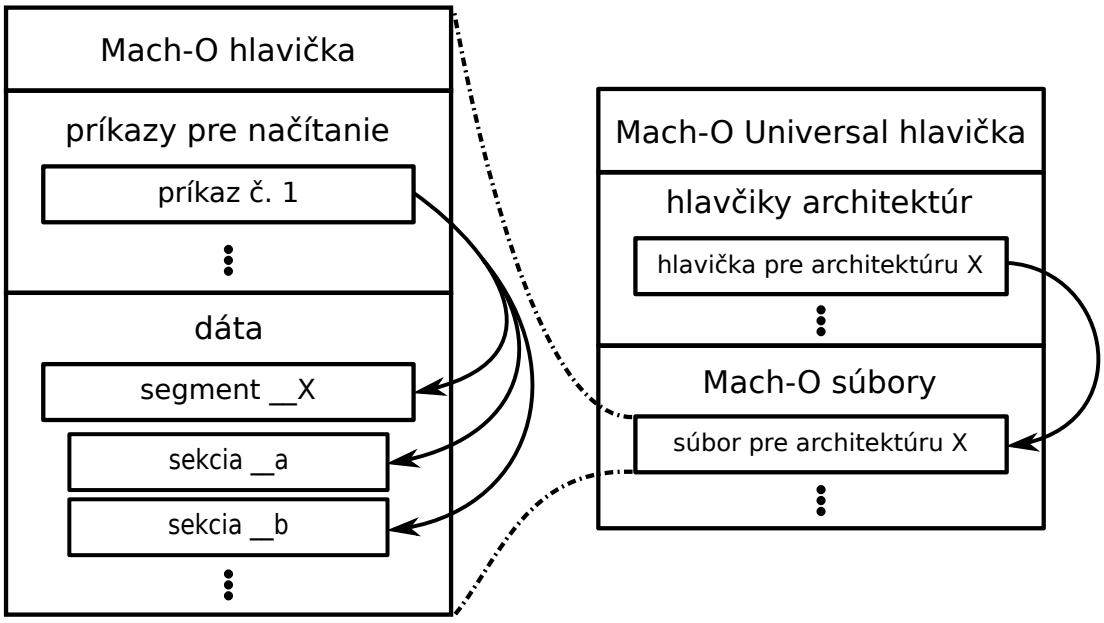
## 5.3 Mach-O

Táto podkapitola bola spracovaná na základe [1] a verejne dostupných zdrojových kódov.

Formát Mach-O je binárny formát vyvíjaný spoločnosťou Apple Inc. a používaný v operačných systémoch OS X a iOS. Formát je primárne použitý pre spustiteľné a linkovateľné súbory a dynamické knižnice, menej často pre uloženie ladiacich informácií, výpis pamäti (anglicky core dump), modulov (anglicky bundles), dynamických linkerov alebo OS X a iOS ovládačov (anglicky kernel extensions).

### 5.3.1 Štruktúra formátu Mach-O

Štruktúra formátu je mierne odlišná od bežnej štruktúry objektových formátov. Na začiatku súboru sa nachádza tradičná hlavička so signatúrou formátu, za ňou nasleduje vopred špecifikovaný počet príkazov pre načítanie binárneho súboru (anglicky load commands). Za týmito príkazmi sa nakoniec nachádzajú samotné dáta. Štruktúra formátu je zobrazená na obrázku 5.4.




Obr. 5.4: Štruktúra formátu Mach-O (vľavo) a Mach-O Universal Binary (vpravo).

#### Signatúra

Signatúra je súčasťou hlavičky a identifikuje formát súboru ako Mach-O alebo Mach-O Universal Binary. Existuje celkom päť signatúr popísaných v tabuľke 5.4.

| Signatúra | Bitová šírka | Usporiadanie bajtov |
|---|---|---|
| `0xFEEDFACE` | 32 bitov | big endian |
| `0xFEEDFACF` | 64 bitov | big endian |
| `0xCEFAEDFE` | 32 bitov | little endian |
| `0xCFFAEDFE` | 64 bitov | little endian |
| `0xCAFEBABE` | univerzálna | univerzálne |

Tabuľka 5.4: Mach-O signatúry

#### Hlavička

Hlavička ďalej poskytuje informácie o cieľovej architektúre, type a vlastnostiach súboru a o celkovom počte a veľkosti príkazov pre načítanie súboru.

| Pozícia | Počet bajtov | Popis |
|---|---|---|
| 0 | 4 | Signatúra formátu. |
| 4 | 4 | Rodina cieľovej architektúry, napr. ARM. |
| 8 | 4 | Bližšia špecifikácia architektúry, napr. ARMv7. |
| 12 | 4 | Funkcia súboru (spustiteľný súbor, knižnica). |
| 16 | 4 | Počet príkazov pre načítanie. |
| 20 | 4 | Veľkosť príkazov pre načítanie súboru v bajtoch. |
| 24 | 4 | Príznaky (anglicky flags). |
| 28 | 4 | Rezervované (len 64-bit verzia formátu Mach-O). |

Tabuľka 5.5: Štruktúra Mach-O hlavičky. Prevzaté z [1], upravené.

### Príkazy pre načítanie súboru

Príkazy pre načítanie súboru (ďalej len príkazy) popisujú celkové rozloženie dát v binárnom súbore, napr. segmenty a sekcie, tabuľky symbolov, exportov a importov, vstupný bod, použité dynamické a statické knižnice, prípadne informácie o použitom zabezpečení a cieľovom operačnom systéme. Medzi najdôležitejšie príkazy patria:

- `LC_SEGMENT` a `LC_SEGMENT_64` – načítanie segmentov a sekcií. Počet príkazov tohto typu odpovedá počtu segmentov v binárnom súbore. Za príkazom pre načítanie segmentov môže nasledovať pole štruktúr obsahujúcich dáta pre načítanie sekcií. Informácia o počte sekcií daného segmentu je obsiahnutá v rámci samotného príkazu.
- `LC_UNIXTHREAD` – stav registrov pred štartom aplikácie (OS X v.10.5 a menej).
- `LC_MAIN` – pozícia vstupného bodu programu v súbore a veľkosť zásobníku pre hlavné vlákno programu (OS X v.10.6 a viac).
- `LC_SYMTAB` – pozícia tabuľky symbolov a tabuľky reťazcov vo vstupnom súbore a ich veľkosti.
- `LC_DYSYMTAB` – začiatočný index jednotlivých typov symbolov v tabuľke symbolov (lokálne symboly, exporty a importy) a pozícia a veľkosť mnohých ďalších, pre spätný preklad menej dôležitých, tabuliek. Tento príkaz je závislý na príkaze `LC_SYMTAB`, ktorý popisuje uchovanie samotných symbolov.
- `LC_DYLD_INFO` a `LC_DYLD_INFO_ONLY` – alternatívny príkaz pre načítanie importov a exportov (OS X v.10.8 a viac). Tento príkaz vie poskytnúť informácie nezávisle na existencii príkazov `LC_SYMTAB` a `LC_DYSYMTAB`.
- `LC_LOAD_DYLIB` – získanie názvov použitých dynamických knižníc. Načítanie importov a exportov je závislé na tomto príkaze. Počet príkazov tohto typu odpovedá počtu použitých dynamických knižníc.

Podľa enumerácie uvedenej v zdrojových súborov projektu LLVM existuje celkom 47, viac či menej dokumentovaných príkazov. Nemusí sa však jednať o všetky existujúce príkazy. S výnimkou niekoľkých prípadov, ktoré si vyžadujú špecifické usporiadanie, keď napr. príkaz `LC_DYSYMTAB` musí pre správne fungovanie predchádzať príkaz `LC_SYMTAB`, nie je poradie príkazov nijak stanovené. Všeobecný formát príkazu popisuje tabuľka 5.6. Zvyšná časť je rozdielna pre každý príkaz a závisí od typu príkazu.

| Pozícia | Počet bajtov | Popis |
|---|---|---|
| 0 | 4 | Typ príkazu. |
| 4 | 4 | Celková veľkosť príkazu v bajtoch. |

Tabuľka 5.6: Štruktúra Mach-O príkazu. Prevzaté z [1], upravené.

### 5.3.2 Mach-O Universal Binary

Formát Mach-O Universal Binary vznikol ako pomôcka pre prechod z architektúry PowerPC na architektúru x86, ktorý firma Apple Inc. uskutočnila v roku 2005. Súbor tohto formátu vie pojať ľubovoľný počet architektúr, väčšinou sa však jedná kombinácie architektúr PowerPC a Intel x86 alebo odlišné verzie ARM architektúr v prípade systému iOS. Formát môže obsahovať všetky typy Mach-O súborov spomenuté vyššie ako aj celé statické knižnice, väčšinou zabalené Unixovým formátom archiver. Štruktúra formátu je zobrazená na obrázku 5.4.

#### Hlavička

Hlavička obsahuje signatúru `0xCAFEBABE` (vždy big endian) a informáciu o počte architektúr obsiahnutých v tomto súbore.

#### Hlavičky architektúr

Za úvodnou hlavičkou nasleduje zoznam hlavičiek architektúr, ktoré špecifikujú pozíciu v súbore a celkovú veľkosť dát pre konkrétnu architektúru. Hlavičky architektúr sú uložené v bajtovom usporiadaní typu big endian.

| Pozícia | Počet bajtov | Popis |
|---|---|---|
| 0 | 4 | Rodina cieľovej architektúry, napr. ARM. |
| 4 | 4 | Bližšia špecifikácia architektúry, napr. ARMv7. |
| 8 | 4 | Pozícia v súbore pre konkrétnu architektúru. |
| 12 | 4 | Vekosť dát v súbore pre konkrétnu architektúru. |
| 16 | 4 | Zarovnanie dát. |

Tabuľka 5.7: Štruktúra Mach-O Universal Binary hlavičky. Prevzaté z [1], upravené.

# Kapitola 6

# Spätný preklad súborov formátu Intel HEX

Obsah tejto kapitoly je klasifikovaný ako utajený, viď. licenčné ujednanie.

# Kapitola 7

# Spätný preklad súborov formátu Portable Executable

Obsah tejto kapitoly je klasifikovaný ako utajený, viď. licenčné ujednanie.

# Kapitola 8

# Spätný preklad súborov formátu Mach-O

Obsah tejto kapitoly je klasifikovaný ako utajený, viď. licenčné ujednanie.

# Kapitola 9

# Záver

V tejto práci boli predstavené základy softvérového reverzného inžinierstva, predovšetkým spätný preklad binárnych súborov. Ďalej bol v práci predstavený rekonfigurovateľný spätný prekladač vyvíjaný spoločnosťou *AVG Technologies CZ, s.r.o.* s hlavným zameraním na spracovanie vstupných súborov. Následne boli predstavené formáty Intel HEX, Portable Executable, Mach-O a Mach-O Universal Binary, ktorých moduly boli následne navrhnuté, implementované a testované.

Do spätného prekladača tak bola úspešne pridaná podpora dvoch resp. troch nových formátov – Intel HEX, Mach-O a Mach-O Universal Binary. Tiež došlo k pridaniu alternatívnej implementácie modulu pre formát Portable Executable založenej na LLVM. Spätný prekladač sa tak môže chváliť najväčším počtom podporovaných formátov nezávisle od softvéru tretích strán a tiež podporou formátov troch na trhu najviac zastúpených rodín operačných systémov – Windows, Unix a OS X respektíve iOS. Nasadenie modulov pre formáty Mach-O a Intel HEX do užívateľského vydania produktu by sa malo uskutočniť v nasledujúcich týždňoch. Modul pre formát Portable Executable bude pred nasadením vyžadovať ešte niekoľko potrebných úprav.

Testovanie overilo schopnosť modulov spracovať súbory daných formátov a poskytnúť informácie potrebné pre spätný preklad alebo nástroj *fileinfo*. V prípade modulu pre formát Mach-O tiež bola testovaná schopnosť správne identifikovať zašifrované binárne súbory a schopnosť vybrať správnu architektúru v prípade formátu Mach-O Universal Binary. Testovanie bolo vykonané na umelo vytvorených vstupných súboroch ale aj reálnych vzorkách súborov získaných z online služby pre spätný preklad.

## 9.1 Budúci vývoj

Vzhľadom na jednoduchosť formátu Intel HEX prakticky neexistuje priestor pre pridanie novej funkcionality v rámci modulu. Ak by v budúcnosti prišla požiadavka na implementáciu podobného ASCII formátu (napr. SREC), bude možné použiť veľkú časť už napísaného kódu. Toto však bude vyžadovať určité úpravy tried, ktoré dnes nijako nepodporujú prípadné pridanie podpory nových formátov.

V prípade ďalšieho vývoja podpory formátu PE pomocou LLVM je najskôr nutné odpovedať na dve otázky. Prvou otázkou je, či vzhľadom na rozsiahle nedostatky knižnice *object* má vôbec význam vo vývoji pokračovať. Knižnica *PeLib* má síce tiež niekoľko nedostatkov a zastaralý kód, ale už implementovaná funkcionalita funguje správne a knižnica je úspešne nasadená v rámci spätného prekladača. V prípade použitia LLVM bude nutné venovať

množstvo času implementácii základných vecí, ako napr. načítanie importov do vnútornej reprezentácie. Výhodou je, že LLVM je aktívne vyvíjaný projekt. Druhá otázka sa týka prístupu k riešeniu samotných problémov. Vlastné riešenie bude krátkodobo efektívne, prinesie však množstvo nového kódu s potrebou údržby. Oprava LLVM knižnice *object* bude výrazne pomalšia a náročnejšia, ale v prípade prijatia navrhovaných opráv ľahko udržateľná a prínosná nielen pre projekt spätného prekladača. Finálne rozhodnutie nie je v kompetencii autora tejto práce, prípadné odporúčanie je však pokračovať vo vývoji formou opráv.

V rámci formátu Mach-O je dostupných niekoľko ďalších príkazov, ktoré je možné spracovať primárne pre výpis v rámci nástroja *fileinfo* či zlepšenie kvality prekladu niektorých binárnych súborov. Veľký problém predstavuje nedostatok oficiálnej kvalitnej dokumentácie popisujúcej štruktúru formátu. Už implementácia niektorých základných častí potrebných pre preklad či zlepšenie jeho kvality vychádza z neoficiálnych článkov, prípadne analýzy dostupných zdrojových kódov OS X alebo reverzného inžinierstva.

Keďže veľké percento univerzálnych binárnych súborov formátu Mach-O tvoria statické knižnice, bude vhodné v budúcnosti pridať podporu formátu *archiver*.

# Literatúra


[1] Apple Computer, Inc.: *Mach-O Runtime Architecture*. 2004.

[2] AVG Technologies CZ, s.r.o.: *Retargetable Decompiler* [online]. 2016. [cit. 2016-01-15]. Dostupné z: https://retdec.com/home/.

[3] Backer Street Software: *REC Studio 4 - Reverse Engineering Compiler* [online]. 2016. [cit. 2016-01-15]. Dostupné z: http://www.backerstreet.com/rec/rec.htm.

[4] C4IT Ltd.: *C4Decompiler, the C Decompiler for Windows x64* [online]. 2016. [cit. 2016-01-15]. Dostupné z: http://www.c4decompiler.com/.

[5] Cifuents, C.: *Reverse Compilation Techniques*. Dizertačná práca, Queensland University of Technology, 1994.

[6] Cryptic Apps SARL: *Hopper FAQ* [online]. 2016. [cit. 2016-01-15]. Dostupné z: http://www.hopperapp.com/faq.html.

[7] Eilam, E.: *Reversing: Secrets of Reverse Engineering*. Indianapolis: Wiley, 2005. ISBN 0-7645-7481-7.

[8] Hex-Rays SA.: *Hex-Rays Decompiler: Support* [online]. 2016. [cit. 2016-01-15]. Dostupné z: https://www.hex-rays.com/products/decompiler/support.shtml.

[9] Intel Corporation: *Hexadecimal Object File Format Specification*. 1988.

[10] Křoustek, J.: *Rekonfigurovatelná analýza strojového kódu*. Dizertačná práca, Fakulta informačních technologií VUT v Brně, 2014.

[11] Levine, J. R.: *Linkers and Loaders*. San Francisco: Morgan Kaufmann, 2000. ISBN 1-55860-496-0.

[12] LLVM Project: *LLVM 3.9 documentation* [online]. 2015. [cit. 2015-12-11]. Dostupné z: http://llvm.org/docs/.

[13] LLVM Project: *The LLVM Compiler Infrastructure Project* [online]. 2015. [cit. 2015-12-11]. Dostupné z: http://llvm.org/.

[14] Microsoft Corporation: *Microsoft Portable Executable and Common Object File Format Specification*. 2013.

[15] Net Applications: *Operating System Market Share* [online]. 2016. [cit. 2016-02-02]. Dostupné z: www.netmarketshare.com/operating-system-market-share.aspx.

[16] SmartDec Project: *About SmartDec* [online]. 2016. [cit. 2016-01-15]. Dostupné z: http://decompilation.info/about.

[17] *Snowman* [online]. 2016. [cit. 2016-01-15]. Dostupné z: https://derevenets.com/.

[18] Ďurfina, L.; Křoustek, J.; Matula, P.; aj.: A Novel Approach to Online Retargetable Machine-Code Decompilation. *Journal of Network and Innovative Computing*, ročník 2, č. 1, 2014: s. 224–232.

# Prílohy

# Zoznam príloh

**A Obsah CD** **32**

# Príloha A

# Obsah CD

CD obsahuje:

- Text práce vo formáte PDF.