Szegedi Tudományegyetem

Informatikai Intézet

Az emulátor fejlesztési folyamat bemutatása a Nintendo Game Boy konzolon

Diplomamunka

Készítette:
Krizsák Tibor
informatika szakos
hallgató

Témavezető:
Dr. Tanács Attila
egyetemi adjunktus

Szeged
2018

Tartalomjegyzék

Feladatkiírás

A Nintendo Game Boy egy 1989-ban bemutatott, 8 bites kézi videojáték-konzol. A konzolban egy Zilog Z80 (az Intel 8080 utódja) processzor működik, kiegészítve néhány specifikus utasítással. A Game Boy emulátor fejlesztésének bemutatása során a CPU utasításait, a GPU renderelésének működését, a memóriakezelést, és a megszakítás-vezérlést kell implementálni. Ahhoz, hogy ezek megfelelően működjenek, a CPU frekvenciájára, illetve a képfrissítési gyorsaságra is tekintettel kell lenni.

A játékkonzol emulátorok fejlesztése során a szűk, ezzel foglalkozó fejlesztői réteg kialakított egy egyértelmű, jól követhető fejlesztési folyamatot. A dolgozatban ezen keresztül kerüljenek bemutatásra a Nintendo Game Boy emulátor fejlesztési fázisai.

A tesztelés a más Zilog Z80 emulátor fejlesztők által készített teszt ROM-okon történjen.

Tartalmi összefoglaló

– **A téma megnevezése:**
Egy emulátor fejlesztési fázisainak bemutatása a Nintendo Game Boy hardveren keresztül, Rust nyelven implementálva.

– **A megadott feladat megfogalmazása:**
A feladat egy Nintendo Game Boy emulátor implementálása, és fejlesztési fázisainak bemutatása. A bemutatás során a CPU utasításait, a GPU renderelésének működését, a memóriakezelést, és a megszakítás-vezérlést kell érinteni, illetve az egyéb kisebb, de a működéshez elengedhetetlen megoldások is megemlítésre kerülnek. Ahhoz, hogy ezek megfelelően működjenek, a CPU frekvenciájára, illetve a képfrissítési gyorsaságra is tekintettel kell lenni.

– **A megoldási mód:**
Az emulátor fejlesztő közösség által összegyűjtött, – *reverse-engineered* – információkra, illetve a processzor gyártója által kiadott technikai dokumentációra hagyatkozva felépítettem és implementáltam a CPU struktúráját, utasításkészletét, majd a többi modult, részegységet. Meghatároztam a modulok közti kommunikációt, időzítéseket, adatfolyamot. A videojáték-, illetve teszt ROM-ok byte-jait sorra beolvasva az emulátor meghatározza a megfelelő műveletet, meghatározott időközönként renderel, illetve kezeli a megszakításokat.

– **Alkalmazott eszközök, módszerek:**
Az emulátor Linux rendszeren, Rust nyelven került implementálásra, a `rustc` fordító, illetve a `cargo` package manager segítségével. A rendereléshez a `minifb` libraryt használtam, ami egy nagyon egyszerű framebuffer használatát teszi lehetővé. A fejlesztésre került egy debugger eszköz, illetve egy memóriatérkép eszköz is, ami nagyban megkönnyítette a hibakeresést.

– **Elért eredmények:**
Az implementált emulátor képes futtatni Memory Banking nélküli videojáték ROM-okat, az inputra az elvárásoknak megfelelően reagálva. A processzor műveletek és a renderelés az eredeti konzollal megegyező eredményt adnak. A közösségi Game Boy teszt ROM-ok szinte mindegyikét sikerrel végrehajtja.

– **Kulcsszavak:**
Nintendo, Game Boy, emulátor, fejlesztés, Rust

Bevezetés

A számítástechnikában az emuláció fogalma nem új keletű. Különböző területeken, különféle problémák megoldására használnak emulátorokat, ugyancsak különböző okokból. A nyomtatóktól kezdve, a DOS-kártyákon keresztül, a többmagos rendszertervezésen át egészen a videojáték konzolokig terjed a paletta - nem túlzás azt állítani, hogy az emulátorok ott vannak a mindennapjainkban.
Ezen diplomamunka a videojáték konzolok emulátorainak fejlesztésére fókuszál. Többféle cél állhat a háttérben, ha valaki ilyen emulátor fejlesztésére adja a fejét: a régi hőskorbeli konzolok digitális megőrzése vagy életre keltése, későbbi szoftverfejlesztés az emulált hardveren, esetleg hobbiként. Az utóbbi évek tendenciája azt mutatja, hogy ez utóbbi ok egyre gyakoribb - az emulátor fejlesztői közösség napról napra nagyobb és aktívabb, szokások és kisebb fejlesztői folklór alakult ki az emulátor készítését illetően - a dolgozat ennek bemutatására helyezi a hangsúlyt.
Az emulátor fejlesztés szemléltetése Nintendo Game Boy kézi videojáték konzolon keresztül fog történni, amely a maga idejében egy igazán sikeres konzol volt, és tulajdonképpen kultusz épült köré. A 8 bites architektúrájából adódóan kevéssé bonyolult felépítéssel rendelkezik, népszerűségéből adódóan jól dokumentált, így az emulálásának implementációjához nincs szükség túl sok *reverse-engineering* gyakorlatra.
A dolgozat első néhány fejezetében az emulátorokról, a Nintendo Game Boy hardveréről, specifikációjáról, illetve a későbbi fejlesztés workflow-járól fog szó esni. Ezekben a fejezetekben van megfogalmazva, illetve leírva az, hogy pontosan mi az az emulátor, milyen hardver emulációjáról van szó, és hogy az emuláció teljes implementálásáig milyen pontokon keresztül vezet az út. A következő nagyobb logikai egység az implementáció. Ennek részeként először bemutatásra kerülnek az alkalmazott eszközök, technológiák, majd az emulátor pontos és elvárt specifikációjának leírását az igazi implementációs szakasz követi.
A processzor modellezése a regiszterek, flagek, és egyéb jellemzők megtervezésével kezdődik, majd következő lépésként az utasításkészlet megvalósításával folytatódik. A CPU-hoz szorosan kapcsolódó memória ez után kerül tárgyalásra. A memória ismertetése után az időzítők, majd a PPU felépítése és működése szerepel. Az implementáció ezen pontján a Boot ROM már futtathatóvá válik, erről is esik majd néhány szó. A fejlesztési részt a joypad jellemzői és megoldásai zárják.
A dolgozat zárásaként bemutatásra kerül az emulátor használata, illetve a fejlesztésből adódó dependenciák, majd végül a teszt ROM-ok jellemzői, futtatásuk, és a futtatási eredményeik.

1. fejezet

Az emulátorok és a Nintendo Game Boy

1.1. Az emulátorokról

Az utóbbi évtizedekben végbement – és jelenleg is tartó – technikai fejlődés következményeként rendkívül gyors a technológiai elavulás. Ennek következményeképp az eszközök életciklusa megrövidül, értékük rohamosan csökken. Gyakran előfordul azonban, hogy szükség van a régi *legacy* rendszerekre, vagy elengedhetetlen a visszafelé kompatibilitás, esetleg szeretnénk az adott hardvert a számítástechnikai jelentősége miatt valamilyen formában megőrizni, használhatóvá tenni. Az emulátorok ezekre a problémákra igyekszenek megoldást kínálni – persze rendkívül sok egyéb felhasználási terület mellett.

1.1.1. Az emulátor fogalma

Definíció szerint olyan hardvert vagy szoftvert nevezünk **emulátornak**, amely lehetővé teszi, hogy egy számítástechnikai rendszer (szokás ezt *host*-nak nevezni) úgy viselkedjen, mint egy másik számítástechnikai rendszer (ez pedig a *guest*). Jellemzően az emulátor a *host* rendszer számára teszi lehetővé olyan szoftver futtatását vagy periféria használatát, amely a *guest* rendszerhez lett kifejlesztve. Röviden megfogalmazva az emulátor egy olyan hardver vagy szoftver, ami egy másik eszközt vagy programot emulál, imitál.

1.1.2. Az emulátorok típusai

Az emulátorok többsége csak a hardver architektúrát emulálja – ha operációs rendszer vagy egyéb szoftver is szükséges az emuláláshoz, akkor azt is biztosítani kell. Ebben az esetben az operációs rendszert és a szoftvert *interpretálni* (értelmezni) fogja az emulátor. A gépi kód *interpreteren* kívül azonban az emulátornak tartalmaznia kell a *guest* hardver minden lehetséges jellemzőjét, és viselkedését is virtuálisan: ha például egy adott memóriahelyre való írás befolyásolja azt, hogy mi jelenik meg a képernyőn, úgy azt is emulálni kell. Habár lehetne az emulációt extrém részletességgel, atomi szinten végezni – például az áramkör adott részei által kibocsátott pontos feszültségingadozás emulálásával, stb. – , ez egyáltalán nem gyakori, az emulátorok általában megállnak a dokumentált hardver specifikáció, és digitális logika szimulációjának szintjén.
Némely hardver hatékony emulálásához extrém pontosság szükséges: az óraciklusokat, nem dokumentált jellemzőket, kiszámíthatatlan analóg elemeket, és *bugokat* mind-mind

implementálni kell. A klasszikus otthoni számítógépek esetében (például a Commodore 64) ez hatványozottan igaz, mert az ezekre a hardverekre írt szoftverek gyakran kihasználtak alacsony szintű programozási trükköket, melyeket főként a videojáték programozók és a *demoscene*[1] fedeztek fel.
Ezzel szemben léteznek azonban olyan platformok, amelyek alig használják a közvetlen hardver elérést, jó példa erre a PlayStation Vita. Ezekben az esetekben elég egy kompatibilitási réteget megvalósítani, amely a *guest* rendszer rendszerhívásait fordítja le a *host* rendszer hívásaira.

1.1.3. Az emulátorok jövője

A videojáték-konzol emulátorok világa, illetve az emulátor fejlesztő közösség helyzete igen érdekes. Az egyik oldalról megvizsgálva azt tapasztalhatjuk, hogy egyre nagyobb népszerűségnek örvendő területről van szó. Ami a másik oldalt illeti – a helyzet nagyon homályos. Újabb és újabb konzolok jelennek meg, egyre rövidebb életciklussal és egyre bonyolultabb architektúrával. Jól mutatja ezt a PlayStation 3 példája: 12 éve, 2006-ban jelent meg, és tökéletes emulátor még nem készült hozzá. A közösség nem tudja tartani a tempót a bonyolultság, és a rövid életciklusokból adódó szoros határidők miatt.
Sokak szerint viszont a jövőben nemhogy nehezebb, hanem inkább könnyebb lesz az emuláció: véleményük szerint a hardver emulációja nem lesz könnyű, viszont az utóbbi években nagyon sokat javult a szoftverek minősége és tisztasága egyaránt. A játékfejlesztők rá vannak kényszerítva az API-k (*Application Programming Interface* – alkalmazásprogramozási interfész) használatára a hardver *bugjainak* kihasználása és a trükközés helyett, és ez lehetőséget adhat az API-kon alapuló emuláció elterjedése felé.
Fontos megemlíteni egy 2010-ben indult közösségi projektet, a RetroArch-ot, amely a videojáték konzol emulátorok számára biztosít egy prezentációs réteget, ún. *frontend*-et, amely egybefogja, és használhatóvá, futtathatóvá teszi a vele kompatibilis emulátorokat. Ez a megoldás nagyban megkönnyíti a felhasználók életét, hiszen több tucatnyi rendszer emulátorát érhetik el egyetlen felületen keresztül, és a fejlesztők számára is jelent egy enyhe szabványosítási törekvést.
Ahogy a fenti két vélemény, és a RetroArch példája is mutatja, sokan sokféleképpen vélekednek az emulátorfejlesztés jövőjéről, nem beszélve a frissen induló közösségi projektekről – szinte biztosan kijelenthető, hogy ez a terület nem fog egyhamar megszűnni.

1.2. Nintendo Game Boy

Egy emulátor fejlesztési folyamatának bemutatására a Nintendo Game Boy tökéletes példa több szempontból is. Elsősorban széleskörűen ismert, ebből adódóan az emulátor fejlesztői közösség által is jól dokumentált, a hardver szinte az utolsó részletig vissza lett fejtve. Ezekből a dokumentációk tehát jó kiindulási alapot nyújtanak. Az is fontos szempont, hogy a hardver a 8 bites érából származik, ami szinte garantálja az egyszerűbb architektúrát (ez persze relatív), így a könnyebb implementálhatóságot. Szintén megemlítendő, hogy a fejlesztői közösség által készített teszt ROM-ok nagyban segítik a

[1] A *demoscene* nemzetközi underground számítástechnikai szubkultúra, amelynek célja különböző számítógépes digitális művészeti alkotások (*demók*) készítése.

hibakeresést, verifikációt.

1.2.1. Története, jelentősége

1.1. ábra. *A Nintendo Game Boy logója*

A Game Boy egy Nintendo által gyártott hordozható videojáték konzol, amit a nagyközönség számára 1989-ben mutattak be. Ez volt a gyártó első 8 bites kézi konzolja, amihez a játékokat cserélhető kazetta formájában (angolul *cartridge*) lehetett megvásárolni.
Az okos marketingnek, és a jó Nintendo *brand*-nek köszönhetően a Game Boy kora legsikeresebb kézi konzolja lett, annak ellenére, hogy a versenytársaihoz (Atari Lynx, Sega Game Gear) mérten elavult technológiát használt. Ez egyben azt is jelentette, hogy a Game Boy-ban használt alkatrészek olcsóbbak, ismertebbek és kiforrottabbak voltak, mint a riválisoké. A tervezők alapgondolata az volt, hogy régebbi technológiát használnak fel innovatív módon. A konzol sikerét az olcsósága, az akkumulátor időtartama, és a platformon elérhető rengeteg játék mennyisége és minősége koronázta meg. Az 1997-ig értékesített 60 milliós példányszám a Game Boy-t a gyártó egyik legsikeresebb termékévé tette. A készülék jellegzetes logója a 1.1-es ábrán látható.

1.2.2. Hardver specifikáció

A hardver specifikációját két logikai egységre lehet osztani: a Game Boy hardverére és a *cartridge* hardverre. Ugyan ezek együtt alkotnak egészet, hiszen egyik sem használható a másik nélkül, ám technikailag két különálló egységről beszélhetünk.

Game Boy

A konzol külseje, és kezelőszervei a 1.2-es ábrán figyelhetők meg, az általa tartalmazott hardver elemek pedig a következők:

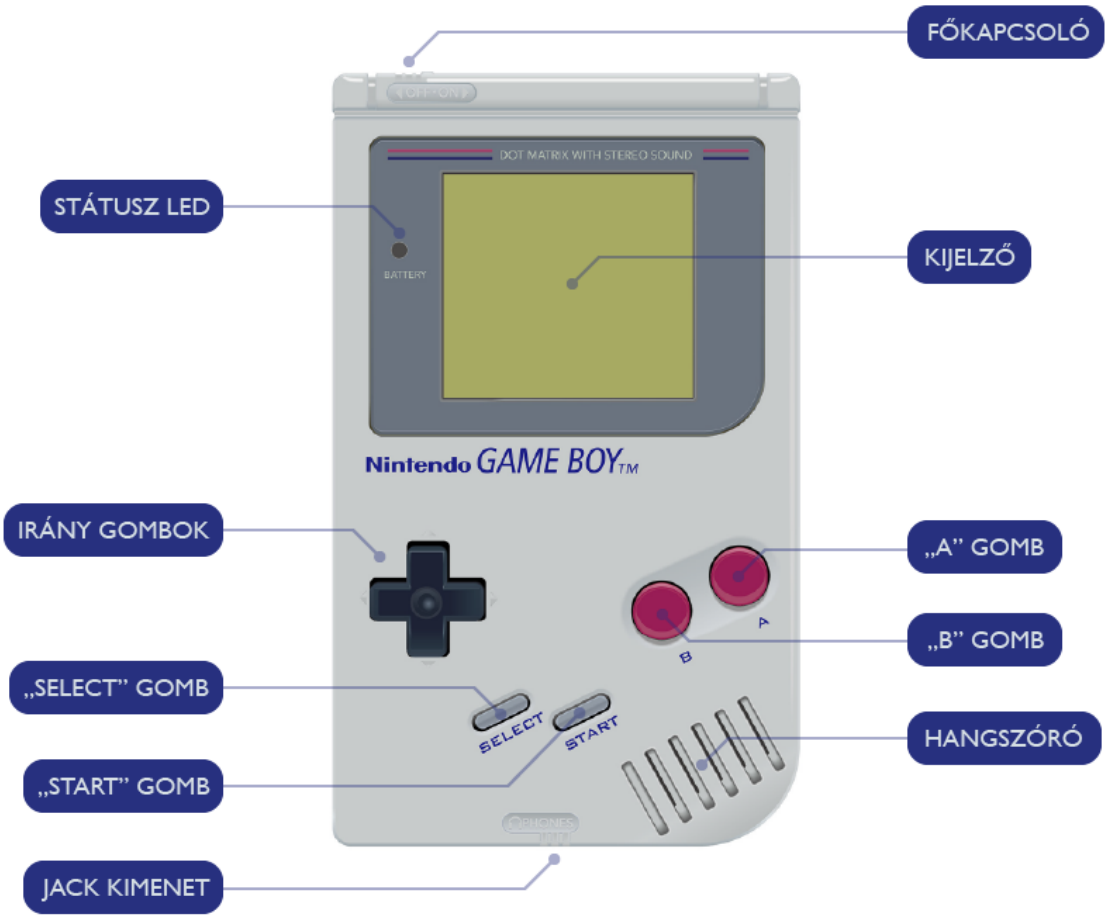
- **CPU**: a 8 bites *Zilog Z80*-as CISC-processzor architektúrán alapuló – annak utasításkészletén enyhén módosított változata – *Sharp LR35902*.
- **RAM**: 8 kB beépített S-RAM
- **VRAM** (video memória): 8 kB beépített
- **ROM**: 256 Byte (Boot ROM-nak fenntartva)

– **Hang**: 2 négyszögjel generátor, 1 programozható 32 mintás 4 bites PCM hullám, 1 fehér zaj, és egy audio bemenet a kazettából. (A külső kazetta bemenetet soha egy piacra dobott játék sem használta.) A jack kimeneten keresztül sztereó hangot ad.

– **Kijelző**: 166 × 144 pixel felbontású LCD kijelző, mérete átlósan 66 mm.

– **Színpaletta**: 4 árnyalat 2 biten tárolva – világos zöldtől a sötét zöldig.

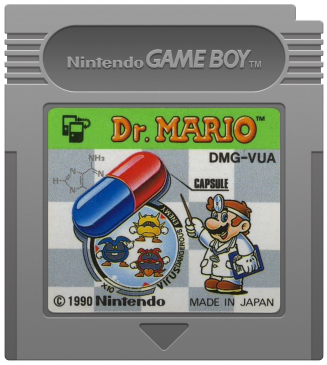
– **Tápellátás**: 4 db AA elem, amely megközelítőleg 14-35 óra játékidőt biztosít.



1.2. ábra. *A Game Boy és részei*

Cartridge

A konzolhoz tartozó játék kazettákat a konzol hátuljába kellett címkével kifelé fordítva becsúsztatni. Ezek a kazetták jól felismerhetőek voltak a jellegzetes (nagyrészt) szürke színükről, illetve az elejükre ragasztott, az adott játékot ábrázoló címkéjükről, ami a 1.3-es ábrán is megfigyelhető. A Game Boy-hoz több típusú kazetta volt forgalomban, melyet az indokolt, hogy némely játék nagyobb erőforrást igényelt a futásához. A konzolnak köztudottan kicsi volt a memória mérete, így a játékfejlesztőknek különféle trükköket kellett bevetniük ahhoz, hogy a játékaikat futásra bírják. Erre a problémára a *Memory Bank Controller* alkalmazása volt a megoldás, ennek használatával a fejlesztők számára nagyobb ROM, illetve *MBC* verziótól függően nagyobb RAM volt elérhető. Az *MBC* részleteiről és típusairól az egyik későbbi fejezetben lesz szó.



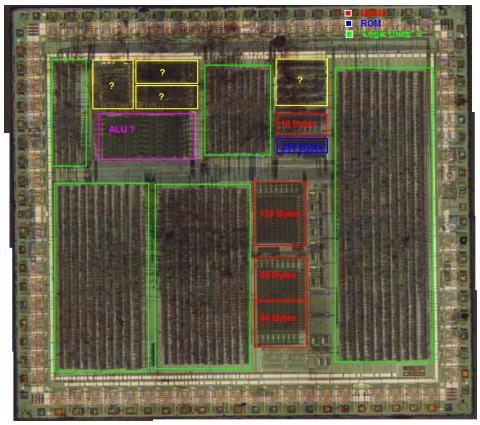
1.3. ábra. *A Dr. Mario játék kazettája*

A kazetták többségében volt egy `CR2025`-ös típusú gombelem is, ami az elmentett játékállások tárolásából adódó erőforrás-ellátásért felelt. Az elem viszont nem tartott örökké – így mikor hosszú idő után ugyan, de lemerült, az összes mentett állás elveszett.
További érdekesség még a *cartridge*-ekkel kapcsolatban a Nintendo által kifejlesztett korabeli (de meglepően hatékony) másolásvédelmi és *homebrew*[2]-fejlesztőket kizáró mechanizmus. Annak érdekében hogy a kazettákat ne lehessen lemásolni és így terjeszteni, illetve a gyártóval kapcsolatba nem lépő hobbi fejlesztők ne tudjanak játékot kiadni a platformra, a Nintendo szokatlan, de hatékony megoldást választott. Amikor a felhasználó behelyezi a kazettát a Game Boy-ba, és bekapcsolja azt, akkor a Boot ROM lefutását követően a játék csak akkor indul el, ha a játék kódjában megtalálhatók a Nintendo logót alkotó byte-ok. Ha ez hiányzik, akkor a játék nem fog futni. A trükk az, hogy ugyan a törvény nem tiltja, hogy játékokat fejlesszenek a hobbi fejlesztők a konzolra, viszont a Nintendo logó felhasználását szerzői jogi törvények védik. Így, ha a hobbi fejlesztő terjesztené a játékát, akkor be kell ágyaznia a Nintendo logót a kódba, amivel viszont szerzői jogi szabályokat sért. Ezzel a Nintendo elérte, hogy a konzolra kiadott játékok minősége magas legyen, hiszen minden játék kiadásáról végső soron ők döntöttek.

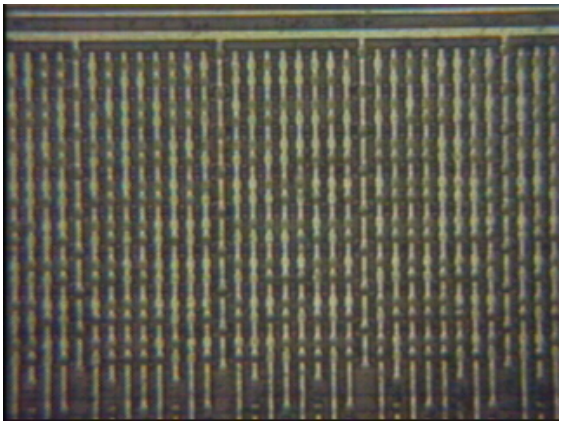
1.2.3. Boot ROM

A Nintendo Game Boy Boot ROM-ja végzi a hardver elindulását követő inicializáló folyamatokat, illetve az előző részben említett másolásvédelmi eljárást. A ROM pontos tartalma egészen 2003-ig ismeretlen volt – ekkor viszont egy *Neviksti* nevű felhasználó publikálta a *cherryroms.com* fórumára a program teljes változatát. *Neviksti* azt is leírta, hogy hogyan sikerült visszafejtenie a kódot: a Game Boy processzor chip tetejének leszedése után mikroszkóppal megvizsgálta az áramkört, majd amint megtalálta a Boot ROM

[2] A *homebrew* fogalmat azokra a videojátékokra vagy egyéb szoftverekre használjuk, amelyeket a fogyasztói réteg készít el a zárt forrású hardverekre, platformokra – tehát nincsenek kapcsolatban a célhardver gyártójával.



(a) *A Sharp LR35902 CPU*



(b) *A Boot ROM bitjei*

1.4. ábra. *A CPU és a Boot ROM*

lehetséges helyét (256 kB ROM van elkülönítve erre a CPU-ban), lefényképezte azt, majd bitről bitre haladva rekonstruálta a bináris állományt. A teljes processzor madártávlati nézete a 1.4-es ábrának *(a)* részén látható, a belenagyított *(b)* ábrán pedig az áramkör Boot ROM-ot tároló része szerepel, amin szemmel is jól láthatók a programot alkotó bitek.

2. fejezet

A fejlesztési folyamat

Ahogy már az előző fejezetekben is említésre került, az emulátor fejlesztői szubkultúrában többé-kevésbé kialakult egyfajta irányelv, amit érdemes követni az emulátor fejlesztésénél. Természetesen olyan leírást nem lehet készíteni ami bármilyen konzol emulátorának fejlesztésére használható – a hardverek különbözősége és a speciális megoldások nem teszik ezt lehetővé. Azt viszont meg lehet tenni, hogy egy általános tervezési mintát meghatározunk, és a tervezésnél - implementálásnál ezt követjük.
Az első teendő mindenképpen a lehető legtöbb tudásanyag összeszedése ilyen-olyan forrásokból: internetről, régi szaklapokból, esetleg magát a hardvert tanulmányozva. Nagy segítséget jelenthet például ha már valaki belekezdett ugyanazon hardver emulátorának fejlesztésébe, hiszen fontos információkkal szolgálhat. Egyes hardver emulátorok köré közösségek is összegyűlnek: így van ez a Game Boy esetén is. Ez a közösség egy honlapon gyűjtötte egybe az elérhető összes – eddig fellelt – információt a konzolról. A legfontosabb dokumentum azonban minden emulátor fejlesztése kapcsán a processzor dokumentációja, hiszen – ahogy majd látni a későbbiekben erre ki is térek – ezt fogjuk először implementálni. Mielőtt az implementációs szakaszba lépnénk, célszerű átgondolni az emulátor leendő struktúráját, működését, illetve az alkalmazott eszközöket. Továbbá az elvárt működést, *input*, *output* adatokat is át kell gondolni a tényleges fejlesztési munkálatok előtt.

2.1. A fő ciklus

A fő ciklus a Game Boy utasítás-végrehajtását emulálja, aminek egy leegyszerűsített modellje bármilyen Neumann-elvű számítógép processzorára illeszkedni fog. Ezt a fő ciklust elterjedtebb nevén **betöltő-dekódoló-végrehajtó** ciklusnak is nevezik. Lépései a következők:

1. A soron következő utasítás betöltése a memóriából az utasításregiszterbe.
2. Az utasításszámláló (másnéven *Program Counter*, vagy PC) beállítása a következő utasítás címére.
3. A beolvasott utasítás típusának meghatározása.
4. Ha az utasítás memóriabeli szót használ, a szó helyének meghatározása.
5. Ha szükséges, a szó beolvasása a CPU egy regiszterébe.

6. Az utasítás végrehajtása.

7. Vissza az 1. pontra.

A fenti szerkezet valamilyen módon minden emulátorban megtalálható, ez a felépítés alapja. A ciklus addig ismétlődik, amíg egy `HALT`, vagy egyéb kilépést/megállást szolgáló utasítás nem érkezik végrehajtásra. Természetesen a megszakításkezelő valamelyest beleszól a ciklus működésébe, de erről majd egy későbbi fejezetben lesz szó.
A Game Boy emulátorban a fenti szerkezet egy egyszerűbb változata működik, ami vázlatszerűen így néz ki:

```
loop {  // endless loop
  let next_byte = fetch_byte();
  let instruction = decode_instruction(next_byte);
  execute(instruction);
}
```

A fenti függvényeket, és azok működését a későbbiekben fogom részletezni.
A fő ciklus megtervezése tipikusan a CPU alap struktúrájának (regiszterek, RAM, stb.) implementálása után következik. Ezek után jön a legtöbb emulátor leghosszabb és legrepetitívebb része: a CPU műveleteinek implementálása. A műveletek után a hardverspecifikus RAM mechanizmusokkal érdemes foglalkozni, majd a megszakításvezérlés, a grafikus emuláció és a joypad emuláció zárja a fejlesztést. Természetesen ez egy elnagyolt *roadmap*, de mankónak, vezetőnek megfelelő.

2.2. Alkalmazott eszközök

A fejlesztéshez alkalmazott eszközök meghatározása fontos tényező, hiszen nagyban megkönnyíthetjük vagy megnehezíthetjük a saját munkánkat. Először is célszerű egy programozási nyelvet választani, lehetőség szerint olyat, amihez léteznek olyan *library*-k, amelyekkel megvalósítható a program. Emellett az is lényeges, hogy a programozási nyelv gyors binárist generáljon – természetesen megvalósíthatjuk az emulátort *Javascript* nyelven is, csak észrevehetően lassabb lesz, mint mondjuk a *C++*-os variánsa.
A programozási nyelv mellett a *debug*-olást nagyban megkönnyíti egy *disassembler*, vagy optimális esetben egy másik emulátorhoz készített *debugger*. A ROM fájlokhoz szükséges lehet még egy *hex editor*[1], hogy pontosan lássuk azt, hogy milyen bájtokkal dolgozunk. Ahhoz hogy lássuk, hogy a memóriában milyen adatok szerepelnek, célszerű egy memória térkép eszközt készíteni a fejlesztés során.

2.2.1. A Rust programozási nyelv

Az emulátor fejlesztéséhez a Rust programozási nyelvet választottam, több okból. Egyrészt ez előtt egy kisebb emulátor projekten dolgoztam a nyelvvel, és már akkor megtetszett az

[1] A *hex editor* egy olyan szoftver, amely segítségével megtekinteni és módosítani lehet egy bináris adatfájlt. A "*hex*" előtag a hexadecimális rövidítésből ered: a bináris fájl bájtjait 16-os számrendszerben mutatja a program.

egyszerűsége, a környezete, a nyelv köré alakult közösség. Másrészt a nyelvet az ehhez hasonló performancia-orientált feladatokra tervezték.
A **Rust** a fejlesztők weboldala szerint egy 2006 óta fejlesztett, rendszerfejlesztésre készített nyelv, amely villám gyorsan dolgozik, megelőzi a szegmentációs hibákat, és garantáltan gátolja a versenyhelyzetek kialakulását. Erősen típusos nyelv, szintaktikailag a C++-hoz hasonlít, viszont hozzá képest biztonságosabb memóriakezelést biztosít a sebesség megtartásával. A Rust világában tehát nincsen null pointer, lógó pointer, és versenyhelyzet sem. A fejlesztését és tervezését a Mozilla kutatói részlege kezdte el, majd idővel közösségi projektté alakult. Jelenleg `1.24.1`-es jelzésű az aktuális verzió.
Fontos még megemlíteni, hogy a *Stack Overflow* weboldalon megrendezett éves fejlesztői kérdőív kitöltések alapján 2016-ban, 2017-ben és 2018-ban is a Rust nyerte a "leginkább kedvelt programozási nyelv" kategóriát. Egyéb érdekesség, hogy jól megfigyelhető, hogy az emulátor fejlesztő közösség túlnyomó többsége vagy C++-ban, vagy Rust-ban fejleszt – ez a nyelv kényelmességének, eleganciájának és sokoldalúságának is köszönhető.
Maga a nyelv szépsége azonban még nem minden – a nyelv mellett a **Cargo** eszköz egy fontos szempont. A Cargo nyilván tartja és rezolválja a Rust projektekben összeszedett függőségeket, illetve *buildeli* a projektet. Két *metadata* fájlban tárolja a projekttel kapcsolatos információkat, melyek alapján beszerzi és buildeli a projekt függőségeit. Ezt követően meghívja és futtatja a `rustc` fordítót a megfelelő paraméterekkel. A Cargo a külső *libraryket*, illetve függőségeket a *crates.io* közösségi központi repozitóriumból szerzi be.

2.2.2. A `minifb` könyvtár

Mivel grafikus programról beszélünk, ezért az ablakkezelés és az emulátor vizuális *outputja* fontos tényező. Ehhez – ha lehetséges – minél egyszerűbb és gyorsabb külső könyvtárat kell használnunk, ha szeretnénk megkönnyíteni és felgyorsítani a munkafolyamatunkat. A `minifb` *crate* ezt teszi lehetővé, hiszen ez egy platformfüggetlen, Rust-ban írt *library*, amivel az operációs rendszer által kínált natív ablakokat lehet megnyitni, és feltölteni egy 32 bites *bufferrel*. Támogatja a billentyűzet és egér eseménykezelést, és némely operációs rendszer esetén (Windows, macOS) a menürendszereket.
A használata nagyon egyszerű:

```
window : Window::new("RUST BOY",            // name
                     160,                   // width
                     144,                   // height
                     WindowOptions {        // other options
                         resize: false,
                         scale: Scale::X2,
                         ..WindowOptions::default()
                     }).unwrap()
}
```

Amint látható, négy kötelezően megadandó paramétere van a `Window` struktúrának, melyek rendre:

- `name`: az ablak címsorában szereplő szöveg,

- `width`: az ablak szélessége pixelben,
- `height`: az ablak magassága pixelben,
- `WindowOptions`: az egyéb ablakbeállításokat tartalmazó struktúra.

A negyedik paraméternél kiválaszthatjuk, hogy a default ablakbeállításokat szeretnénk-e – amennyiben igen, `WindowOptions::default()`-ot kell megadni. Ha saját beállításokat kívánunk megadni ebben a `WindowOptions` struktúrában, rendre ezek közül választhatunk:

- `borderless`: ezzel megadható, hogy az ablaknak legyen-e kerete vagy sem,
- `title`: ezzel megadható, hogy az ablaknak legyen-e címe vagy sem
- `resize`: ezzel megadható, hogy az ablak átméretezhető legyen-e vagy sem
- `scale`: ezzel a struktúrával megadható, hogy az ablak mekkora nagyítással jelenjen meg, választható opciók: `X1`, `X2`, `X4`, `X8`, `X16`, `X32`.

A konstruktor meghívását követően az ablak tartalmát (és a `framebuffert`) a következőképpen frissíthetjük:

```
window.update_with_buffer(&framebuffer).unwrap();
```

ahol a `&framebuffer` egy `&[u32]` típusú, `u32` számokat tároló, `width * height` méretű tömbre mutató referencia. A tömbben lévő számok tárolják el az adott pixel színét az ablakban: hexadecimálisan megadva az első két karaktert figyelmen kívül hagyjuk, majd az utána következő 6 karakter adja a szín hexadecimális megfelelőjét:

`FF FF FF FF`

A fentiek alapján látszik, hogy a második `FF` tag a piros (`R`), a harmadik `FF` tag a zöld (`B`), a negyedik `FF` tag pedig a kék (`B`) színért felel. Külön-külön tehát az `RGB` kódokat, míg együtt a hexadecimális színkódot kapjuk.

2.2.3. Fejlesztői környezet

A fejlesztést *elementary OS*[2] rendszeren végeztem. Az emulátor fejlesztés sajátosságai miatt feleslegesnek éreztem egy IDE[3] használatát, hiszen ha a programkód szintaxisa megfelelő, onnantól kezdve a hibakeresést az IDE-k által kínált eszközök sem tudják megkönnyíteni, ahhoz saját *debuggert* kell írni. Ilyen fejlesztői környezet használata helyett tehát a klasszikusnak mondható szövegszerkesztő (Atom, Rust *linterrel*[4]) és terminál párost használtam, a `rustc` fordító *warningjaira* és *errorjaira* hagyatkozva.
A `rustc` fordító *targetjeként* a `stable-x86_64-unknown-linux-gnu` beállítást használtam (alapbeállítás), ami a "hagyományos" 64 bites Linux disztribúciókra optimalizált fordítási paraméterezés. A fordítást, futtatást és a külső függőségek (*libraryk*) beszerzését a Cargo eszközzel valósítottam meg.

[2] Az *elementary OS* egy *Ubuntu* alapú Linux disztribúció.
[3] Az integrált fejlesztői környezet (angolul IDE, azaz Integrated Development Environment) a neve a számítógép-programozást jelentősen megkönnyítő, részben automatizáló programoknak.
[4] Olyan eszközöket nevezünk *linternek*, amelyek a forráskódot analizálva programozási hibákat, *bugokat*, stílusbeli hibákat, vagy gyanús felépítéseket jeleznek a felhasználónak.

2.2.4. *Debugger*

2.3. A feladat specifikációja

Nyilatkozat

Alulírott ..... szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet ..... Tanszékén készítettem, ... diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, 2018. március 15. .....
aláírás

Köszönetnyilvánítás

Irodalomjegyzék

[1] J. L. Gischer, The equational theory of pomsets. *Theoret. Comput. Sci.*, **61**(1988), 199–224.

[2] J.-E. Pin, *Varieties of Formal Languages*, Plenum Publishing Corp., New York, 1986.