# Aplikace pro porovnávání a úpravu .hex a .s19 souborů

Bc. Jakub Anděl

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Jakub Anděl** |
| Osobní číslo: | **A22288** |
| Studijní program: | **N0613A140022 Informační technologie** |
| Specializace: | **Softwarové inženýrství** |
| Forma studia: | **Prezenční** |
| Téma práce: | **Aplikace pro porovnávání a úpravu .hex a .s19 souborů** |
| Téma práce anglicky: | **Application for Comparing and Editing .hex and .s19 Files** |

## Zásady pro vypracování

1. Podrobně se seznamte se strukturami souborů .s19 a .hex.
2. Seznamte se s existujícími aplikacemi pro porovnávání souborů.
3. Navrhněte vlastní srovnávací aplikaci včetně uživatelského rozhraní.
4. Naprogramujte tuto aplikaci a doplňte ji vhodnými podpůrnými funkcemi usnadňujícími porovnávání.
5. Ověřte funkčnost aplikace na vybraných .s19 a .hex souborech.
6. Vytvořenou aplikaci podrobně zdokumentujte.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:


1. TANNA, Sunil. Binary, Octal and Hexadecimal for Programming & Computer Science. CreateSpace Independent Publishing Platform, 2018. ISBN 9781722300548.
2. SANCHEZ, Julio a Maria P. CANTON. Microcontrollers: High-Performance Systems and Programming. Taylor & Francis, 2013. ISBN 9781466566651.
3. Porty, bajty, osmibity: počítače na koleni. Praha: CZ.NIC, z.s.p.o., 2019. CZ.NIC. ISBN 978-80-88168-39-3.
4. J. PRICE, Mark. C# 11 and .NET 7 – Modern Cross-Platform Development Fundamentals – Seventh Edition. 7th ed. Packt Publishing, 2022. ISBN 1803237805.
5. BAIDACHNYI, Sergii. Developing Windows 10 Applications with C#. CreateSpace Independent Publishing Platform, 2016. ISBN 978-1522894919.
6. COHN, Ronald, RUSSELL, Jesse, ed. Srec (File Format). Book on Demand, 2013. ISBN 978-5511896397.
7. Intel HEX File Format – Developer Help. MICROCHIP TECHNOLOGY, INC. Intel HEX File Format [online]. 2021 [cit. 2023--10-30]. Dostupné z: https://microchipdeveloper.com/ipe:sqtp-hex-file-format
8. SB-Projects. Motorola Sxx records format [online]. 2023 [cit. 2023-10-30]. Dostupné z: https://www.sbprojects.net/knowledge/fileformats/motorola.php


Vedoucí diplomové práce: **Ing. Pavel Pokorný, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**


**doc. Ing. Jiří Vojtěšek, Ph.D.** v.r.
děkan

**prof. Mgr. Roman Jašek, Ph.D., DBA** v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 13.5.2024

Jakub Anděl, v.r.
podpis studenta

## ABSTRAKT

Předmětem diplomové práce je vytvoření desktopové aplikace pro porovnání .hex a .s19 souborů. Teoretická část se zabývá popisem jednotlivých formátů souborů, analýzou a porovnání již existujících aplikací pro porovnání hexadecimálních souborů a představením použitých technologií. Praktická část řeší návrh a implementaci desktopové aplikace, testováním její funkcionality, další možná vylepšení a srovnáním s již existujícími aplikacemi. Výsledkem této práce je desktopová aplikace pro porovnání souborů .hex a .s19, vyvinutá pomocí platformy WPF .NET a programovacího jazyka C#.



Klíčová slova: WPF (rozhraní),C# (programovací jazyk),Microsoft Visual C#.NET (software),desktopové aplikace,Intel (mikrokontroléry),Motorola (mikrokontroléry), hexadecimální soubory



## ABSTRACT

The subject of this thesis is the development of a desktop application for comparing .hex and .s19 files. The theoretical part deals with the description of the individual file formats, analysis and comparison of existing applications for comparing hexadecimal files, and introduction of the technologies used. The practical part addresses the design and implementation of the desktop application, testing its functionality, further possible improvements and comparison with existing applications. The result of this work is a desktop application for comparing .hex and .s19 files, developed using the WPF .NET platform and the C# programming language.



Keywords: Windows Presentation Foundation (interface), C# (programming language), Microsoft Visual C#.NET (software), desktop applications,Intel (microcontrollers), Motorola (microcontrollers), hexadecimal files

Chtěl bych poděkovat svému vedoucímu diplomové práce Ing. Pavlu Pokornému, Ph.D. za odborné vedení, za pomoc a rady při zpracování této práce. Dále bych rád poděkoval své rodině a přítelkyni, kteří by mi byli oporou během celého studia.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

# OBSAH

# ÚVOD

Hexadecimální soubory zůstávají klíčovým prvkem v mnoha odvětvích technologie a softwarového vývoje a pro profese pohybující v těchto oborech je důležité chápat a mít schopnost s nimi pracovat, jelikož mohou pomoct při porozumění fungování systémů a detailnější analýzu dat. Jejich využití je se uplatní zejména při práci s nízkoúrovňovými programovacími jazyky a interakci s hardwarem. Například v oblasti vestavěných systémů jsou tyto soubory stále běžně využívány pro programování mikrokontrolérů a dalších elektronických zařízení. Hexadecimální formát umožňuje přesnou specifikaci paměťových adres a dat pro správné načítání a provádění programu. Dále se vývojáři mohou setkat s tímto typem souborů v oblasti ladění a debuggování softwaru, kdy při analýze chyb nebo optimalizaci výkonu je často nutné prozkoumat obsah paměti nebo sledovat změny v registrech a jiných hardwarových zařízeních. V bezpečnostní sféře pak slouží soubory jako nástroj pro analýzu malwaru, identifikaci zranitelností a zkoumání síťové komunikace. Pomáhají tak vytvořit obraz paměti nebo sledovat podezřelé aktivity v síti. Pro práci s tímto typem souborů se využívají hexadecimální editory, ve kterých je možné analyzovat obsah, modifikovat data a porovnat soubory různých formátů a mojí úlohou je takový editor podporující formáty Intel HEX a S19 navrhnout a vytvořit.

Teoretická část práce se zabývá popisem běžně používaných souborových formátů Intel HEX a Motorola SRecord. Hlavním cílem je detailní popis struktury záznamů, různých druhů, které se mohou v těchto formátech vyskytovat, a procesů pro převod na čitelný text a výpočet kontrolního součtu. Další kapitolou je analýza vybraných existujících hexadecimálních editorů, které jsou již na trhu běžně zavedené, a srovnání nejen funkcí, ale taktéž na základě použitelnosti.

Praktická část diplomové práce se soustředí na návrh aplikace, definování funkčních a nefunkčních požadavků a vytvoření uživatelského rozhraní. Následuje podrobná dokumentace, která popisuje implementaci výsledného programu, strukturu souborů, použité podpůrné třídy a implementované funkce. Na základě výsledků testování aplikace na poskytnutých datech se pak vyhodnotí celkový stav a navrhují se možná rozšíření stávajících funkcí a přidání nových funkcionalit pro zlepšení výsledné aplikace. Na základě výsledků testování aplikace na poskytnutých datech se pak vyhodnotí celkový stav a navrhují se možná rozšíření stávajících funkcí a přidání nových funkcionalit pro zlepšení výsledné aplikace.

Závěrečná část praktické části obsahuje srovnání vytvořené aplikace s existujícími nástroji na základě jejich funkcí a možností. Tímto srovnáním se zkoumají přednosti a nedostatky různých softwarových řešení a zjišťuje se, jakou hodnotu a konkurenční výhodu přináší nově vytvořená aplikace ve srovnání s existujícími alternativami.

# I. TEORETICKÁ ČÁST

# 1 SOUBOROVÝ FORMÁT S19

Tento souborový formát pro práci s mikroprocesory a mikrokontrolery v hexadecimálním zápisu s příponou .s19 je zařazen do rodiny S-Record (SREC), kterou vyvinula společnost Motorola. Tento formát je také znám pod názvy Exorciser, Exormacs nebo Exormax. [1] Hlavním cílem tohoto formátu je kódovat binární informace pomocí ASCII znaků. Existují celkem tři typy SREC, přičemž základním je S19, který využívá 16bitové adresy a maximální velikost souboru v tomto formátu je omezena na 64 kB. Dalším typem je formát S28 s 24bitovými adresami, který umožňuje pracovat se soubory o velikosti až 16 MB. Největším formátem je S37, který používá 32bitové adresy a umožňuje manipulaci se soubory o velikosti až 4 GB. [2]

## 1.1 Obsah souboru

Soubor se skládá z binárních dat, kde každý bajt je zakódován v hexadecimálním formátu o 2 znacích a první znak označuje 4 bity vyššího řádu a druhý znak pak 4 bity nižšího řádu. Každý řádek v souboru je nazýván záznamem a obsahuje pět individuálních sekcí. [3]

První sekcí je identifikátor tohoto formátu, který na začátku záznamu začíná písmenem „S“ následovaným dvěma číslicemi od 0 do 9. Například v případě formátu S19 označuje první číslice 1 datový záznam s 16bitovou adresou a druhá číslice 9 signalizuje znak konce řádku (End of line) na konci záznamu. Samotné názvy formátů SREC jsou odvozeny právě od těchto identifikátorů, jako například u formátu S28, kde se používají S2 a S8, nebo u formátu S37, kde jsou použity S3 a S7.

Dále je uváděn počet bajtů v záznamu, kromě samotného počtu a identifikátoru. Tato hodnota je zapsána v hexadecimálním formátu a může nabývat hodnot od „00“ do „FF“. Počet bajtů vždy přesahuje 3 bajty, protože 2 bajty tvoří 16bitové adresové pole a 1 bajt je vyhrazen pro kontrolní součet.

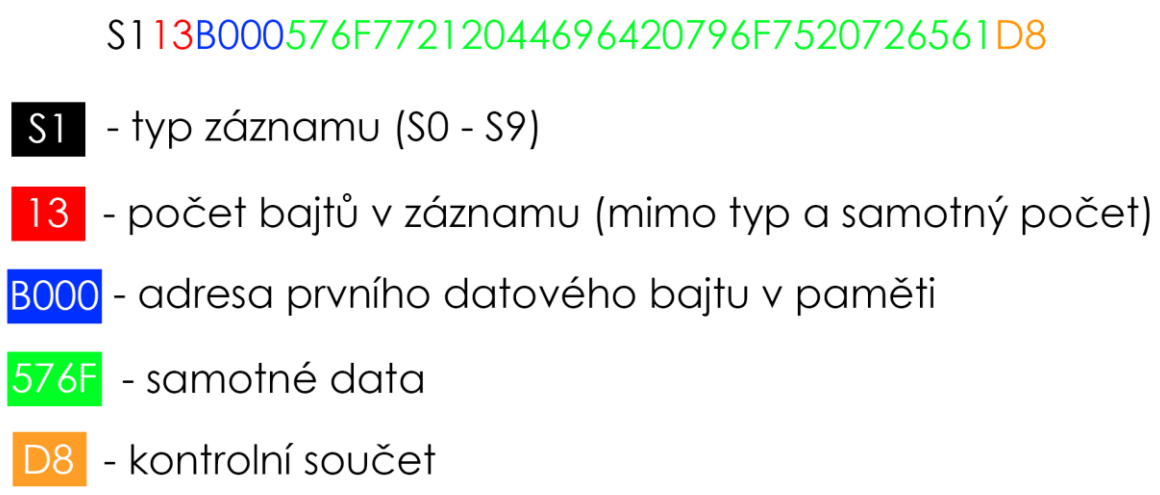
U souboru s příponou. s19 následuje za počtem bajtů adresa v paměti, kde je uložen první datový bajt. Pořadí adres v souboru není klíčové. Délka adresového pole závisí na typu

SREC; S1 záznamy mají délku adresy o 2 bajtech, S2 3 bajty a S3 ještě o bajt více.


Obrázek 1 Délka adres u S19 souboru [vlastní tvorba]

Ve čtvrté sekci záznamu se nacházejí samotná data, představující například spustitelný kód, paměťově čitelná data nebo popisné informace. Velikost dat může být i nulová. První datový bajt je uložen na adrese uvedené v předchozí sekci, a s postupným ukládáním dat se adresa inkrementuje na adresu pro následující datový bajt, dokud nejsou všechna data v paměti uložena. Obvykle mají záznamy délku 16 bajtů.

Konec řádku záznamu zahrnuje kontrolní součet, reprezentovaný dvěma hexadecimálními číslicemi. Tato hodnota je vypočítána jako nejméně významný bajt jedničkového doplňku součtu všech sekcí (počet bajtů + všechny adresní bajty + všechny datové bajty). [3] [4]




Obrázek 2. Struktura jednoho záznamu S19 souboru [vlastní tvorba]

## 1.2 Typy záznamů v S19 souboru

Pro ilustraci toho, jak vypadá soubor ve formátu S19 a jaké typy záznamů mohou být v něm obsaženy, jsem použil úryvek kódu z referenčního manuálu pro program SRecord, který byl původně napsán Peterem Millerem a Scottem Finneranem a je v současnosti udržován. [1] Formát S19 může obsahovat tři druhy záznamů, které budou níže popsány. Všechny tyto záznamy začínají prefixem „S“, následovaným počtem bajtů v záznamu, adresou v paměti a kontrolním součtem. Některé z nich můžou obsahovat i data.

První typ je označen jako S0 a funguje jako hlavička každého bloku záznamů. Může obsahovat popisné informace k následujícím blokům, například textový popisek „HDR“ v ASCII. Délka záznamu je 6 bajtů (06 v desítkové soustavě). Adresní pole je obvykle nastaveno na nulu, tedy na začátek paměti.

Druhým typem je S1, který obsahuje data spolu s 2bajtovou adresou, kde mají být data v paměti uložena. V tomto případě má záznam 10 bajtů mimo typ, samotný počet (podle druhého páru znaků) a data, která jsou zapsána hned za hlavičkou. Pro kontrolu: 10 v hexadecimální soustavě odpovídá 16 v desítkové soustavě, což je počet dvojic následujících za počtem bajtů (modrá část + zelená část + oranžová část). Po převodu dává datová část řetězec znaků „Hello, World“, což je podrobněji popsáno v následující kapitole 1.3.

Existují celkem tři typy datových záznamů: S1, S2 a S3. Typy S2 a S3 se liší od typu S1 v délce adresového pole. U typu S2 má adresové pole délku 3 bajty, což odpovídá 6 hexadecimálním znakům v adresovém poli, zatímco u typu S3 má pole adresy délku 4 bajty, tedy je dvakrát delší než u typu S1. S delší adresou souvisí také větší bitový počet a změněný kontrolní součet. Další rozdíl spočívá v ukončení těchto bloků: záznam typu S2 končí typem S8, zatímco záznam typu S3 očekává na konci typ S7. Oba ukončovací typy mají stejně velikou adresu jako datové typy v jejich bloku. [2]

Dalšími typy, které můžeme nalézt v souboru formátu S19, jsou záznamy S5 a S6. Tyto dva typy jsou volitelné a neobsahují žádná data. Jejich účel spočívá v identifikaci počtu řádků S1, S2 a S3 v celém bloku. Obsahují pouze prefix, počet párů znaků za počtem bajtů, kontrolní součet a hodnotu na adrese. Odlišují se od sebe velikostí adresy, kde typ S5 používá 2 bajty a typ S6 3 bajty.

Blok záznamů ve formátu S19 je uzavřen záznamem typu S9. Na rozdíl od úvodního záznamu S0 tento záznam neobsahuje žádnou datovou část a slouží pouze k obalení celého bloku záznamů. [1]

```
S0060000484452 1B
S1100000 48656C6C6F2C20576F726C640A9D
S5030001FB
S9030000FC
```

Obrázek 3. Příklad bloku záznamu S19 souboru [vlastní tvorba]

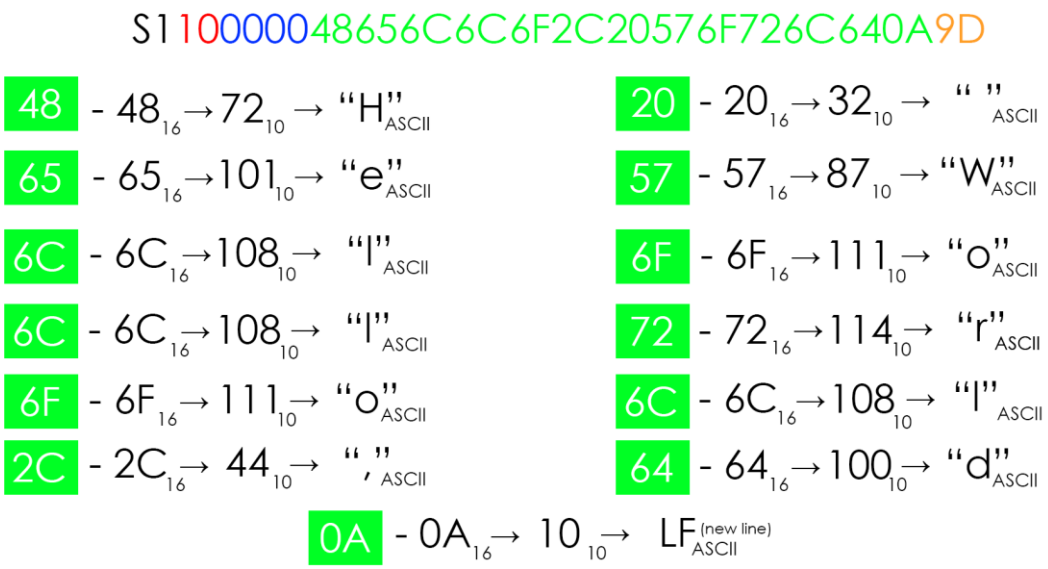
## 1.3 Překlad do čitelného textu

Využil jsem druhý řádek s prefixem S1 z předchozí kapitoly jako ukázku. Tento řádek je nejdelší a obsahuje data, která vytvářejí smysluplný text. Proto je vhodný pro překlad do srozumitelného textu.

```
S1100000 48656C6C6F2C20576F726C640A9D
```

Obrázek 4. Ukázkový datový záznam v S19 souboru [vlastní tvorba]

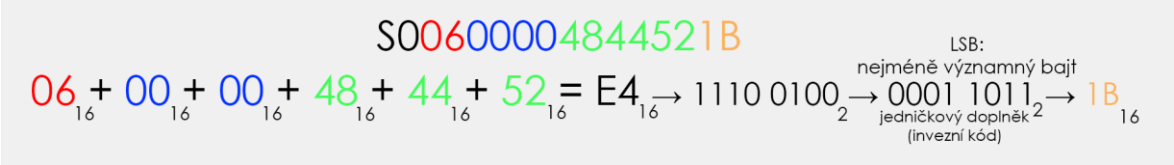
Podle prvního páru znaků jsme identifikovali typ záznamu, který obsahuje data a má adresový prostor o velikosti 2 bajty. Tento adresový prostor je uložen na adrese 0. Pro překlad do čitelného textu je klíčová až sekce od 9. znaku až po poslední dva znaky (označené zeleně na obrázku), které představují kontrolní součet. Celá datová sekce je rozdělena do párů znaků, které jsou postupně převedeny ze šestnáctkové soustavy na desítkovou. Poté se příslušná hodnota v desítkové soustavě hledá v ASCII tabulce a přiřazuje se k ní určitý znak. Tímto postupem vzniká čitelný text. V tomto případě se jedná o text „Hello, World“ a je zakončen netisknutelným znakem LF, který značí nový řádek.




Obrázek 5. Postup překladu datových bajtů na čitelný text v S19 [vlastní tvorba]

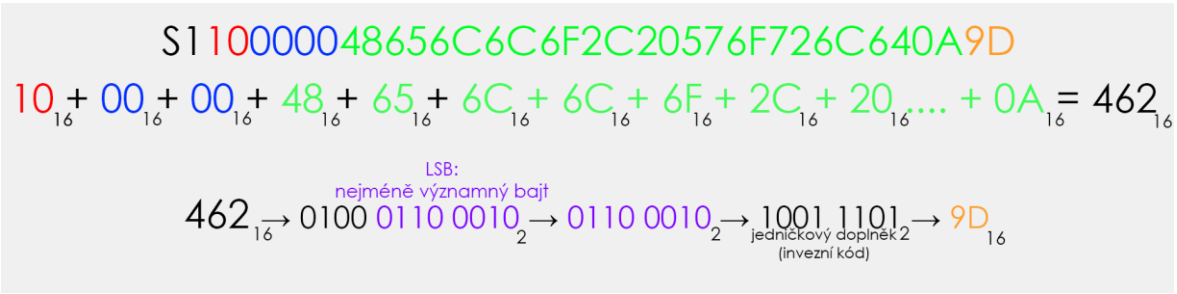
## 1.4 Výpočet kontrolního součtu

Kontrolní výpočet se provádí součtem všech párů (dvojic) znaků v šestnáctkové soustavě, s výjimkou první a poslední dvojice znaků. Výsledná suma se převede do binární podoby, která má délku 8 bitů (1 bajt). Následně se použije jedničkový doplněk, což znamená negaci všech bitů. Prakticky to znamená, že jedničky se změní na nuly a naopak. Výsledek inverzního kódu se opět převede z binárního na hexadecimální format. [2]




Obrázek 6. Výpočet kontrolního součtu hlavičky záznamu [vlastní tvorba]

V případě, že je součet všech bajtů větší než 8 bitů, což se obvykle týká záznamů typu S1, vezme se v úvahu pouze nejméně významný bajt (LSB), tedy posledních 8 číslic. S těmito číslicemi se pracuje stejným způsobem, jak bylo popsáno v předchozím postupu. Číslice se znegují a převedou zpět do hexadecimální podoby, což vytvoří pár znaků. Tento pár by měl být totožný s posledním párem v záznamu, který je na obrázku 5 a obrázku 6 označen oranžově.




Obrázek 7. Výpočet kontrolního součtu u S19 [vlastní tvorba]

# 2 SOUBOROVÝ FORMÁT HEX

Intel HEX je souborový formát, známý také jako Intel MCS-86 Object format, navržen pro mikroprocesory společnosti Intel s 8bitovou, 16bitovou a 32bitovou architekturou. [1] Často se používá při programování mikrokontrolérů nebo jako vstupní formát pro Programmable Read Only Memory (PROM) a hardwarové emulátory. Obvykle je generován kompilátorem nebo assemblerem, který převádí zdrojový kód z jazyka C nebo symbolických adres na strojový kód a následně jej ukládá do formátu HEX. [5]

Tento formát patří mezi nejrozšířenější souborové formáty a je podporován ve většině systémů a vývojových nástrojů. Původně byl Intel HEX navržen pro 16bitové adresy s omezením velikosti adresového prostoru na 64 kB. Postupem času byl vylepšen, umožňující práci s 20bitovými adresami a zvyšující limit velikosti adres na 1 MB. Později byla přidána podpora pro 32bitové adresové pole, což znamená, že maximální velikost adresového prostoru může být až 4 GB. Tato flexibilita z něj činí univerzální nástroj pro reprezentaci strojového kódu v různých kontextech v oblasti embedded systémů a mikrokontrolerů. [6]

## 2.1 Obsah souboru

Soubor formátu Intel HEX je strukturován do záznamů, a většina z nich obsahuje šest různých sekcí: délku záznamu, typ záznamu, startovací adresu, data a kontrolní součet. Typicky se vytváří kompilátorem nebo assemblerem při převodu zdrojového kódu, a výsledek je uložen do souboru ve formátu HEX. [7]

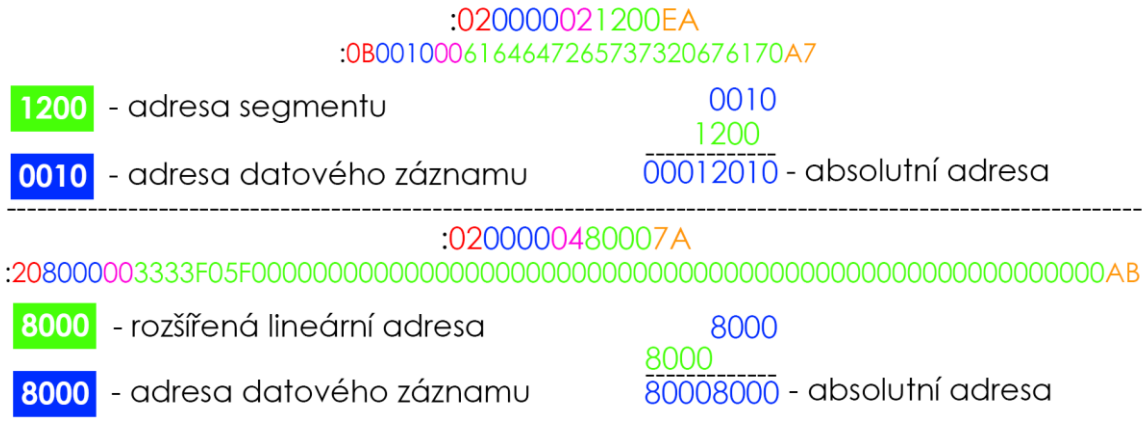
Identifikovat formát INHX16 lze na základě prvního znaku záznamu, který je zakódovaný jako ASCII dvojtečka (3A v hexadecimálním zápisu). [7]

Pole pro délku záznamu následuje po dvojtečce a specifikuje celkový počet datových bajtů. Obvykle mají záznamy délku 16 bajtů, ale může být libovolná hodnota od 0 do 255 (FF). Na rozdíl od Motoroly nezahrnuje délka adresového pole do počtu bajtů, pouze obsah datového pole. [7] [8]

Adresa, obvykle reprezentovaná dvojicí znaků (2 bajty), označuje místo, kde začínají data záznamu. Toto pole se obvykle využívá u datových záznamů, zatímco u jiných typů záznamů může být pouze zakódováno jako čtyři nuly. [7]

V hexadecimálním souboru se však mohou vyskytovat záznamy typu 02 a 04, které určují rozšířenou adresu segmentu pro 16 nebo 32bitové formáty, respektive lineární adresu pro

32bitové formáty. U těchto typů záznamů se ignoruje hodnota adresy v řetězci. Při čtení typu 02 je adresa segmentu obsažena v datovém poli na bitech 4-19 a aplikuje se pro následující datové záznamy, dokud není přečten další rozšířený záznam. Absolutní adresa se spočítá tak, že se k adrese datového záznamu přičte segment posunutý doleva o 16 bitů a doplní se zleva nulami v závislosti na formátu. Lineární adresa se také vyskytuje v datovém poli a jedná se o horních 16 bitech absolutní adresy, takže se absolutní adresa z tohoto záznamu skládá z rozšířené lineární adresy prvních 16 bitů a adrese z datového záznamu dolních 16 bitů. Pokud před těmito záznamy neexistuje žádný záznam typu 00, hodnota horních 16 bitů budou čtyři nuly. [9]

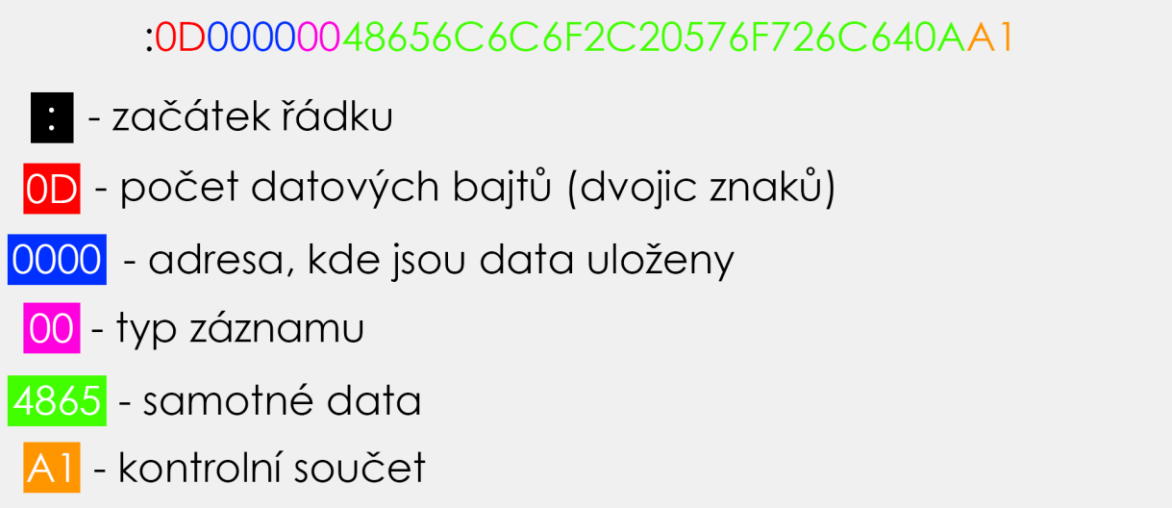



Obrázek 8 Výpočet absolutní adresy ze záznamů typu 02 a 04 [vlastní tvorba]

V případě S19 zjistíme typ záznamu podle prvních dvou znaků na řádku záznamu, kdy první písmeno označuje, že se jedná o formát SRecord od Motoroly a druhý znak určuje typ záznamu. U Intel HEX je typ záznamu za adresou, a pro 8bitový formát jsou podporovány především dva druhy záznamů a to typy „00“ a „01“.

Následují data, jejichž počet je určen počtem datových bajtů. Typ „00“ obsahuje zakódované informace vhodné k dalšímu zpracování. Toto pole může být prázdné, například u typu End of File (01), kde po informaci, že se jedná o konec souboru, následuje kontrolní součet. [7]

Poslední bajt řádku reprezentují kontrolní součet, který se vypočítá jako dvojkový doplněk součtu všech hexadecimální párů (kromě dvojtečky a posledních dvou znaků). Ověření záznamu proběhne porovnáním vypočítaného kontrolního součtu s posledními dvěma znaky. Liší-li se, záznam obsahuje chybu. [6] [8]




Obrázek 9. Struktura jednoho záznamu HEX souboru [vlastní tvorba]

## 2.2 Typy záznamů v HEX souboru

V standardním formátu HEX souborů pro 8bitové mikroprocesory se běžně setkáváme pouze s dvěma typy záznamů: datovými záznamy označenými „00“ a záznamem konce souboru označeným „01“. Avšak pro 16bitové a 32bitové mikroprocesory jsou k dispozici další čtyři typy záznamů, které rozšiřují možnosti formátu. Těmito typy jsou Extended Segment Address (16bit a 32bit), Start Segment Address (16 a 32bit), Extended Linear Address (pouze 32bit) a Start Linear Address Record (pouze 32bit). [6]

Extended Segment Address Records (neboli HEX86) slouží k přednastavení rozšířené adresy segmentu, což umožňuje pracovat se soubory až do velikosti 1 MB. Tento typ se nejčastěji objevuje na začátku souboru a má označení „02“. Záznam se skládá z identifikátoru začátku řádku, počtu datových bajtů (vždy 2 bajty), adresové pole, které je nulové, identifikátor záznamu, základní adresy horního segmentu (USBA) a kontrolního součtu. Adresa v tomto záznamu je nulová a není důležitá, podstatné jsou až hodnoty za identifikátorem záznamu "02" v datovém poli, kde je uložená adresa segmentu o dvou bajtech. Když se tento typ čte, adresa segmentu se uloží a aplikuje na následující datové záznamy, dokud není přepsán jiným záznamem s typem "02". Absolutní paměťová adresa se získá tak, že se sečte adresa datového záznamu (00) s USBA z datového pole tohoto záznamu a k výsledku se na začátek přidají tři nuly. Ve výchozím nastavení je adresa segmentu nulová. [9]

Typ „03“ je určen k určení počáteční adresy pro procesory Intel, například 8086. Tento záznam definuje počáteční adresu kódu, která se načte do registrů CS (Segment kódu programu) a IP (Instruction Pointer) procesoru. Počet datových bajtů je vždy 4 (04 v hexade-

cimálním zápisu), a pole adresy je nulové. Datový typ obsahuje celkem 4 bajty, přičemž první dva bajty jsou hodnota pro registr CS a druhé dva bajty pro registr IP. [6]

Extended Linear Address Records (neboli HEX386) jsou záznamy s 32bitovou adresou s typem „04“, obsahující horních 16 bitů (bity 16-31) adresy v části dat. Při čtení těchto záznamů se zaznamená rozšířená lineární adresa z datového pole a použije se na následující záznamy. Tato adresa zůstává v platnosti, dokud není změněna jiným rozšiřujícím záznamem. Absolutní paměťová adresa se získá přičtením adresového pole záznamu k posunuté adrese záznamu rozšířeného lineárního adresového záznamu. [9]

Typ „05“ je specifický pro 32bitové procesory od Intelu, jako například 80386. Startovací adresa kódu se načte do registru EIP (Extended Instruction Pointer). Počet bajtů je vždy 4, a adresa je nulová. V datovém poli se nachází 4bajtová lineární adresa, od které se začne vykonávat kód. Tento typ záznamu se může objevit kdekoliv v souboru, ale ve většině případů je ignorován, například pokud procesor není 32bitový, protože neobsahuje data potřebná k přehrátí paměti. [6]

```
:040000000102030 4F8
:02000002120 0EA
:0400000320003239 6E
:02000004FFFFFC
:0400000500000CD2A
:00000001FF
```

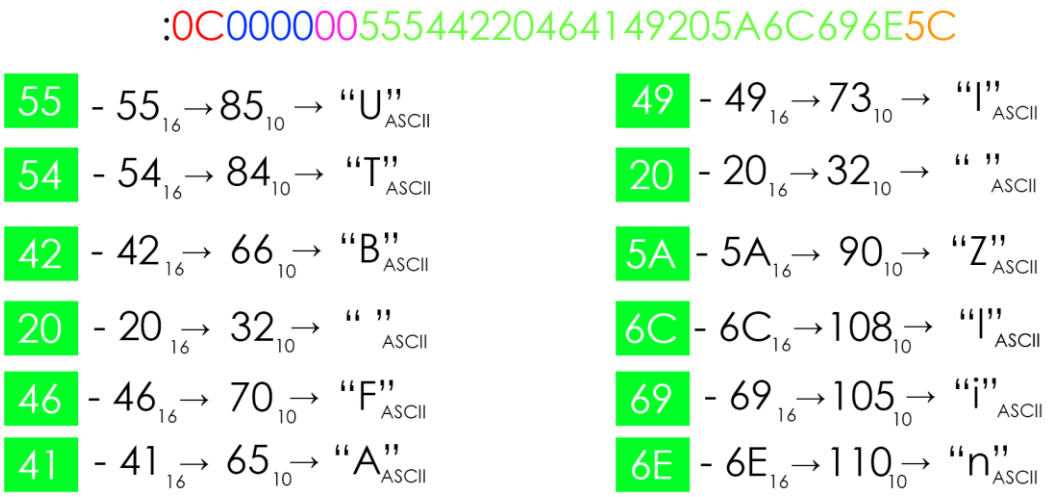
Obrázek 10. Všechny typy HEX formátu [vlastní tvorba]

## 2.3 Překlad do čitelného textu

Při převodu datového záznamu do textové podoby ve formátu Intel HEX platí stejné principy jako u formátu S19. K ilustraci jsem zvolil převod jiných dat, konkrétně textu „UTB FAI Zlin“. Diakritika byla vynechána kvůli omezením základní ASCII tabulky.

Datový typ je označen znaky „00“ a převod probíhá od 10. znaku záznamu až po konec, s výjimkou posledních dvou znaků, které reprezentují kontrolní součet. Záznam obsahuje 12 datových bajtů, kde první pár „0C“ je převeden do desítkové soustavy. Adresa je nastaveno na nule a datová část je dále rozdělena na 12 dvojic, které jsou postupně převedeny z hexa-

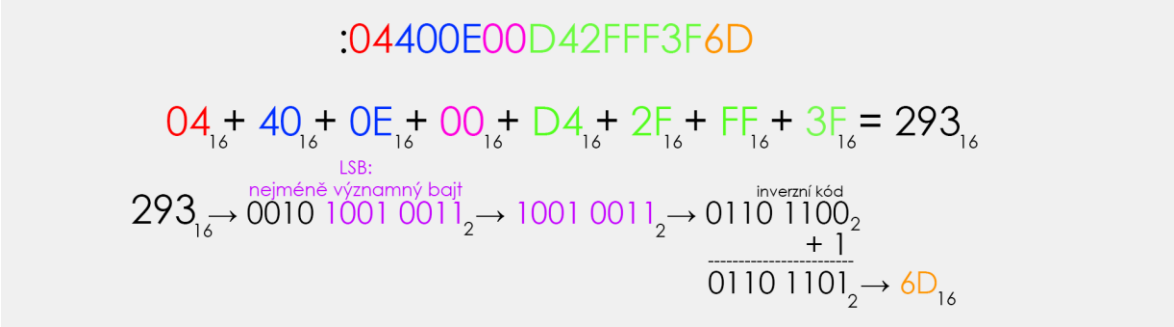
decimální do desítkové soustavy. Následně jsou tato čísla vyhledána v ASCII tabulce, kde každému číslu odpovídá určitý znak.




Obrázek 11. Postup překladu datových bajtů na čitelný text v HEX [vlastní tvorba]

## 2.4 Výpočet kontrolního součtu

Kontrolní součet v HEX souboru je vypočítán tak, že se sečtou všechny hexadecimální páry v záznamu s výjimkou posledních dvou znaků. Suma se následně převede na binární formát a na tu se aplikuje dvojkový doplněk. Postup dvojkového doplňku zahrnuje inverzi všech bitů (změna nul na jedničky a naopak) a pak se k výsledku přičte jednička. Výsledek se nakonec převede z binární podoby do hexadecimální. Pokud je součet párových znaků větší než 1 bajt ve dvojkové soustavě, použije se pro výpočet kontrolního součtu nejméně významný bajt (LSB). Postup zůstává stejný, tj. všechny bity jsou invertovány, k výsledku se přičte jednička, a výsledek se převede do hexadecimálního formátu. Vypočítaný výsledek a poslední dva znaky v záznamu by měly být identické; v opačném případě značí chybu v záznamu. [6] [7]




Obrázek 12. Výpočet kontrolního součtu u HEX [vlastní tvorba]

# 3 EXISTUJÍCÍ APLIKACE PRO POROVNÁNÍ SOUBORŮ

Existuje několik užitečných aplikací navržených speciálně pro porovnávání hexadecimálních souborů, které jsou významné pro programátory, vývojáře a odborníky v oblasti bezpečnosti. Jednou z těchto aplikací je Beyond Compare, který nabízí pokročilé možnosti porovnání souborů včetně hexadecimálního porovnání. Umožňuje uživatelům rychle identifikovat rozdíly mezi dvěma soubory a snadno provádět synchronizaci změn.

Pro práci s hexadecimálními soubory je také vhodný nástroj Hexinator. Tato aplikace je zaměřena na analýzu hexadecimálních dat a umožňuje uživatelům rychle procházet, editovat a analyzovat soubory na úrovni bajtů. S pokročilými funkcemi pro vyhledávání a filtry je Hexinator ideální pro práci s hexadecimálními kódy.

Pro uživatele hledající minimalistický a snadno použitelný hex editor existuje ImHex. Tento nástroj je navržen s důrazem na jednoduchost a přehlednost, což usnadňuje prozkoumávání hexadecimálních souborů. ImHex je vhodný pro ty, kteří potřebují rychle a snadno analyzovat hexadecimální data.

Celkově tyto aplikace – Beyond Compare, Hexinator a ImHex – poskytují rozsáhlé možnosti pro porovnání a analýzu hexadecimálních souborů, přičemž každá z nich má své specifické vlastnosti, které mohou vyhovovat různým potřebám uživatelů.

## 3.1 Beyond Compare

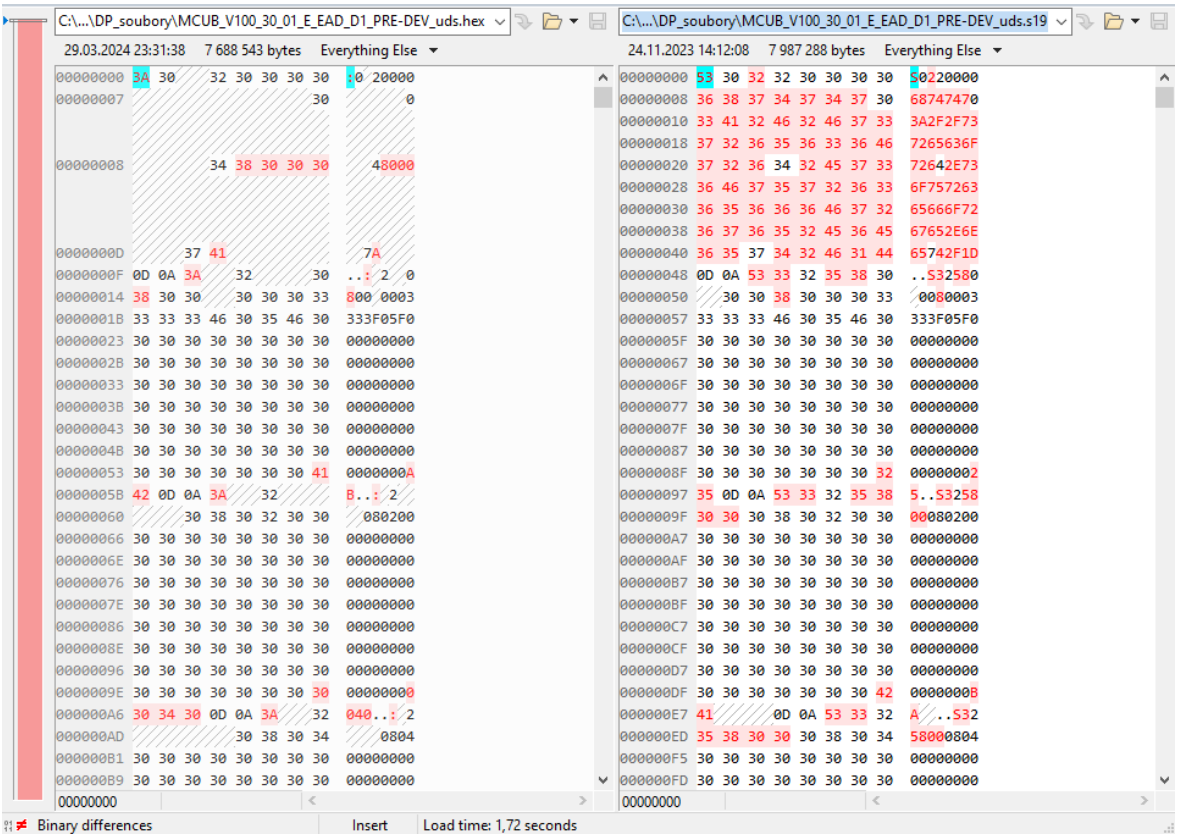
Beyond Compare je proprietární nástroj od společnosti Scooter Software, který poskytuje širokou škálu funkcí pro uživatele a vývojáře pro práci se soubory. Tento program je dostupný pro platformy Windows, macOS a Linux. Přestože se jedná o komerční produkt, nabízí různé typy licencí, včetně jednorázové licence pro jednoho uživatele, pro jedno pracoviště a pro celou společnost, což umožňuje využívání softwaru na celém světě po jednorázovém poplatku. Pro zájemce je také k dispozici bezplatná 30denní zkušební verze. Aktuální verze je 5, ale volně dostupná zkušební verze má číslo 4. [10]

Mezi klíčové výhody patří schopnost porovnávat nejen soubory, ale také celé složky. Tento nástroj rovněž podporuje třícestné slučování, což umožňuje pracovat s dvěma novými verzemi, které mají společného předka, a zobrazuje potenciální výsledky sloučení. Dále umožňuje přímou úpravu souborů, obsahuje masky a pravidla pro ignorování určitých pasáží během porovnání a umí synchronizovat složky. Beyond Compare se vyznačuje spolehlivostí při práci s velkými soubory, včetně porovnávání obrázkových souborů, registrů,

tabulek, hexadecimálních a MP3 souborů. Program nejen zobrazuje obsah lokálních složek, ale také dokáže pracovat s archivovacími formáty, jako je ZIP, jako s běžnými složkami. Kromě toho umožňuje porovnávat adresáře přes FTP nebo SFTP a pracovat s cloudovými platformami, jako jsou Dropbox nebo Amazon S3. Za zmínku stojí i funkce konverze mezi formáty a předzpracování, kdy lze spustit skript před zobrazením souboru nebo extrahovat text z dokumentů Microsoft Office. [11]

Hlavní nevýhodou tohoto nástroje je, že se jedná o placený a proprietární software, i když licence se platí pouze jednou a pokrývá všechny hlavní operační systémy. Verze 4 nemá možnost využívat tmavý režim (tato funkce byla přidána až od verze 5), chybí podpora pro dotykové obrazovky a ve Windows 11 chybí podpora kontextového menu. Starší verze rovněž nejsou kompatibilní s procesory od Apple (M1/M2). [12]

Beyond Compare má možnost porovnávat soubory s přístupem k datům odkudkoliv. Program automaticky volí nejvhodnější způsob porovnání a zobrazuje identické pasáže ze dvou souborů tak, že umožňuje uživateli provádět editace. Tato funkce je užitečná jak pro obyčejné soubory, například zdrojové kódy, tak i pro binární data nebo hexadecimální soubory.

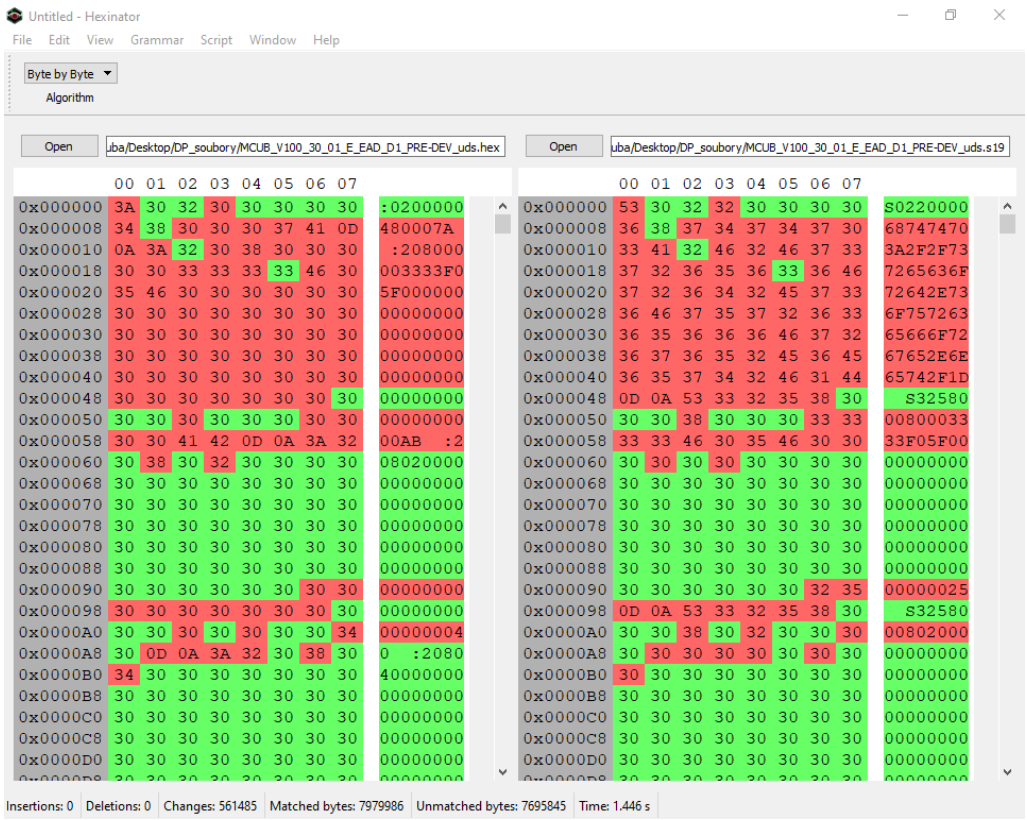



Obrázek 13 Porovnání stejného souboru v S19 a HEX v Beyond Compare

Tento nástroj umožňuje kombinovat změny z dvou verzí souboru nebo složky do jednoho výstupu, což usnadňuje řešení konfliktů mezi verzemi souborů. Lze ho také integrovat s verzovacími nástroji, jako jsou Git nebo Team Foundation Server z Visual Studia, a efektivněji tak porovnávat a kombinovat soubory. Díky uživatelskému rozhraní pro synchronizaci složek může uživatel snadno sladit rozdíly v datech a kopírovat je na disky, FTP servery nebo do zip souborů. Beyond Compare umožňuje filtraci obsahu, aby se minimalizovaly nežádoucí změny. Navíc podporuje automatizaci repetitivních úloh pomocí skriptovacího jazyka, a tyto skripty lze volat přímo z příkazového řádku, což umožňuje naplánovat synchronizaci na vhodnou dobu. [10]

## 3.2 Hexinator

Hexinator je proprietární software, který umožňuje manipulaci s binárními a hexadecimálními soubory na platformě Windows a Linux. Aplikaci lze stáhnout zdarma, přičemž tato bezplatná verze obsahuje některá omezení ve srovnání s plně placenou verzí. Verze zdarma umožňuje editaci souborů bez omezení velikosti, inkrementální hledání řetězců a využívání masek, zobrazování všech řetězců v binárním souboru o určité délce a jejich kódování a dekódování běžných datových typů.




Obrázek 14 Porovnání stejného souboru v S19 a HEX v Hexinatoru

Placená verze Hexinatoru nabízí rozšířené funkce, jako je dekódování souborů pomocí slovníků, tvorba vlastních slovníků, automatizace úloh a rozšíření slovníků pomocí skriptů v Lua a Pythonu. Dále umožňuje zobrazit histogram souboru, vypočítat kontrolní součty pro vybrané bajty a porovnávat kódování znaků. [13] [14]

Na webové stránce aplikace je prezentována možnost stáhnout verzi zdarma, avšak při otevření aplikace je upřesněno, že se jedná o zkušební verzi s 7denní platností. Aplikace obsahuje dvě okna pro výběr souborů k porovnání a nabízí možnost porovnání souborů buď bajt po bajtu nebo pomocí pokročilých porovnávacích metod. Ačkoliv porovnání souborů by mělo být dostupné pouze v plně placené verzi, aplikace umožňuje zobrazit výsledky porovnání. Nicméně, manipulace s obsahem souboru není v bezplatné verzi povolena. Editovat soubor lze jedině tak, že se při startu aplikace vybere možnost „New File“ a zvolí příslušný soubor, kde je už možné bajty v souboru editovat. Základní verze této aplikace umožňuje vytvářet nové soubory a volit jejich kódování a hodnoty na jednotlivých bajtech.

## 3.3 ImHex

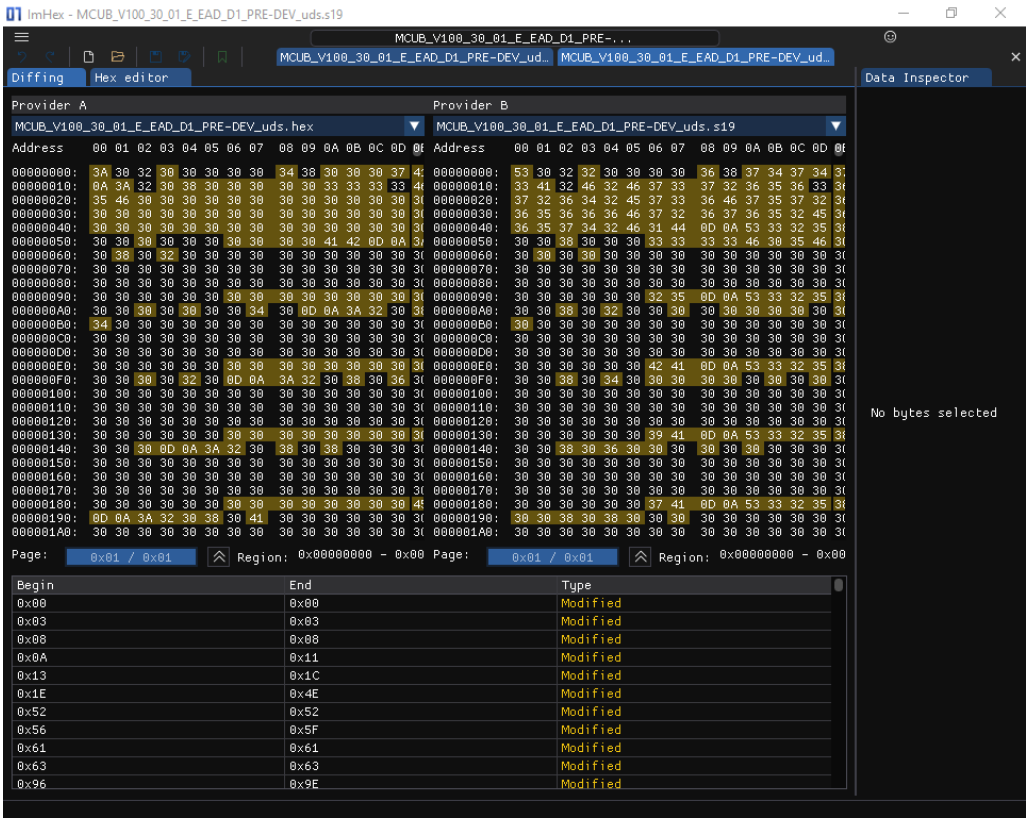
ImHex je bezplatný editor hexadecimálních souborů s otevřeným zdrojovým kódem pod licencí GPLv2, dostupný na platformách Windows, macOS a Linux. Tato aplikace je součástí operačního systému Kali Linux, vytvořeného pro penetrační testování a etické hacking. Má rovněž webovou aplikaci, ale nemá možnost přímého porovnávání souborů. Uživatelé na platformě Windows mohou stáhnout portable verzi aplikace, a v budoucnu je plánováno zveřejnění ImHex na herním obchodě Steam.

ImHex umožňuje zobrazení a úpravu vybraných hexadecimálních souborů a poskytuje funkce jako byte patching, patch management, kopírování bajtů ve formě pole pro různé programovací jazyky a vyhledávání pomocí řetězců a hexadecimálních dat. Pokud jsou otevřeny více souborů, lze snadno přepínat mezi nimi pomocí záložek. Aplikace zvládá pracovat s velkými soubory, nastavit variabilní počet sloupců a volit vlastní kódování.

Vytvořena byla také speciální syntaxe s názvem „pattern language,“ založená na C++, která umožňuje nastavování zvýrazňování, dekódování a analýzu různých souborových formátů. To je užitečné pro reverzní inženýry, kteří potřebují analyzovat různé instrukční sady procesorů, od x86 přes ARM Cortex-M Thumb až po PowerPC.

ImHex poskytuje statistickou analýzu obsahu vybraného souboru, včetně velikosti, metadat, bajtové distribuce a entropie. Kromě toho nabízí možnost porovnávání souborů, kde se zvýrazňují odlišné části. [15] [16]

Nástroj „Tools“ v ImHex obsahuje mnoho užitečných funkcí, včetně ASCII tabulky, kalkulačky, možnosti práce s barvami a převodu do HEX nebo RGBA, tvorby regex pravidel, grafického zobrazení funkcí a různých nástrojů pro manipulaci se soubory.




Obrázek 15. Porovnání stejného souboru v S19 a HEX v ImHex

## 3.4 Srovnání aplikací

Když se všechny tři nástroje porovnají, Beyond Compare sice nabízí nejširší paletu funkcí pro tři nejdůležitější operační systémy (Windows, Linux a macOS), kdy si poradí s porovnáním i složek, třícestným slučováním, synchronizováním složek, konverzi mezi formáty nebo podporu nejen pro hexadecimální soubory, ale taky audio soubory nebo registry. Pro běžného uživatele je ale úskalím fakt, že se jedná o placený software a zkušební verze je jen na měsíc. Verze 4 již není podporována pro uživatele počítačů Apple s procesory M1 a M2, takže jediná možnost pro ně je zakoupit si verzi 5.

Kompromisem se může stát nástroj Hexinator, který je sice také placený, ale verze zdarma není zkušební a chybí ji jen některé funkce, jako jsou slovníky, automatizace úloh nebo skriptování v jazycích Lua nebo Pythonu. Ač je možné porovnávat dva hexadecimální soubory, editovat ho přímo v programu je pouze v placené verzi, i když se přímo na stránkách programu píše, že je to možné v obou verzích.

Kompletně zadarmo a s otevřeným kódem je program ImHex, jehož uživatelské rozhraní je postaveno do záložek, a kromě základních možností manipulace a komparace hexadecimálních souborů, také pracovat s vlastním jazykem, využívat statistikou analýzu obsahu souborů, případně další užitečné funkce, které ani jeden z předchozích programů nenabízí a můžou se uživateli hodit.

| | **Beyond Compare 4** | **Hexinator** | **ImHex** |
|---|---|---|---|
| **Operační systémy** | Windows, Linux a macOs | Windows a Linux | Windows, Linux, macOS |
| **Licence** | Proprietární/Shareware | Proprietární s verzí zdarma | Freeware/open-source |
| **Porovnání souborů** | Textové, tabulkové, hexadecimální, audio, obrázkové | Hexadecimální | Hexadecimální (pouze v placené verzi) |
| **Pravidla pro porovnání** | Ano | Ne | Ne |
| **Editace** | Ano (mimo porovnání) | Ano (pouze v placené verzi) | Ano |
| **Zobrazení obsahu v ASCII** | Ano | Ano | Ano |
| **Hledání v obsahu** | Ano | Ano | Ano |
| **Zvýraznění stejných dat** | Ano | Ne | Ne |
| **Konverze formátu** | Ano | Ne | Ne |
| **Statistická analýza** | Ne | Ne | Ano |

| **obsahu** | | | |
|---|---|---|---|
| **Informace o soubo-ru** | Ano | Ne | Ano |

Tabulka 1 Srovnání existujících aplikací pro porovnání souborů

# 4 POPIS POUŽITÝCH TECHNOLOGIÍ

V této kapitole se detailněji zabývám popisem použitých technologií v rámci implementace aplikace. Zmíním se o klíčových technologiích, které tvoří základní infrastrukturu projektu, a jejich využití při vývoji softwarového řešení. Konkrétně budu popisovat programovací jazyk C# a platformu .NET, která zahrnuje širokou škálu knihoven a nástrojů pro vývoj aplikací. Dále se zaměřím na technologie Windows Presentation Foundation (WPF) a Extensible Application Markup Language (XAML), které byly použity pro implementaci uživatelského rozhraní aplikace.

## 4.1 C#

Programovací jazyk C# je objektově orientovaným a typově bezpečným programovacím jazykem vyvinutým společností Microsoft. Tento jazyk je součástí rodiny jazyka C a svou syntaxí vychází z jazyků C, C++ a Java. [17] C# je podporován na různých platformách, zejména na desktopových operačních systémech jako jsou Windows 7-11, Linux, macOS, Android a iOS. Umí vytvářet desktopové, webové a mobilní aplikace, včetně programů běžících v příkazové řádce. [18] Hlavním vývojovým prostředím je Visual Studio, taktéž od společnosti Microsoft.

Tento jazyk nabízí řadu funkcí, které usnadňují vývoj robustních aplikací a řeší záležitosti jako správa paměti pomocí Garbage Collection, která automaticky uvolňuje nepoužívanou paměť. Dále poskytuje zpracování výjimek pro detekci a opravu chyb v programu. Díky typové bezpečnosti není možné číst z neinicializovaných proměnných, přistupovat k polím mimo jejich rozsah nebo provádět neplatné převody mezi typy. [17] Kompilátor upozorní na chybu a program nelze zkompilovat, dokud není chyba opravena.

C# má jednotný typový systém, kde všechny typy, včetně primitivních jako int nebo double, dědí od kořenového typu object. Všechny tyto typy sdílejí stejné operace a hodnoty lze ukládat a manipulovat s nimi. Jazyk také umožňuje použití uživatelsky definovaných referenčních typů jako jsou třída, rozhraní, delegát a záznam, stejně jako vestavěných referenčních typů dynamic, object a string, spolu s hodnotovými typy. [17] [19]

I přesto, že je C# primárně objektově orientovaný jazyk, využívá některé vlastnosti funkcionálního programování, zejména možnost předávat funkce jako hodnoty pomocí delegátů nebo jako návratové typy. Dále umožňuje využít lambda výrazy pro použití anonymních funkcí během běhu programu. [18]

## 4.2 ASP .NET

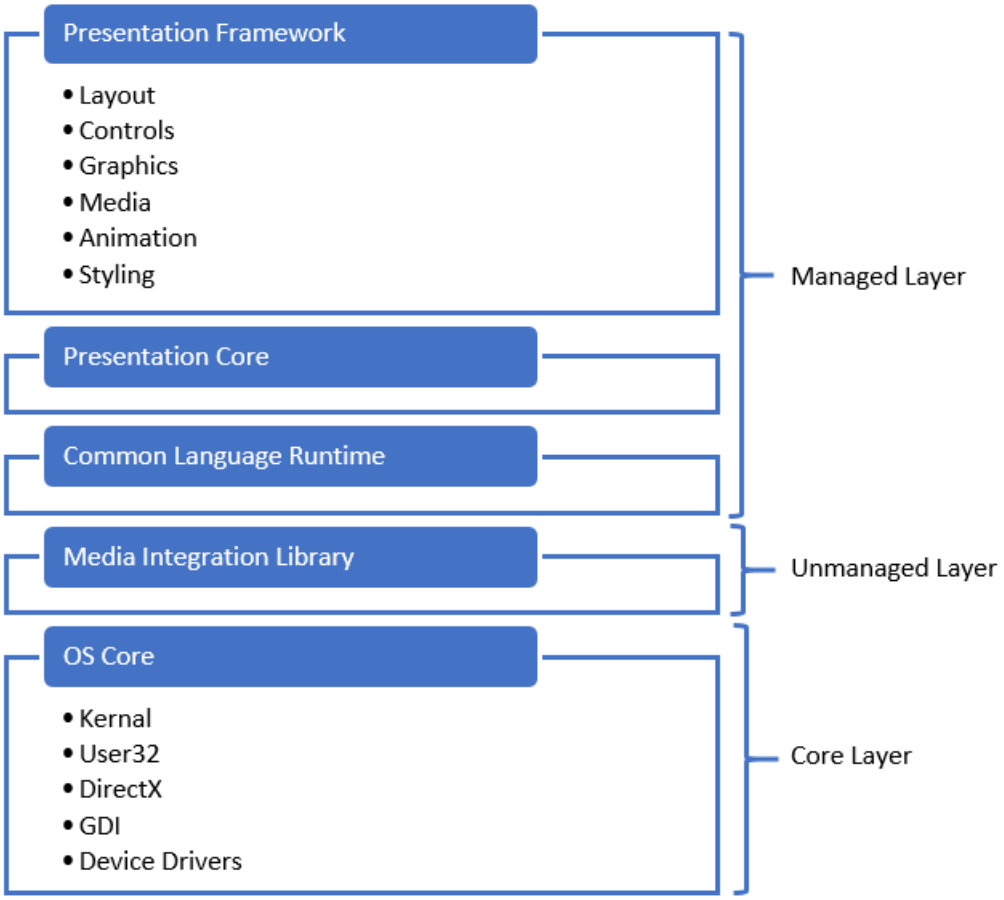
V roce 2002 Microsoft uvedl platformu .NET (Dotnet) s otevřeným kódem, která se dodnes těší velké oblibě mezi vývojáři softwaru. [20] Cílem bylo vytvořit univerzální prostředí pro programování v libovolném jazyce. Od samého počátku byl důraz kladen především na operační systém. Tato platforma podporuje širokou škálu programovacích jazyků, avšak hlavními jsou C#, F# a VB .NET, což je nová verze jazyka Visual Basic postavená na této platformě. Platforma podporuje Common Language Infrastructure (CLI), což je platformně nezávislý systém umožňující přeložit zdrojový kód do Common Intermediate Language (CIL), což je jazyk nejnižší úrovně bez závislosti na konkrétním programovacím jazyce. Kromě jazyků přímo podporovaných společností Microsoft lze kompilovat do CIL i další programovací jazyky. Mezi nejznámějšími zástupci CLI jsou rozšíření jazyka C++ jako C++/CLI, IronPython, PowerBuilder nebo Eiffel. [21]

V rámci tohoto frameworku je možné vytvářet webové, mobilní, desktopové a IoT aplikace, stejně jako programy pro příkazovou řádku. Tyto aplikace lze vytvářet v různých aplikačních frameworkách, z nichž každý má svůj specifický účel, a dohromady tvoří tyto varianty. NETu jeden ekosystém. [22] Původně pro vývoj softwaru pro operační systém Windows sloužil .NET Framework, který je dodnes podporován a aktualizován. Avšak po verzi 4.8 byl nahrazen .NET Core, který zahrnuje další frameworky jako ASP.NET pro webové aplikace, Windows Forms a Windows Presentation Foundation pro desktopové aplikace [23], Universal Windows Platform (UWP) pro tvorbu univerzálních aplikací pro zařízení s Windows 10/11, Windows 10 Mobile, HoloLens a Xbox One bez nutnosti přepisování zdrojového kódu. [24] Existuje také Xamarin pro mobilní platformy (iOS, Android a Windows) [25] a psaní skriptů do videoher v prostředí Visual Studia a jazyka C# v enginu Unity. [26] S příchodem .NET 6 Microsoft představil nástupce Xamarinu nazvaný MAUI, který umožňuje vývoj multiplatformních aplikací (iOS, Android, macOS a Windows) s použitím jednoho zdrojového kódu. [27]

## 4.3 WPF a XAML

Windows Presentation Foundation (WPF) je framework z ekosystému .NET pro tvorbu desktopových aplikací na platformu Windows. Prvně byl uveden v .NET Frameworku verzi 3.0. [28] Svým příchodem se lišil tím, že se choval jinak než v té době dostupné frameworky pro tvorbu uživatelských rozhraní jako Microsoft Foundation Class (MFC) a Windows Forms, které fungovaly jen obalovaly DLL knihovny User32 (tradiční vzhled systé-

mu Windows) a GDI/GDI+, jenž měl na starost vykreslování tvarů, textu a obrázků. WPF jen minimálně využívalo User32. Dříve se v GUI frameworcích nerozlišovalo mezitím, jak se aplikace chová a jak vypadá, a tak byl vzhled i logika napsány ve stejném programovacím jazyce, což bylo pro programátora náročnější. V WPF jsou UI prvky navrhovány jazykem XAML a jejich chování řešeno v procedurálních jazycích jako je C# nebo VB.NET, a tak je chování rozděleno od vzhledu. K tomu začalo využívat technologii DirectX jako svůj grafický engine pro vykreslování a aplikace tak mohly využívat hardwarovou akceleraci. [29]




Obrázek 16 Diagram architektury WPF [30]

WPF byl vytvořen, aby nahradil Windows Forms a jejich hlavní rozdíl je v tom, jak zpracovávají grafiku. WPF používá vektorový grafický systém, který dovoluje vyšší kvalitu, škálovatelnou grafiku, kterou lze animovat a dále s ní manipulovat. Naopak WinForms používají rastrový grafický systém, kterého limituje nižší kvalita a omezené možnosti animování. Dále se liší v tom, že WPF nabízí data binding, což má na starost, že se hodnota UI prvku automaticky zaktualizuje, pokud se změní data, dále styly a šablony, kdy je možné například nastavit, jak má nějaký prvek vypadat v případě, že se zmáčkne tlačítko nebo

nastavení šablony pro celý řádek tabulky. Dále nabízí WPF podporu multimédií jako je audio, video nebo animace, za to Windows Forms k tomu potřebuje další knihovny nebo pluginy třetích stran. [31]

Tvorba uživatelského rozhraní programu ve WPF probíhá v programovacím jazyce Extensible Application Markup Language (XAML) a poprvé se objevila jako součást Microsoft .NET Frameworku 3.5.[29] Kromě WPF se využívá také v UWP a Xamarinu. Jedná se o deskriptivní programovací jazyk vytvořený společností Microsoft, jehož účelem je tvorba uživatelského rozhraní pomocí značkovacího jazyka, který je podobný formátu XML pro serializaci dat. Vývojové prostředí frameworku WPF, Visual Studio, nabízí Designer, ve kterém lze vybrat ovládací prvek a vizuálně navrhnout vzhled rozhraní, nebo je možné psát kód. Typické uživatelské rozhraní pro Windows se skládá z okna (Window) nebo stránky (Page). XAML používá formát XML pro elementy a atributy, přičemž každý element reprezentuje viditelný objekt, který je instancí typu, definovaného oborem názvů (namespace), který fyzicky existuje v sestavě (DLL) knihovny .NET. Každý element v XAML začíná levou ostrou závorkou (<) a končí pravou ostrou závorkou (>). Je nutné, aby každá značka byla uzavřená. Jednotlivé objekty, jako například textové pole (TextBlock) nebo tlačítko (Button), mají nastavitelné atributy a lze jim nastavit obsah, velikost, barvu, velikost písma, font a další vlastnosti. Stejně jako ve Windows Forms, ovládací prvky mohou mít události v podobě kliknutí, vybrání textu, zmáčknutí tlačítka na klávesnici, kliknutí myší a další. Tyto události mají event handler, který se zavolá a spustí kód v pozadí.

Při vytvoření nového souboru XAML se rovněž vytvoří soubor stejného jména (s koncovkou .cs), ve kterém se provádí veškerá aplikační logika dané obrazovky, včetně nastavení obrazovky při načtení, zachycení uživatelských událostí jako je stisknutí tlačítka nebo nastavení vzhledu a chování jednotlivých vizuálních objektů. Aby bylo možné přistupovat v kódu k těmto objektům, je nutné jim nastavit unikátní identifikátor v podobě jména pomocí přidání atributu "x: Name". Takovému kódu, který se stará o aplikační logiku obrazovky, se říká Code-Behind. [32]

# II. PRAKTICKÁ ČÁST

# 5 NÁVRH

Praktická část této diplomové práce se zaměřuje na návrh a implementaci desktopové aplikace pro práci s hexadecimálními soubory. Účelem aplikace je korektní zobrazení obsahu tohoto druhu souborů v souladu s jejich formátem, umožnit editaci dat a jejich následné uložení, provést porovnání obsahu dvou souborů a vygenerovat report identifikující odlišné či stejné bajty. Během rešerše existujících hexadecimálních editorů jsem zjistil, že většina z nich se soustředí především na obecné zobrazení obsahu souborů bez ohledu na jejich konkrétní formát. Aplikace má záměr pomoct uživatelům, kteří potřebují analyzovat a modifikovat soubory ve formátu Intel HEX a Motorola SRecord, jenž se používají na programování mikroprocesorů. Vedle implementace užitečných funkcí pro usnadnění práce uživatelů bude klíčovým hlediskem vývoje také důraz na intuitivní a přehledné grafické uživatelské rozhraní, optimalizaci výkonu aplikace pro zpracování souborů libovolné velikosti a různých typů záznamů.

## 5.1 Funkční a nefunkční požadavky

Pro desktopovou aplikaci je důležité definovat funkční a nefunkční požadavky, které určují, jak má aplikace fungovat a jaké vlastnosti by měla mít. Funkční požadavky popisují požadované funkcionality a chování aplikace, zatímco nefunkční požadavky se zaměřují na vlastnosti, jako je výkon, stabilitu a uživatelskou přívětivost.

### 5.1.1 Funkční požadavky

Funkční požadavky jsou rozděleny do částí podle jejich funkcionality. Zmíněné jsou ty požadavky, které jsou pro danou stránku specifické, některé však mohou být sdíleny napříč různými stránkami.

**Požadavky pro celou aplikaci**:

- Aplikace by měla mít možnost vybrat soubor, který bude zpracován.
- Aplikace by měla si ukládat posledních 5 otevřených souborů a zobrazovat je na stránce.
- Lze dynamicky měnit jazyk a barevný režim aplikace a tyto volby ukládat.
- Aplikace by měla v menu mít možnost pro ukončení.

- Aplikace musí umět uložit na vybrané místo uživatelem nový soubor na základě již otevřeného souboru.
- Aplikace by měla zobrazovat metadata, obsažené bloky daty a adresní rozsah otevřeného souboru.

**Požadavky pro zobrazení obsahu**:

- Stránka musí obsahovat název aktuálně otevřeného souboru spolu s jeho formátem.
- Tabulka pro zobrazení obsahu má obsahovat celkem 18 sloupců, po jednom pro adresu s ASCII a 16 sloupců pro data.
- Hlavička tabulky by měla být vizuálně odlišena od ostatních řádků.
- Text ve sloupci ASCII má být zarovnaný pod sebou.
- Při modifikování obsahu souboru by se měla kontrolovat validita vložených hexadecimálních číslic.
- Adresa záznamu by měla být ve formátu "0x00000000".
- Na těchto stránkách by mělo být implementováno hledání ve všech sloupcích včetně přechodu na konkrétní adrese.
- V případě většího počtu nálezů by měla aplikace umožňovat přechod na další nalezený prvek.

**Požadavky pro porovnávání**:

- Stránka musí obsahovat název a jeho formát pro oba vybrané soubory.
- Uživatel si může volit mezi zobrazením stejných nebo odlišných bajtů.
- Stránka by měla obsahovat tlačítko pro navrácení všech prvků stránky do původního stavu.
- Stránka by měla obsahovat podrobnosti datových bloků pro oba vybrané soubory.
- Uživatel by měl mít možnost synchronizovat si obě tabulky a pohybovat se tak v nich najednou.
- Shody nebo rozdíly by měly být barevně označeny.
- Data vyskytující se pouze v jednom souboru by měla být v tabulce odlišně barevně označeny než v druhé tabulce.

**Požadavky pro generování reportu**:

- Uživatel by měl mít možnost si vybrat mezi generováním souhrnu, pouze shodných nebo odlišných bajtů do formátu PDF.
- Reporty by měly být ukládány vygenerované složce s časovým razítkem.
- Report by měl obsahovat nadpis, čas generování a cesty k jednotlivým souborům.
- Vygenerování reportů pro data vyskytující se pouze v jednom souboru.

### 5.1.2 Nefunkční požadavky

- Aplikace by měla být spustitelná na operačním systému Windows bez předchozí instalace programu.
- Aplikace musí podporovat otevírání souborů s příponou .s19 a .hex libovolné velikosti.
- Aplikace by měla umět zobrazit korektně obsah hexadecimálních souborů s různými typy záznamů a jejich zápisem.
- Doba načítání tabulek by neměla být příliš dlouhá, jinak by to mohlo negativně ovlivnit uživatelský zážitek.
- Aplikace by měla být responzivní vůči různým velikostem monitorů.
- Přidání nových prvků do aplikace by nemělo výrazně ovlivnit zdrojový kód.
- I nový uživatel by se měl v aplikaci snadno zorientovat.
- Požadavek by měl zaručit, že systém nebude selhávat.
- Pro správné fungování celého systému je nutná existence souborů pro nastavení aplikace ve složce Dokumenty.

## 5.2 Uživatelské rozhraní

Při návrhu uživatelského rozhraní aplikace jsem se zaměřil na vytvoření intuitivního a uživatelsky přívětivého prostředí, které umožní uživatelům snadno a efektivně pracovat s hexadecimálními soubory. Cílem bylo vytvořit responzivní moderní vzhled aplikace, který bude splňovat standardy prostředí, jako je podpora tmavého režimu a možnost přepínání mezi tmavým a světlým režimem. Návrhy jednotlivých stránek jsem si nejprve vytvořil v grafickém editoru a podle nich vytvořil aktuální vizuál aplikace.

První částí návrhu bylo rozložení hlavního okna, které slouží jako základ celé aplikace a obal pro jednotlivé stránky. Rozhodl jsem se využít framework WPF UI a jeho kostry pro celkový vzhled, přičemž texty grafických prvků by měly automaticky reagovat na vybraný jazyk.

Další důležitou součástí návrhu je horní a boční menu. Levé menu poskytuje přístup ke všem hlavním funkcím aplikace, jako je zobrazení obsahu souborů, informace o nich, porovnávání a nastavení. Horní menu obsahuje možnosti pro otevření a uložení souboru.

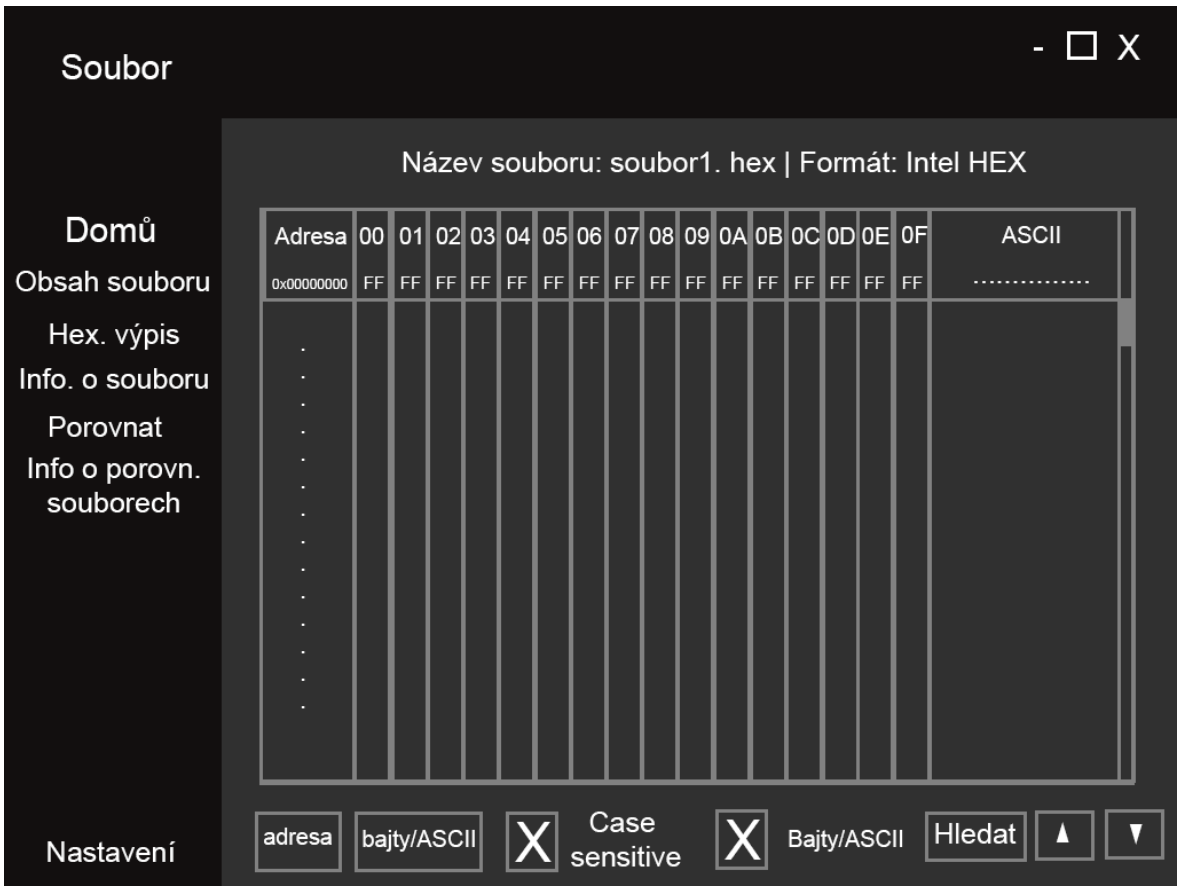
Úvodní obrazovka aplikace zobrazuje název „Hexmurai“, pod ním text „Poslední otevřené soubory“ a seznam posledních pěti otevřených souborů.


Obrázek 17 Návrh úvodní obrazovky aplikace

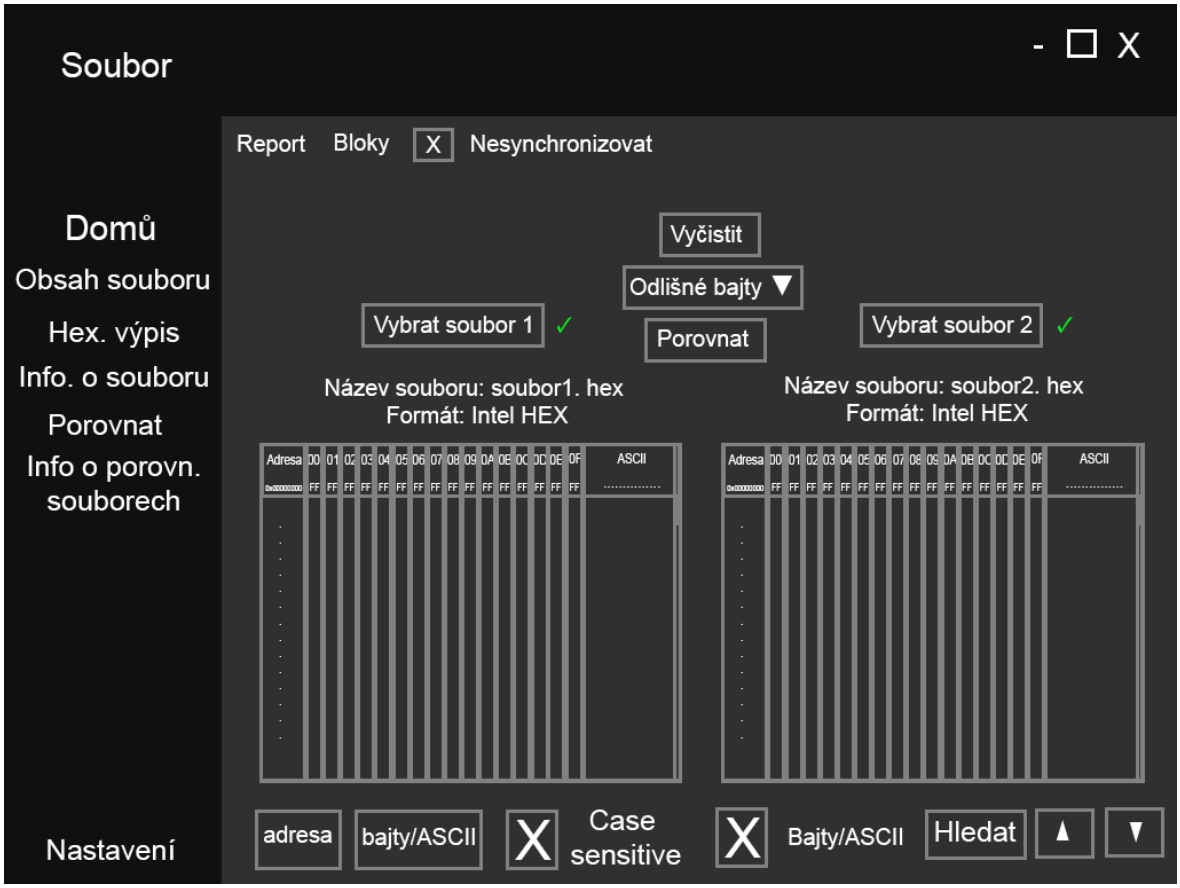
Stránky pro zobrazení obsahu a porovnání souborů obsahují název a formát otevřeného souboru a data jsou vizualizována pomocí tabulek s 18 sloupci. Spodní část doplňuje lišta s textovými poli a tlačítky pro hledání v obsahu. Stránka pro porovnání zobrazuje tlačítka pro výběr dvou souborů, spuštění porovnání a volbu režimu komparace. Horní menu nabízí

možnosti pro generování reportu, zobrazení datových bloků souborů a přepínač pro synchronizaci pohybu tabulek.

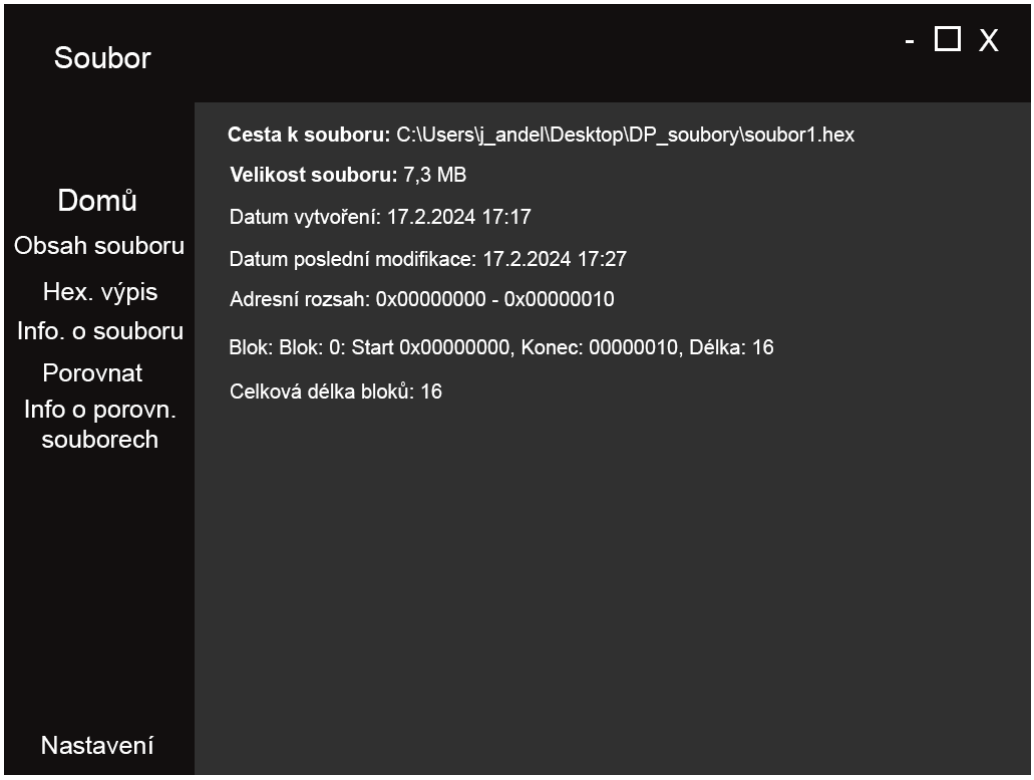
Stránka pro zobrazení informací o souborech je rozdělena na dvě části, kde je vlevo zobrazen název informace tučně a vedle ní je uvedena její hodnota.




Obrázek 18 Návrh obrazovky pro zobrazení obsahu souboru



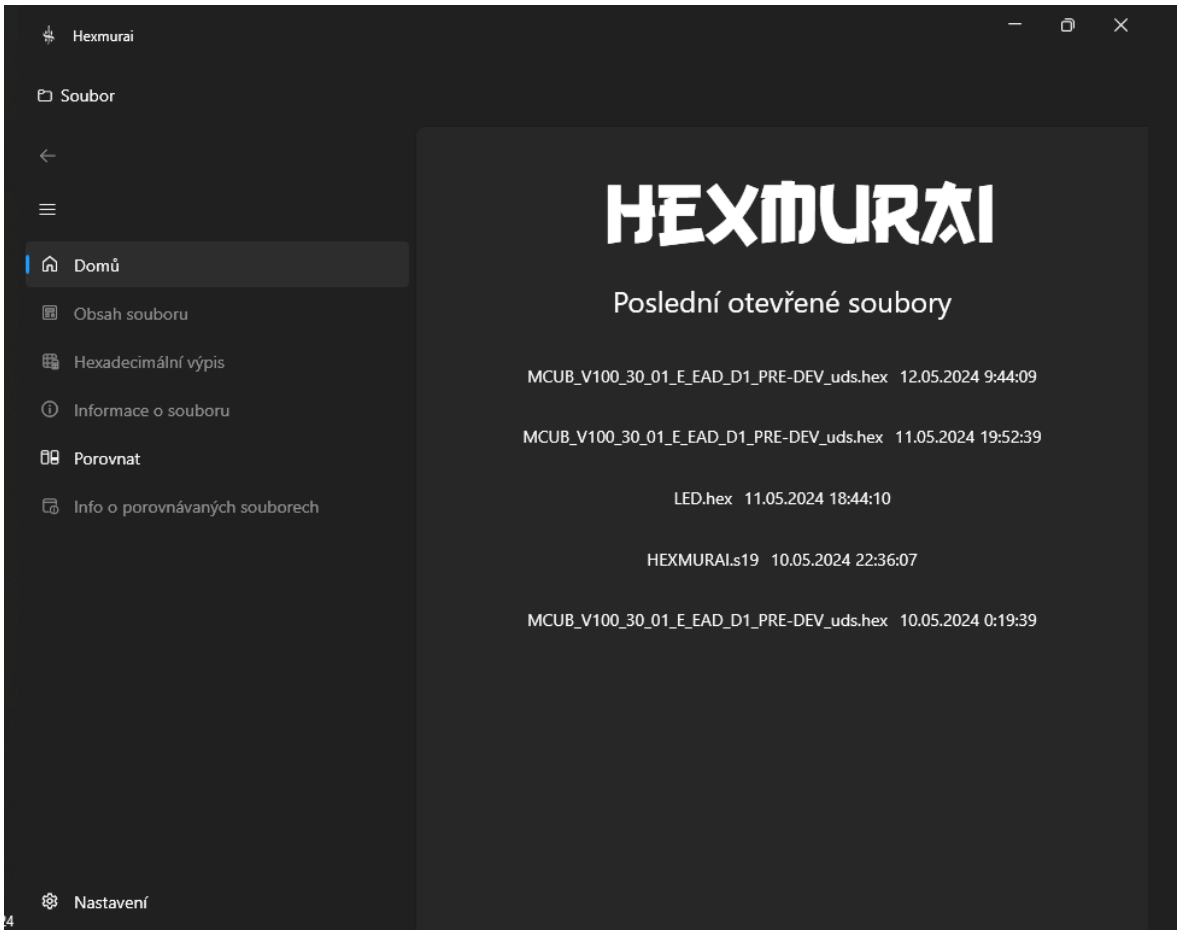

Obrázek 19 Návrh obrazovky pro porovnání dvou souborů




Obrázek 20 Návrh pro zobrazení informací o souboru

# 6 IMPLEMENTACE

Desktopová aplikace pro porovnání a úpravu .hex a .s19 souborů je vytvořena ve vývojovém prostředí Microsoft Visual Studio 2022 na platformě WPF.NET 8.0. Aplikační logika je řešena programovacím jazykem C# a vzhled značkovacím jazykem XAML. Aplikaci je možné používat pouze v operačním systému Windows. Během vývoje byly použity tři NuGet balíčky – framework WPF UI pro vzhled, QuestPDF pro generování reportu a middleware StaticFiles pro zjištění typu internetového média (MIME).




Obrázek 21 Úvodní obrazovka aplikace

## 6.1 Souborová struktura aplikace

Hlavní okno aplikace *MainWindow* se nachází v kořenovém adresáři aplikace a obsahuje jednotlivé stránky (Pages), které jsou umístěny ve složce *Views*. Všechny stránky aplikace obsahují horní menu pro výběr a uložení souboru a boční nabídku pro navigaci na požadované stránky. Toto menu lze sbalit pomocí horního tlačítka s ikonou tří svislých čár. Složka Views obsahuje podsložku *Pages*, kde se nacházejí soubory ve formátu XAML pro zobra-

zení obsahu. Tyto soubory obsahují kód aplikační logiky (code-behind) ve stejnojmenném souboru, avšak s příponou xaml.cs.

Ve složce Models jsou umístěny soubory, které pomocí tříd reprezentují datové struktury, se kterými aplikace pracuje. Jedním z těchto souborů je model *HexData* reprezentující jeden řádek tabulky v hexadecimálním výstupu. Obsahuje další vlastnosti, které se využívají například pro zvýraznění rozdílů mezi soubory nebo určení, zda se bajty při porovnání nacházejí pouze v jednom souboru. Dále je zde statická třída s funkcí pro navrácení adresního rozsahu aplikace a struktura pro zobrazení posledních 5 otevřených souborů.

Jedním z požadavků bylo implementovat změnu jazyka za běhu do češtiny, angličtiny a němčiny. Texty těchto jazyků jsou uloženy ve složce Resources ve formátu XAML jako slovník, kde každý řetězec má svůj klíč. Při změně jazyka se načte do prvku pro zobrazení řetězec podle klíče z daného slovníku. Aplikace také umožňuje uložení volby jazyka, což znamená, že při dalším spuštění aplikace se automaticky nastaví preferovaný jazyk.

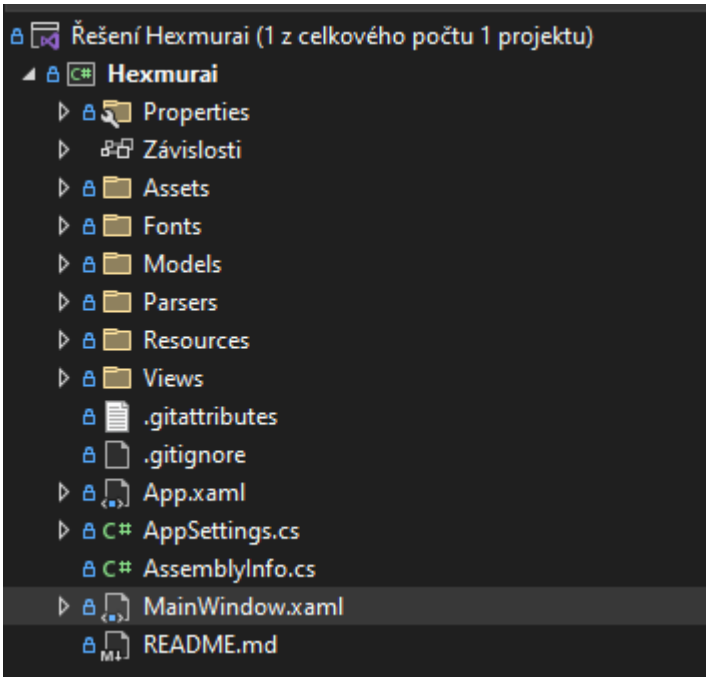
Pro správné otevření a zobrazení vybraných hexadecimálních souborů v tabulce je nejprve zapotřebí soubor přečíst po řádcích a tyto řádky správně zpracovat podle formátu. Ve složce Parsers jsou umístěny třídy s funkcemi pro zpracování datových bloků. Do těchto bloků se nejprve ukládají data, a poté se při zobrazování z nich čte a dále se upravují pro korektní zobrazení.

Složky Assets a Fonts obsahují soubory, které slouží pro zobrazení, konkrétně ikonu aplikace na horní liště okna a externí písma Gang of Three a Century Gothic, která se využívají pro nadpis a text pro název a čas otevření posledních pěti souborů.

V kořenovém adresáři, mimo složky a soubory generované prostředím a verzovacím softwarem, se nachází soubory pro nastavení při spuštění, konkrétně *App.xaml* a *AppSettings.cs*. V prvním jmenovaném jsou definovány zdrojové soubory použité v aplikaci, včetně použitých písem, slovníků pro překlad textů a výchozí hodnoty motivu aplikace. Obsahuje také aplikační logiku pro tento XAML soubor a funkce, které se vykonávají při spuštění aplikace. Při startu se definuje nastavení balíčku QuestPDF pro generování PDF, včetně licence, možností cachování a debuggování.

Jelikož aplikace načítá zvolené možnosti v nastavení i po vypnutí ze systémové složky Dokumenty ve formátu JSON, tak při spuštění programu probíhá kontrola, zda v počítači existuje tento systémový adresář složka se jménem programu. Pokud ne, vytvoří se a uloží

do ní JSON soubor *appsettings*, který obsahuje hodnoty pro motiv aplikace a aktuálně zvolený jazyk (výchozí hodnoty jsou tmavý režim a čeština).



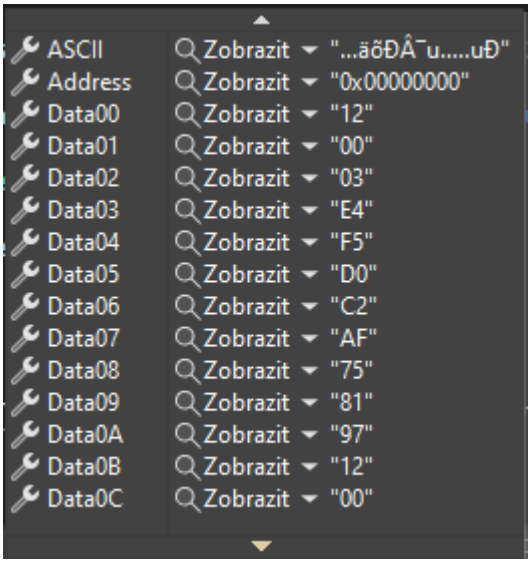

Obrázek 22 Souborová struktura aplikace

## 6.2 Model HexData

Model *HexData* je klíčovou součástí aplikace určené k zobrazení a manipulaci dat hexadecimálního souboru. Každá instance tohoto modelu představuje jeden řádek v tabulce zobrazení. Tento model obsahuje vlastnosti pro adresu a 15 párů znaků, představujících jednotlivé bajty dat. Jakmile se hodnota některé z těchto datových vlastností změní, spustí se událost *PropertyChanged*. Tím je umožněno informovat o změnách v datových polích, což automaticky aktualizuje hodnoty nejen v kolekci dat, ale také v tabulce. Na konci každého řádku je vlastnost ASCII, která obsahuje přeložený řetězec bajtů do kódování ASCII.

Kromě toho model obsahuje další vlastnosti pro porovnání dat mezi soubory. Tyto vlastnosti určují, zda se bajty na určité adrese vyskytují pouze v prvním nebo druhém souboru, a zda mají být zvýrazněny shody nebo rozdíly. Pro označení, které bajty se neshodují, obsahuje model pole boolovských hodnot *IsDifferent*, *IsSame*, *IsDifferentReport* a *IsSameReport*. Tato pole indikují rozdíly a shody mezi daty při porovnávání mezi soubory a následně při generování reportů.

Konstruktor třídy inicializuje tato pole a k nim definuje metody *SetDifference*, *SetSame*, *SetDifferenceReport* a *SetSameReport*, které slouží k nastavení hodnot v těchto polích a indikátorům rozdílů a shod.




Obrázek 23 Část hodnot modelu HexData

## 6.3 Parsery

Složka *Parsers* obsahuje třídy *HexFile* a HexFileBlock, ve kterých je vytvořená logika pro zpracování záznamů popsaných souborových formátů.

Třída *HexFile* reprezentuje celý hexadecimální soubor a umožňuje manipulaci s jeho daty a poskytuje metody pro nastavení dat na určité adrese a získání dat z dané adresy. Metoda *SetData* umožňuje nastavit data na určité adrese v souboru. Pokud jsou data již v souboru obsažena, provede se aktualizace těchto dat. Metoda *GetData* slouží k získání dat z dané adresy a určité velikosti. Dále obsahuje interní metody pro odstranění přepsaných bloků a udržování správné struktury dat.

Třída *HexFileBlock* představuje jednotlivý blok dat v hexadecimálním souboru a obsahuje informace o počáteční adrese bloku, jeho velikosti a samotná data. Tato třída umožňuje nastavovat data na základě adresy nebo offsetu, přidávat data na konec bloku a udržovat konzistenci dat v rámci bloku. Vnitřní implementace této třídy zajišťuje správnou manipulaci s daty a zachování integrity bloků v rámci hexadecimálního souboru.

Jelikož formáty Intel HEX a SRecord se od sebe liší úvodními znaky, typy záznamů, délkou adresy a celkovým uspořádáním jednotlivých položek, jejich zpracování je implementováno odděleně ve třídách *IntelHexParser* a *S19Parser*. Následující popis detailně pojednává o funkcionalitě těchto tříd.

### 6.3.1 IntelHEXParser

Statická třída *IntelHEXParser* inicializuje číselné konstanty pro všech 6 typů Intel HEX záznamů, dále inicializuje indexy, kde se v záznamu vyskytuje úvodní kód (dvojtečka), počet bajtů, adresa, typ záznamu a samotná data a pak jejich délka včetně kontrolního součtu. Samotná operace načtení souboru je realizována pomocí asynchronní metody *LoadAsync*, která přijímá instanci Stream. Během načítání souboru je vytvořena nová instance třídy *HexFile*, která bude obsahovat data z načteného souboru. Následně je inicializován *StreamReader* pro čtení ze streamu.

Během každé iterace smyčky metoda postupně načítá jednotlivé řádky ze souboru a zpracovává je. Každý řádek představuje jeden záznam v hexadecimálním formátu a v případě prázdného řádku se iterace přesouvá na následující řádek. Pokud není řádek prázdný, provádí se zpracování pomocí metody *ProcessLine*.

Tato metoda slouží k analýze jednoho řádku hexadecimálního záznamu a převodu tohoto řádku na instanci třídy *Record*, která obsahuje informace o typu záznamu, adrese a datech. Nejprve jsou provedeny kontroly integrity řádku a její délky. Pokud je menší než suma indexu dat a velikosti kontrolního součtu, vyvolá se výjimka s chybou „Neplatná délka záznamu“. Dále se kontroluje první znak řádku, který by měl být úvodním znakem, v případě Intel HEX dvojtečka. Pokud se nejedná o očekávaný znak, vyvolá se výjimka s informací o neplatném prvním znaku.

Následně jsou z řádku extrahovány informace o počtu bajtů, adrese, typu záznamu a samotná data. Počet bajtů je získán z řádku jako řetězec, ze kterého jsou extrahovány dva znaky za prvním znakem. Tato hodnota je poté převedena do formátu hexadecimálního řetězce. Adresa následuje za počtem bajtů a je reprezentována čtyřmi znaky. Typ záznamu, který má délku 1 bajtu, je získán jako následující dva znaky za adresou v řetězci řádku a je převeden na celé číslo ve formátu hexadecimálního řetězce. Po získání těchto hodnot jsou provedeny další kontroly, jako je porovnání délky řádku s očekávanou délkou. Očekávaná délka je součtem indexu dat, velikosti kontrolního součtu a dvojnásobku počtu bajtů. Pokud délky nesouhlasí, vyvolá se výjimka s chybou „Neplatná délka záznamu“.

Následně jsou extrahována samotná data a vypočten kontrolní součet. Data v Intel HEX začínají na indexu 9 a nacházejí se v řetězci až do předposledního znaku. Poté jsou tato data převedena na bajty. Pro každý bajt dat je vypočítán kontrolní součet, který je součtem počtu bajtů, adresy, kódu typu záznamu a jednotlivých bajtů dat.

Nakonec jsou porovnány vypočítaný a skutečný kontrolní součet, a pokud se neshodují, vyvolá se výjimka s informací o neplatném kontrolním součtu. Nakonec je vytvořena a vrácena instance třídy *Record* s informacemi o typu záznamu, adrese a datech.

V průběhu zpracování záznamů formátu Intel HEX může dojít na záznamy 02 a 04 obsahující rozšířené adresy, jejichž hodnoty se ukládají do proměnných *extendedSegmentAddress* a *extendedLinearAddress*. Pro získání těchto adres slouží metody *GetExtendedSegmentAddress* a *GetExtendedLinearAddress*, kde se nejprve provádějí kontroly, zda je délka dat v záznamu 2 bajty a pokud ano, tak se provede dekódování rozšířené adresy ze záznamů. V případě rozšířené segmentové adresy se první bajt v poli dat posune o 8 bitů doleva (posun o 1 bajt) a druhý bajt je posunut o 4 bity doleva. Tyto hodnoty jsou následně sloučeny pomocí bitového operátoru OR. U rozšířené lineární adresy je rozdíl v posunutí, první bajt je posunut o 24 bitů doleva a druhý bajt je posunut o 16 bitů doleva. Pokud je detekován ko-

nec souboru (EOF záznam), cyklus se ukončí. Metoda vrací načtená data v instanci třídy *HexFile*.

### 6.3.2 S19Parser

Třída S19Parser slouží k zpracování souborů ve formátu Motorola SRecord a sdílí mnoho vlastnost a metod s třídou *IntelHEXParser*. Hlavní rozdíl spočívá v typech a délce adresy záznamů. Inicializuje číselné konstanty pro indexy, kde se v řetězci nachází typ záznamu, počet bajtů a adresa, stejně jako jejich délku.

Třída zpracovává pouze řádky, jejichž typ záznamu je S1, S2 a S3, což jsou datové záznamy obsahující bajty představující data. Ostatní typy záznamů jsou ignorovány. Zpracování probíhá v metodě *ProcessLine*, která transformuje řádek na instanci třídy *Record*. Při zpracování řádku se kontroluje, zda je záznam kompletní nebo zda byl zkrácen. Pokud je délka řetězce menší než suma pozice, na které začíná adresa, délky adresy a velikosti kontrolního součtu, vyvolá se výjimka s textem „Zkrácený záznam“.

Z řetězce se extrahují první dva znaky, což je typ záznamu, poté následuje počet bajtů o délce 4 znaků, které následují za typem záznamu, a adresa, jejíž délka závisí na typu záznamu (4 bajty pro S1, 6 bajtů pro S2 a 8 bajtů pro S3). Pokud dojde k chybě při převodu, vyvolá se výjimka s popisem chyby.

Dalším krokem je ověření, zda délka řádku odpovídá očekávané délce určené počtem bajtů dat. Pokud se délka řádku neshoduje s očekávanou délkou, vyvolá se výjimka s chybou „Neplatná délka záznamu“.

Následně se vypočítá kontrolní součet. K tomu se používají počet bajtů, adresa a samotná data. Kontrolní součet se vypočítá jako suma všech těchto hodnot, na které se aplikuje negace. Poté se extrahují samotná data z řetězce a převedou se na bajty. Kontroluje se také správnost každého bajtu v rámci extrakce dat. Nakonec se porovnají vypočtený a skutečný kontrolní součet. Pokud se oba součty neshodují, vyvolá se výjimka s informací o neplatném kontrolním součtu.

## 6.4 Otevření souboru

Pro otevření souboru ke zpracování uživatel vybere možnost „Otevřít“ z horní hlavní nabídky. Následně se zobrazí dialog umožňující vybrat soubory pouze s příponami .s19 a .hex. Po výběru souboru se aplikace přepne na stránku pro hexadecimální výstup. Zde dochází k čtení a zpracování souboru řádek po řádku. Během tohoto procesu je na obrazovce zobrazen točící ukazatel průběhu, který je skryt, jakmile je soubor zpracován.

Po dokončení zpracování je obsah souboru načten do tabulky a zobrazen na obrazovce. Aktivují se také tlačítka v bočním menu umožňující přechod na další stránky, které se týkají otevřeného souboru. Tyto stránky zahrnují zobrazení obsahu souboru, jeho metadata a statistiky. Dále se do textového souboru v adresáři Dokumentů zapíše cesta k otevřenému souboru spolu s časovým razítkem tohoto zápisu. Tento soubor obsahuje informace o posledních 5 otevřených souborech v aplikaci na daném počítači spolu s časem jejich otevření.

```
1  private void openFile_Click(object sender, RoutedEventArgs e)
2  {
3      OpenFileDialog openFileDialog = new OpenFileDialog();
4      openFileDialog.Filter = "Hex Files (.hex, .s19)|*.hex;*.s19";
5      openFileDialog.Title = Application.Current.Resources["openFileString"].ToString();
6
7      if (openFileDialog.ShowDialog() == true)
8      {
9          FileName = openFileDialog.FileName;
10         fileLocation = openFileDialog.FileName;
11         fileContent.IsEnabled = true;
12         fileContentHex.IsEnabled = true;
13         SaveLastOpenedFiles();
14         saveFileAs.IsEnabled = true;
15         infoFile.IsEnabled = true;
16         RootNavigation.Navigate(typeof(ContentHexPage));
17     }
18 }
```

Obrázek 24 Kód pro otevření souboru

## 6.5 Čtení souboru

V aplikaci je možné si vybrat, zda zobrazit soubor, nebo zda ho přečíst řádek po řádku, zpracovat do příslušného formátu a zobrazit až datové bajty z těchto záznamů. Při otevření

souboru se automaticky aplikace přepne na hexadecimální výstup, v bočním menu je ale možnost přepnutí na běžné zobrazení obsahu.

### 6.5.1 Hexadecimální výstup

Po otevření souboru a přesměrování na stránku *ContentHexPage* se spustí proces zpracování obsahu hexadecimálního souboru v souladu s formátem Intel HEX nebo Motorola SRecord. Při této operaci se zobrazí točící indikátor průběhu a v horní části stránky se zobrazí název souboru a jeho formát podle přípony.

Pro zobrazení obsahu souboru v aplikaci je využíván vizuální prvek DataGrid, který je konfigurován tak, aby byl schopen zobrazit velký objem dat. Pro zvýšení výkonu aplikace byly provedeny úpravy tohoto prvku, jako je virtualizace řádků a sloupců, nastavení maximální výšky a zakázání některých nechtěných funkcí, jako je řazení řádků a sloupců.

První řádek tabulky obsahuje záhlaví s názvy jednotlivých sloupců Adresa daného záznamu je obsažena v prvním sloupci a každý řádek je hexadecimálně inkrementován o hodnotu jedna. Sloupce 2 až 17 obsahují jednotlivé datové bajty ve formátu dvojice šestnáctkových čísel (1 bajt). Poslední sloupec zobrazuje ASCII výstup, kde každý bajt je přeložen do odpovídajícího ASCII znaku.

Zdrojem dat pro DataGrid je *ObservableCollection* s názvem *hexDataList*, která automaticky aktualizuje hodnoty v kolekci i na zobrazení. V kódu této stránky jsou inicializovány další prvky, jako seznam všech řádků, slovník pro ukládání řádků v jejich původní formě, proměnná pro název souboru a událost, která při změně hodnoty v seznamu řádků automaticky předá změněný seznam do kódu hlavního okna (MainWindow) připraveného pro případné uložení. V konstruktoru jsou uvedeny události, které jsou spuštěny při jejich vyvolání. Metoda *HandleFileNameChanged* je připojena k události *FileNameChanged* hlavního okna aplikace a kdykoliv dojde ke změně názvu souboru v hlavním okně (otevření souboru), spustí se funkce *HandleFileNameChanged*, kde se aktualizuje lokální proměnná pro název souboru na této stránce a asynchronně se načte hexadecimální výpis pro nový soubor. Dále je připojena událost *SizeChanged* aktuálního okna, která spustí metodu *Window_SizeChanged*, kde se vypočítá maximální výška DataGridu, aby se zobrazil korektně na základě nové výšky okna.

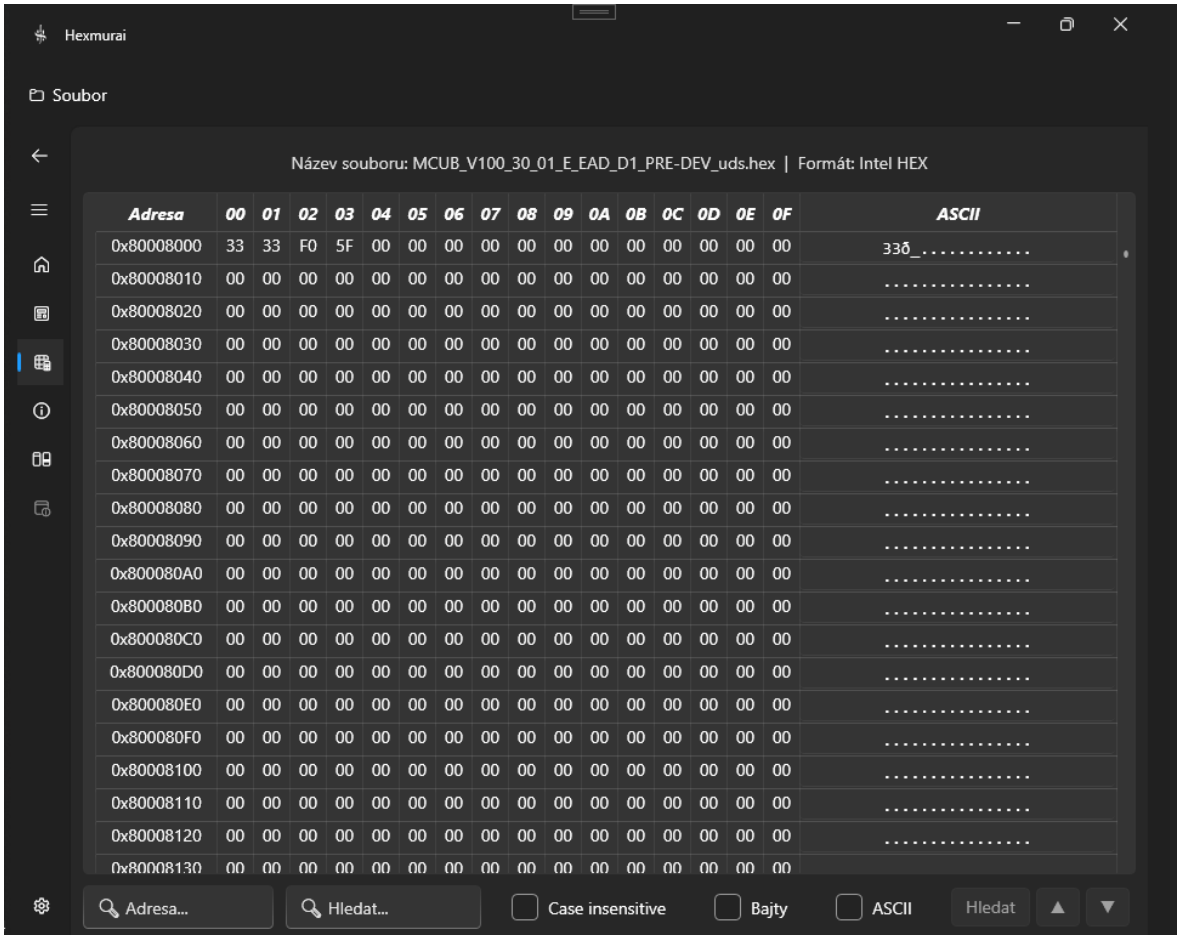
Zpracování hexadecimálního výstupu je implementováno v asynchronní metodě *LoadDataGrid*, která očekává parametr s cestou k souboru. Před čtením souboru se vyprázdní ko-

lekce, vrátí se vizuální prvky do původního stavu a zkontroluje se, zda parametr není prázdný. Poté se extrahuje z cesty název souboru a podle přípony se určí formát. Následně se přečtou všechny řádky v souboru a uloží do kolekce hexLines; přípona určuje podle jakého formátu bude obsah zpracován. Pro každý řádek se provede zpracování podle typu záznamu a uložení pouze datových záznamů. Poté se použije příslušný parser pro daný formát, který ukládá data do bloků. Z těchto bloků se získávají data podle startovací adresy bloku a velikosti bloku, vypočte se offset, nastaví se hodnota všech bajtů v řádku na hodnotu „FF“ a postupně se dosazují bajty z bloků na příslušné adresy. Na konci řádku se provede výpočet ASCII. Po uložení do kolekce se všechny řádky vypíší do tabulky prostřednictvím vlákna uživatelského rozhraní. Přečtené řádky souboru se načtou do proměnné *FileContent*, která vyvolá událost a umožní tak tento výstup uložit jako nový soubor. Po výpisu se povolí možnost vyhledávání.


Obrázek 25 Hexadecimální výstup otevřeného souboru

### 6.5.2 Obsah souboru

Po načtení hexadecimálního výstupu na obrazovku je uživateli umožněno zobrazit obsah otevřeného souboru z bočního menu. Po výběru této možnosti se aktuální stránka přepne na stránku s obsahem otevřeného souboru (*ContentPage*). Vizuálně stránka pro zobrazení obsahu souboru je totožná s tou pro hexadecimální výpis, liší se pouze v tom, jak jsou data v tabulce interpretovány.




Obrázek 26 Obsah souboru

Čtení obsahu souboru je implementováno v asynchronní metodě *LoadDataGrid*, která očekává parametr s cestou k souboru. Nejprve se kontroluje, zda je zadaná cesta neprázdná, inicializují se vizuální prvky a vyčistí se příslušné seznamy pro případ zpracování dalšího souboru. Následně se z cesty souboru vybere jeho název souboru spolu s příponou a tyto informace jsou zobrazeny uživateli. Před zpracováním souboru se zobrazí tabulka a točící indikátor průběhu. Po načtení souboru je jeho obsah zpracován a zobrazen v tabulce. Pro přidání každého řádku do tabulky je využívána kolekce *ObservableCollection*, která automaticky aktualizuje zobrazení tabulky při jakékoliv změně dat.

Čtení obsahu souboru probíhá pomocí třídy *FileStream*, která čte bajty ze souboru postupně. Každý přečtený bajt je převeden do hexadecimální podoby a přidán do řádku. Po každých 16 bajtech je vytvořen řádek tabulky obsahující adresu, hexadecimální data a ASCII reprezentaci. Tento řádek je následně přidán do seznamu řádků tabulky. Aby bylo možné správně aktualizovat uživatelské rozhraní, je přidání nového řádku prováděno přes dispečera aplikace. Nakonec je provedeno zavření *FileStream* a zobrazení všech načtených řádků v tabulce.

```
string hexLine = $"0x{addressOffset:X6} {hexLineBuilder}";
hexLineBuilder.Clear();
addressOffset += 16;
string[] hexPairs = hexLine.Split(' ');
var asciiData = hexPairs.Skip(1).Take(16).Select(hex =>
{
    if (hex == "")
    {
        return ".";
    }
    else
    {
        int asciiValue = Convert.ToInt32(hex, 16);
        char asciiChar = asciiValue < 32 || asciiValue > 126 ? '.' :
(char)asciiValue;
        return asciiChar.ToString();
    }
}).ToArray();
var asciiString = string.Join("", asciiData);
HexData hexData = new HexData
{
    Address = hexPairs[0],
    Data00 = hexPairs.Length > 1 ? hexPairs[1] : "  ",
    Data01 = hexPairs.Length > 2 ? hexPairs[2] : "  ",
    Data02 = hexPairs.Length > 3 ? hexPairs[3] : "  ",
    Data03 = hexPairs.Length > 4 ? hexPairs[4] : "  ",
    Data04 = hexPairs.Length > 5 ? hexPairs[5] : "  ",
    Data05 = hexPairs.Length > 6 ? hexPairs[6] : "  ",
    Data06 = hexPairs.Length > 7 ? hexPairs[7] : "  ",
    Data07 = hexPairs.Length > 8 ? hexPairs[8] : "  ",
    Data08 = hexPairs.Length > 9 ? hexPairs[9] : "  ",
    Data09 = hexPairs.Length > 10 ? hexPairs[10] : "  ",
    Data0A = hexPairs.Length > 11 ? hexPairs[11] : "  ",
    Data0B = hexPairs.Length > 12 ? hexPairs[12] : "  ",
    Data0C = hexPairs.Length > 13 ? hexPairs[13] : "  ",
    Data0D = hexPairs.Length > 14 ? hexPairs[14] : "  ",
    Data0E = hexPairs.Length > 15 ? hexPairs[15] : "  ",
    Data0F = hexPairs.Length > 16 ? hexPairs[16] : "  ",
    ASCII = asciiString
};
    Application.Current.Dispatcher.Invoke(() =>
{
    hexDataList.Add(hexData);
});
});
```

Obrázek 27 Kód pro zpracování jednoho řádku

## 6.6 Informace o souboru

Další možností v bočním menu je tlačítko s textem „Informace o souboru“, která uživatele přesměruje na stránku "InfoPage". Zde jsou prezentovány detailní informace o vybraném

souboru, zahrnující kompletní cestu k souboru, jeho velikost, metadata jako datum vytvoření a datum poslední modifikace, adresní rozsah obsahu souboru a výpis bloků, ve kterých se nacházejí data. Každý blok je popsán počáteční a koncovou adresou a délkou jak v hexadecimální, tak decimálním formátu. Posledním údajem je celková délka bloků, kde je zobrazena suma délek jednotlivých bloků. Před načtením stránky je zobrazen indikátor průběhu, který se skryje po dokončení načtení informací.

Vzhled stránky je vytvořen pomocí prvku *ListView*, který je umístěn v horní části stránky. Každá informace je prezentována jako jedna položka v seznamu a je reprezentována dvojicí *PropertyName* a *PropertyValue*. Název informace je zobrazen tučně (PropertyName), zatímco hodnota informace (PropertyValue) je umístěna na pravé straně.

```xaml
<ui:ListView x:Name="lstFileInfo" Grid.Row="0" Margin="10">
    <ui:ListView.ItemTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding PropertyName}" FontWeight="Bold"/>
                <TextBlock Text=" "/>
                <TextBlock Text="{Binding PropertyValue}"/>
            </StackPanel>
        </DataTemplate>
    </ui:ListView.ItemTemplate>
</ui:ListView>
```

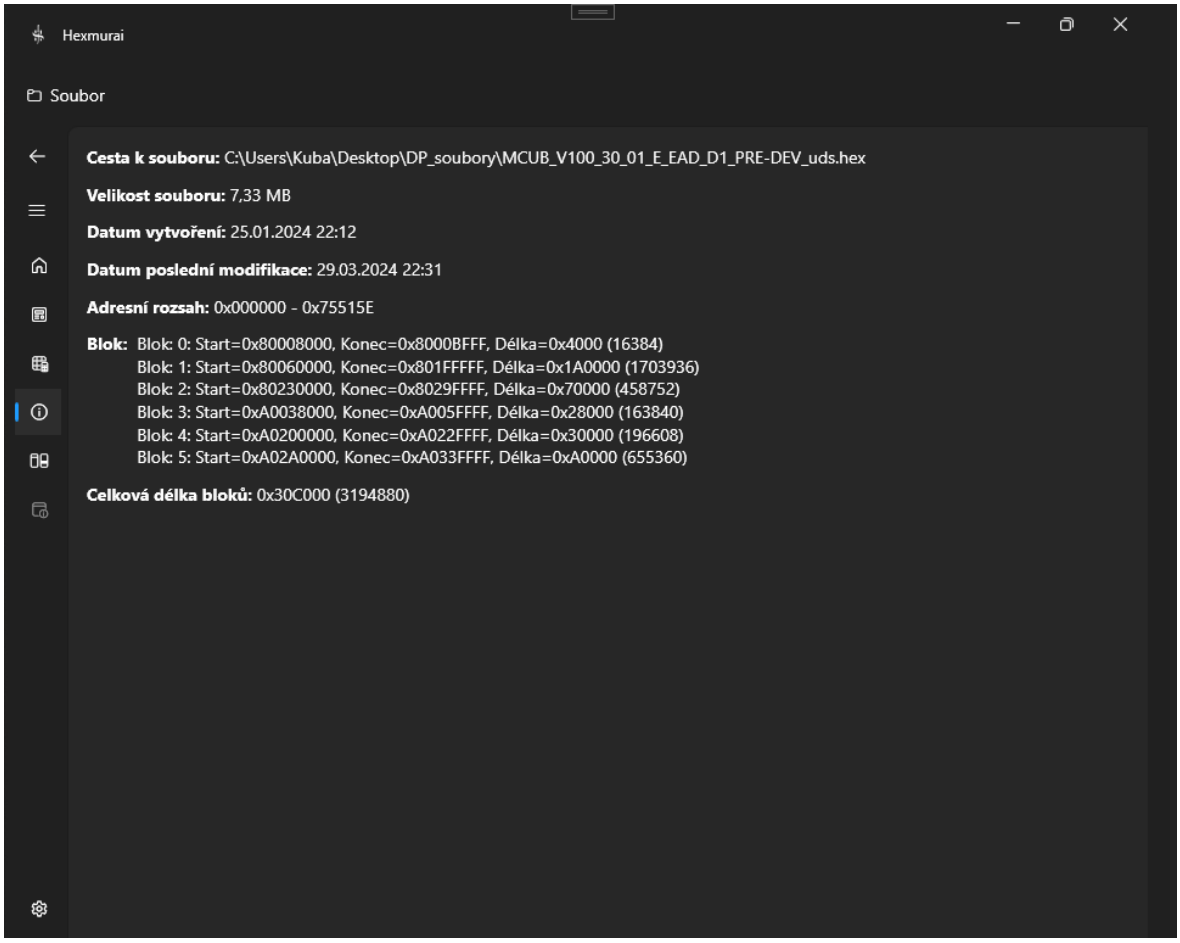
Obrázek 28 Kód pro zobrazení informací ze seznamu v XAML

V kódu této stránky je inicializována třída *FileInfoProperty*, která slouží k reprezentaci jednotlivých položek informací o souboru. Každá položka obsahuje výše uvedené vlastnosti reprezentující název a hodnotu informace.

Při načítání stránky jsou volány metody pro nastavení jazyka a načtení informací o souboru. Pokud se změní jazyk aplikace, informace o souboru jsou aktualizovány již s přeloženými texty. Pokud dojde ke změně názvu souboru, informace jsou znovu načteny pro nový soubor.

Metoda *LoadFileInfo* načte informace o souboru na základě zadaného názvu souboru. Získají se metadata o souboru, jako je jeho velikost, a tyto informace se přidají do kolekce

*fileInfoProperties*, která slouží jako zdroj dat pro zobrazení v *ListView* na stránce. Provádí se analýza obsahu souboru a získávají se informace o blocích paměti, ve kterých se vyskytují data. Celková délka těchto bloků je také spočítána a přidána do seznamu informací. Nakonec jsou všechny informace zobrazeny v *ListView* na stránce.




Obrázek 29 Informace o souboru

## 6.7 Editace a ukládání souboru

Funkce úpravy souboru je přístupná na stránce zobrazující hexadecimální výstup a umožňuje uživatelům editovat jednotlivé bajty v souboru a následně tyto úpravy uložit.

Editace hodnoty datového bajtu se spustí dvojklikem na požadovanou buňku, což vyvolá textové pole s označenou aktuální hodnotou. Uživatel smaže původní hodnotu a napíše novou, kterou potvrdí stisknutím klávesy Enter. Před vložením nové hodnoty je původní hodnota uložena do proměnné, aby bylo možné obnovit původní hodnotu, pokud by uživatel zkoušel vložit prázdnou hodnotu.

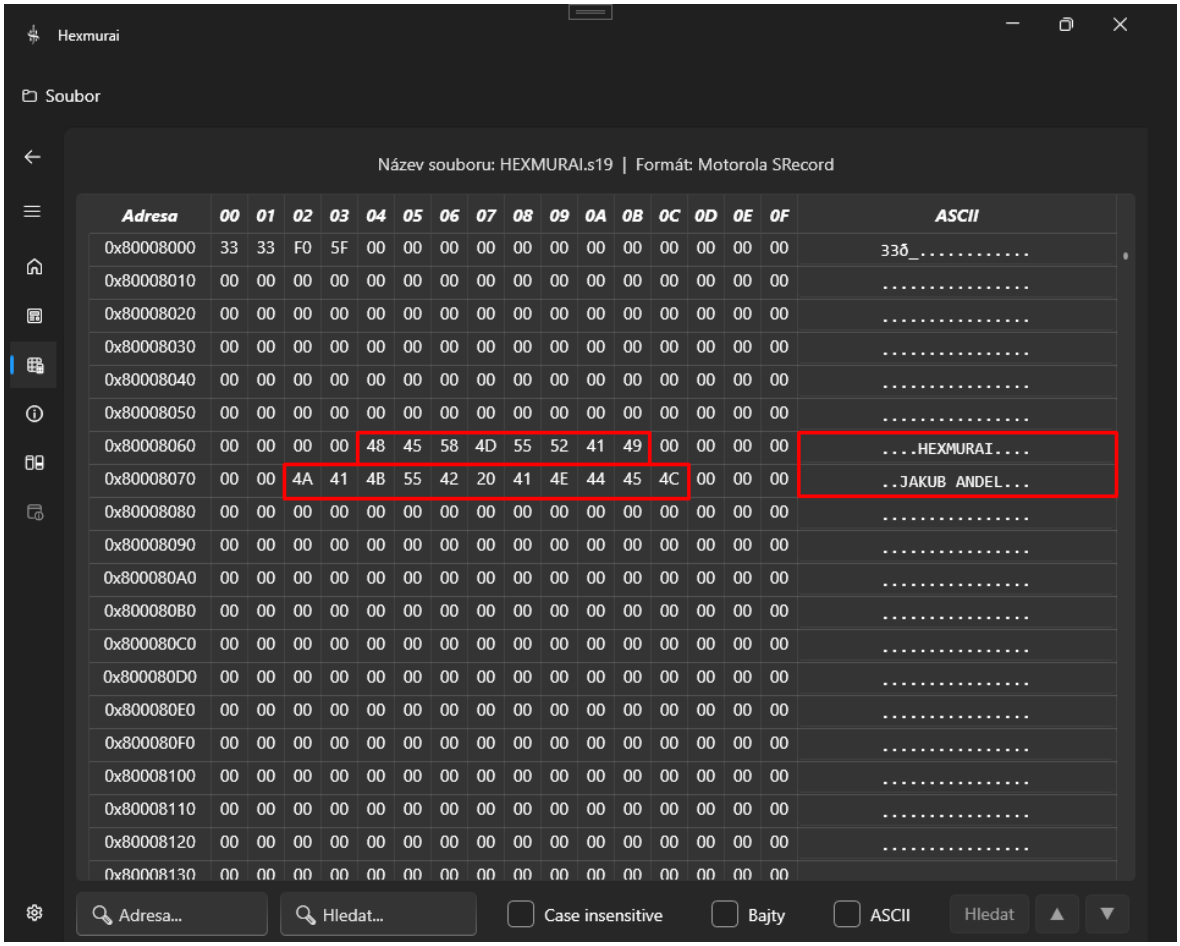
Po stisku klávesy Enter se spustí událost DataGridu s názvem *DataGrid_CellEditEnding*, která kontroluje platnost vložené hodnoty. Pokud je textové pole prázdné nebo obsahuje pouze jeden znak, editace se neprovádí a původní hodnota je obnovena do buňky.

Následně se načte řádek obsahující modifikovanou hodnotu a vypočítá se ASCII pro celý řádek. Poté se spočítá adresa a index datového bajtu v tomto řádku a vyhledá se odpovídající řádek v kolekci datových řádků. Na základě přípony souboru se určí formát a podle něho se vytvoří nový záznam s aktualizovanými datovými bajty a přepočítaným kontrolním součtem. Nakonec se aktualizovaná kolekce s řádky vloží zpět do proměnné *FileContent*, čímž se zaktualizuje obsah souboru.

```
if (editedData != hexByte)
{
    bytePairs[dataByteIndex] = editedData;
    string newByteString = string.Join("", bytePairs);
    string newDataString = string.Empty;
    string checksum = string.Empty;
    if (extension == ".hex")
    {
        newDataString = line.Substring(0, 9) + newByteString + line.Substring(line.Length - 2);
        checksum = CalculateIntelChecksum(newDataString);
    }
    else if (extension == ".s19")
    {
        newDataString = line.Substring(0, 2) + line.Substring(2, 2) + line.Substring(4, addressSize) + newByteString + line.Substring(line.Length - 2);
        checksum = CalculateMotorolaChecksum(newDataString);
    }
    newDataString = newDataString.Substring(0, newDataString.Length - 2) + checksum;
    dataHexLines[entry.Key] = newDataString;
    int indexLine = hexLines.IndexOf(line);
    hexLines[indexLine] = newDataString;
    break;
}
```

Obrázek 30 Kód pro editaci datových bajtů v záznamu

V hlavním menu se nachází možnost „Uložit jako“, která zobrazí *SaveFileDialog* pro výběr umístění, kam se uloží soubor. Po potvrzení umístění a zadání názvu se obsah aktualizované kolekce uloží do nového souboru. Po úspěšném uložení se zobrazí informační dialogové okno. Tvorba nového souboru probíhá v hlavním vlákně aplikace pomocí dispečera a tím je tak zajištěna synchronizace s uživatelským rozhraním s uživatelským rozhraním.

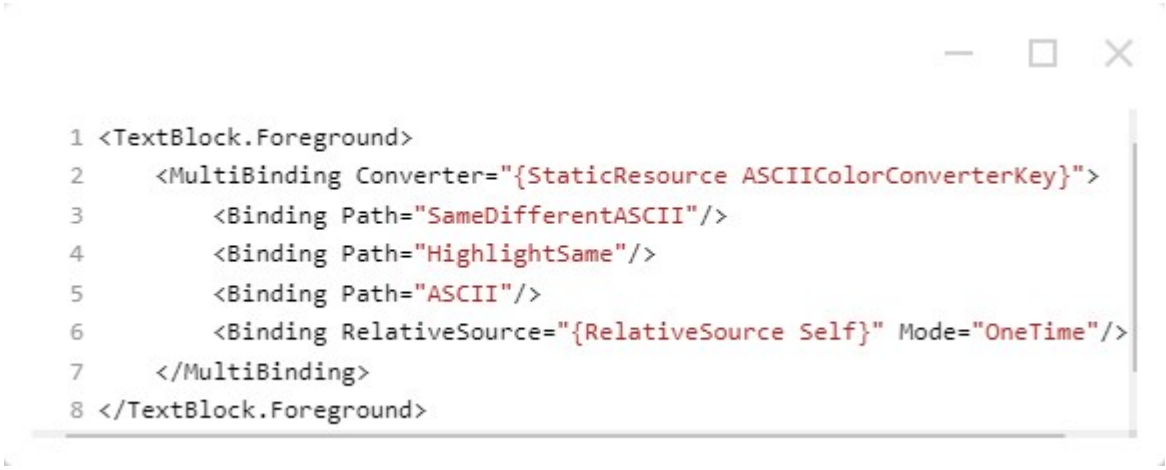



Obrázek 31 Hexadecimální výpis uloženého souboru s modifikovanými bajty

## 6.8 Porovnání souborů

Porovnání hexadecimálního výstupu dvou souborů je možné na stránce *ComparePage*, na které se vyskytují ve dvou sloupcích dva tlačítka pro výběr souborů s indikátorem, zda je soubor vybrán či nikoliv. Dále se při výběru zobrazí název pod tlačítkem název a jeho formát. Uprostřed v horní části se mezi tlačítky pro výběr souborů nachází tlačítko pro návrat stránky do původního stavu a vyprázdnění tabulek, dále rozbalovací menu pro výběr, zda se má porovnávat stejné nebo odlišné bajty a tlačítko „Porovnat“. Tlačítko pro zahájení porovnání je povoleno v momentě, kdy jsou vybrány oba soubory a zvolen režim porovnání. Po zmáčknutí tlačítka se zobrazí na obrazovce indikátor průběhu a dvě tabulky, do kterých se po načtení a zpracování vepíše hexadecimální výpis jednotlivých souborů, barevně zvýrazní buňky, které se buď shodují, nebo liší, a to platí i pro jednotlivé znaky ve sloupci ASCII. Na stránce se v horní části v menu nachází tlačítka pro generování reportu v PDF, zobrazení vyskakovacího okna s výpisem datových bloků obou souborů a přepínač syn-

chronizace, který při aktivování přesune obě tabulky na první stejnou adresu, případně na stejný offset a při vertikálním pohybu myši se posunují obě tabulky najednou.



```
<TextBlock.Foreground>
    <MultiBinding Converter="{StaticResource ASCIIColorConverterKey}">
        <Binding Path="SameDifferentASCII"/>
        <Binding Path="HighlightSame"/>
        <Binding Path="ASCII"/>
        <Binding RelativeSource="{RelativeSource Self}" Mode="OneTime"/>
    </MultiBinding>
</TextBlock.Foreground>
```

Obrázek 32 Definice Multibinding překladače pro sloupec ASCII

Funkcionalitu pro porovnání řeší metoda *compareButton_Click*, kde se kontroluje, zda jsou vybrané soubory a zvolen režim porovnání. Jednotlivé vizuální prvky se vrátí do původního stavu, vyprázdní se kolekce, se kterými se bude pracovat, a zobrazí se indikátor průběhu. Následuje asynchronní čtení hexadecimálních dat ze souborů, po kterém se do *Hashsetů* uloží adresy obou výpisů a jejich průnik (společné adresy). Poté následuje zvýraznění jednotlivých bajtů podle výběru režimu. V případě, že neexistují společné adresy nebo alespoň jedna je jiná, tato informace se zobrazí v horní části stránky. Dále se nastaví, aby se obě tabulky přesměrovaly na nejbližší stejnou adresu, a připravují se informace pro porovnávané soubory. Kromě těch, co byly popsány v kapitole „Informace o souboru“, je stránka *InfoComparePage* doplněna o počet shodných a odlišných bajtů. Počet těchto bajtů je vypočten na základě sumy kladných hodnot v polích *IsSameReport* a *IsDifferentReport*, které primárně slouží pro generování reportu, ale pro tyto účely bylo nutné je použít. Nakonec se zobrazí zpráva, že obsahy souborů jsou stejné, pokud je výsledná suma rozdílných bajtů nulová.

Určení shod a rozdílů v kódu funguje tak, že model *HexData* obsahuje dva boolenovské pole s názvem *IsSame* a *IsDifferent* o velikosti 16 prvků, takže každá hodnota patří k jednomu datovému bajtu v řádku. Procházení obou tabulek a určení hodnot probíhá v metodách *HighlightSame* a *HighlightDifferent*, v první jmenované funkci se převedou hodnoty z tabulek slovníků a klíčem je jejich adresa, spojí se společné adresy obou tabulek do jednoho *HashSetu*, u kterého bylo zjištěno během vývoje, že je na průchod rychlejší než List, především u většího počtu položek. Metoda *HighlightDifferent* funguje na stejném princi-

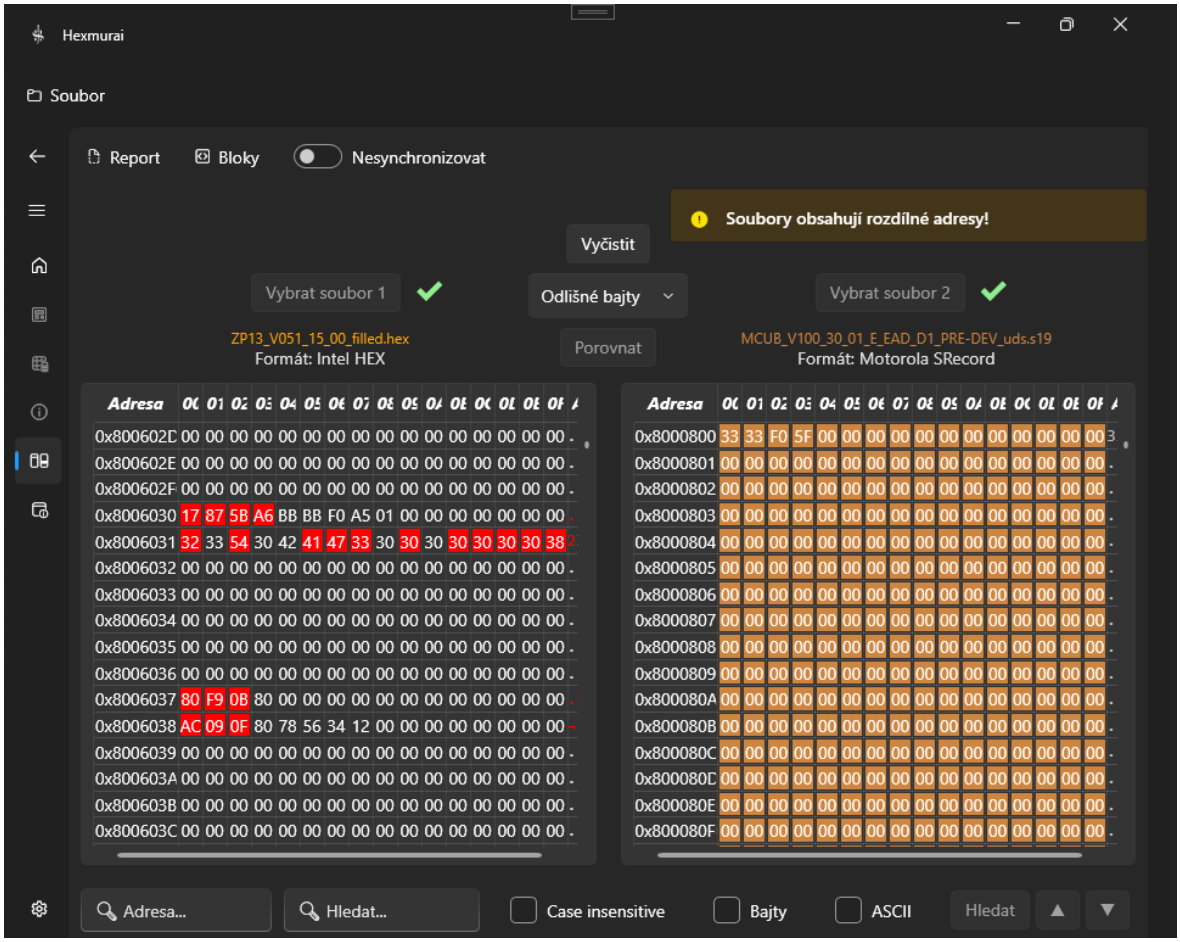
pu, liší se v tom, že neporovnává pouze ve stejných adresách, ale bere v potaz i to, že rozdílem je i situace, když adresy nejsou shodné a prochází se tak všechny adresy. Při průchodu adres se vyhledají oba řádky na stejné adrese, kontroluje se, že se jedná o porovnání a v případě shody nebo odlišnosti obou bajtů na stejné adrese se zapíše kladná hodnota (*true*) do polí *IsSame* nebo *IsDifferent*. Mimo bajtů se nastaví k jednotlivým řádkům do proměnné *SameDifferentASCII* hodnotu z druhé tabulky, která se bude dále porovnávat a podle ní se určí, který znak ASCII se zvýrazní barevně.

Zobrazení zvýrazněných bajtů je vytvořeno pomocí *MultiBinding* překladačů, které jsou definovány v XAML u stylu jednotlivých sloupců pro data a nastavují tak barvu pozadí pro příslušné bajty. Tyto sloupce řeší překladač s názvem *CellColorConverter*, který očekává parametr s indexem daného sloupce (od nuly po 15), boolenovské pole s hodnotami, poté proměnné *HighlightSame* indikující volbu shod nebo rozdílů, *OnlyInFile1* a *OnlyInFile2* pro případ, že by soubor obsahoval adresy vyskytující jen v jednom souboru či druhém souboru a zvýrazní se tak příslušnou barvou v jedné nebo druhé tabulce. Implementaci této třídy *CellColorConverter* dědící od *IMultiValueConverter* se vyskytuje v aplikačním kódu stránky pro přehlednost a obsahuje metodu *Convert*, ve které se kontroluje počet poskytnutých parametrů v poli objektů, které byly definovány v XAML a mimo to se testuje taktéž jejich datový typ. Po kontrole indexu sloupce, a zda se porovnají shody či rozdíly probíhá výběr z pole *differences* a nastavení příslušných barev pro daný režim. Průhledná barva se nastaví, pokud je hodnota v poli nepravda. Pokud se adresa vyskytuje pouze v souboru 1, označí se pozadí všech bajtů na oranžovou barvu, v souboru 2 na hnědou.

```
if (!HighlightSame)
{
    bool difference = differences[index];
    if (OnlyInFile1)
    {
        return difference ? Brushes.Orange : Brushes.Transparent;
    }
    else if (OnlyInFile2)
    {
        return difference ? Brushes.Peru : Brushes.Transparent;
    }
    else
    {
        return difference ? Brushes.Red : Brushes.Transparent;
    }
}
```

Obrázek 33 Nastavení pozadí datových bajtů v CellColorConverter

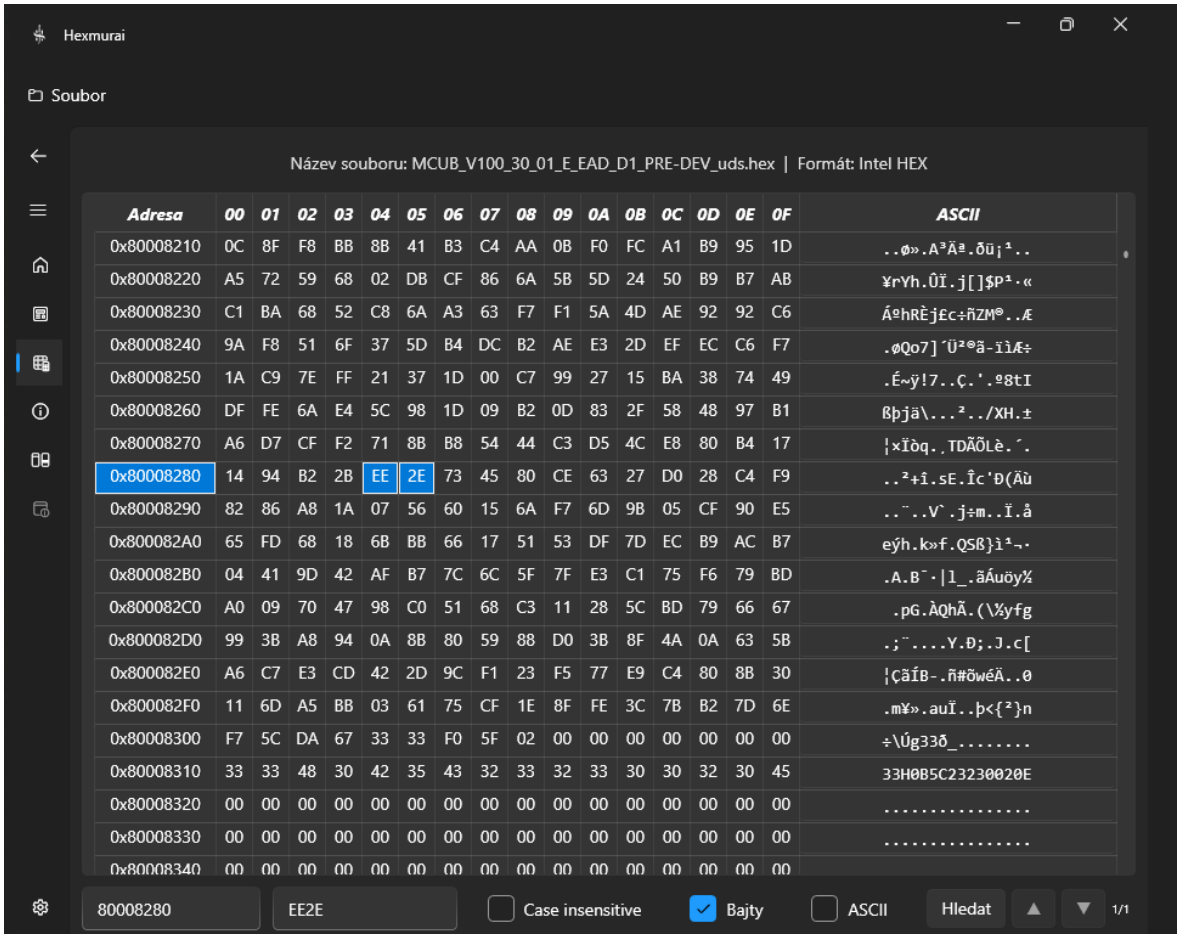
Sloupec ASCII obsahuje taktéž *MultiBinding* překládač, ale nastavuje se tím barva písma pro každý znak (*Foreground*), nazývá se *ASCIIColorConverter* a přebírá proměnné z modelu *SameDifferentASCII*, které reprezentuje hodnotu z posledního sloupce v daném řádku v druhé tabulce, pak *HighlightSame*, které je stejné jako u zvýraznění dat, dále aktuální hodnota a posledním parametrem je *TextBlock*, jenž je vizuální prvek, ve kterém se řetězce zobrazují. Stejně jako u předchozího překladače probíhá v metodě *Convert* kontrola přijatých parametrů a probíhá čtení znak po znaku obou řetězců, v případě shody se znak zbarví do zelena, opačně do červena. Jelikož aplikace podporuje dynamickou změnu režimu, je výchozí barva stříbrná, aby při přepnutí pořád byly znaky stále viditelné.




Obrázek 34 Porovnání dvou souborů v režimu odlišných bajtů

## 6.9 Hledání

Vyhledávací textová pole pro hledání v souborech se nachází na stránkách „Obsah souboru“, „Hexadecimální výstup" a „Porovnat“ ve spodní části aplikace. Skládá se z pole pro vyhledání konkrétní adresy, jednoho bajtu či sekvence bajtů nebo řetězce v ASCII sloupci. Aktivace tlačítka pro hledání vyžaduje vyplnění alespoň jednoho z textových polí a zaškrtávací volbu „Bajty“ nebo „ASCII“. Program umožňuje vybrat pouze jednu možnost a deaktivuje druhou, nedovoluje tedy kombinovat hledání bajtu a řetězce v ASCII současně. Uživatel může hledat bajty nebo řetězce na konkrétní adrese a volit, zda se má ignorovat velikost písmen. Při výběru hledání bajtů se provádí kontrola, zda se jedná o platný hexadecimální znak o sudé délce; v případě neplatného znaku se zobrazí dialogové okno s informací a hledání se neprovádí. Po správném vyplnění vstupů a dokončení hledání se v liště zobrazí počet nalezených výskytů ve formátu 1/x, přičemž uživatel může mezi nimi přepínat pomocí dvou tlačítek vpravo.




Obrázek 35 Hledání sekvence bajtů o délce 4 znaky na konkrétní adrese

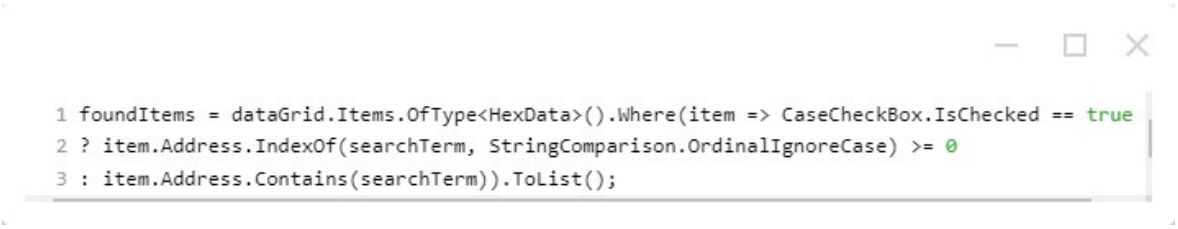
Algoritmus hledání je identický na všech stránkách, s výjimkou stránky „Porovnat“, kde je upraven pro vyhledávání ve dvou tabulkách a úpravu přepínání. Nalezené položky se ukládají do seznamu *foundItems*. Jsou definovány proměnné *searchTextCount* pro počet výskytů a *currentFoundIndex* popisující index aktuálně označeného výskytu; výchozí hodnota je -1. Logika je implementována v metodě *findButton_Click*, která se spustí po kliknutí na tlačítko *findButton*. Před kontrolou textových polí se vynuluje počet nalezených výskytů a index aktuálního řádku, ve kterém se nachází hledaný bajt nebo znak, pokračuje se kontrolou, zda se jedná o hledání adresy, bajtu nebo znaku. Během hledání se všechny nalezené řádky vkládají do seznamu. Pokud není seznam po dokončení hledání prázdný, nastaví se index aktuálního výskytu na nulu, zobrazí se počet výskytů a pomocí metody *SelectFoundItem* se označí první buňka obsahující hledaný výraz a tabulka se posune na odpovídající řádek Pokud je nalezených prvků více než jeden, jsou aktivována tlačítka pro přepínání mezi nimi.

### 6.9.1 Adresa

Pro hledání konkrétní adresy ve souboru se nejprve ověřuje, zda je textové pole pro adresu vyplněno a současně je prázdný vstup pro hledaný řetězec. Jestliže je tato podmínka splněna, adresa určená k hledání se získává ze vstupu a ukládá se do proměnné *searchTerm*. Poté pomocí dotazu LINQ se vyhledává v položkách DataGridu řádek, kde se hledaný termín vyskytuje. Pokud není zaškrtnuta volba „Case insensitive“, hledání probíhá bez ohledu na velikost písmen pomocí metody *IndexOf* s parametrem *StringComparison.OrdinalIgnoreCase*, jinak se hledá přesná shoda pomocí metody *Contains*. Výsledné položky jsou ukládány do seznamu *foundItems*. Pokud není seznam prázdný, první nalezená adresa v tabulce se označí a tabulka se na ni přesune v metodě *SelectFoundItem*.

Asynchronní metoda *SelectFoundItem* přijímá povinný číselný parametr pro index hledané položky v seznamu a volitelný parametr pro aktuální znak v ASCII. V těle funkce se do proměnné *selectedItem* načte hledaný prvek podle přijatého indexu, zruší se výběr všech buněk a pomocí metody *ScrollIntoView(selectedItem)* se tabulka přesune k tomuto prvku. Pro správné zpracování přesunu a zobrazení je přidáno asynchronní čekání 100 milisekund, po kterém se vybere první sloupec vyhledaného řádku, který reprezentuje adresu.

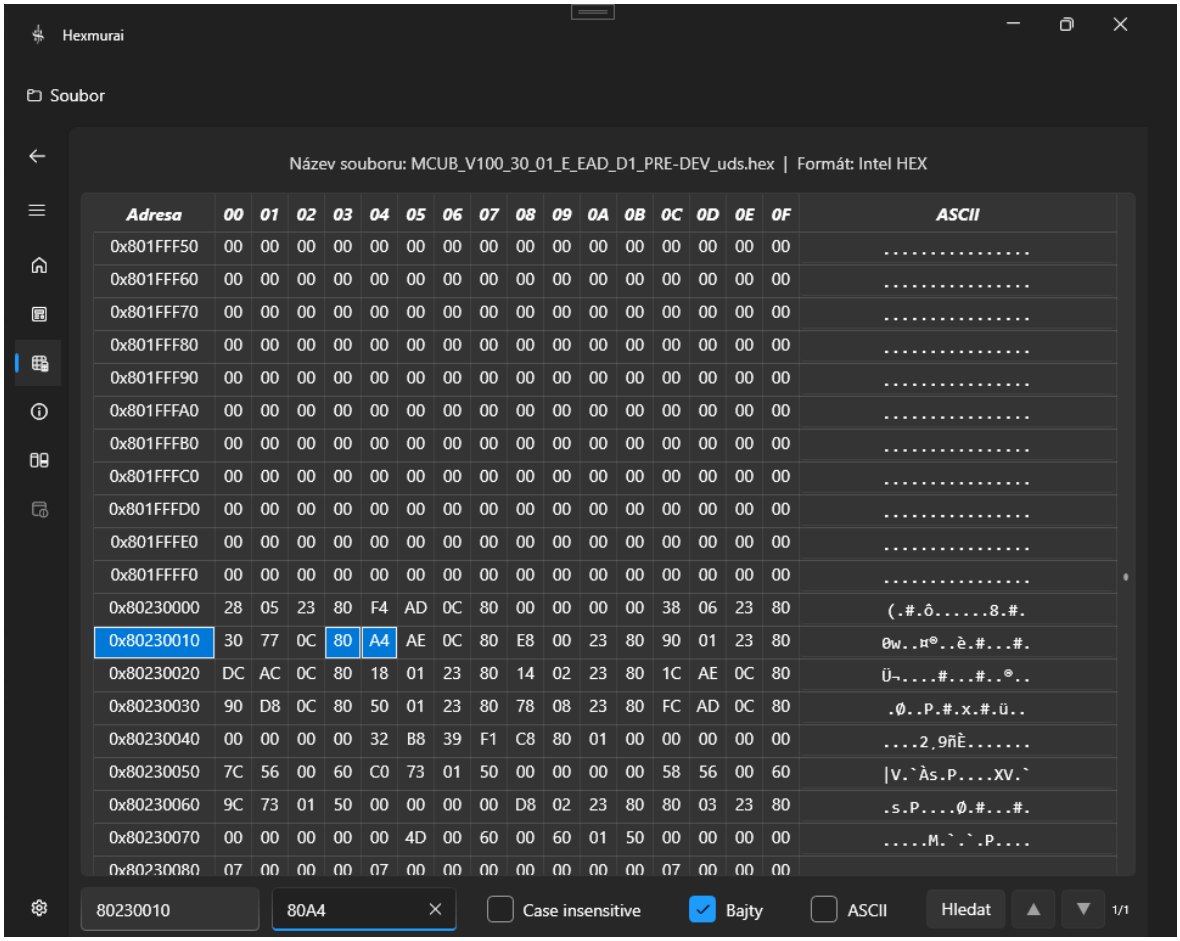
Přechod mezi nalezenými adresami je implementován v metodách *SelectNextCellOrRow* a *SelectPreviousCellOrRow*, které jsou volány po stisku tlačítka pro posun nahoru nebo dolů. Stejně jako v dalších metodách se zde provádí kontrola vstupů, u adresy se načte aktuální číslo výskytu a inkrementuje nebo dekrementuje se index aktuálně nalezeného řádku. Pokud je tento index větší než celkový počet hledaných výrazů nebo nulový, nastaví se mu hodnota na číslo řádku s prvním nebo posledním výskytem. V opačném případě se zavolá metoda *SelectFoundItem* se změněným indexem řádku.




Obrázek 36 LINQ kód pro hledání adresy

### 6.9.2 Datové bajty

Další možností vyhledávání jsou datové bajty, které mohou být buď hledány jako jeden pár nebo sekvence číslic v hexadecimálním formátu, přičemž zadávaný vstup musí obsahovat sudý počet znaků a pouze hexadecimální číslice bez mezer. Před samotným vyhledáváním je provedena kontrola, zda vstup splňuje tato pravidla; pokud ne, vyhledávání se neuskuteční. Pro zadání hledaných dat slouží druhé textové pole zleva, a je třeba zvolit zaškrtávací tlačítko s nápisem „Bajty“ pro vyhledávání v příslušných sloupcích.




Obrázek 37 Označení nalezených bajtů

Pokud je zadán platný řetězec dat, ve funkci *findButton_Click* se kontroluje, zda se má hledat na konkrétní adrese. V případě potvrzení se řádky pro danou adresu vyfiltrují, a následně se procházejí a rozlišuje se, zda se jedná o dvouznakový výraz nebo sekvenci, a zda se má při hledání ignorovat velikost písmen. U sekvence dat se jednotlivé bajty sloučí do jednoho řetězce, jejichž bajty jsou odděleny mezerou po dvou znacích. Následně se zjišťuje, zda se v tomto řetězci složeném z bajtů vyskytuje hledaný výraz, a pokud ano, přidá se do seznamu *foundItems*. Regulárním výrazem se zjišťuje, kolikrát se hledaný termín v ce-

lém řádku vyskytuje, a tato suma těchto počtů se zobrazí uživateli k informaci o počtu nalezených výskytů. Při hledání pouze jednoho bajtu se prochází sloupec po sloupci, a po zjištění počtu výskytů se z řádků složí datový řetězec s využitím regulárního výrazu.

```
StringBuilder combinedData = new StringBuilder();
combinedData.Append(item.Data00).Append(" ")
.Append(item.Data01).Append(" ")
.Append(item.Data02).Append(" ")
.Append(item.Data03).Append(" ")
.Append(item.Data04).Append(" ")
.Append(item.Data05).Append(" ")
.Append(item.Data06).Append(" ")
.Append(item.Data07).Append(" ")
.Append(item.Data08).Append(" ")
.Append(item.Data09).Append(" ")
.Append(item.Data0A).Append(" ")
.Append(item.Data0B).Append(" ")
.Append(item.Data0C).Append(" ")
.Append(item.Data0D).Append(" ")
.Append(item.Data0E).Append(" ")
.Append(item.Data0F);
string formattedSearchTerm = string.Join(" ", Enumerable.Range(0, searchTerm.Length / 2).Select(i => searchTerm.Substring(i * 2, 2)));
string combinedString = combinedData.ToString();
int count = Regex.Matches(combinedString, Regex.Escape(formattedSearchTerm)).Count;
searchTextCount += count;
if (combinedString.IndexOf(formattedSearchTerm) >= 0)
{
    foundItems.Add(item);
}
```

Obrázek 38 Kód pro vyhledávání sekvence bajtů

Při zvýraznění jednotlivých buněk v tabulce ve funkci *SelectFoundItem* se nejprve přetransformuje text na velká písmena a opět se zjišťuje délka hledaného výrazu. Sekvence o délce delší než 2 znaky se rozdělí do seznamu po párech, získají se všechny bajty v odpovídajícím řádku, ze kterých se vytvoří datový řetězec. Následně se v tomto řetězci hledá první výskyt hledané sekvence. Cílem cyklu je získat startovací index, který odpovídá prvnímu sloupci, jehož buňka se označí, a poté po vypočtení konečného indexu se vyberou buňky, které se nacházejí mezi těmito indexy. V případě zadání adresy se vybere první sloupec. Buňka jednoho bajtu se označí průchodem celého řádku, a získání hodnot mezi adresou a posledním sloupcem se porovnává požadovaný pár s tím aktuálním; v případě shody se označí buňka podle indexu právě iterovaného sloupce.

Při přepínání mezi nalezenými bajty ve funkcích *SelectNextCellOrRow* a *SelectPreviousCellOrRow* se získá aktuální řádek a následuje stejný postup jako v předchozím odstavci, ale pokud se další bajt či sekvence v řádku nenachází, vyhledávání pokračuje v dalším řádku odpovídajícího směru přepínání. Tento proces probíhá, dokud není nalezen další výskyt.

### 6.9.3 ASCII

Pro vyhledání řetězce v ASCII sloupci je nutné zvolit příslušný režim hledání na spodní liště a do vstupu napsat hledaný termín. Jelikož je možné kombinovat hledání řetězce na konkrétní adrese, kontroluje se, zda je vyplněn vstup pro adresu a v případě vyplnění je zahrnut do hledání v kódu, který je identický s tím pro hledání adresy, ale rozšířen o vlastnost modelu reprezentující ASCII řetězec pro daný řádek tabulky. Pro zjištění počtu výskytů hledaného termínu v řetězci se používá regulární výraz. Pokud jsou nalezené výskyty termínu v řetězci, probíhají stejné operace jako u jiných sloupců, ale navíc se získává celkový počet nalezených výskytů v celém řádku. Tento počet se poté využije při přepínání na další nebo předchozí výskyt hledaného termínu.

Přepínání mezi jednotlivými výskyty funguje tak, že se aktualizuje aktuální index výskytu termínu v řetězci. Pokud další výskyt termínu není nalezen, aplikace volá metodu pro zvýraznění hledaného prvku, přičemž přebírá volitelný parametr pro číslo aktuálního řádku a aktuálního výskytu termínu. Jestliže další výskyt v řádku není nalezen, index aktuálního řádku se zvětší nebo zmenší na základě směru přepínání, a následuje opětovný výpočet počtu výskytů termínu v řádku. Poté se nastaví index aktuálního výskytu na začátek nebo konec řádku.

V metodě *SelectFoundItem* je zvýrazněn konkrétní znak v textu. To se provede získáním hodnoty z buňky posledního sloupce daného záznamu a následně nalezením prvku TextBox, který umožňuje zvýraznění části textu. Jestliže je tento prvek s názvem *tbASCII* v buňce nalezeno, volá se metoda *IndexOfOccurence* s parametry pro text řádku, hledaný termín a aktuální index výskytu. Tato metoda vrátí startovací index dalšího nebo předchozího výskytu termínu, který se nastaví TextBoxu jako začátek výběru, spolu s délkou odpovídající hledanému termínu.

```
foundItems = dataGrid.Items.OfType<HexData>().Where(item =>
{
    if (CaseCheckBox.IsChecked == true)
    {
        return item.Address.IndexOf(address, StringComparison.OrdinalIgnoreCase) >= 0 && item.ASCII.IndexOf(searchTerm, StringComparison.OrdinalIgnoreCase) >= 0;
    }
    else
    {
        return item.Address.Contains(address) && item.ASCII.Contains(searchTerm);
    }
}).ToList();
```

Obrázek 39 LINQ kód pro hledání řetězce ASCII

## 6.10 Generování reportu

Jakmile je dokončeno porovnání souboru a jsou zvýrazněny bajty, je povolena v horním menu možnost Report, která obsahuje tři volby generování reportu, a to ve formě souhrnu a zobrazení pouze stejných či odlišných bajtů. Po vybrání volby se zobrazí dialogové okno pro výběr adresáře, kam se mají výsledné dokumenty uložit. Pro generování PDF dokumentu byla zvolena knihovna QuestPDF a to především z důvodu, že během vývoje bylo vyzkoušeno více knihoven, ale většina z nich měla problém vytvořit dokumenty s velkým počtem stránek v řádu tisíců a nesplňovaly požadavky na vizuální zobrazení tabulek. Z toho důvodu byla použita tato knihovna s rozdělením dokumentů na více částí včetně rozdělení seznamu pro určení shodných a odlišných bajtů na menší seznamy o velikosti 10 tisíc položek.

```
1 public static IEnumerable<List<T>> SplitIntoChunks<T>(this List<T> source, int chunkSize)
2 {
3     for (int i = 0; i < source.Count; i += chunkSize)
4     {
5         yield return source.GetRange(i, Math.Min(chunkSize, source.Count - i));
6     }
7 }
```

Obrázek 40 Statická metoda pro rozdělení seznamu na menší části

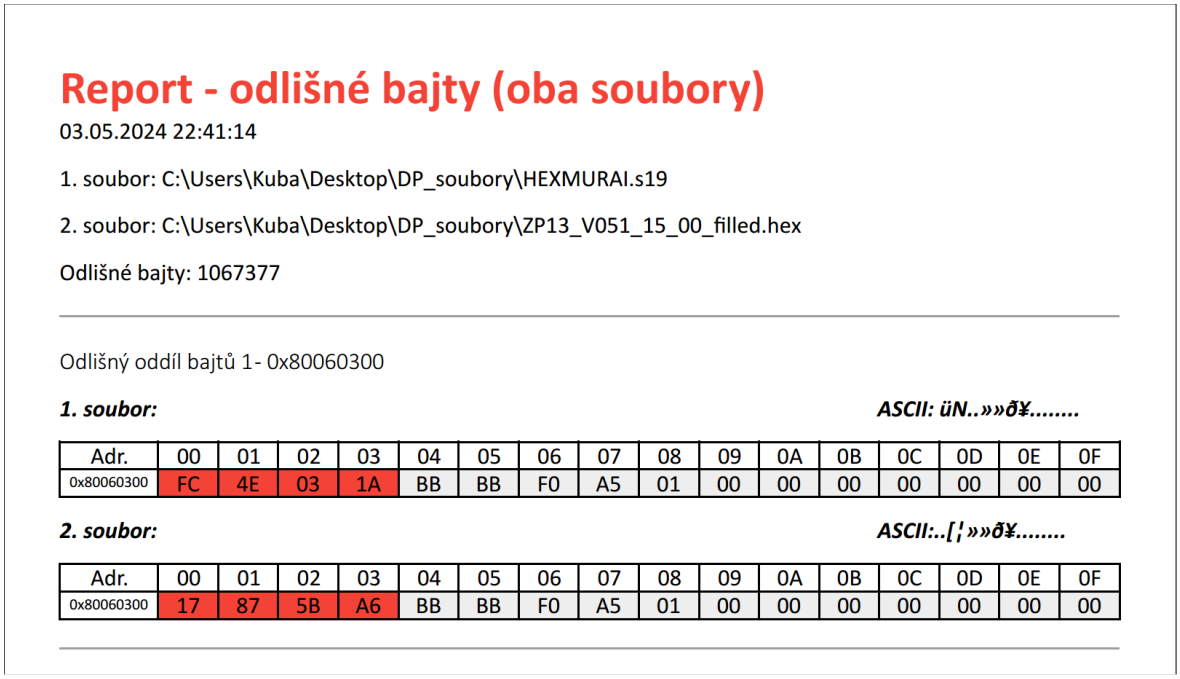
U všech tří typů reportů se vytvoří ve vybraném adresáři složka s názvem „Report“ a aktuální časové razítko pro přehlednější identifikaci, v této složce jsou pak uloženy výsledné dokumenty. Během procesu tvorby souborů je uživatel o aktuálním informován v pravém horním rohu ve žlutém informačním pruhu, kde je zobrazen číslo aktuálně zpracovaného dokumentu spolu s celkovým počtem. Aby bylo možné zobrazit v tabulce v reportu jednotlivé bajty, má model *HexData* boolenovské pole *IsSameReport* a *IsDifferentReport*, protože v případě užití polí *IsSame* a *IsDifferent* by došlo ke změně režimu porovnání a zvýraznění odpovídajících sloupců.

Prvním typem je souhrn, který obsahuje nadpis, cesty obou souborů a počty jak stejných, tak odlišných bajtů. V asynchronní metodě *generateReport_Click* pro zachycení stisku tohoto tlačítka opět dochází k zavolání metod *HighlightSame* a *HighlightDifferences*, v tomto případě se ale zapisuje do výše zmíněných polí a zjišťuje se celkový počet kladných hodnot v obou polích. Během testování bylo zjištěno, že průchod seznam o délce přibližně

200 tisíc položek není optimální a trvá velice dlouho, bylo nutné celý algoritmus optimalizovat, a to použitím *HashSetů* a slovníků, a to i přes to, že se u souhrnu pouze vypočítávala suma hodnot.

Následujícím typem reportu je zobrazení pouze shodných bajtů na adresách, které se vyskytovaly v obou souborech. Tento dokument obsahuje nadpis, cesty k oběma souborům, počet shodných bajtů, ohraničená sekce s textem s aktuální adresou, přehledně zobrazený výpis dvou řádků na stejných adresách v tabulce, která má totožnou hlavičku, jaká je použita v aplikaci, se zeleně označenými shodnými bajty, které mají vzhled definovány ve statické metodě *BlockGreen*, jenž se volá pro pozadí buňky tabulky a u každého řádku je zobrazen příslušný ASCII řetězec. Kód je podobný jako u souhrnu, tentokrát se ale volala funkce *HighlightSame*, po jejím skončení se do *Hashsetu* uložily jako řetězce společné adresy a následně vyfiltrovaly pouze ty řádky na základě shodných adres. Pro zlepšení výkonu je filtrování prováděno paralelně a použit přístup popsán výše v podobě rozdělení seznamu označených dat na menší části a procházení seznamu za použití *CollectionsMarshal.AsSpan*, které se ukázalo jako nejrychlejší možnost jako iterovat v tak velkém seznamu. Po každém průchodu 10 tisíců položek se inkrementuje číslo aktuálně zpracovávaného reportu, které je použito v názvu souboru a taktéž se aktualizuje na stránce. Po zpracování posledního dokumentu se zobrazí dialogové okno se zprávou, že je generování dokončeno.

Vytvoření reportu pouze odlišných bajtů se liší od předchozího typu tím, se neporovnává jen na stejných adresách, ale taktéž se jako rozdíl označuje celý řádek, jehož adresa se nevyskytuje v obou souborech. V metodě *generateReportDiff_Click* se zpracovávají jednak společné adresy, tak adresy vyskytující pouze v daných souborech. Kvůli zmenšení paměťové náročnosti byly jednotlivé seznamy *filteredData1*, *dataOnlyInHexData1* a *dataOnlyInHexData2* rozděleny na části po 10 tisíc položek a postupně se procházely. Nejprve se tvoří reporty pro společné adresy, které obsahují stejné položky jako u shodných bajtů, akorát je zobrazen počet odlišných bajtů. Po úspěšném vytvoření všech dokumentů se následně generují dokumenty s adresami, které obsahuje pouze první soubor a následně druhý soubor, pokud existují.



Report - odlišné bajty (oba soubory)

03.05.2024 22:41:14

1. soubor: C:\Users\Kuba\Desktop\DP_soubory\HEXMURAI.s19

2. soubor: C:\Users\Kuba\Desktop\DP_soubory\ZP13_V051_15_00_filled.hex

Odlišné bajty: 1067377

Odlišný oddíl bajtů 1- 0x80060300

***1. soubor:*** ***ASCII: üN..»»ð¥.....***

| Adr. | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x80060300 | FC | 4E | 03 | 1A | BB | BB | F0 | A5 | 01 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

***2. soubor:*** ***ASCII:..[¦»»ð¥.....***

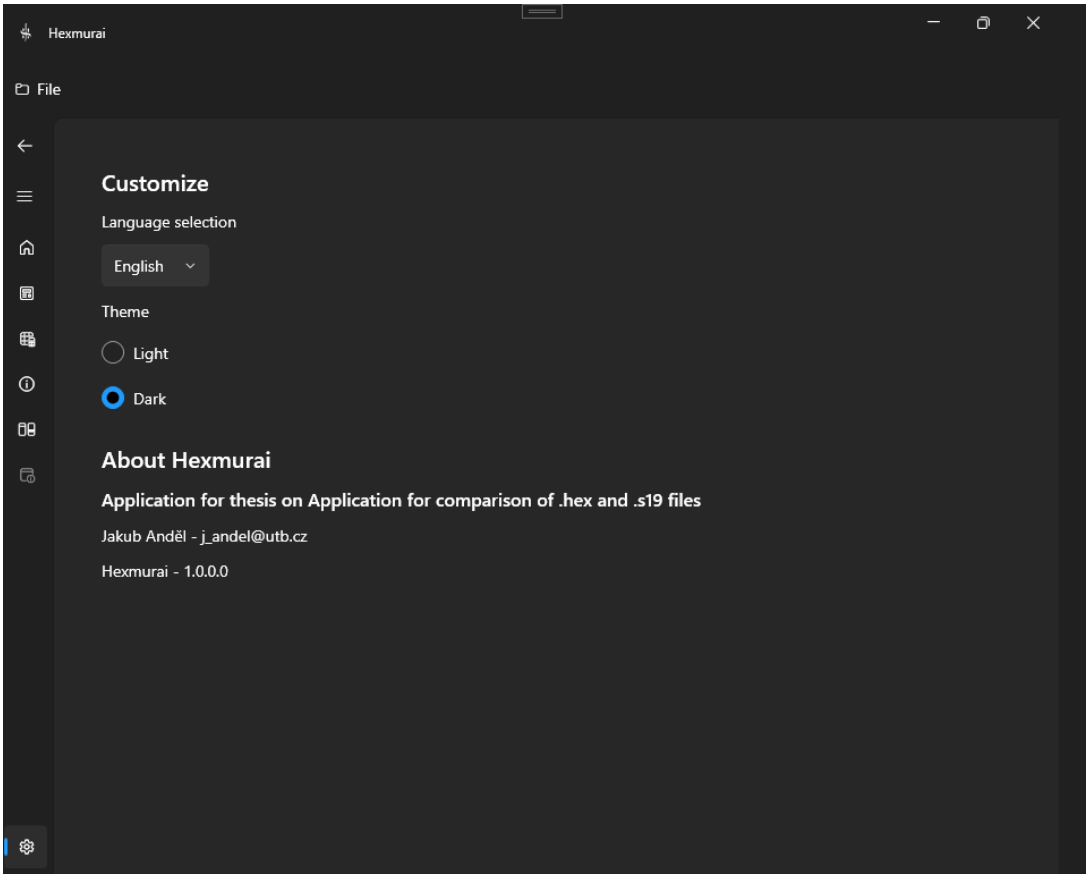
| Adr. | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x80060300 | 17 | 87 | 5B | A6 | BB | BB | F0 | A5 | 01 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

Obrázek 41 Report zobrazující odlišné bajty

## 6.11 Možnosti v nastavení

Stránka pro nastavení aplikace (*SettingsPage*) vychází z ukázkového programu frameworku WPF UI, který umožňuje uživatelům měnit barevný režim, popis aplikace s a uvést aktuální verzi programu. Tuto stránku jsem rozšířil o možnost změny jazyka pro celou aplikaci.

Při přechodu na tuto stránku se v konstruktoru načítá JSON soubor umístěný v adresáři Dokumenty, který obsahuje aktuální volby pro režim a jazyk. Režim se nastaví pomocí třídy *ApplicationThemeManager* z WPF UI, která očekává *enum* s možností tmavého nebo světlého režimu. Po jeho změně se režim aktualizuje pro celou aplikaci a následně se řetězec pro jazyk uloží do JSON souboru. Jazyk má také svou hodnotu v tomto souboru ("cz", "en", "de"), podle které se určuje, který XAML soubor s texty v daném jazyce se vybere. Tento soubor je poté přidán do aktuálních zdrojů (Resources) aplikace a nahrazuje ten původní. Vybraný jazyk se uloží do souboru JSONu, aby aplikace pracovala v daném jazyce i po opětovném spuštění. Poté se vyvolá událost *LanguageChanged*, což upozorní ostatní části aplikace, že se jazyk změnil, a nahradí všechny texty těmi z XAML souboru. Změna jazyka se projeví i při zobrazení dialogových oken a ve vygenerovaných reportech.




Obrázek 42 Stránka nastavení v anglickém jazyce

# 7 ZHODNOCENÍ A TESTOVÁNÍ

Pro účely testování jsem obdržel čtyři hexadecimální soubory od vedoucího mé diplomové práce. Dva z nich byly ve formátu Intel HEX a zbývající dva ve formátu Motorola SRecord. Kvůli jejich větší velikosti jsem musel brát v potaz také výkon aplikace při čtení a zobrazení těchto souborů, aby byla schopná s nimi pracovat bez komplikací. Pro kontrolu výsledků zobrazení jsem použil webovou aplikaci HexEd.it, která umožňuje zobrazit hexadecimální výstup Intel HEX včetně neuspořádaných záznamů a záznamů s rozšířenými adresami. Během rešerše hexadecimálních editorů jsem zjistil, že velká většina z nich nenabízí možnost zobrazit hexadecimální výpis, ale pouze obsah souboru, proto jsem musel spoléhat na výsledky z této webové aplikace.

Výsledky z této stránky byly velmi užitečné, protože mi pomohly odhalit chyby ve zobrazení obsahu souborů. Tyto chyby se objevily při otevření jiných hexadecimálních souborů z internetu, které obsahovaly zkrácené řádky s nenulovými adresami. To vedlo k vytvoření duplicitních řádků nebo zobrazení dat na nesprávných adresách. Na základě toho se mi podařilo opravit tyto chyby a aplikace korektně zobrazí obsah ve formátech Intel HEX a Motorola SRecord.

Pro ověření funkcionality vyhledávání a generování reportů byly použity taktéž soubory dostupné na internetu kvůli jejich velikosti a rozsahu tabulek. Jakmile aplikace úspěšně vyhledávala různé hledané řetězce a správně zobrazovala počet jejich výskytů v souboru a generovala reporty ve formátu PDF, byly testovány i poskytnuté soubory.

Během testování bylo zjištěno, že aplikace bez problémů otevře běžně velké hexadecimální soubory, zobrazí posledních 5 otevřených souborů, přečte a zobrazí jejich obsah a hexadecimální výpis, umožní přechod na konkrétní adresu, vyhledá bajt nebo sekvenci bajtů, případně řetězec v ASCII, umožní úpravu bajtů ve výpisu, uloží do nového souboru, zobrazí informace o souboru, porovná dva běžně velké hexadecimální soubory, zobrazí shodné a odlišné bajty, umožní vyhledávání v obou souborech najednou, hledání dalších výskytů, synchronizaci pohybu v tabulkách během porovnání, generování reportů a zobrazení informací o porovnávaných souborech. Dále umožní dynamickou změnu barevného režimu a jazyka.

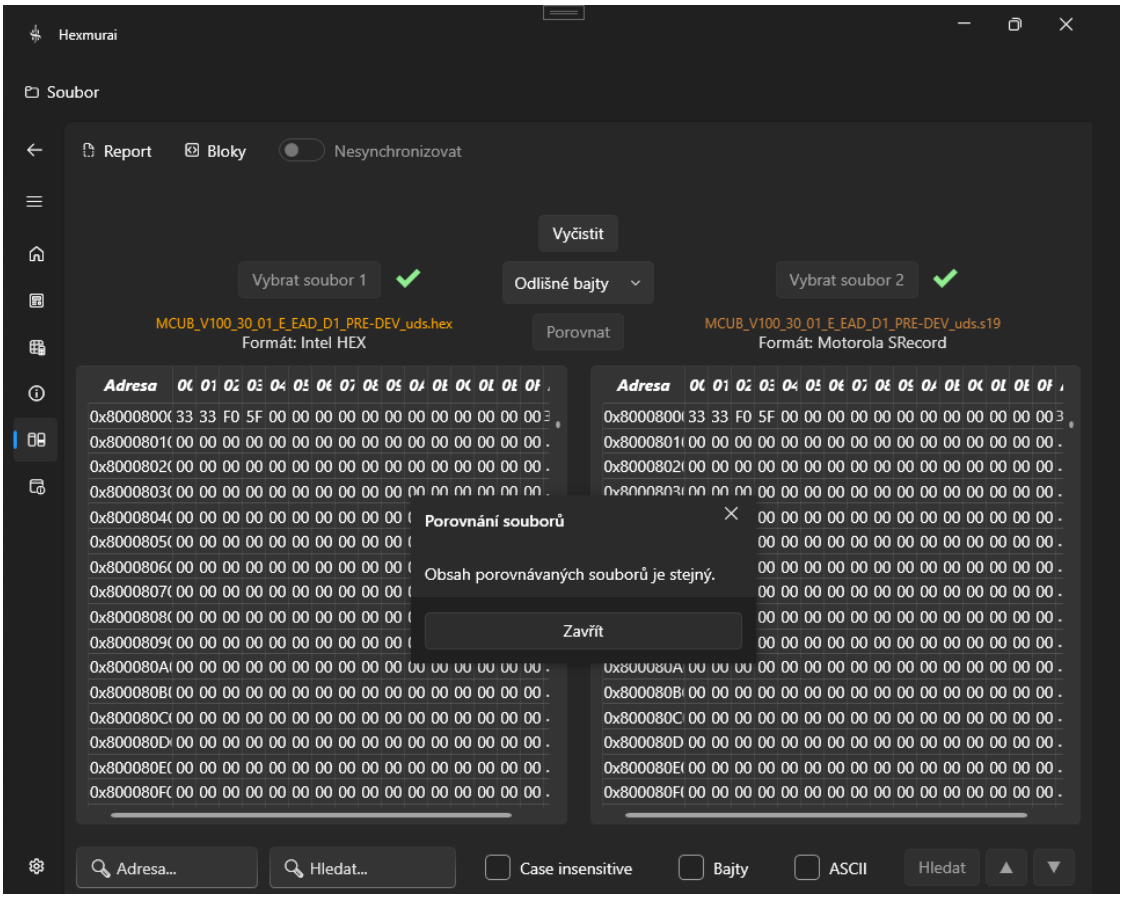
Problémem pro aplikaci mohou představovat výrazně větší soubory (5 a více MB), které obsahují velké množství řádků až v řadech statisíců, které vyžadují velký výpočetní čas při procházení, a taktéž se projeví zvýšená paměťová zátěž při generování reportů v PDF. Po-

skytnuté soubory mají velikost přes 7 MB a aplikace s nimi umí pracovat, veškeré operace se dějí v řádech jednotek až desítek sekund, až na generování reportu, které trvá u větších souborů v řádech minut až hodin v závislosti na počtu shodných a rozdílných bajtů. Řešení v podobě rozdělení seznamu na menší části o velikosti 10 tisíc se ukázalo jako uspokojující, jelikož při procházení celého seznamu se zahltila celá paměť počítače a program přestal pracovat.

Aplikace má také omezení spojená s menšími obrazovkami. Na nižším rozlišení je obtížné zobrazit vedle sebe dvě tabulky s 18 sloupci, aby bylo možné kompletně přečíst sloupec ASCII. Částečným řešením tohoto problému je sbalení bočního menu, avšak toto řešení nemusí spolehlivě fungovat na velmi nízkých rozlišeních.

## 7.1 Stejné soubory v jiném formátu

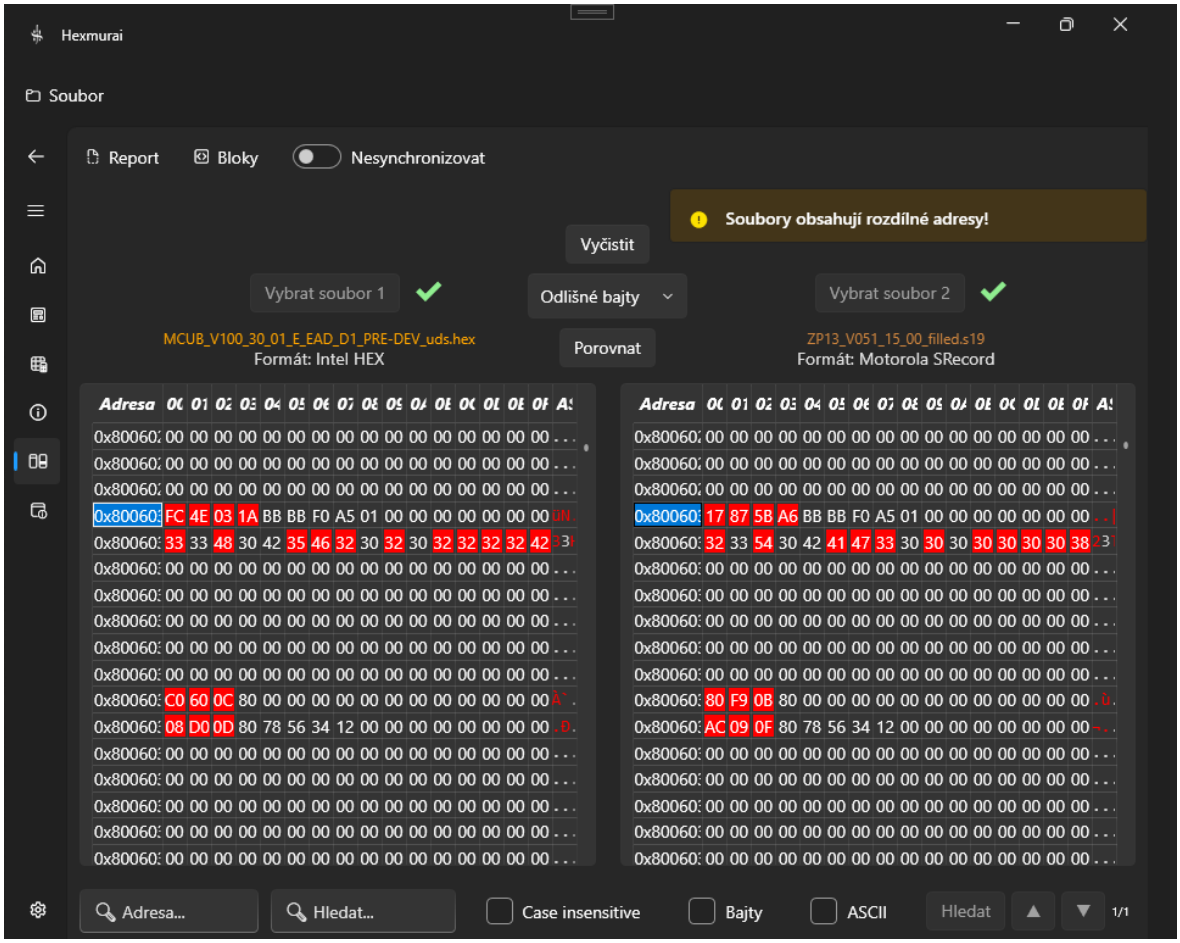
Pokud porovnáváme dva soubory se stejným obsahem, tak po načtení do tabulek se zobrazí vyskakovací okno se zprávou, že obsah porovnávaných souborů je stejný a žádný bajt se barevně nezvýrazní.




Obrázek 43 Porovnání stejných souborů v jiném formátu

## 7.2 Odlišné soubory v jiném formátu

V případě odlišných obsahů se buňky, které se liší na stejných adresách v obou souborech, zvýrazní červeně. Dále se v první tabulce zvýrazní oranžově buňky obsahující data, která se nevyskytují v druhé tabulce, a naopak, taková data v druhé tabulce se zvýrazní hnědě. Aplikace také upozorní uživatele v pravém horním rohu stránky na informaci, že oba soubory buď obsahují adresy, které se nevyskytují v obou souborech, což znamená, že se soubory liší nejen v hodnotách, ale i v adresách, nebo se úplně liší v adresách a každý bajt je tak odlišný.




Obrázek 44 Porovnání odlišných souborů v jiném formátu

# 8 DALŠÍ MOŽNÁ ROZŠÍŘENÍ

I přesto, že aplikace dokáže správně zobrazovat obsah hexadecimálních souborů bez ohledu na jejich formátování a typ záznamů, umožňuje jejich vzájemné porovnání nebo úpravu jednotlivých bajtů, stále existuje prostor pro vylepšení současných funkcí a úpravu uživatelského rozhraní. Nové funkcionality by mohly zvýšit uživatelský komfort při práci s tímto typem souborů.

Kromě implementace nových funkcí je důležité věnovat také pozornost i výkonnostní stránce aplikace, zejména pokud jde o využití paměti počítače. Cílem je zajistit, aby aplikace byla schopna rychle zpracovávat hexadecimální soubory libovolné délky bez zbytečného čekání. I když se během vývoje podařilo odstranit některá omezení spojená s velkým množstvím položek v seznamu, stále může dojít k delším časům načítání obsahu souboru a generování reportů. To může negativně ovlivnit celkovou uživatelskou zkušenost.

## 8.1 Rozšíření stávajících funkcí

Rozšíření stávajících funkcí může výrazně přispět ke zlepšení uživatelské zkušenosti a efektivity aplikace. Jednou z navrhovaných inovací je implementace možnosti editovat buňky přímo v tabulkách na stránkách určených pro zobrazení obsahu a pro porovnání dvou souborů. Pro komplexní úpravu souborů by bylo vhodné také možnost přidávání a mazání jednotlivých buněk v tabulce či celých řádků, čímž uživatel bude mít možnost manipulovat s daty bez omezení. Tato rozšíření by měla být prováděna tak, aby se změny automaticky synchronizovaly napříč ostatními stránkami aplikace, což přispěje k plynulé a efektivní práci uživatelů.

Další možností rozšíření je přidání možnosti vizualizace jak shodných, tak odlišných bajtů při porovnávání souborů a generování reportu s tímto zobrazením. Tento krok má za cíl optimalizovat proces porovnávání souborů tím, že umožní uživatelům získat obě informace během jednoho průchodu, místo nutnosti spouštět porovnání dvakrát.

Další potenciální vylepšení spočívá v implementaci funkce, která umožní uživatelům při výběru buňky v tabulce zvýraznit příslušný znak ve sloupci ASCII, který odpovídá vybranému datovému bajtu. Tato funkce zvýší uživatelskou přehlednost a usnadní identifikaci konkrétních znaků.

Rozšíření stránky s podrobnými informacemi o použitých typech záznamů a analýze obsahu, včetně entropie a zastoupení bajtů, nabídne uživatelům hlubší vhled do struktury a ob-

sahu hexadecimálních souborů. Tato rozšíření budou užitečná pro uživatele, kteří se zajímají o podrobnější analýzu a interpretaci dat.

## 8.2 Přidání nových funkcí

Pokud by se změnil aktuální přístup k otevírání souborů na použití záložek, umožnilo by to rychlejší přepínání mezi otevřenými soubory a urychlilo by to práci při modifikaci více souborů současně, což je efektivnější než otevírat soubory postupně.

Další užitečnou funkcionalitou by mohlo být začlenění ověření kontrolního součtu záznamu. Tato funkce by umožnila uživatelům kontrolovat integritu datového záznamu pomocí kontrolního součtu. Uživatelé by do textového pole na stránce zadali záznam, který by se poté přepočítal kontrolním součtem podle formátu na základě úvodního znaku a porovnal by se s aktuálním. V případě shody by se zobrazilo dialogové okno s informací, že je záznam validní. V opačném případě by aplikace signalizovala, že kontrolní součet neodpovídá, a poskytla by správný kontrolní součet.

Další potenciální funkcí, která by mohla být implementována, je možnost převodu mezi formáty .hex a .s19. Stránka pro tuto funkci by obsahovala tlačítko pro výběr souboru a tlačítko pro potvrzení konverze do opačného formátu. Při provedení konverze by z každého řádku v souboru vznikl nový záznam podle daného formátu. Výzvou této funkce by bylo zajistit, aby obsah obou souborů byl po konverzi identický.

Jelikož aplikace primárně slouží pro práci s hexadecimálními soubory, mohlo by být vhodné implementovat kalkulačku pro převod z hexadecimální soustavy do dalších a pro různé aritmetické operace s čísly této soustavy, jako je sčítání, odčítání, násobení a dělení. Uživatelé by také mohli využít převodník na ASCII, aby mohli při editaci vkládat do buněk správná data. Tato funkce by zvýšila flexibilitu uživatelů při práci s hexadecimálními daty a usnadnila by manipulaci s čísly této soustavy.

# 9 SROVNÁNÍ S EXISTUJÍCÍMI PROGRAMY

Tato kapitola se zaměřuje na podrobné srovnání vytvořené aplikace Hexmurai s několika stávajícími nástroji určenými pro porovnávání hexadecimálních souborů, konkrétně Beyond Compare, Hexinator a ImHex. Cílem této kapitoly je zhodnotit funkčnost a výkonnost aplikace ve srovnání s těmito osvědčenými nástroji, přičemž se zaměříme na několik klíčových aspektů. Poskytnuté soubory budou použity jako testovací subjekty.

Prvním bodem srovnání bude obsah souboru, kde bude porovnávána schopnost každého nástroje zobrazit a analyzovat obsah hexadecimálního souboru. Dále se podíváme na hexadecimální výstup, tj. způsob, jakým každý nástroj prezentuje hexadecimální data uživateli a umožňuje jim provádět manipulace a úpravy.

Další bod bude věnováno zhodnocení funkcí, jaké další možnosti a nástroje každá aplikace nabízí nad rámec základního porovnání souborů. Toto srovnání pomůže lépe pochopit, jak naše aplikace stojí v konkurenci, která již je zavedena na trhu a kde je možné případně zlepšit její výkon a uživatelskou přívětivost.

## 9.1 Obsah souboru

V programu Beyond Compare verze 4 se vybraný soubor načetl během 0,05 sekundy. Obsah souboru je prezentován v tabulce bez hlavičky, zahrnující adresu a ASCII hodnoty. Nad tabulkou je uvedeno datum poslední modifikace a velikost souboru v bajtech. Vyhledávání je možné pouze na úrovni bajtů, a to buď jednotlivých, nebo sekvencí. Aplikace nepodporuje přechod na konkrétní adresu ani vyhledávání v ASCII, ale umožňuje editaci jednotlivých bajtů včetně jejich mazání. Porovnání mezi soubory umožňuje volbu mezi režimy zobrazení shodných a odlišných bajtů buď současně, nebo odděleně. V režimu odlišných bajtů jsou rozdíly zvýrazněny červeně a mezi nimi lze přecházet pomocí tlačítek v menu. Beyond Compare nabízí možnost pro generování reportu pro obsah souboru včetně možnosti nastavení, jak má report vypadat a co má obsahovat, ale při generování reportu u těch soubor program přestal program pracovat.

Hexinator načetl a porovnal obsahy obou souborů bajt po bajtu za zhruba 1,1 sekundy. Na rozdíl od Beyond Compare neumožňuje volbu mezi zobrazením shodných a odlišných bajtů. Hexinator zvýrazňuje shody zeleně a rozdíly červeně. Na dolní liště zobrazuje počet shodných a odlišných bajtů spolu s počtem přidaných a smazaných řádků, což je funkce,

která je k dispozici pouze v placené verzi. Program neumožňuje vyhledávání v obsahu ani editaci bajtů.

ImHex při porovnání obsahu zvýrazňuje rozdíly, ale nepodporuje volbu mezi zobrazením shodných a odlišných bajtů. Umožňuje editaci jednotlivých bajtů, ale chybí funkce pro vyhledávání v obsahu při porovnání nebo přechod na konkrétní adresu. Pod tabulkou zobrazuje adresní rozsah obsahu a další podrobné informace o souboru a statistiky jsou dostupné z horního menu.

Aplikace Hexmurai, na rozdíl od ostatních programů, nenabízí možnost porovnání obsahů dvou souborů. Zobrazuje obsah jednoho souboru bez možnosti editace, ale umožňuje vyhledávání bajtů nebo ASCII řetězců a přechod na konkrétní adresu. Jednotlivé informace a statistiky o souboru jsou zobrazeny na stránce „Informace o souboru“.

## 9.2 Hexadecimální výstup

Ani jeden z existujících programů, s výjimkou Hexmurai, nepodporuje hexadecimální výstup souborů ve formátech Intel HEX a Motorola SRecord.

Na stránce určené pro hexadecimální výpis je k dispozici funkce editace jednotlivých bajtů, včetně možnosti uložení provedených změn. Dále je zde implementováno vyhledávání adresy, bajtů a řetězců v ASCII formátu. Informace o souboru, jako jsou podrobnosti o datových blocích a celková délka těchto bloků, jsou k nalezení na stránce s názvem „Informace o souboru“.

Kromě toho je možné provést porovnání dvou hexadecimálních výstupů, s možností volby zobrazení shodných nebo odlišných bajtů. U porovnání lze taktéž vyhledávat a přechod na konkrétní adresu, a také generování reportu o shodných a odlišných bajtech ve formátu PDF.

| | Beyond Compare 4 | Hexinator | ImHex | Hexmurai |
|---|---|---|---|---|
| **Zobrazení obsahu** | Ano | Ano | Ano | Ano |
| **Editace obsahu** | Ano | Ne | Ano | Ne |
| **Vyhledávání** | Ano (pouze bajty) | Ne | Ano (bajty a adresa u jednoho souboru) | Ano |
| **Informace o souboru** | Ne (datum poslední modifikace a velikost) | Ne | Ano | Ano |
| **Porovnání obsahu** | Ano | Ano | Ano | Ne |
| **Zobrazení shodných bajtů** | Ano | Ano | Ne | Ne |
| **Zobrazení odlišných bajtů** | Ano | Ano | Ano | Ne |
| **Hexadecimální výstup** | Ne | Ne | Ne | Ano |
| **Editace hex výpisu** | Ne | Ne | Ne | Ano |
| **Vyhledávání v hex výpisu** | Ne | Ne | Ne | Ano |
| **Generování reportu** | Ano | Ne | Ne | Ano |

Tabulka 2 Srovnání vytvořené aplikace s existujícími

# ZÁVĚR

Cílem této diplomové práce bylo navrhnout, vytvořit a otestovat desktopovou aplikaci pro porovnání a úpravu .s19 a .hex souborů a to včetně zobrazení informací o souboru, generování reportu a hledání v jejich obsahu.

Teoretická část se zaměřuje na popis formátů Intel HEX a S19, které má aplikace podporovat, včetně typů záznamů, procesu překladu do čitelného textu a výpočtu kontrolního součtu. Dále zahrnuje rešerši existujících aplikací pro porovnání souborů, kde jsou detailně popsány funkce vybraných nástrojů a následně porovnány v tabulce. Poslední kapitola se věnuje popisu technologií použitých při vývoji této aplikace.

Praktická část se zabývá návrhem a implementací desktopové aplikace. Při návrhu byly stanoveny funkční a nefunkční požadavky, které měla aplikace splňovat, a vytvořeny návrhy jednotlivých stránek, podle kterých byl vytvořen vzhled aplikace. Poté následuje podrobná dokumentace obsahující popis implementace aplikace včetně struktury souborů, modelů reprezentujících jednotlivé řádky tabulky a pomocných tříd pro manipulaci s datovými bloky záznamů. Dále jsou zde popsány implementované funkcionality spolu s ukázkami kódu. Aplikace byla vyvíjena v prostředí Microsoft Visual Studio 2022 s využitím platformy WPF .NET 8.0, programovacího jazyka C# a značkovacího jazyka XAML.

Další pasáží praktické části bylo zhodnocení a testování vytvořené aplikace, které bylo prováděno během celého vývoje. Byly mi poskytnuté testovací soubory, na kterých měla být aplikace otestována, ale kvůli absenci několika druhů záznamů byly využity hexadecimální soubory z internetu, které poskytovaly širší spektrum typů záznamů a jejich zápisů ve formátech Intel HEX a S19. To umožnilo doladit aplikaci pro všechny podporované varianty záznamů. Během testování byly zjištěny potíže po výkonové stránce u souborů vyšší velikosti projevované pády a zamrznutí aplikace, které ale se podařilo vyřešit změnou přístupu pro procházení seznamu s velkým počtem položek bez těchto projevů Z výsledků testování vyplývá, že aplikace je úspěšná při porovnávání souborů se stejným či odlišným obsahem v popsaných formátech. Nicméně, je důležité zdůraznit, že aplikace má několik estetických a technických omezení, která však neovlivňují vážně její funkčnost ani uživatelský zážitek.

V závěru praktické části byly navrhnuty další funkcionality k implementaci a možné rozšíření existujících funkcí v podobě přidání stránky pro ověřování kontrolního součtu záznamu a převodu mezi souborovými formátu, které by mohly zvýšit použitelnost vytvořené

aplikace. Dále byla aplikace pro porovnání a úpravu hexadecimálních souborů porovnána s existujícím nástroji na trhu a z výsledků srovnání je patrné, že nabízí podobné množství funkcí a v ohledu zobrazení hexadecimálního výpisu je dokonce předčí kvůli absenci podpory těchto nástrojů pro záznamy konkrétních souborových formátů.

Během vypracovávání jsem si prohloubil své znalosti s vývojem desktopových aplikací v jazyce C# a algoritmizování některých komplexních funkcí. Osvojil jsem si i práci s platformou WPF pro operační systém Windows a nabyl nové znalosti se značkovacím jazykem XAML. Z výsledku aplikace jsem spokojený a věřím, že ji uživatelé budou vnímat jako užitečnou.

# SEZNAM POUŽITÉ LITERATURY

[1] SRecord: Reference Manual. In: FINNERAN, Scott a Peter MILLER. *SRecord - Reference Manual* [online pdf]. 18 October 2022n. l. [cit. 2024-01-28]. Dostupné z: https://srecord.sourceforge.net/reference-1.65.pdf

[2] Motorola Sxx records format. *SB-Projects* [online]. c2001, Last updated: 31 December 2023 [cit. 2024-01-28]. Dostupné z: https://www.sbprojects.net/knowledge/fileformats/motorola.php

[3] Motorola S-record Format. *Lucid Technologies* [online]. c1996-2022 [cit. 2024-01-28]. Dostupné z: https://www.lucidtechnologies.info/moto.htm

[4] M68HC11EVB EVALUATION BOARD USER'S MANUAL. In: *M68HC11EVB.pdf* [online]. ©1986,1996 [cit. 2024-01-28]. Dostupné z: https://www.nxp.com/docs/en/user-guide/M68HC11EVB.pdf

[5] WONG, Jimmy. *Format of IntelHex* [online]. 2021 [cit. 2024-01-28]. Dostupné z: https://jimmywongiot.com/2021/04/20/format-of-intelhex/

[6] Intel HEX format. *SB-Projects* [online]. c2001, Last updated: 31 December 2023 [cit. 2024-01-28]. Dostupné z: https://www.sbprojects.net/knowledge/fileformats/intelhex.php

[7] Hexadecimal Object File Format Specification. In: *Intel HEX Standard* [online]. c1998 [cit. 2024-01-28]. Dostupné z: https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2012/ads264_mws228/Final%20Report/Final%20Report/Intel%20HEX%20Standard.pdf

[8] Intel HEX-record Format. *Lucid Technologies* [online]. c1996-2022 [cit. 2024-01-28]. Dostupné z: https://www.lucidtechnologies.info/intel.htm

[9] GENERAL: Intel HEX File Format. *ARM Developer* [online]. c1995-2024 [cit. 2024-01-28]. Dostupné z: https://developer.arm.com/documentation/ka003292/latest/

[10] Scooter Software | Home of Beyond Compare. *Scooter Software | Home of Beyond Compare* [online]. c2024 [cit. 2024-01-29]. Dostupné z:

https://www.scootersoftware.com/

[11] Beyond Compare Review - Slant. *Slant* [online]. 2023 [cit. 2024-01-29]. Dostupné z: https://www.slant.co/options/4394/~beyond-compare-review

[12] Beyond Compare Features List Comparison. *Scooter Software | Home of Beyond Compare* [online]. c2024 [cit. 2024-01-29]. Dostupné z: https://www.scootersoftware.com/kb/feature_compare

[13] Hexinator. *Welcome to Hexinator* [online]. c2014-2024 [cit. 2024-01-29]. Dostupné z: https://hexinator.com

[14] Hexinator. TEODOROVICI, Mihaela. *Softpedia* [online]. c2001-2024, January 9, 2020 [cit. 2024-01-29]. Dostupné z: https://www.softpedia.com/get/Programming/File-Editors/Hexinator.shtml

[15] ImHex - ImHex. *ImHex - ImHex* [online]. 2023 [cit. 2024-01-29]. Dostupné z: https://docs.werwolv.net/imhex

[16] ImHex - Free and Open Source Hex Editor. *ImHex - Free and Open Source Hex Editor* [online]. 2023 [cit. 2024-01-29]. Dostupné z: https://imhex.werwolv.net

[17] HEJLSBERG, Anders, Mads TORGERSEN, Scott WILTAMUTH a Peter GOLDE. *C# Programming Language*. 4th ed. Addison-Wesley, 2010. ISBN 9781098121952.

[18] ALBAHARI, Joseph. *C# 10 in a Nutshell: The Definitive Reference*. O'Reilly Media, 2022. ISBN 9781098121952.

[19] MICROSOFT. Reference types (C# reference) - C# Reference - C#. *Microsoft Learn* [online]. 2024 [cit. 2024-04-11]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/reference-types

[20] TROELSEN, Andrew a Philip JAPIKSE. *Pro C# 10 with .NET 6*. 11th ed. APress, 2022. ISBN 9781484278680.

[21] What is .NET? An Overview of the Platform. CHIARELLI, Andrea. *Auth0 Blog* [online]. 2021 [cit. 2024-04-11]. Dostupné z: https://auth0.com/blog/what-is-dotnet-platform-overview/

[22] Úvod do technologie .NET. MICROSOFT. *Microsoft Learn* [online]. 2024 [cit. 2024-04-11]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/core/introduction

[23] Průvodce pro desktop pro .NET, .NET Core a .NET Framework. MICROSOFT. *Microsoft Learn* [online]. c2024 [cit. 2024-04-11]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/desktop/

[24] What's a Universal Windows Platform (UWP) app? MICROSOFT. *.NET* [online]. 2023 [cit. 2024-04-11]. Dostupné z: https://learn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide

[25] Co je Xamarin? MICROSOFT. *Microsoft Learn* [online]. 2024 [cit. 2024-04-11]. Dostupné z: https://learn.microsoft.com/cs-cz/xamarin/get-started/what-is-xamarin

[26] Unity-Real-Time Development Platform. MICROSOFT. *.NET* [online]. c2024 [cit. 2024-04-11]. Dostupné z: https://dotnet.microsoft.com/en-us/apps/games/unity

[27] MICROSOFT. Co je .NET MAUI? *.NET* [online]. 2024 [cit. 2024-04-11]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/maui/what-is-maui?view=net-maui-8.0

[28] *WPF Tutorialspoint* [online]. Tutorialspoint, 2018 [cit. 2024-04-12]. Dostupné z: https://www.tutorialspoint.com/wpf/wpf_tutorial.pdf

[29] MACDONALD, Matthew. *Pro WPF 4.5 in C#: Windows Presentation Foundation in .NET 4.5*. 4th ed. Apress, 2012. ISBN 978-1430243656.

[30] CHOWDHURY, Kunal. *Mastering Visual Studio 2017*. Packt Publishing, 2017. ISBN 9781787281905.

[31] WPF vs WinForms – Which One is Right for Your Project? *ByteHide* [online]. 2023 [cit. 2024-04-12]. Dostupné z: https://www.bytehide.com/blog/wpf-vs-winforms

[32] CHAND, Mahesh, HOBBS, Sam, ed. *Programming XAML* [online]. C# Corner, 2014 [cit. 2024-04-11]. Dostupné z: https://pdfcoffee.com/programming-xaml-beginners-guide-pdf-free.html

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| HEX | Intel HEX formát nebo hexadecimální soubor/soustava |
| SREC | Motorola SRecord |
| ASCII | American Standard Code for Information Interchange |
| kB | kilobajt |
| MB | megabajt |
| GB | gigabajt |
| LSB | nejméně významný bit |
| PROM | Programable Read Only Memory |
| USBA | Upper Segment Base Address |
| CS | Code segment |
| IP | Instruction pointer |
| EIP | Extended Instruction Pointer |
| MP3 | MPEG-1 Audio Layer III/MPEG-2 Audio Layer III |
| ZIP | Souborový formát pro kompresi a archivaci dat |
| FTP | File Transfer Protocol |
| SFTP | Secure File Transfer Protocol |
| ARM | Advanced RISC Machine |
| RGBA | Red-Green-Blue-Alpha |
| WPF | Windows Presentation Foundation |
| XAML | Extensible Application Markup Language |
| VB .NET | Visual Basic .NET |
| CLI | Command Language Interface |
| CIL | Common Intermediate Language |
| UWP | Universal Windows Platform |

| | |
|---|---|
| MAUI | Multi-platform App User Interface |
| MFC | Microsoft Foundation Classes |
| DLL | Dynamic link library |
| GUI | Graphics user interface |
| GDI | Graphics device interface |
| MIME | Multipurpose Internet Mail Extensions |
| JSON | JavaScript Object Notation |
| UI | User Interface |
| PDF | Portable Document Format |
| LINQ | Language-Integrated Query |

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# SEZNAM PŘÍLOH

# PŘÍLOHA P I: CD-ROM

Přiložené CD obsahuje:

- Zdrojový kód desktopové aplikace zabalený ve formátu .zip
- Adresář se spustitelným souborem .exe
- Diplomovou práci ve formátu .pdf