

# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER SYSTEMS

## **EMULÁTOR HERNÍ KONZOLE GAMEBOY ADVANCE PRO MOBILNÍ PLATFORMY**

BAKALÁŘSKÁ PRÁCE

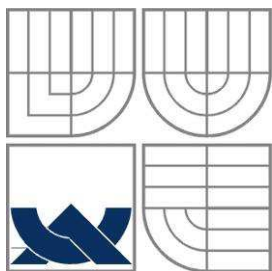
BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

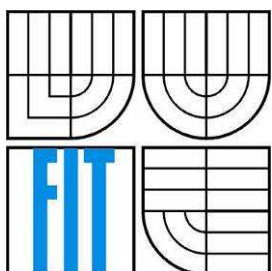
Hynek Vilímek

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER SYSTEMS

# EMULÁTOR HERNÍ KONZOLE GAMEBOY ADVANCE PRO MOBILNÍ PLATFORMY

GAMEBOY ADVANCE EMULATOR FOR MOBILE PLATFORMS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

Hynek Vilímek

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Václav Šimek

BRNO 2013

## Abstrakt

Konzole GameBoy Advance vyšla roku 2001 z dílny firmy Nintendo a za dobu své existence zažila fenomenální úspěch. Ve své době ovládala majoritní část trhu a také díky tomu se stala synonymem pro přenosné herní konzole. Tuto popularitu jí získaly především pro ni vytvořené hry. Ty se staly natolik populární, že i v dnešní době existuje poptávka po hraní těchto her. Už se však nehrají na původních konzolách, ale na současných mobilních zařízeních. A jelikož jsou dnešní mobilní zařízení technologicky naprosto odlišná od konzole, tak je nutné spouštět tyto hry skrz emulátor.

## Abstract

The GameBoy Advance console was released by Nintendo in 2001 and it has experienced phenomenal success during its existence. It used to control the majority of the market at one time and partly due to this it has become synonymous with portable gaming consoles. This popularity was gained especially thanks to the games developed for it. They have become so popular that even today there is a demand for them. These games, however, are not played on the original consoles but on today's mobile devices. And since current mobile devices are technologically very different from the console, it is necessary to run these games via an emulator.

## **Klíčová slova**

Herní konzole, GameBoy Advance, emulátor, procesor ARM7TDMI

## **Keywords**

Gaming console, GameBoy Advance, emulator, processor ARM7TDMI

## **Citace**

Hynek Vilímek: Emulátor herní konzole GameBoy Advance pro mobilní platformy, bakalářská práce, Brno, FIT VUT v Brně, 2013

# Emulátor herní konzole GameBoy Advance pro mobilní platformy

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Václava Šimka.

Další informace mi poskytl Ing. Petr Šimon.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....  
Hynek Vilímek

15. 5. 2013

## Poděkování

Děkuji svému vedoucímu, panu Ing. Václavu Šimkovi za vstřícnost a ochotu. Dále bych chtěl poděkovat panu Ing. Petrovi Šimonovi za poskytnutí cenných informací a rad.

© Hynek Vilímek, 2013

*Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.*

# Obsah

Obsah .....	1
Úvod .....	2
Teorie .....	4
2.1 Emulace .....	4
2.2 Game Boy Advance.....	5
2.2.1 Architektura .....	6
2.2.2 Paměťový subsystém.....	19
2.2.3 Grafické rozhraní .....	21
2.2.4 Vstupní moduly.....	24
2.2.5 Časové spínače .....	24
2.2.6 Přímý přístup do paměti .....	25
2.3 Mobilní platforma Android.....	25
Implementace .....	26
2.4 Vstupní bod .....	26
2.5 Procesor .....	26
2.6 Paměťový subsystém.....	28
2.7 Grafické rozhraní .....	29
Testování funkčnosti emulátoru.....	31
Závěr .....	33
Citovaná literatura .....	34
Diagram tříd .....	35
Fragment kódu .....	37

## Kapitola 1

# Úvod

Konzole GameBoy Advance zažila ve své době obrovský úspěch, a to především mezi příslušníky mladší generace. Této generaci totiž dala do ruky možnost hrát na tu dobu kvalitní a především barevné hry kdekoliv a kdykoliv. Hry se v té době stávaly čím dál tím více žádanějším zbožím, a tak velmi mnoho příslušníků mladší generace toužilo po vlastnění této všemi milované konzole. Její popularita také rostla díky dětskému seriálu japonského původu Pokémon. Obrovskou popularitu tohoto seriálu dokazují i série celovečerních filmů rozvíjejících příběhy hlavních hrdinů. Mladší generace tak toužila „žít“ jako postavy tohoto fenoménu a tato konzole jim to skrz její hry umožňovala na velmi vysoké úrovni.

S nástupem chytřejších mobilů a dalších, modernějších zařízení tento nadměrný zájem o tuto konzolu uvaldal. Co však neuvadalo, byla popularita jejich her. Stále více lidí se však rádo k těmto hrám vrací a i dnes je chtějí hrát. Tato poptávka hráčské komunity vyústila až v tvorbu emulátorů. Ty byly nejdříve nedokonalé, a protože přenosná zařízení samozřejmě nebyla na takové úrovni, jakou známe dnes, tak byly primárně vytvářeny pro osobní počítače. Až s masovějším nástupem chytrých přenosných zařízení vzniknul tlak a potřeba na vznik emulátorů i pro tyto zařízení.

V současnosti existuje několik emulátorů pro mobilní zařízení. Mým cílem tedy není překonat tyto dlouhodobě a organizovaně vyvíjené emulátory, na kterých spolupracuje tým lidí a které jsou již vyzkoušené a prověřené. Mojí pohnutkou byla především zapálenost a nadšenost z této konzole a chuť vytvořit vlastní, alespoň z části funkční emulátor, se kterým bych já mohl zavzpomínat na éru této konzole.

Při výběru mobilní platformy jsem se rozhodoval ryze pragmaticky, a proto jsem si vybral platformu Android. Ta se, co se týče popularity, řadí k tomu nejlepšímu, co dnešní trh s mobilními zařízeními nabízí. Dalším pádným důvodem také bylo moje vlastnictví tohoto zařízení.

**Kapitola 2** pojednává a objasňuje principy fungování konzole. Jelikož jsem se rozhodl vytvořit celý model konzole, tak je zde i popisováno, jak funguje součásti procesoru a jak jsou spolu provázány.

**V kapitole 3** popisují způsob mojí implementace, případná úskalí, která se při vývoji objevila, a jejich řešení. Jelikož by velmi detailní popis mohl být kontraproduktivní, tak se zmiňuji pouze o důležitých součástích a souvislostech.

**V kapitole 4** se věnuji způsobu testování a shrnutí jejich výsledků.

## Kapitola 2

# Teorie

### 2.1 Emulace

Informace k této kapitole jsou čerpány z [1] a [2].

Emulace je proces, který slouží k napodobení systému na jiném hostitelském systému. K tomu je zapotřebí speciální software, který tvoří mezivrstvu mezi oběma systémy. Tato mezivrstva může pracovat na několika úrovních, čímž emulovanému systému dovoluje přístup k různým zdrojům. Typicky můžeme na jednom hostitelském systému provozovat mnoho emulovaných systémů. Podle Church-Turingovi teze lze totiž každý výpočet uskutečnit algoritmem běžícím na počítači, za předpokladu, že máme dostatek času a paměti. Díky tomu předpokladu tedy můžeme emulovat libovolný systém na libovolném systému.

Samotná emulace by nebyla mnoho užitečná, pokud by nám nepřinášela žádné výhody. Tou nejdůležitější je především úspora zdrojů. Tedy například lepší využití výpočetního výkonu počítače paralelním emulováním více systémů, peněžní úspora při pořizování zařízení, pro které vyvíjíme software, či úspora času. Emulátory mají navíc široké využití vestavěných systémů, kdy dovolují vytvářené programy důkladně otestovat bez nutnosti jejich nahrání do cílového systému.

Samotnou emulaci lze rozdělit na několik samostatných mezikroků – na emulaci procesoru, přístupu do paměti a na emulaci ostatních vstupně-výstupních modulů. Emulace procesoru bývá realizována jako interpret instrukcí daného procesoru. Při samotné interpretaci je však zapotřebí implementovat veškeré vedlejší účinky. Tyto účinky většinou nebývají zdokumentovány a tak se při vývoji hojně využívá reverzního inženýrství. Dalším řešením může být převod emulovaného kódu na kód hostitelského systému. Tento způsob je však velmi náročný na implementaci a samotná interpretace bývá podstatně jednodušší. Emulace přístupu do paměti primárně řeší převod adres, rychlost přístupů do různých typů paměti a mapování registrů do paměti. Emulace vstupně-výstupních modulů již plně záleží na daném zařízení. Jako typický příklad vstupně-výstupního modulu je klávesnice, či grafický adaptér.

Ačkoliv emulace poskytuje mnohé výhody, nelze přehlédnout ani její nevýhody. Tou největší je právě časová a paměťová náročnost. Velmi náročná je také případná interpretace procesoru. Progra-

my pro tato zařízení jsou navíc psány s úsilím získat z daného hardwaru co největší výkon, a proto vývojáři často využívali nezdokumentovaných vlastností – takzvaných vedlejších účinků. Emulátor se také může stát nástrojem k porušení duševního vlastnictví, jelikož použití emulátoru velmi zjednodušuje přístup jak k softwaru, tak i k hardwaru, a není tak nutné jej vlastnit.

## 2.2 Game Boy Advance

Informace k této kapitole jsou čerpány z [3] a [4].

Game Boy Advance je 32bitová kapesní herní konzole vyrobená firmou Nintendo. Jedná se o přímého nástupce konzole Game Boy Color. Tato konzole byla poprvé uvedena roku 2001. Od doby uvedení bylo provedeno několik inovací této konzole. Tyto úpravy však nebyly žádného zásadního charakteru a jsou známy v podobě zařízení Game Boy Advance SP a Game Boy Micro. Od doby uvedení na trh si konzole získala velkou oblibu, bylo pro ni vyvinuto mnoho her a v tomto odvětví měla dominantní postavení. To dokazuje i sumarizační zpráva z roku 2013, kde firma uvádí 81.51 milionů prodaných kusů. Jeho distribuce byla ukončena roku 2008.

Základní model Game Boy Advance byl 14.45 cm dlouhý, 2.45 cm široký, 8.2 cm vysoký a jeho hmotnost byla 140 g. Jeho součástí byla také TFT LCD obrazovka s úhlopříčkou 7.4 cm a s rozlišením 240×160 bodů. Dále obsahoval 32bitový procesor ARM7TDMI pracující na frekvenci 16.8Mhz a pro účely zpětné kompatibility s hrami určenými pro konzolu Game Boy Color obsahoval ještě 8bitový koprocesor Z80. Díky tomu bylo možno spustit veškeré hry určené pro konzolu Game Boy Color i na této konzole. Další nedílnou součástí byla vnitřní 32 KB operační paměť umístěná v procesoru, 96KB video paměť umístěná taktéž v procesoru, 256 KB operační paměť umístěná na základní desce, 1 KB paměti určené pro definici a zobrazování pohyblivých objektů a 1 KB paměť určená pro palety barev. Konzole disponuje podporou dvou barevných módů s maximálním počtem 512 nebo 32768 barev. A nakonec dva 8bitové digitálně analogové převodníky pro podporu stereo zvuku. Napájení bylo obstaráváno dvěma bateriemi velikosti AA. Původní vzhled konzole je možné si prohlédnout na obrázku 2.1.

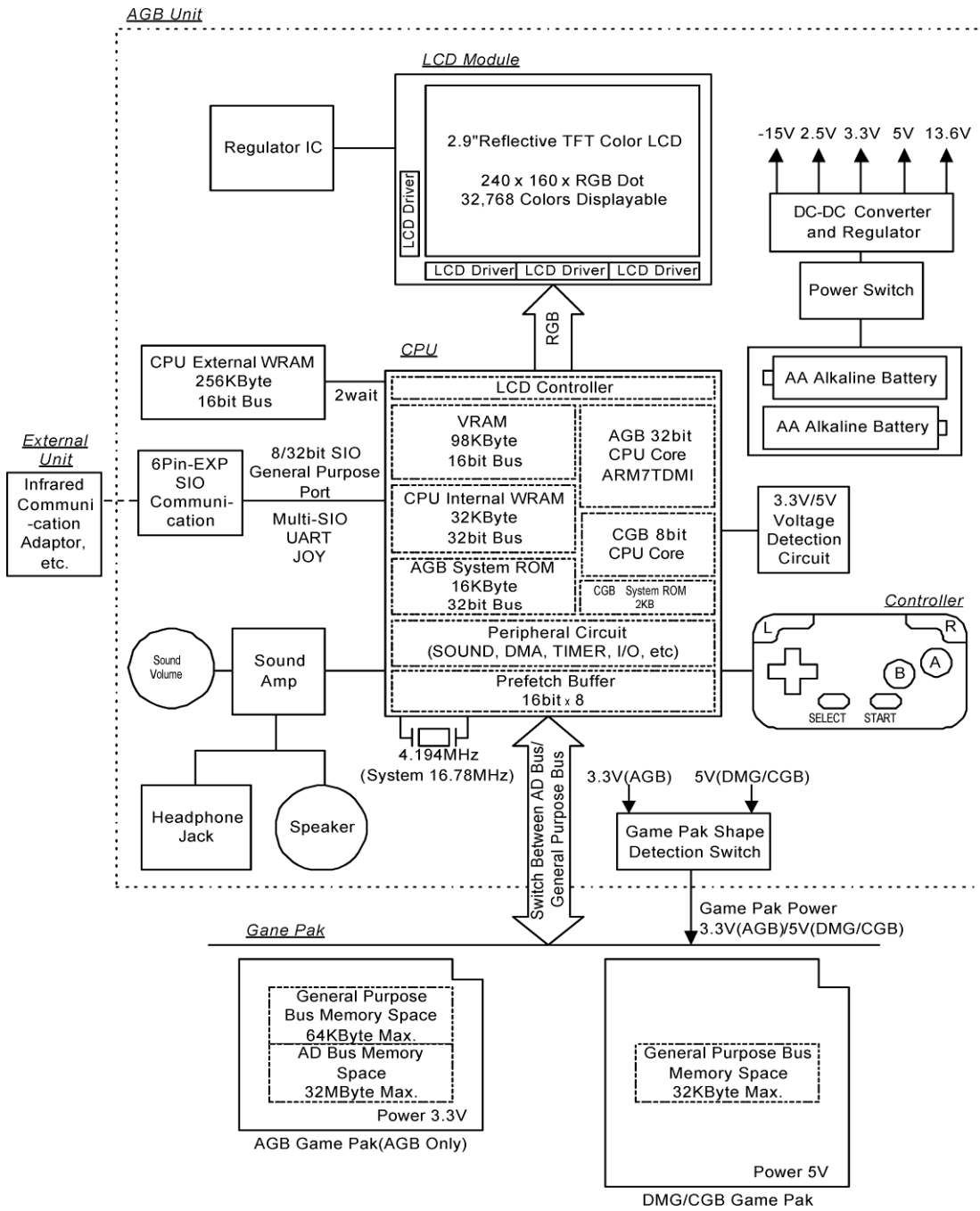


**Obrázek 2.1 – GameBoy Advance**

Prvního vylepšení se Game Boy Advance dočkal v podobě konzole Game Boy Advance SP. Změny se týkaly designu, podporou dobíjecího lithium-iontového akumulátoru a kvalitnějšího LCD displeje. Druhé vylepšení v podobě konzole GameBoy Micro bylo pouze kosmetického rázu a nedosáhlo tak výrazné popularity.

### **2.2.1 Architektura**

Informace k této kapitole jsou čerpány z [5], [6], [7], [8] a [9]. V následujících kapitolách se budu věnovat pouze těm částem architektury, kterým bylo nutné se v rámci implementace věnovat. Kompletní přehled všech součástí GameBoy Advance je možné vidět na obrázku 2.2.



Obrázek 2.2 – blokový diagram konzole GameBoy Advance

## **Procesor ARM7TDMI**

Konzole obsahuje procesor ARM7TDMI, což je 32bitový RISC procesor vytvořen firmou ARM, který byl navržen jak pro vysoký výkon, tak pro malou energetickou náročnost. Toho je dosaženo pomocí dvou módů pro zpracování instrukcí – 32bitovém ARM režimu a 16bitovém THUMB režimu. Procesor také podporuje technologii zřetěženého zpracování instrukcí (pipelining), která dovoluje zpracovávat až tři instrukce současně – první z nich je čtena z paměti, druhá je dekodována a třetí je prováděna. Procesor je Von Neumannova typu s jedinou 32bitovou datovou sběrnicí. Tato sběrnice je sdílená a je tedy používána jak pro přenos dat, tak i pro přenos instrukcí.

Mezi přednosti THUMB režimu patří vyšší výkonnost a menší velikost programového kódu. Nehledě na šířku instrukcí procesor dovoluje používat 32bitové registry, 32bitový přístup do paměti a také 32bitové aritmetické a logické operace. Na druhou stranu nejsou instrukce tak multifunkční jako v ARM režimu a většina instrukcí THUMB režimu využívá pouze poloviční paletu všech dostupných registrů.

Za přednost ARM režimu lze především pokládat vyšší funkcionalitu instrukcí a také možnost využívat u všech instrukcí celou sadu registrů. Jeho nevýhodami je větší náročnost vykonávání instrukcí a vzhledem k dvojnásobné velikosti instrukce i větší velikost programového kódu.

Pro dosažení nejlepší výkonnosti je zapotřebí oba režimy kombinovat. Samotné přepnutí je realizováno speciální instrukcí a její vykonání je z tohoto důvodu optimalizováno. Navíc oba režimy používají stejnou sadu registrů, což zjednodušuje výměnu dat mezi nimi. V některých případech může být procesor navíc automaticky přepnut do ARM režimu. To nastává například při zpracování výjimky, kdy se procesor během provádění výjimky přepne do ARM režimu a po dokončení obsluhy výjimky se procesor přepne zpátky do původního režimu.

Procesor lze provozovat v 6 módech.

- **Uživatelský mód** – Tento mód je základním módem procesoru a většina procesorového času je strávena právě v tomto módu.
- **Systémový mód** – Tento mód lze popsat jako privilegovaný uživatelský mód, který je určen pro běh operačního systému. Konzole GameBoy Advance do tohoto módu vstupuje při vykonávání základních vstupně-výstupních funkcí.
- **Mód rychlého přerušování** – Do tohoto privilegovaného módu procesor přejde v momentě, kdy je vyvolán rychlý požadavek na přerušování. V základním nastavení konzole je tento mód nevyužit.
- **Mód přerušování** – Do tohoto privilegovaného módu je procesor přepnut vždy, když dojde k požadavku na přerušování. Všechny přerušování v konzole se zpracovávají v tomto módu.

- **Dozorčí mód** – V tomto privilegovaném módu se procesor nachází vždy, když je vykonána instrukce softwarového přerušení.
- **Mód selhání** – Do tohoto privilegovaného módu procesor vstoupí v momentě, kdy se nepovedou načíst data nebo instrukce z paměti.
- **Nedefinovaný mód** – Do tohoto privilegovaného módu vstupuje procesor ve chvíli, kdy je zpracovávána nedefinovaná instrukce.

Všechny módy kromě uživatelského módu jsou privilegované a mají tedy přístup ke všem systémovým zdrojům a mohou libovolně měnit mód. Uživatelský mód může vstoupit do jiného módu jen v případě vyvolání výjimky, která má být zpracována v jiném módu.

## Registrová sada

Registrová sada se skládá z 37 registrů o velikosti 32bitů. 31 registrů je univerzálních a zbylých 6 plní funkci registrů příznaků. Některé registry jsou však naklonovány a jejich obsah se tak liší v závislosti na módu, ve kterém se procesor právě nachází. Z tohoto důvodu je možno vždy přistupovat pouze k 15 univerzálním registrům.

Hlavní registr příznaků (CPSR) je jediný a jeho obsah se tak neliší v žádném módu procesoru. Tento registr ukládá stav znaménkového příznaku (SIGN FLAG), nulového příznaku (ZERO FLAG), příznaku přenosu (CARRY FLAG) a příznaku přetečení (OVERFLOW FLAG). Dále je zde uložen příznak režimu procesoru, příznaky povolení přerušení a mód, ve kterém se procesor právě nachází. Další registry příznaků (SPSR) slouží k uložení obsahu hlavního registru příznaků. Registr SPSR není dostupný v uživatelském a systémovém módu. Vždy, když procesor začne obsluhovat výjimku, tak se aktuální obsah CPSR uloží do registru SPSR, který přísluší módu, do kterého bylo právě vstoupeno, a když procesor opouští výjimku, tak je obsah registru SPSR nahrán zpět do registru CPSR. Tento automatický způsob ukládání registru příznaků je vhodný pro jednoduché zpracovávání výjimek, avšak nelze jej použít v případě, kdy chceme výjimky zanořovat do sebe. V tom případě je případné uložení toho registru plně v moci tvůrce.

Ostatní registry jsou označovány číselně R0 až R15. Registry R0 až R12 včetně nemají žádný výjimečný význam a mohou být použity libovolným způsobem. Registr R13 plní roli ukazatele do zásobníku (SP). Registr R14 (LR) je využíván pro uložení návratové adresy pro vykonávání subrutin. Oba dva registry jsou v módu rychlého přerušení, v dozorčím módu, v módu selhání, v módu přerušení a v nedefinovaném módu unikátní. Registr R15 plní funkci čítače instrukcí (PC) a z registrů se speciálním významem je jediný, který je ve všech módech totožný.

## Instrukční sada ARM režimu

Každá instrukce ARM režimu obsahuje 4bitové pole, které určuje, za jakých okolností se daná instrukce má vykonat, a je umístěno na horních bitech instrukce. Kompletní přehled významů podmínkových kódů si lze prohlédnout níže na přiložené tabulce 2.1.

Operační kód	Slovní význam	Stav příznaků nutný k provedení
0000	Rovno	ZERO FLAG je nastaven
0001	Ne rovno	ZERO FLAG není nastaven
0010	Došlo k přenosu	CARRY FLAG je nastaven
0011	Nedošlo k přenosu	CARRY FLAG není nastaven
0100	Záporná hodnota	SIGN FLAG je nastaven
0101	Kladná hodnota	SIGN FLAG není nastaven
0110	Došlo k přetečení	OVERFLOW FLAG je nastaven
0111	Nedošlo k přetečení	OVERFLOW FLAG není nastaven
1000	Beznaménkové porovnání větší než	OVERFLOW FLAG je nastaven a ZERO FLAG není nastaven
1001	Beznaménkové porovnání menší nebo rovno než	CARRY FLAG není nastaven a ZERO FLAG je nastaven
1010	Znaménkové porovnání větší nebo rovno než	SIGN FLAG a OVERFLOW FLAG si jsou rovny.
1011	Znaménkové porovnání menší než	SIGN FLAG a OVERFLOW FLAG si nejsou rovny.
1100	Znaménkové porovnání větší než	ZERO FLAG není nastaven a SIGN FLAG a OVERFLOW FLAG si jsou rovny.
1101	Znaménkové porovnání menší nebo rovno	ZERO FLAG je nastaven a SIGN FLAG a OVERFLOW FLAG si nejsou rovny.
1110	Vždy je provedeno	Libovolný stav
1111	Nedefinovaný operační kód	

**Tabulka 2.1 – podmínkové kódy instrukcí ARM**

## **Skokové instrukce**

Dále se instrukční sada skládá z instrukcí skoků. Jejich popis lze vidět v následujícím seznamu s jejich mnemotechnickými zkratkami.

- **B** – Základní instrukce skoku.
- **BL** – Instrukce skoku do podprogramu. Tato instrukce navíc od instrukce skoku uloží obsah programového čítače na zásobník, aby tak byl umožněn případný návrat z podprogramu.
- **BX** – Instrukce skoku s případným přepnutím do THUMB režimu.
- **BLX** – Instrukce skoku do podprogramu s případným přepnutím do THUMB režimu.

Případné přepínání do THUMB režimu je závislé na obsahu registru, který je právě určen onou instrukcí. Za povšimnutí stojí, že mezi instrukcemi skoku nenalezneme žádnou speciální instrukci podmíněného skoku. To je dáno právě tím, že všechny instrukce jsou standardně již podmíněné, a tak není nutné, aby ARM režim těmito instrukcemi oplýval.

## **Instrukce zpracovávající data**

Další součástí instrukční sady jsou instrukce určené ke zpracování dat. Do této skupiny lze zařadit 16 instrukcí instrukční sady. Jejich přehled lze vidět v následujícím seznamu včetně jejich mnemotechnických zkratk.

- **AND** – Instrukce vykonávající logickou operaci a s dvěma operandy.
- **EOR** – Instrukce vykonávající logickou operaci exklusivní nebo s dvojicí operandů.
- **SUB** – Instrukce odečtení dvou operandů.
- **RSB** – Instrukce odečtení dvou operandů v opačném pořadí.
- **ADD** – Instrukce sečtení dvou operandů.
- **ADC** – Instrukce sečtení dvou operandů a příznaku přenosu.
- **SBC** – Instrukce odečtení dvou operandů a příznaku přenosu.
- **RSC** – Instrukce odečtení dvou operandů v opačném pořadí a příznaku přenosu.
- **TST** – Instrukce porovnávací dvojici operandů. Porovnání je uskutečněno operací AND a podle výsledku je pozměněn registr příznaků.
- **TEQ** – Instrukce porovnávací dvojici operandů. Porovnání je provedeno pomocí operace EOR a podle výsledku je pozměněn registr příznaků.
- **CMP** – Instrukce porovnávací dvojici operandů. Ty jsou od sebe odečteny a následně je dle výsledku pozměněn registr příznaků.
- **CMN** – Instrukce porovnávací dvojici operandů. Operandů jsou sečteny a dle výsledku se pozmění registr příznaků.
- **ORR** – Instrukce provádějící logickou operaci nebo s dvojicí operandů.
- **MOV** – Instrukce přiřazení hodnoty operandu do cílového registru.

- **BIC** – Instrukce vynulování konkrétního bitu.
- **MVN** – Instrukce vykonávající přiřazení opačné hodnoty operandu do cílového registru.

Všechny instrukce kromě těch, které porovnávají operandy, definují cílový registr, do kterého bude uložen výsledek operace. Kromě instrukcí MOV a MVN definují všechny první operand jako registr. U instrukcí MOV a MVN z významu jejich funkce je nutný pouze jeden operand. Všechny ostatní instrukce využívají i druhý operand. Tímto operandem může být jak registr, tak přímo definovaná hodnota. Tyto instrukce mohou mít vliv na obsah registrů příznaků. Tato možnost však musí být explicitně určena v instrukci. V každé instrukci může být druhý operand posouván. Posouván může být jak registrový, tak přímý operand, avšak registrový operand může být posouván logicky doleva, logicky doprava, aritmeticky doprava a rotovat doprava, kdyžto přímý operand může být pouze rotován doprava.

### ***Instrukce pro manipulaci s registry příznaků***

Instrukční sada dále obsahuje dvě instrukce pro manipulaci s registry příznaků. První z nich je instrukce pro přesun hodnoty z registru kopie registru příznaků (SPSR) do univerzálního registru (MRS). Instrukce vždy operuje s registrem SPSR daného módu, přičemž pokud tento registr v některém módu neexistuje, tak tato instrukce nemá význam. To se dle kapitoly o registrové sadě týká pouze uživatelského módu a systémového módu. Druhá z nich je instrukce pro přesun hodnoty do registru SPSR (MSR). Hodnotou může být jak přímá hodnota, tak obsah registru. Dále lze v této instrukci určit, které konkrétní příznaky se mají přepsat i do hlavního registru příznaků CPSR.

### ***Instrukce násobení***

Další součástí jsou instrukce určené k vykonání operace násobení. Jejich přehled lze vidět na následujícím seznamu i s jejich mnemotechnickými zkratkami.

- **MUL** – Instrukce násobení dvou 32bitových operandů. Do cílového registru se ukládá pouze spodních 32 bitů výsledku operace a horních 32 bitů výsledku je ignorováno.
- **MLA** – Instrukce násobení dvou 32bitových operandů a následné přičtení výsledku operace k obsahu libovolného univerzálního registru. Do cílového registru se ukládá pouze spodních 32 bitů výsledku operace a horních 32 bitů výsledku je ignorováno.
- **UMULL** – Instrukce bezznaménkového násobení dvou 32bitových operandů. Výsledkem je 64bitová hodnota, která je uložena ve dvou 32bitových registrech.
- **UMLAL** – Instrukce bezznaménkového násobení dvou 32bitových operandů a následné přičtení obsahu virtuálního 64bitového registru, jehož roli plní dva univerzální registry, které jsou zároveň cílovými registry, neboť i výsledek operace je 64bitový.
- **SMULL** – Instrukce znaménkového násobení dvou 32bitových operandů. Výsledkem je 64bitová hodnota, která je uložena ve dvou 32bitových registrech.

- **SMLAL** – Instrukce znaménkového násobení dvou 32bitových operandů a následné přičtení obsahu virtuálního 64bitového registru, jehož roli plní dva univerzální registry. Ty jsou zároveň cílovými registry výsledku, neboť výsledná hodnota je též 64bitová.
- **SMLAxy** – Instrukce znaménkového násobení dvou 16bitových operandů a následné přičtení obsahu 32bitového registru, jehož funkci plní libovolný univerzální registr. Výsledkem je 32bitová hodnota, která je uložena v jediném registru.
- **SMLAWy** – Instrukce znaménkového násobení 32bitového operandu a 16bitového operandu a následné přičtení obsahu libovolného univerzálního registru. Výsledkem je 32bitová hodnota, a proto je ignorováno jakékoliv případné přetečení výsledku.
- **SMULWy** – Instrukce znaménkového násobení 32bitového operandu a 16bitového operandu. Výsledkem je 32bitová hodnota, a proto je ignorováno jakékoliv případné přetečení.
- **SMLALxy** – Instrukce znaménkového násobení dvou 16bitových operandů. Výsledek této operace je přičten k 64bitové hodnotě, kterou definují dva univerzální 32bitové registry.
- **SMULxy** – Instrukce znaménkového násobení dvou 16bitových operandů. Výsledkem je 32bitová hodnota, která je uložena v jediném 32bitovém registru.

Tyto instrukce mohou mít vliv na registr příznaků. Toto nastavení však musí být explicitně nastaveno v instrukci. Všechny tyto instrukce mohou mít jako operand pouze univerzální registr a nesmí to být programový čítač (registr R15). I v případě instrukcí, které operují s 16bitovými operandy, jsou použity univerzální 32bitové registry. V instrukci však lze nastavit, zda se má jako 16bitový operand použít horní nebo dolní část registru.

### ***Instrukce přistupující k paměti***

Poslední důležitou součástí instrukční sady jsou instrukce pro manipulaci s pamětí. Tyto instrukce jsou si velmi podobné, co se funkčnosti týče, avšak každá z nich je vhodná na jiný druh přístupu k paměti. Všechny instrukce používají minimálně dva registry. V prvním z nich je definována báze adresy, která bude využita pro čtení či zapisování, a druhý z nich plní funkci zdrojového nebo cílového registru. Všechny instrukce navíc používají logickou adresu (offset). Pomocí logické adresy se určuje finální adresa pro přístup do paměti. Logická adresa může být odečítána nebo přičítána k báze adresy. Nehledě na způsob posunu adresy lze nastavit, aby se tato operace vykonala před nebo až po samotném přístupu do paměti. Instrukce nedovolují nezarovnaný přístup do paměti. Základní instrukce pro manipulaci s daty lze vidět v následujícím seznamu i s jejich mnemotechnickými zkratkami.

- **LDR** – Instrukce pro načtení dat z adresy definované báze adresou a logickou adresou do univerzálního registru.
- **STR** – Instrukce pro uložení dat z registru do paměti na adresu, která je definována báze adresou a logickou adresou.

V obou případech je možné definovat logickou adresu obsahem některého z registrů, nebo přímou hodnotou. V případě definování obsahem registru lze tuto hodnotu před samotným vykonáním operace logicky posunout doleva, logicky posunout doprava, aritmeticky posunout doprava nebo rotovat doprava. Obě instrukce mohou pracovat jak v 8bitovém, tak v 32bitovém režimu přístupu do paměti.

Další instrukce rozšiřující způsob přístupu do paměti jsou tyto následující.

- **STRH** – Instrukce uložení 16bitové hodnoty.
- **LDRD** – Instrukce načtení 64bitové hodnoty. Instrukcí je definován pouze jeden cílový registr, a proto je v tomto případě druhá část načtené 64bitové hodnoty uložena v registru bezprostředně následujícím.
- **STRD** – Instrukce uložení 64bitové hodnoty. Instrukcí je definován pouze jeden zdrojový registr, a proto je jako druhý zdrojový registr vybrán bezprostřední následník tohoto registru v registrové sadě.
- **LDRH** – Instrukce načtení bezznaménkové 16bitové hodnoty.
- **LDRSB** – Instrukce načtení znaménkové 8bitové hodnoty.
- **LDRSH** – Načtení znaménkové 16bitové hodnoty.
- **LDM** – Instrukce blokového načtení paměti do registrů.
- **STM** – Instrukce blokového uložení registrů do paměti.

Cílové nebo zdrojové registry instrukcí s blokovým přístupem do paměti jsou definovány jako 15bitové pole, kde každý bit odpovídá právě jednomu registru a určuje tak, zda je odpovídající registr vybrán k operaci, či nikoliv. Logická adresa je určena implicitně, neboť lze takto operovat pouze s celými 32bitovými registry.

### **Instrukční sada THUMB režimu**

Instrukční sada THUMB režimu je co do objemu a funkčnosti instrukční sady ARM režimu chudší. Téměř všechny instrukce mohou pracovat pouze s prvními 8 univerzálními registry. Pro využití všech registrů je nutné použít více jak jednu instrukci. Chybí zde také možnost nastavit, aby instrukce ovlivnila či neovlivnila registr příznaků. Také posuvy operandů nelze vykonávat v rámci jiné instrukce, ale pouze jako samostatnou operaci k tomu určené instrukce.

## ***Instrukce posuvů***

První částí instrukční sady THUMB režimu jsou instrukce určené pro posuny s operandem určeným přímou hodnotou, který definuje počet jednotek, o které má být posun proveden. Tyto instrukce jsou uvedeny v následujícím seznamu se svými mnemotechnickými zkratkami.

- **LSL** – Instrukce logického posunu vlevo.
- **LSR** – Instrukce logického posunu vpravo.
- **ASR** – Instrukce aritmetického posunu vpravo.

Tyto instrukce využívají zdrojový registr a cílový registr. Obsah zdrojového registru je posunut o definovanou hodnotu v instrukci a výsledek operace je uložen v cílovém registru. Zároveň při vykonávání instrukce dochází k ovlivnění registru příznaků a to konkrétně nulového příznaku, znaménkového příznaku a příznaku přenosu. K ovlivnění příznaku přenosu nedochází jen v případě instrukce logického posunu vlevo o 0 pozic.

## ***Instrukce zpracovávající data***

Další důležitou skupinou instrukcí jsou instrukce určené k zpracování dat.

- **ADD** – Tato instrukce má několik variant:
  - Sečtení obsahu dvou registrů a uložení výsledku do cílového registru.
  - Sečtení registru a přímého operandu a uložení do cílového registru. Obor hodnot přímo definovaného operandu je pouze v rozmezí 0 až 7.
  - Přičtení přímo definované hodnoty k obsahu registru. Její obor hodnot je v rozmezí 0 až 255.
- **SUB** – Tato instrukce má několik variant:
  - Odečtení obsahu dvou registrů a uložení výsledku do cílového registru.
  - Odečtení obsahu registru a přímého operandu a uložení do cílového registru. Obor hodnot přímo definovaného operandu je v rozmezí 0 až 7.
  - Odečtení přímé hodnoty od obsahu registru. Obor hodnot přímo definovaného operandu je v rozmezí 0 až 255.
- **MOV** – Instrukce přiřazení přímé hodnoty do registru.
- **CMP** – Instrukce porovnání přímé hodnoty a obsahu registru. Obor hodnot přímo definované hodnoty je v rozmezí 0 až 255.

Pro zpracování dat jsou také určeny následující instrukce. Jejich společným rysem je využívání pouze dvou registrů – cílového registru a zdrojového registru. Jako první operand je proto používán obsah cílového registru a jako druhý operand je využíván druhý registr.

- **AND** – Instrukce provádějící logickou operaci a.

- **EOR** – Instrukce vykonávající logickou operaci exklusivní nebo.
- **LSL** – Instrukce logického posunu vlevo. Počet jednotek, o které má být posun proveden, je určen spodními 8 bity druhého operandu.
- **LSR** – Instrukce logického posunu vpravo. Počet jednotek, o které má být posun proveden, je definován spodními 8 bity druhého operandu.
- **ASR** – Instrukce aritmetického posunu vpravo. Počet jednotek posunu je definován spodními 8 bity druhého operandu.
- **ADC** – Instrukce vykonávající aritmetickou operaci sčítání s přičtením příznaku přenosu.
- **SBC** – Instrukce provádějící aritmetickou operaci odečítání s případným odečtením příznaku přenosu.
- **ROR** – Instrukce rotace vpravo. Počet jednotek je definován spodními 8 bity druhého operandu.
- **TST** – Instrukce porovnání dvou operandů operací pomocí instrukce AND. Nemění obsahy registrů.
- **NEG** – Instrukce přiřazení záporné hodnoty druhého operandu do cílového registru.
- **CMP** – Instrukce porovnání operandů. To je uskutečněno odečtením druhého operandu od prvního. Nijak nemění obsahy registrů.
- **CMN** – Instrukce porovnání operandů. To je provedeno sečtením obou operandů. Nijak nemění obsahy registrů.
- **ORR** – Instrukce vykonávající logickou operaci nebo.
- **MUL** – Instrukce vykonávající aritmetickou operaci násobení. Oba dva operandy jsou definovány na 32 bitech a výsledek operace je též uložen v jediném, 32bitovém registru.
- **BIC** – Instrukce určená k nulování konkrétních bitů.
- **MVN** – Instrukce přiřazení bitové negace druhého operandu do cílového registru.

Všechny instrukce ovlivňují znaménkový příznak a nulový příznak. Instrukce ADC, SBC, NEG, CMP, CMN, ADD, SUB navíc ovlivňují i příznak přenosu a příznak přetečení. Příznak přenosu je také měněn při použití instrukcí LSL, LSR, ASR, ROR, avšak pouze pokud se jedná o nenulové posunutí. Příznak přenosu může být také měněn při vykonání instrukce MUL.

Jelikož by bylo nepraktické striktní snížení počtu využitelných registrů o polovinu, tak instrukční sada THUMB režimu disponuje několika instrukcemi, které manipulují se všemi registry a v kombinaci s ostatními instrukcemi tak lze plnohodnotně využívat plnou sadu registrů. Přehled těchto instrukcí lze vidět v následujícím seznamu.

- **ADD** – Instrukce vykonávající aritmetickou operaci sčítání. K obsahu cílového registru se přičte obsah druhého registru. Neovlivňuje registr příznaků.
- **CMP** – Instrukce porovnání obsahu dvou registrů jejich odečtením. Následně je dle výsledku operace pozměněn registr příznaků.

- **MOV** – Instrukce přiřazení obsahu registru cílovému registru.
- **BX** – Instrukce vykonávající přepnutí procesoru do ARM režimu. Adresa první instrukce vykonané v ARM režimu je uložena ve zdrojovém registru.
- **BLX** – Instrukce provádějící přepnutí procesoru do ARM režimu s uložením aktuálního obsahu programového čítače (registr R15) do registru určeného pro uložení zpětné adresy (registr R14).

Přepnutí do režimu ARM není automaticky provedeno v rámci vykonání k tomu určené instrukce, ale závisí na hodnotě 0. bitu zdrojového registru. Jestliže je jeho hodnota 0, tak k přepnutí dojde, jinak nikoliv.

### ***Instrukce přistupující k paměti***

Součástí instrukční sady THUMB režimu jsou také instrukce pro přístup do paměti. Lze ukládat 8bitová, 16bitová a 32bitová data. Dále lze načítat znaménkově nebo bezznaménkově 8bitová a 16bitová data. Tyto instrukce operují s registrem, jehož obsah definuje bázovou adresu, zdrojovým registrem v případě ukládání nebo cílovým registrem v případě načítání a logickou adresou (offsetem). Ta může být určena jak přímou hodnotou, tak obsahem registru a je interpretována různě v závislosti na velikosti ukládané nebo načítané hodnoty tak, aby nedošlo k nezarovnanému přístupu do paměti. Tedy při paměťové operaci s 8bitovou hodnotou je offset interpretován po krocích o jednu jednotku, s 16bitovou operací je offset interpretován po krocích o dvě jednotky a s 32bitovou hodnotou je offset interpretován po krocích o čtyři jednotky. Lze též využít instrukce pro blokový přístup do paměti, kdy zdrojové či cílové registry jsou definovány v bitovém poli, kde každý bit určuje konkrétní registr, který má či nemá být součástí operace.

Pro manipulaci s pamětí jsou k dispozici také instrukce s konkrétnějšími funkcemi. Těmi jsou načtení dat z adresy, která je definována relativně pomocí logické adresy a obsahu programového čítače. Dále instrukce pro čtení a zápis s daty, která jsou definována relativně pomocí logické adresy a ukazatele do zásobníku. Dále je možné použít instrukce ke zjištění adresy pomocí logické adresy, kdy funkci bázového registru plní buď programový čítač, nebo ukazatel do zásobníku, přičtení logické adresy k ukazateli do zásobníku a instrukce pro vložení dat do zásobníku a odebrání dat ze zásobníku. Instrukce pro odebrání dat ze zásobníku také může automaticky načítat data ze zásobníku do programového čítače (registr R15) a instrukce pro vkládání dat do zásobníku může automaticky vložit obsah registru určeného pro uložení návratové adresy (registr R14). Jinak lze využívat zase pouze poloviční sadu registrů.

### **Instrukce podmíněného skoku**

Dále lze v instrukční sadě THUMB režimu nalézt instrukce podmíněného skoku. Délka skoku může být až o 256 jednotek a lze samozřejmě provést skok jak dopředu, tak dozadu. Jejich výčet lze vidět v následujícím seznamu včetně jejich mnemotechnických zkratk.

- **BEQ** – Instrukce skoku, kdy skok je proveden, jestliže je nastaven nulový příznak.
- **BNE** – Instrukce skoku, kdy skok je proveden, jestliže nulový příznak není nastaven.
- **BCS** – Instrukce skoku, kdy skok je proveden v případě, že je nastaven příznak přenosu.
- **BCC** – Instrukce skoku, kdy skok je proveden, jestliže není nastaven příznak přenosu.
- **BMI** – Instrukce skoku, jejíž provedení nastane, jestliže je nastaven znaménkový příznak.
- **BPL** – Instrukce skoku, jejíž provedení nastane, když není nastaven znaménkový příznak.
- **BVS** – Instrukce skoku, kdy skok nastane, když je nastaven příznak přetečení.
- **BVC** – Instrukce skoku, kdy skok je proveden, když příznak přetečení není nastaven.
- **BHI** – Instrukce skoku, jejíž provedení nastane, když je nastaven příznak přenosu a není nastaven nulový příznak.
- **BLS** – Instrukce skoku, kdy skok je proveden, jestliže je nastaven příznak přenosu a je nastaven nulový příznak.
- **BGE** – Instrukce skoku, kdy skok je proveden, jestliže si jsou rovný znaménkový příznak a příznak přetečení.
- **BLT** – Instrukce skoku, kdy skok je uskutečněn, když si znaménkový příznak a příznak přetečení nejsou rovný.
- **BGT** – Instrukce skoku, kdy skok je proveden, jestliže není nastaven nulový příznak a znaménkový příznak a příznak přetečení si jsou rovný.
- **BLE** – Instrukce skoku, kdy skok je proveden, jestliže nulový příznak je nastaven nebo znaménkový příznak a příznak přetečení si nejsou rovný.

Také lze využít instrukci nepodmíněného skoku (**B**), u které může být délka skoku až o 2048 jednotek vpřed nebo vzad. Navíc je tu i možnost využít instrukci nepodmíněného dlouhého skoku (**BL**). Tato instrukce je jako jediná instrukce THUMB režimu 32bitová, a proto musí být využito dvou po sobě následujících 16bitových instrukcí. V nich je definována 22bitová logická adresa, díky které lze skočit až o 4194304 jednotek dopředu nebo dozadu. Tato instrukce také ukládá zpětnou adresu do registru k tomuto účelu určenému (R14).

## 2.2.2 Paměťový subsystém

Informace k této kapitole jsou čerpány z [8], [9] a [10].

Přestože procesor ARM7TDMI umožňuje použití formátu dat jak typu Little-endian, tak i Big-endian a výběr mezi nimi pomocí externího obvodu, tak tento obvod v konzole GameBoy Advance chybí a formát dat je tak vždy typu Little-endian. Přístup do paměti může být 8, 16, či 32bitový. Procesor však nepovoluje nezarovnaný přístup a v případě, že adresa je nezarovnaná, tak jsou patřičné bity ignorovány a čtení je provedeno ze zarovnané části. Z pohledu výkonnosti je důležité si uvědomit, že sběrnice připojící externí ROM paměť je pouze 16bitová a proto je rozumné používat THUMB instrukce, protože jejich operační kódy jsou na rozdíl od ARM instrukcí načteny v jednom výpočetním cyklu.

Mapování paměti je zobrazeno v tabulce 2.2.

Adresa	Popis
<b>Interní paměť</b>	
0000000 – 0003FFF	Systémová ROM paměť – BIOS
2000000 – 203FFFF	Pracovní RAM paměť umístěná na desce
3000000 – 3007FFF	Pracovní RAM paměť umístěná v čipu procesoru
4000000 – 40003FE	Mapování vstupních a výstupních registrů
5000000 – 50003FF	RAM paměť určená pro uložení palety barev
6000000 – 6017FFF	Video RAM paměť
7000000 – 70003FF	Paměť pro definici pohyblivých objektů
<b>Externí paměť</b>	
8000000 – 9FFFFFFF	Externí ROM / Flash ROM paměť s hrou. Sekce 0
A000000 – BFFFFFFF	Externí ROM / Flash ROM paměť s hrou. Sekce 1. Jedná se o zrcadlení dat ze sekce 0.
C000000 – DFFFFFFF	Externí ROM / Flash ROM paměť s hrou. Sekce 2. Jedná se o zrcadlení dat ze sekce 0.
E000000 – E0FFFFFF	Externí SRAM/Flash ROM paměť sloužící pro uložení hry.

**Tabulka 2.2 – paměťový prostor**

Systémová ROM paměť, pracovní RAM paměť v čipu procesoru, paměť pro vstupní a výstupní registry a paměť pro definici pohyblivých objektů je přístupná pomocí 32bitové sběrnice. Umožňují čtení v 8, 16 a 32bitovém přístupu. Zápis do pracovní RAM paměti procesoru a paměti pro vstupní a výstupní registry je možný v 8, 16 i 32bitovém režimu. Zápis do paměti pro definici pohyblivých

objektů je možný pouze v 16 a 32bitovém režimu. Zápis do systémové paměti není možný. Přístup do těchto pamětí je nejrychlejší a trvá jeden výpočetní cyklus.

Zbývající paměti jsou dostupné na 16bitové sběrnici. Pouze externí SRAM/Flash ROM paměť je připojena 8bitovou sběrnici. Vyjma externí SRAM/Flash ROM paměti umožňují všechny čtení v 8, 16, a 32bitovém režimu. Zápis v 16 a 32bitovém režimu umožňuje RAM paměť pro paletu barev, video RAM paměť, případně SRAM/Flash ROM paměť, pokud je použita. Běžná externí ROM paměť zápis neumožňuje. Externí SRAM/Flash ROM paměť z důvodu připojení 8bitovou sběrnici dovoluje pouze 8bitový přístup jak pro čtení, tak pro zápis.

Veškerá paměť kromě SRAM/Flash ROM je taktéž přístupná pomocí přímého přístupu do paměti (DMA) a to v 16 a 32bitovém módu. Paměti určené pro grafický výstup – tedy video paměť, paměť pro definici pohyblivých objektů a paměť s paletou barev, jsou přístupné pouze v době probíhajícího horizontálního nebo vertikálního zpětného běhu. Pro přístup během horizontální zpětného běhu pro paměť s definicemi pohyblivých objektů musí být navíc přístup povolen v registru určeném k řízení displeje (DISPCNT). Tento přístup však redukuje možný počet zobrazených pohyblivých předmětů. Pro procesor se zdají tyto paměti přístupné kdykoliv, avšak pokud k nim přistupuje ve stavu nedostupnosti, tak procesor počká až do doby, kdy budou dostupné. Pro větší dostupnost paměti pro grafický výstup jsou implementovány virtuální – skryté řádky. Těchto řádků je celkem 68 a během jejich zpracovávání je paměť taktéž dostupná.

### 2.2.3 Grafické rozhraní

Nejdůležitějším registrem, který ovládá nastavení grafického rozhraní, je kontrolní registr obrazovky (DISPCNT). Použití registru lze nejlépe ilustrovat tabulkou 2.3.

Bity	Popis
0–2	Definice používaného grafického módu.
3	Určení, zda má být použit kompatibilní mód pro hry určené pro konzolu Game Boy Color
4	Slouží k výběru ze dvou kreslících pláten. Toto nastavení je použitelné pouze v některých obrázkových módech.
5	Povolování přístupu k paměti pro definici pohyblivých objektů během horizontálního zpětného běhu. Toto nastavení snižuje počet zobrazitelných pohyblivých objektů.
6	Nastavení způsobu mapování dat pohyblivých předmětů ve video paměti. Způsob mapování může být buď jednorozměrný, či dvourozměrný.
7	Povolování rychlého přístupu ke všem částem grafické paměti.
8	Povolení využívání vrstvy 0, která je využívána pro vykreslování pozadí.
9	Povolení využívání vrstvy 1, která je využívána pro vykreslování pozadí.
10	Povolení využívání vrstvy 2, která je využívána pro vykreslování pozadí.
11	Povolení využívání vrstvy 3, která je využívána pro vykreslování pozadí.
12	Povolení vrstvy určené pro pohyblivé objekty.
13	Povolení používání okna 0.
14	Povolení používání okna 1.
15	Povolení používání okna pro pohyblivé objekty.

**Tabulka 2.3 – registr DISPCNT**

Veškerý grafický výstup je řízen pomocí oken. Ty slouží k rozdělení obrazovky na podseky. Prvním dvěma okny lze pomocí registrů WIN0H, WIN0V, WIN1H, WIN1V nastavit umístění, výšku a šířku. Umístění a velikost okna pro pohyblivé objekty je definováno viditelnými objekty na obrazovce. Každé okno může obsahovat libovolné vrstvy v libovolném počtu. Okno 0 má nejvyšší prioritu zobrazení, a proto v případě překrývání oken je zobrazen právě jeho obsah. Další v pořadí oken dle priority následuje okno 1 a nejnižší prioritu má okno pro pohyblivé objekty.

## **Vykreslovací vrstvy**

Každá vrstva určená pro vykreslování pozadí má vlastní kontrolní registr (BGXCNT). Tyto registry jsou použitelné pouze v dlaždicových módech vykreslování. V těchto registrech se nastavuje priorita vzájemného zobrazení, počáteční adresa dlaždic, počáteční adresa popisovacích dat dlaždic a hlavně velikost samotné mapy. Tedy kolik dlaždic je celkově obsaženo v pozadí. Každá vrstva totiž má registry pro horizontální (BGXHOFs) a vertikální posun (BGXVOFS), a proto může být velikost mapy větší, než zobrazovaná výšeč. Těmito registry lze tedy modifikovat souřadnice zobrazovaných bodů, avšak tato funkce je možná pouze v dlaždicových módech. Pro vrstvy 2 a 3 lze navíc využít registrů určených ke složitějším transformacím souřadnic. Lze modifikovat například sklon objektů a také jejich měřítko.

## **Způsob vykreslování**

Způsob, jakým jsou vykreslovány informace, je z velké části definován zvoleným grafickým módem. K dispozici jich je celkem 6, avšak používán může být současně vždy jen jeden. Jejich výčet lze vidět v následujícím seznamu.

- Dlaždicové módy
  - Jednoduchý dlaždicový mód
  - Dlaždicový mód s možností transformací dlaždic v jedné vrstvě
  - Dlaždicový mód s možností transformací dlaždic ve dvou vrstvách
- Obrázkové módy
  - Standardní obrázkový mód s 16bitovou paletou barev využívající plného rozlišení
  - Obrázkový mód s 8bitovou paletou barev využívající plného rozlišení
  - Obrázkový mód s 16bitovou paletou barev a se zmenšeným rozlišením

Dlaždicové módy rozdělují obrazovku na mnoho dlaždic. Naproti tomu obrázkové módy obrazovku nerozdělují a celý displej je definován jedním obrázkem. Tyto módy především ovlivňují způsob, jakým jsou uloženy informace o pozadí.

## **Dlaždicové módy**

V obou dlaždicových módech je mapa pozadí tvořena dlaždicemi o rozměrech 8x8 bodů. Samotné záznamy o vzhledu jsou dvojího druhu – popisný záznam celé dlaždice a vlastní záznam dlaždice. Ve vlastním záznamu jsou uloženy indexy do palety barev a reprezentují tak vzhled dlaždice. Tyto indexy mohou mít 4bitovou nebo 8bitovou hloubku. Hloubka souvisí i s počtem dostupných palet. S 4bitovou hloubkou je možno mít definováno 16 palet barev, kdy každá obsahuje 16 barev, zatímco s 8bitovou hloubkou můžeme mít pouze jednu paletu barev, která však obsahuje 256 barev. Záznamy jsou organizovány po řádcích odshora dolů a každý řádek zprava doleva.

Popisný záznam celé dlaždice si liší v závislosti na zvoleném způsobu zobrazení pozadí. V jednoduchém dlaždicovém přístupu je záznam velký 2B. Obsahuje pořadové číslo dlaždice, zda má být dlaždice horizontálně, nebo vertikálně převrácena a index definující paletu barev. Ten však není v případě použití jedné palety používán. V režimu povolujícím transformace pozadí je tento záznam pouze 1B velký a obsahuje pořadové číslo dlaždice. Z toho také vyplývá, že je v tomto režimu možno použít pouze jednu paletu s 256 barvami.

### ***Obrázkové módy***

V obrázkových módech je paměť strukturována jako jedno pole. Samotné uspořádání je také definováno po řádcích zprava doleva. V rámci této kategorie se módy liší jak samotným rozsahem palety barev, tak i počtem zobrazovaných řádků a sloupců (rozlišením). Paleta barev může obsahovat buď 32 768 barev, nebo 256 barev. Při použití větší palety s maximálním rozlišením je většina video paměti obsazena obrázkem a zařízení není schopno zároveň zobrazovat pohyblivé objekty. Tento mód se proto hodí pouze ke statickému zobrazení. Dále je možno použít menší paletu s maximálním rozlišením. Tento mód dovoluje určitou dynamičnost, neboť je možné používat dvě vrstvy pro vykreslování. Jedna vrstva je zobrazena a druhá se na pozadí překresluje. Poslední možností je použití zmenšeného rozlišení s maximální šířkou palety. Tento mód taktéž používá dvě vrstvy pro vykreslování stejným způsobem, jako předešlý.

### ***Vykreslování pohyblivých objektů***

Nejdůležitější součástí grafického rozhraní a her jsou však pohyblivé objekty. Na displeji může být zobrazeno současně až 128 pohyblivých objektů a je jim přidělena oddělená část video paměti. V té je uložen samotný vzhled dlaždice objektu. Velikost dlaždice je 8x8 bodů a barevná hloubka může být, stejně jako u pozadí, 4bitová nebo 8bitová. Objekty jsou tvořeny jednou či více dlaždicemi. Více dlaždicemi jsou tvořeny v případě, když chceme vytvořit pohyblivý objekt větší než pouze 8x8 bodů. Mapování více dlaždic na jeden pohyblivý objekt závisí na nastavení v kontrolního registru zobrazení (DISPCNT). První možností je jednodimenzionální uložení, kdy jsou dlaždice seřazeny sekvenčně za sebou. Druhou možností je dvojdimeznionální uložení, kdy je paměť rozdělena do řádků po 32 dlaždicích, a proto dlaždice v řádku jsou uloženy sekvenčně za sebou a index dlaždic ve sloupci se liší vždy právě o číslici 32.

Každý pohyblivý objekt je popsán třemi 16bitovými atributy. Ty jsou uloženy v paměti pro pohyblivé objekty a definují pozici objektu, index dlaždice objektu, zda se má objekt rotovat, index pro speciální registry určené k rotaci, průhlednost, velikost samotného objektu, prioritu, barevný mód, index použité palety barev a zda má být objekt vertikálně nebo horizontálně převrácen. Pro složitější transformace, jako jsou rotace a měnění měřítko, jsou k dispozici skupiny po 4 registrech. Tyto skupi-

ny lze chápat jako čtvrtý atribut pohyblivých objektů, který však není vázán přímo k jednomu pohyblivému objektu, ale může být na něj libovolněkrát odkazováno. Tento odkaz je uložen v atributu 1.

### ***Paleta barev***

Poslední částí grafického rozhraní je paleta barev. Pro ni je vyčleněna separátní část paměti a tato paměť je navíc rozdělena na část určenou pouze pro pozadí a na část určenou pouze pro pohyblivé objekty. Každá tato část může být navíc interpretována jako 16 různých palet, kdy každá z nich obsahuje 16 barev, nebo jako jediná paleta o obsahu 256 barev. Speciální význam má barva na nulté pozici v paletě. Ta má funkci transparentní barvy a proto tedy každá paleta obsahuje pouze 15, případně 255, viditelných barev. Barva na nulté pozici vrstvy 0 má navíc funkci barvy pozadí a je tedy vykreslena v případě, že žádná netransparentní barva není vykreslena. Každá barva je definována 2B a je uložena v systému zobrazení RGB. Pro každou barvu je tedy k dispozici 5 bitů a jsou definovány v následujícím pořadí – červená, zelená, modrá.

### **2.2.4 Vstupní moduly**

Nejdůležitějším vstupním modulem pro komunikaci s uživatelem je klávesnice. Ta se skládá ze 4 směrových kláves (doprava, doleva, nahoru dolů) a 6 dalších kláves. Samotnou reakci na uživatelský vstup lze realizovat dvojím způsobem. Prvním způsobem je opakované snímání registru KEYINPUT, ve kterém je uložen stav všech tlačítek. Toto snímání by se mělo opakovat každý snímek, kdy snímková frekvence displeje je 60Hz. Druhým způsobem je využití hardwarového přerušování. Povolení přerušování každé klávesy se nastavuje v registru KEYCNT. Druhý způsob je mnohem méně efektivní, neboť klávesnicové výjimky jsou spouštěny v nízko-napěťovém módu.

### **2.2.5 Časové spínače**

Nedílnou součástí konzole jsou také čtyři 16bitové inkrementální časové spínače. Každý ze spínačů je konfigurován vlastním registrem. Díky němu lze zvolit ze 4 druhů děličů frekvence ( $F/1$ ,  $F/64$ ,  $F/256$ ,  $F/1024$ ) nebo zvolit speciální mód, kdy se následující spínač inkrementuje v případě přetečení předchozího spínače. Toto nastavení samozřejmě není z logiky věci možné vynutit pro první z nich. Dále lze také nastavit, aby spínač vyvolal, či nevyvolal přerušování při svém přetečení. Nejčastějším využitím časových spínačů je řízení zvukových výstupů.

## 2.2.6 Přímý přístup do paměti

Důležitou součástí pro přenosy dat z externí paměti s hrou do zařízení je přímý přístup do paměti (DMA). K dispozici jsou až čtyři DMA kanály 0-3, přičemž každý kanál má přiřazenou prioritu zpracování. Kanál s nižší prioritou zpracování začíná svoji činnost až po dokončení přenosu kanálu s vyšší prioritou. Priorita kanálů je určena staticky – kanál 0 má největší prioritu a kanál 3 má nejmenší prioritu.

Každému kanálu jsou k dispozici 4 registry. Registr se zdrojovou počáteční adresou, registr s cílovou počáteční adresou, počet přenášených dat a kontrolní registr. V něm lze nastavit, jak má přenos dat probíhat. Lze určit, zda se má cílová adresa inkrementovat, dekrementovat nebo nechat fixní, zda se má zdrojová adresa inkrementovat, dekrementovat nebo nechat fixní, zda přenos dat má probíhat v 16bitovém nebo 32bitovém režimu, zda má být po skončení přenosu spuštěna výjimka a kdy má být přenos dat spuštěn. Přenos může být časován přímo na okamžik spuštění, na příští horizontální zpětný běh nebo na vertikální zpětný běh.

## 2.3 Mobilní platforma Android

Informace k této kapitole jsem čerpal z [11] a [12].

Mobilní platforma Android je v současnosti jedním z nejvíce rozšířených a jedním z nejrychleji se vyvíjejících mobilních platform současnosti. Tato skutečnost je do jisté míry způsobena otevřeností platformy, čímž mimochodem také přilákala mnoho dalších přispěvatelů z komerční sféry, a také operačním systémem založeným na jádru Linuxu. Popularitě navíc pomohlo využití mezivrstvy mezi aplikacemi a vlastním jádrem v podobě virtuálního stroje zvaného Dalvik. Ten totiž dává platformě větší pružnost a samozřejmě urychluje i samotný vývoj aplikace. Spouštěný kód je totiž upravený Java byte kód, takzvaný Dalvik byte kód. Samotný vývoj aplikací je tedy prováděn hlavně v jazyce Java. Části aplikace mohou být napsány nativně v jazyce C/C++, avšak tento přístup je doporučován pouze v případě výpočetně náročných operací, které nealokují mnoho paměti. Platforma Android tedy poskytuje především vyšší abstrakci vývojářům aplikací.

## Kapitola 3

# Implementace

V rámci implementace jsem se rozhodl implementovat samotný procesor, grafický subsystém a paměťový subsystém. Implementace zvukového podsystému nebyla vůbec započata. V implementaci jsem vycházel z oficiálních dokumentací [7], z neoficiálních dokumentací [9] [8] a z některých již fungujících emulátorů [10] [13]. Přehled hlavních tříd emulátoru lze vidět v příloze A.

### 2.4 Vstupní bod

Hlavní třídou emulátoru je třída `Gameboy`. Ta slouží jako vstupní bod k celému emulátoru a spojuje všechny části do jednoho celku. Stará se o synchronizaci postupů uvnitř emulátoru tak, aby žádná část nepředháněla jinou. Veškerá emulace je spouštěna a řízena následující metodou.

```
□ public void runEmulation();
```

Metoda je spouštěna v samostatném vlákně určeném pouze k emulaci. Emulace probíhá v nekonečné smyčce. V ní se vždy přidělí procesoru určitý počet výpočetních cyklů, které má vykonat. Dále se zde přepíná mezi stavy, kdy probíhá horizontální zpětný běh a kdy neprobíhá. V případě, že má nastat stav horizontálního zpětného běhu, tak spouští vykreslování jednoho řádku.

### 2.5 Procesor

Jak bylo uvedeno v kapitole 2.2.1 – Procesor ARM7TDMI, použitý procesor je schopen pracovat ve dvou režimech – v 16bitovém (THUMB) a 32bitovém (ARM). Implementace 16bitového režimu procesoru se nachází ve třídě `ThumbProcessor` a implementace 32bitového režimu ve třídě `ArmProcessor`. Z důvodu sdílení některých metod oběma režimy jsou tyto třídy potomky abstraktní třídy `AbstractProcessor`. Tato třída některé metody jak implementuje, tak i pouze deklaruje. Výhodou je navíc možnost definovat společné proměnné obou režimů staticky právě v této abstraktní třídě. Třída `AbstractProcessor` deklaruje pro své potomky následující metodu.

□ `public void run(int cycles)`

Její činností je interpretovat takový počet instrukcí, aby počet cyklů nutných k vykonání se rovnal jedinému parametru funkce.

Obě třídy implementující režimy se skládají především z metod, vykonávajících konkrétní instrukce. Zajímavější částí procesu interpretace instrukcí je jejich dekodování. V THUMB režimu (třída `ThumbProcessor`) se o dekodování instrukcí stará následující metoda.

□ `private void opRunner()`

Jelikož lze pomocí horních 8 bitů instrukce určit u většiny instrukcí exaktně, o jakou instrukci se jedná, tak je implementace dekodování návrhově čistá a lehce srozumitelná. Samotné dekodování je tak uskutečněno pomocí dvou řídicích konstrukcí `switch`. První z nich tedy rozlišuje instrukce dle horních 8 bitů a kromě instrukcí aritmeticko-logické jednotky jsou tyto instrukce již rozlišeny. Pro instrukce aritmeticko-logické jednotky jsou navíc vyhrazeny další 2 bity a podle nich lze již absolutně určit konkrétní instrukci.

Z důvodu komplexnosti instrukcí ARM režimu (třída `ArmProcessor`) bylo vhodnější rozdělit dekodování instrukcí do několika fází. O první fázi se stará následující metoda.

□ `private void opRunner();`

Metoda rozlišuje instrukce pouze dle 25. až 27. bitu. Rozděluje tedy instrukce do následujících 8 skupin:

- Instrukce pro zpracování dat s použitím registrů jako operandů.
- Instrukce pro zpracování dat s použitím přímé hodnoty jako operandu.
- Instrukce pro přístup do paměti s použitím přímé hodnoty jako operandu.
- Instrukce pro přístup do paměti s použitím registru jako operandu.
- Instrukce pro blokový přístup do paměti.
- Instrukce skoku, vyjma skoku s případným přepnutím do THUMB režimu.
- Instrukce softwarového přerušování.

Jak je zmíněno v kapitole „Instrukční sada ARM režimu“, tak před samotným provedením instrukce lze modifikovat operandy posuny nebo rotacemi. Pro tento účel je ve třídě implementována následující metoda plnící roli válcového posouvače.

□ `private int shifting(int operand);`

Operandem této metody je spodních 12 bitů instrukce. Tyto bity totiž definují způsob, jakým má být provedena modifikace. Návrátovou hodnotou metody je právě zmodifikovaná hodnota.

V případě instrukcí, které jako operand používají přímou hodnotu, se této metody nevyužívá. Tyto instrukce totiž nemají stejnou interpretaci spodních 12 bitů, a proto nelze využít obecnějšího přístupu.

První dvě uvedené skupiny jsou si zpracováním velmi podobné. Liší se pouze ve způsobu a možnostech využití posunů operandu. Po případném posunu operandu má jejich dekódování na starosti následující metoda.

```
□ private void opNormalDataProcess(int operand);
```

Jejím jediným parametrem je operand modifikovaný případnými předchozími posuny. Metoda následně provádí dekódování pomocí 4 bitů instrukce a zároveň již tyto instrukce i vykonává.

Podobně je postupováno i u následujících dvou skupin určených pro přístup do paměti. Jejich konkrétní dekódování je prováděno následující metodou.

```
□ opNormalLoadStore(int offset);
```

Metoda dekóduje a také sama provádí instrukce přístupu do paměti. Jediný parametr metody určuje logickou adresu (offset), který má být při přístupu do paměti použit.

Další skupiny již mají každá vlastní metodu, která se stará o druhou fázi dekódování a následně i o provedení instrukce.

## 2.6 Paměťový subsystém

Veškeré paměťové operace jsou v režii třídy `Memory`. Jelikož je samotný adresový prostor rozdělen na několik samostatných částí, tak byl v implementaci zvolen podobný přístup, a proto je každé části přiděleno separátní pole bytů. Každá adresa, na které má být prováděna paměťová operace, je maskována tak, aby nedošlo ke čtení mimo pole. Třída `Memory` nabízí tyto metody pro čtení z určené adresy.

- `public byte read8(long address);`
- `public short read16(long address);`
- `public long read32(long address);`

Třída `Memory` také disponuje metodami pro zápis do paměti.

- `public void write8(byte value, long address);`
- `public void write16(short value, long address);`
- `public void write32(int value, long address);`

Tyto metody jsou používány především samotným procesorem. Samotné sekce paměti lze identifikovat dle 24. až 28. bitu adresy. Tato 4bitová hodnota tedy slouží jako identifikátor každé sekce, nicméně je využito pouze 10 identifikátorů. Zbylé identifikátory jsou nevyužity a jsou tedy nedefinovány. Čtení z nedefinované adresy však není nevalidní operací a jejím výsledkem je obsah programového čítače.

Každý přístup do paměti spotřebuje různý počet výpočetních cyklů. Tento problém je taktéž v režii třídy `Memory` a je řešen pomocí jedné proměnné. Ta je v případě, že si procesor vyžádá tuto hodnotu, vynulována.

## 2.7 Grafické rozhraní

Hlavní třídou grafického rozhraní je třída `VideoManager`. Tato třída vytváří spojovací článek mezi třídou starající se o samotné vykreslování (třída `GraphicCore`) a hlavní třídou emulátoru `Gameboy`. Součástí třídy `VideoManager` jsou následující metody.

□ `public void renderSingleLine();`

Tato metoda vynucuje vykreslení jednoho řádku displeje. Samotné vykreslení je delegováno na třídu, která implementuje rozhraní `IGraphicCore`.

□ `public void enterHBlank(AbstractProcessor processor);`

Tato metoda simuluje vstup do fáze zpětného horizontálního běhu. Mění vstupně-výstupní registr sloužící pro detekci stavu displeje (`DISPSTAT`). Dále spouští ty DMA přenosy, které jsou časovány právě na dobu zpětného horizontálního běhu.

□ `public void leaveHBlank(AbstractProcessor processor);`

Tato metoda simuluje ukončení horizontálního zpětného běhu. Mění jak vstupně-výstupní registr sloužící pro detekci stavu displeje (`DISPSTAT`), tak i registr detekující počet vykreslených řádků (`VCOUNT`). Dále řídí jak vstup, tak i výstup systému z vertikálního zpětného běhu.

□ `private void enterVBlank(AbstractProcessor processor);`

Tato metoda simuluje vstup do fáze vertikálního zpětného běhu. Operuje s registrem detekujícím stav displeje (`DISPSTAT`). V této fázi je celý displej vykreslen a měl by se zobrazit. Dále se v této fázi spouští ty DMA přenosy, které jsou časovány právě na dobu zpětného vertikálního běhu.

□ `private void leaveVBlank(AbstractProcessor processor);`

Tato metoda simuluje ukončení vertikálního zpětného běhu. Operuje s registrem detekujícím stav displeje (`DISPSTAT`). V této fázi se také vynucuje aktualizace registrů dle uživatelských vstupů.

Samotné jádro vykreslování je implementováno ve třídě `GraphicCore`. Součástí této třídy je dvoudimenzionální pole, které reprezentuje celý displej. Jelikož jsou barvy v konzole definované pomocí RGB modelu jako 5bitové hodnoty, je nutné tuto reprezentaci převést na model, který by byl použitelný na platformě Android. Tento převod je realizován pomocí vyhledávací tabulky, kdy jejím vstupem je barevná hodnota definovaná konzolou a jejím výstupem je barevná hodnota, ve které je každá barva definována na 8 bitech a navíc je zde další 8bitová hodnota, která určuje průhlednost. Tento barevný model má vyhrazeno 32 bitů pro každý bod obrazu a je běžně podporován platformou. Třída `GraphicCore` implementuje rozhraní `IGraphicCore`, které definuje tuto metodu.

```
□ public void renderSingleLine(int line);
```

Metoda se stará o vykreslování řádků do interního pole. Její jediný parametr určuje řádek, který má být vykreslen.

Ve třídě `GraphicCore` je implementováno mnoho metod, které jsou vyvolány pouze při použití konkrétních zobrazovacích módů. Tyto metody si jsou však velmi podobné, co se funkčnosti týče, jelikož i zobrazovací módy konzole si jsou v lecčems podobné. Jejich sloučení by však z důvodu přehlednosti nebylo vhodné.

## Kapitola 4

# Testování funkčnosti emulátoru

K ověřování funkčnosti jsem již během prvních fází vývoje využíval komparaci výsledků s již fungujícími řešeními [10] a [13]. Tyto řešení mají v sobě zabudovaný disassembler instrukcí a poskytují možnost si nechat zobrazit libovolný registr, libovolnou část paměti a libovolný vstupně-výstupní registr a to vše přehledně a i s případnými doplňujícími popisy konkrétních bitů.

Jako prostředek testování jsem tedy použil několik volně dostupných her určených právě pro konzolu GameBoy Advance. Těmito hrami byly především populární hra Zelda – A Link to the Past a další populární hra Pokemon Emerald. Pro testování funkčnosti procesoru jsem do tříd implementoval navíc metody určené ke krokování instrukcí. Dále pro účely testování je možné programově vložit adresy instrukcí, na kterých má být pozastaven běh emulace tak, aby během pozastavení mohla proběhnout analýza stavu. Testování funkčnosti grafického rozhraní samostatně bylo velmi improvizované. Části kódu jsem vždy vyjmul z kontextu a nad množinou smyšlených dat, která reprezentovala obsahy video paměti a některých vstupně-výstupních registrů, jsem zkoušel, zda dojde k očekávanému výsledku.

Rychlost emulátoru jsem zkoušel pomocí mírně modifikovaného kódu hry Zelda – A Link to the Past. Fragment tohoto kódu lze vidět v příloze B. V tomto fragmentu kódu jsem nechal proběhnout celkem 100 000 instrukcí a měřil dobu trvání. Měření bylo prováděno na osobním mobilním zařízení Huawei Ascend G300, což je v současnosti průměrně výkonné zařízení. Vykonání tohoto fragmentu trvalo okolo 32 000 ms. Z těchto hodnot lze usuzovat, že rychlost emulace procesoru není optimální, neboť dle výrobce je procesor schopen vykonat až 130 milionů instrukcí za sekundu. Tato hodnota samozřejmě musí být brána s rezervou, avšak výkon emulátoru ani zdaleka nedosahuje této rychlosti, protože je schopen vykonat maximálně kolem 3125 instrukcí za sekundu.

Dále je nutné vzít v úvahu, že tvůrci her se často snažili získat maximum z daných zdrojů, a tak využívali veškerých možností, jak toho dosáhnout. Tyto způsoby jsou často špatně zdokumentované a v některých případech se může jednat i o chyby přístroje.

Nepříjemným problémem byla také absence bezznaménkového číselného typu v jazyce Java. Tento neduh výrazně ztížil testování, neboť je nutné u aritmetických operací rozlišovat, zda má být hodnota interpretována jako znaménková nebo bezznaménková.

Díky krokování instrukcí mohu prohlásit, že interpretace je z velké části funkční. Díky testování grafického rozhraní lze předpokládat, že grafický výstup bude určitým specifictějším způsobem fungovat. Rozhodně však nelze předpokládat naprosto věrohodný grafický výstup. Bohužel však nemůžu o emulátoru jako celku prohlásit, že je funkční.

## Kapitola 5

# Závěr

Cílem práce bylo navrhnout funkční emulátor herní konzole GameBoy Advance a dále ho zprovoznit na některé mobilní platformě. Celý návrh byl rozdělen do 3 částí – části starající se o přístup do paměti, části, která emuluje samotný procesor a části, která se stará o vykreslování, neboť tyto 3 části jsou základními stavebními kameny konzole.

Očekávaným výsledkem práce samozřejmě nemohl být emulátor takových kvalit, aby mohl konkurovat již hotovým emulátorům, neboť ty jsou vyvíjeny vícečlennými týmy po dobu několika let. Zcela nepopíratelným přínosem je však ucelený popis základních prvků a způsobu fungování konzole. Dále implementace emulátoru procesoru ARM7TDMI v jazyce Java jistě stojí za pozornost.

To, že se emulátor nakonec nepovedlo posunout do fáze, aby byl z větší části funkční, je samozřejmě negativum. Za příčinou tohoto stavu jistě stojí rozhodnutí vytvořit vlastní emulátor a nepoužít tak žádnou volně dostupnou implementaci procesoru. Mým cílem však bylo proniknout do hloubky fungování této konzole, a proto jsem nechtěl této možnosti využít a tak jsem se pouze inspiroval některými implementacemi.

Zkoušení části, která se stará o emulaci procesoru, bylo prováděno na privátním zařízení fungujícím na platformě Android. V rámci toho jsem nabyl dojmu, že i přesto, že je doporučováno vyvíjet aplikace pro platformu Android pouze v Javě a využívat tak bonusy virtuálního stroje, tak rychlost emulace procesoru byla velmi nedostačující. Toto samozřejmě mohlo být způsobeno i nižší výkonností zařízení, případně i vnitřní chybou, avšak hlavní příčinou dle mého je právě využití virtuálního stroje. Pro dosažení vyšší výkonnosti by tak bylo vhodné implementovat co největší část aplikace nativně. Dále by bylo vhodné grafické rozhraní plně hardwarově akcelarovat.

## Citovaná literatura

1. **Svatopluk Vít.** Emu- co? Aneb lehký úvod do emulace. *Linux Expres*. [Online] [Citace: 20. Březen 2013.] <http://www.linuxexpres.cz/praxe/emu-co-aneb-lehky-uvod-do-emulace>.
2. **Blass, Andreas and Gurevich, Yuri.** Algorithms: A Quest for Absolute Definitions. [Online] 2003. [Cited: Duben 5, 2013.] <http://research.microsoft.com/en-us/um/people/gurevich/Opera/164.pdf>.
3. **Nintendo Co.** Consolidated Sales Transition by Region. [Online] 2013. [Citace: 21. Březen 2013.] [http://www.nintendo.co.jp/ir/library/historical\\_data/pdf/consolidated\\_sales\\_e1212.pdf](http://www.nintendo.co.jp/ir/library/historical_data/pdf/consolidated_sales_e1212.pdf).
4. —. Technical specification. *web archive*. [Online] [Citace: 22. Březen 2013.] <http://web.archive.org/web/20071014004636/http://www.nintendo.com/techspecgba>.
5. **ARM.** [Online] [Citace: 22. Březen 2013.] <http://infocenter.arm.com/help/topic/com.arm.doc.ddi0210c/index.html>.
6. —. [Online] [Citace: 23. Březen 2013.] <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0210c/ch02s06s01.html>.
7. —. Architecture reference manual. [Online] [Citace: 25. Březen 2013.] sekce A3-4. [https://silver.arm.com/download/ARM\\_Architecture/AR550-DA-70002-r0p0-00rel0/DDI%2001001.pdf](https://silver.arm.com/download/ARM_Architecture/AR550-DA-70002-r0p0-00rel0/DDI%2001001.pdf).
8. GBATEK reference. [Online] [Citace: 26. Březen 2013.] <http://nocash.emubase.de/gbatek.htm>.
9. **Happ, Tom.** CowBite Virtual Hardware Specifications. [Online] [Citace: 2. Duben 2013.] [http://www.zap.pe.kr/projects/arm7tdmi/docs/gba\\_spec.htm](http://www.zap.pe.kr/projects/arm7tdmi/docs/gba_spec.htm).
10. **Linscott, Gary.** Gameboy Advance Emulator. [Online] [Citace: 2. Duben 2013.] <http://www.forwardcoding.com/projects/gbaemu.html>.
11. **Ujbányai, Miroslav.** *Programujeme pro Android*. Praha : Grada, 2012. 978-80-247-3995-3.
12. **Google.** About. *Android*. [Online] [Citace: 20. Duben 2013.] <http://developer.android.com/about/index.html>.
13. Visual Boy Advance. [Online] [Citace: 1. Duben 2013.] <http://vba.ngemu.com/downloads.shtml>.

**Příloha A**

# **Diagram tříd**



## Příloha B

# Fragment kódu

```
08000000: EA00002E    b 0x080000C0
080000C0: E3A00012    mov r0, #0x00000012
080000C4: E129F000    msr cpsr, r0
080000C8: E59FD028    ldr sp, =#0x03007FA0
080000CC: E3A0001F    mov r0, #0x0000001F
080000D0: E129F000    msr cpsr, r0
080000D4: E59FD018    ldr sp, =#0x03007F00
080000D8: E59F101C    ldr r1, =#0x03007FFC
080000DC: E28F0020    add r0, pc, #0x00000020
080000E0: E5810000    str r0, [r1]
080000E4: E59F1014    ldr r1, =#0x0800B361
080000E8: E1A0E00F    mov lr, pc
080000EC: E12FFF11    bx  r1
0800B360: B570        push {r4, r5, r6, lr}
0800B362: F000        bl
0800B364: F8AF        bl 0x0800B4C4
0800B4C4: B500        push {lr}
0800B4C6: B082        add sp, #-0x008
0800B4C8: F000        bl
0800B4CA: FA0C        bl 0x0800B8E4
0800B8E4: B510        push {r4, lr}
0800B8E6: 4A1C        ldr r2, =#0x00040400
0800B8E8: 2100        mov r1, #0x00
0800B8EA: 8011        strh r1, [r2]
0800B8EC: 481B        ldr r0, =#0x02020400
0800B8EE: 8001        strh r1, [r0]
0800B8F0: 481B        ldr r0, =#0x00B00400
0800B8F2: 8001        strh r1, [r0]
0800B8F4: 481B        ldr r0, =#0xC5FF0400
0800B8F6: 8001        strh r1, [r0]
0800B8F8: 8011        strh r1, [r2]
0800B8FA: 491B        ldr r1, =#0x7FFF0000
0800B8FC: E7F4        b 0x0800B8E8
```