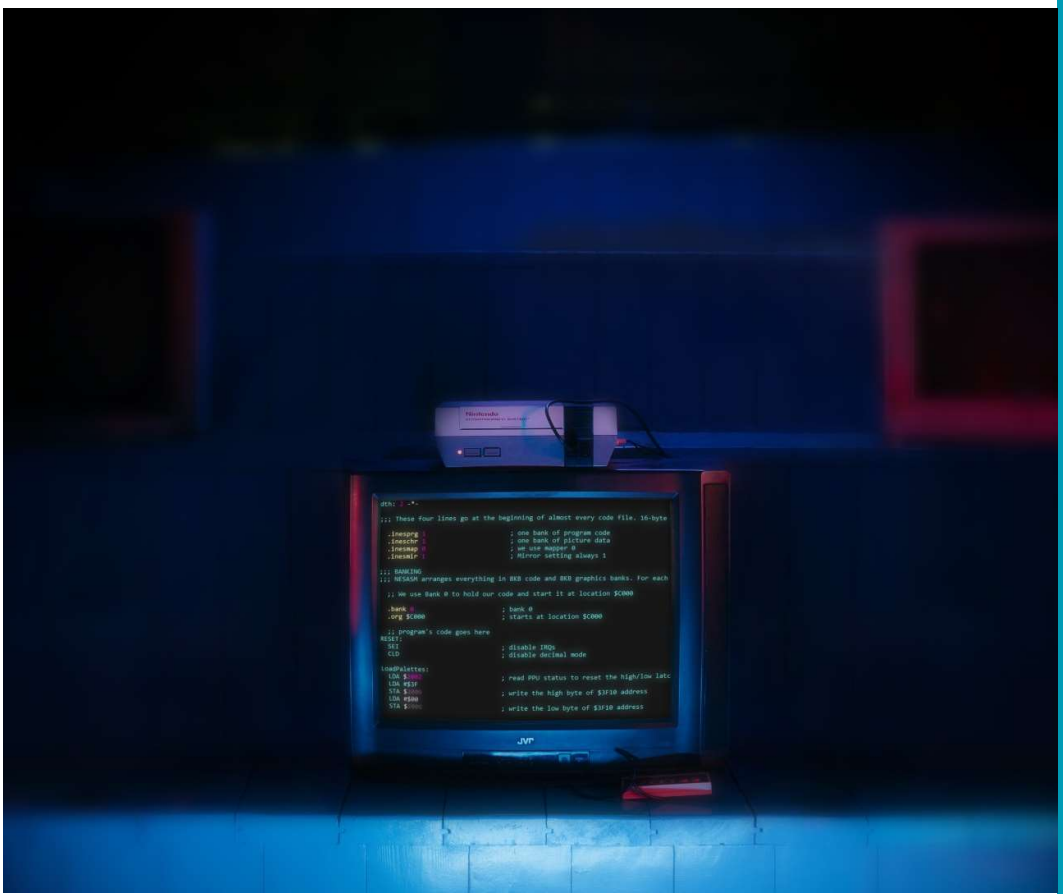
Patrik Broms

Nintendo Entertainment System emulointiympäristön kehittäminen



Insinööri (AMK)

Tieto- ja viestintätekniikka

Syksy 2020

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Broms Patrik

Työn nimi: Nintendo Entertainment System emulointiympäristön kehittäminen

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: emulointi, NES, tietokonearkkitehtuuri, tietokonegrafiikka

Työn tavoitteena oli kehittää Nintendo Entertainment System -emulaattori. Tavoitteena oli tarjota alusta, jota voidaan käyttää NES-kasettien kehityksessä apuna. Työssä käytettiin apuna julkisesti saatavilla olevia alustan ja sen komponenttien dokumentaatioita. Lisäksi emulaattoria kehitettiin julkisesti saatavilla olevien avoimen lähdekoodin NES-kasettien avulla.

Työn edellytyksenä oli tukea Windows- ja Linux-käyttöjärjestelmiä. Työ toteutettiin C-ohjelmointikielen C99 versiolla ja työssä käytettiin alustasta riippumattomia ohjelmistokirjastoja.

Opinnäytetyö toteutettiin yksi komponentti kerrallaan. Komponentit vastaavat isoja kokonaisuuksia alkuperäisen alustan toiminnasta, kuten prosessoria ja kasettia. Jokaisen komponentin toimintaa pyrittiin visualisoimaan työssä kehitetyssä käyttöliittymässä. Visualisoitavia kohteita olivat alustan eri muistit ja ajettavan koodin tila. Työssä toteutettuja komponentteja olivat prosessorin osoiteväylä, kasetti, prosessori ja kuvankäsittely-yksikkö.

Prosessorin osoiteväylän vastuulla on yhdistää alustan komponentit yhteen liikuttamalla tietoa näiden välillä. Osoiteväylä pystyy kerrallaan kuljettamaan 8-bittisen luvun, ja se kattaa 16-bittisen muistiavaruuden. Tätä osoiteväylää emulaattorissa ajettava koodi pystyy käsittelemään suoraan.

Kasetissa sijaitsee emulaattorissa ajettava koodi, kuvat ja äänet. NES-alustalla on monia erilaisia kasetti tyyppejä, joiden avulla alustan toimintaa on voitu muokata. Tätä varten kasettikomponentti sisältää muistikartoittajia, jotka valitaan kasetin tyypin mukaan. Kasetit ovat alkuperäisessä alustassa erillisiä liitettäviä komponentteja ja emulaattorissa ne ovat digitaalisia tiedostoja.

Prosessori hakee kasetista operaatioita ja suorittaa ne. Operaatioiden avulla voidaan muokata joko prosessorin omaa tilaa, muisteja tai muiden komponenttien tilaa. Prosessorin operaatiot perustuvat 6502-ohjelmointikielen ohjeisiin. Työhön toteutettiin ajettavan koodin visualisointi.

Kuvankäsittely-yksikkö tuottaa alustan kuvan. Kuvan tuottaminen koostuu taustakuvista ja edessä olevista spritekuvista. NES-alusta ei pysty piirtämään yksittäisiä pikseleitä, vaan piirtämiseen käytetään kasetista saatuja kuvia. Kuvankäsittely-yksikkö sisältää myös oman osoiteväylän. Tämän avulla kuvankäsittely-yksikkö yhdistää sen sisäiset muistit ja kasetissa sijaitsevat kuvat.

Kun työhön vaaditut komponentit oli toteutettu, emulaattoria testattiin NES-kasettien avulla. Testauksen avulla pystyttiin parantamaan emulaatio tarkkuutta ja korjaamaan komponenteissa olevia virheitä. Testauksen apuna oli työhön kehitetty käyttöliittymä. Käyttöliittymästä pystyy katsomaan mitä muutoksia prosessorin suorittamat operaatiot tekevät, ja miltä kasetissa sijaitsevat kuvamuistit näyttävät.

Abstract

Author: Broms Patrik

Title of the Publication: Development of Nintendo Entertainment System Emulation Environment

Degree Title: Bachelor of Engineering, Information and Communication Technology

Keywords: emulation, NES, computer architecture, computer graphics

The goal of this bachelor's thesis was to develop Nintendo Entertainment's system emulator. The objective was to offer a platform, which could be used to help in the development of NES-cartridges. Publicly available documentation on the platform and its components were used to help with the project. Additionally, the emulator was developed with the help of publicly available open-source NES-cartridges.

The prerequisite of the project was to support Window and Linux operating systems. The Project was implemented with C99 programming language.

The project was implemented component by component. Components correspond to major entities of the original platform, for example, processor or cartridge. Each component's functionality has been visualized in the project's user interface. Major subjects of the visualization were the platform's different memories and the state of the executed code. Components that were implemented in the project were processors address bus, cartridge, processor, and picture processing unit.

The processors address bus connects the platform`s components together. It moves 8-bit sized data between different components, and it covers a 16-bit address space. Executable code addresses this address bus directly.

A cartridge contains executable code, pictures, and sounds. NES has many kinds of cartridges, which can be used to customize how the platform operates. For this reason, the cartridge component contains memory mappers, which are selected according to the type of cartridge. In the original platform, cartridges were physical components, but in the emulator, they are digital files.

The processor executes operations that are fetched from a cartridge. The processor`s operations can modify the state of the processor, memories, or other components. These operations are based on the 6502-programming language. This project also implemented a visualization of the executable code`s state.

The picture processing unit renders the picture to the screen. Rendering is done in two phases. The first phase is to render the background images and the second one is to render sprite images to the foreground. NES can't render individual pixels, only full images from the cassette`s memory can be rendered. The picture processing unit has its own address bus which is used to connect the unit's internal memories and images located in the cassette.

After the required components were implemented, the emulator was tested with available NES-cassettes. This improved the emulation accuracy and exposed mistakes in the components. The project`s user interface assisted with the testing. From the interface, one can see what changes the operations performed by the processor make, and what the images in the cartridge look like.

Sisällysluettelo

Symboliluettelo

ANSI-standardi	American National Standards Institute valvoo standarien kehittymistä yhdysvalloissa.
APU	Audio Processing Unit.
Assembler	Muuttaa konekielen binäärimuotoon tietokoneelle suoritettavaksi.
GPU	Graphics Prosessor Unit, grafiikkaprosessori
ImGUI	Immediate Graphical User Interface, välittömän muodon käyttöliittymä.
NES	Nintendo Entertainment System.
NES-Kasetti	Alustan suoritettava tieto sijaitsee kasetissa. Alkuperäisessä konsolissa kasetit ovat fyysisiä komponentteja ja emulaattori ympäristössä kasetit ovat tiedostoja.
PC	Personal Computer, henkilökohtainen tietokone. Yleistermi tietokoneesta
NTSC	National Television System Committee. Television värijärjestelmä.
RAM	Random Access Memory, keskusmuisti
ROM	Read Only Memory, lukumuisti
Simulointi	Ohjelmistossa alkuperäisen asian jäljittelyä
Sprite	Tietokonegrafiikassa käytetty nimitys kuville, joissa ei ole taustaa.
Tekstuuri	Tietokonegrafiikassa tekstuurit kertovat, millä väreillä malli piirretään.
Verteksi	Tietokonegrafiikassa verteksit muodostavat näytölle piirrettävän kuvan mallin.

1 Johdanto

Opinnäytetyössä toteutettiin Nintendo Entertainment System -emulaattori Windows- ja Linux-alustoille. Tavoitteena oli tehdä ohjelma, jolla voidaan suorittaa NES-kasetteja ja visualisoida alustan tilaa. Ohjelman avulla voidaan helpottaa NES-kasettien kehitystyötä, virheiden etsimistä ja alustan ymmärtämistä. Työ toteutetaan henkilökohtaiseen käyttöön ja aiheeseen päädyin omasta kiinnostuksesta emulaattoreiden toimintaan. Koska NES-emulointi ei ole uusi asia ja alustalle erilaisilla toteutuksilla olevia emulaattoreita on paljon, päätettiin toteutetussa työssä keskittyä visualisoimaan alustan toimintaa sen muistien ja suoritettavan koodin visualisoinnin kautta. Emulaattorien toteutus käsittelee tietotekniikassa monia osa-alueita, kuten prosessorin toimintaa, konekieliä, tietokonearkkitehtuureja ja grafiikka- ja käyttöliittymäohjelmointia.

NES on vuonna 1983 Nintendon julkaisema pelikonsoli. NES julkaistiin alun perin Japanissa, josta se myöhemmin levisi Amerikkaan ja Eurooppaan. Alun perin NES julkaistiin 17 pelin kanssa [1]. Nykypäivään mennessä alustalle on virallisesti lisensoitu 715 peliä, joista viimeinen julkaistiin vuonna 1994 [2]. Tunnettuja NES - pelejä ovat esimerkiksi Super Mario Bros., Contra ja Duck Hunt. Näiden pelien lisäksi alustalla on vahva ”Homebrew”-kulttuuri. Homebrew-kulttuurilla tarkoitetaan pelejä, jotka eivät ole Nintendon lisensoimia. Näitä ”tee se itse”-pelejä tuotetaan yleisimmin emulaattoreiden avulla, koska ne eivät vaadi pelin lataamista fyysiseen kasettiin tai komplekseja fyysisiä komponentteja [3]. Homebrew pelien määrää on vaikea arvioida, mutta tunnettuja on vähintään parikymmentä. Näihin kuuluvat 2019 julkaistu Micro Mages, 2010 julkaistu Battle kid: Fortress of Peril ja 2016 julkaistu Haunted: Halloween ’85.

Työssä toteutettiin NES-alustan fyysiset komponentit ohjelmistossa ja tehtiin työkaluja, joilla voidaan visualisoida yksittäisten komponenttien tilaa. Työssä vaadittuja komponentteja olivat esimerkiksi prosessori, kuvankäsittely-yksikkö ja kasetin lukija. Työn tavoitteena oli toteuttaa emulaattori, jolla pystytään suorittamaan yleisimpiä kasettityyppejä. Yleisimmillä kasettityypeillä tarkoitetaan NROM-, UxROM- ja MMC1-kartoittajia käyttäviä kasetteja [4]. Kasetteja käsitellään tarkemmin luvussa 6.

Työ perustuu julkisesti saatavilla olevaan NES-alustan ja sen komponenttien dokumentointiin. Tällaisiin kuuluvat esimerkiksi MOS Technology 6502 -prosessorin dokumentointi, nesdev.com-yhteisön keräämä dokumentointi ja NESDoc-dokumentti. Tärkein työssä käytetty lähde on nesdev.com-verkkosivusto. Tälle sivustolle on koottu kaikkea alustaan liittyvää dokumentaatiota ja

projekteja. Lisäksi sivustolla on aktiiviset foorumit, missä yhteisö keskustelee esimerkiksi NES-pelien ja emulaattorien kehittämisestä. [5.]

2 Emulointi

Emuloinnilla tarkoitetaan toisessa järjestelmässä alkuperäisen järjestelmän toiminnan jäljittelyä [6]. Esimerkiksi tässä työssä jäljitellään NES-alustan toimintaa PC-alustalla. Emuloinnin avulla voidaan mahdollistaa mm. vanhalle tietokonearkkitehtuurille tehdyn ohjelman toiminta nykyisellä systeemillä. Emulointi voi tuoda uusia käyttäjiä vanhoille systeemeille, joita ei nykyisin kehitetä tai myydä. Vaikka tässä työssä emuloidaan NES-alustan fyysisiä komponentteja, myös tietokoneohjelmistoja voidaan emuloida.

NES-alustalla on pitkäaikainen emulointihistoria. Vuonna 1996 julkaistiin ensimmäinen suosittu NES-emulaattori nimeltään iNES. iNES-emulaattori kehitti nykyisten NES-emulaattoreiden käyttämän .NES-kasettiformaatin ja loi pohjan rikkaalle emulointikulttuurille. [7.] Myös itse Nintendo on lisensoinut pelejä NES-emulaattoreiden käyttöä varten heidän muilla alustoillaan, kuten Wii ja Nintendo 3Ds.

Emulaattori voidaan toteuttaa useilla eri tavoilla, mutta yleisesti emulaattorilla emuloidaan alustan komponentteja korkealla tai alhaisella tasolla [8].

Korkean tason emulaattori pyrkii nopeisiin ohjelmiin toteuttamalla vain päällepäin näkyvä komponentin toiminta. Tämän tyylistä emulointia voidaan kutsua myös simuloinniksi. Tämä johtaa emulaation tarkkuuden heikentymiseen. Korkean tason emulointi on tärkeää etenkin moderneissa ja nopeissa järjestelmissä, joihin alhaisen tason emulointi on liian hidasta. Korkean tason emulaattoreita käytetään muun muassa GPU-komponenttien emuloinnissa, joissa GPU-kutsut muutetaan natiivin järjestelmän kutsuihin. [8.]

Alhaisen tason emuloinnissa emuloidaan komponentteja kellotarkasti. Tämä tarkoittaa, että komponenttia matkitaan mahdollisimman tarkasti jokaisella kellotuksella. Tämän tason emulointi on reaaliajassa mahdollista vain vanhemmissa ja hitaimmissa systeemeissä. [8.]

Tässä opinnäytetyössä on tarkoitus toteuttaa alhaisen tason emulaattori, missä emuloitavat komponentit on toteutettu riittävällä tarkkuudella alkuperäiseen konsoliin verrattuna. Täysin kellotarkkaa toteutusta ei kuitenkaan haeta ohjelman yksinkertaisuuden säilyttämiseksi. Lisäksi työhön toteutetaan NES komponenteille työkaluja, joilla voi monitoroida yksittäisen komponentin toimintaa.

3 Työssä käytetyt työkalut

OpenGL-kirjastoa lukuun ottamatta, työ toteutettiin avoimien lähdekoodikirjastojen avulla. Käyttämällä vapaasti saatavilla olevia kirjastoja pystyin takaamaan koodin jaettavuuden ja monille alustoille mahdollisen siirrettävyyden. Ohjelmistokirjastojen käytöllä pystyttiin myös yksinkertaistamaan monia ominaisuuksia, kuten ikkunan avaamista.

Työssä päätettiin käyttää ohjelman ikkunan avaamiseen ja äänien tuottamiseen Simple Directmedia Layer (SDL) -kirjastoa. SDL-kirjasto mahdollistaa alustariippumattomat ikkunat ja alhaisen tason pääsyn tietokoneen ääniin, näppäimistöön ja peliohjaimiin. Lisäksi SDL-kirjasto mahdollistaa OpenGL-grafiikkakirjaston funktioiden lataamisen ja käyttämisen. SDL-kirjasto tukee virallisesti Windows-, Mac OS X-, Linux-, iOS- ja Android-käyttöjärjestelmiä. [9.]

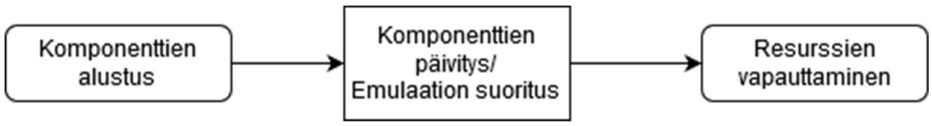
Grafiikan piirtämiseen käytin OpenGL-grafiikkakirjastoa. OpenGL ei ole avoin lähdekoodikirjasto, mutta se mahdollistaa 2D- ja 3D-grafiikoiden piirtämisen monilla eri alustoilla. OpenGL-kirjasto on korkean tason rajapinta tietokoneen grafiikankäsittely-yksikölle. Käyttäjien on mahdollista käyttää grafiikkakäsittely-yksikön ominaisuuksia OpenGL:n avulla. Tämän avulla grafiikkaan liittyvää koodia voidaan käyttää monessa eri käyttöjärjestelmässä ja ympäristössä. OpenGL on Khronos Groupin ylläpitämä kirjasto, ja sen uusin versio on 4.6. Tässä opinnäytetyössä käytettiin versiota 3.2, koska modernimmille versioille ei tullut tarvetta. [10.]

Ohjelmointikielenä työssä käytettiin C99-ohjelmointikieltä. C99 on vuonna 1999 julkaistu versio 1989 ANSI-standardisoidusta C-ohjelmointikielestä. C on proseduraalinen, staattisesti tyypitetty ja yleiseen tarkoitukseen tarkoitettu kieli. C-ohjelmointikieltä käytetään esimerkiksi käyttöjärjestelmien, sulautettujen järjestelmien, tietokonekielikääntäjien ja tietokonesovelluksien ohjelmoinnissa. [11.] Valitsin C99-ohjelmointikielen tähän opinnäytetyöhön sen nopeuden ja helposti siirrettävyyden takia. Lisäksi kieli on itselleni tuttu.

Ohjelman käyttöliittymän piirtämiseen käytin Nuklear-kirjastoa. Nuklear on ImGUI-tyylinen käyttöliittymäkirjasto. Se on suunniteltu helposti projektiin integroitavaksi, ja sitä pystyy käyttämään kaikkien grafiikkarajapintojen kanssa. Lisäksi sillä ei ole ollenkaan riippuvuutta muuhun koodiin tai käyttöjärjestelmään. Se toimii generoimalla käyttöliittymän verteksidatan ja antamalla käyttäjälle tarvittavat tekstuurit. Käyttäjän vastuulla on antaa valitun grafiikkakirjaston käyttöön tekstuurit ja piirtää verteksit tekstuurille tai ikkunaan. [12.] Valitsin Nuklear-kirjaston sen helpon käytettävyyden ja kaikille alustoille siirrettävyyden takia.

4 Ohjelman runko

Projektin alussa kehitettiin emulaattorille runko. Emulaattorit toimivat hyvin tyypillisessä ohjelmarungossa. Emulaattorit voidaan ohjelmien tavoin jakaa kolmeen hyvin tyypilliseen osaan: alustukseen, komponenttien päivitykseen ja resurssien vapauttamiseen. Tämä rakenne on kuvattu kuvassa 1. Ohjelman komponenttien alustus tapahtuu käynnistyksen yhteydessä ja resurssien vapauttaminen ohjelman sulkeutuessa. Komponenttien päivitys toteutuu ajoitetussa silmukassa. Komponentit ovat ”kellotettuja” tietyllä taajuudella, minkä takia niitä ei päivitetä jokaisella silmukalla. Tämä ohjelman rakenne antaa myös suunnan komponenttien suunnittelulle. Jokaisella komponentilla on ohjelman runkoa vastaava alustus, päivitys ja resurssien vapautusfunktio, joita kutsutaan ohjelman rungosta.



Kuva 1. Emulaattorin toimintavaiheet

4.1 Alustaminen

Ohjelman alustamista suoritetaan omassa funktiossaan, joka huolehtii kaikkien komponenttien alustamisesta ennen varsinaista emuloinnin suorittamista. Normaalisti kolmannen osapuolen ohjelmistokirjastot tuodaan ohjelmaan linkkaamalla sen kirjaston binääridata omaan kirjastoon. OpenGL-kirjasto kuitenkin vaatii funktioiden dynaamista latausta. Tässä ohjelmassa käytetään SDL-kirjastoa lataamaan funktioita. Alustuksessa myös avataan ikkuna SDL-kirjaston avulla ja asetetaan SDL-attribuutit. Seuraavaksi ladataan haluttu kasetti emuloitavaksi ja alustetaan ohjelman komponentit, joita ovat prosessori ja PPU, kuvan 2 näyttämällä tavalla.

```
static void
initialize(char* rom) {

    // Initialize SDL, backbuffers and OpenGL
    SDL_Init( SDL_INIT_VIDEO | SDL_INIT_JOYSTICK);
    SDL_GL_SetAttribute( SDL_GL_DOUBLEBUFFER, 1 );
    SDL_GL_SetAttribute( SDL_GL_ACCELERATED_VISUAL, 1 );
    SDL_GL_SetAttribute( SDL_GL_RED_SIZE, 8 );
    SDL_GL_SetAttribute( SDL_GL_GREEN_SIZE, 8 );
    SDL_GL_SetAttribute( SDL_GL_BLUE_SIZE, 8 );
    SDL_GL_SetAttribute( SDL_GL_ALPHA_SIZE, 8 );

    SDL_GL_SetAttribute( SDL_GL_CONTEXT_MAJOR_VERSION, 3 );
    SDL_GL_SetAttribute( SDL_GL_CONTEXT_MINOR_VERSION, 2 );
    SDL_GL_SetAttribute( SDL_GL_CONTEXT_PROFILE_MASK, SDL_GL_CONTEXT_PROFILE_CORE );

    if(!SDL_SetHint(SDL_HINT_NO_SIGNAL_HANDLERS, "1")) {
        printf("failed to set hint!\n");
    }

    window = SDL_CreateWindow("nes-emu",
            SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
            SCREEN_WIDTH, SCREEN_HEIGHT, SDL_WINDOW_OPENGL | SDL_WINDOW_SHOWN);

    assert(window);
    SDL_GL_CreateContext(window);

    // Load cartridge, init cpu, ppu and gamepad
    cartridge_load(rom);
    cpu_reset();
    ppu_init();
    debugger_init(window);
    gamepad_init();

    LOG_COLOR(CONSOLE_COLOR_BLUE ,"all initialized");
}
```

Kuva 2. Ohjelman alustuskoodi

4.2 Päivitys

Alustuksen jälkeen ohjelmaa päivitetään niin kauan, kunnes se suljetaan. Päivitys tapahtuu jatkuvasti pyörivässä silmukassa. Silmukan alussa luetaan SDL-kirjaston avulla näppäimistön ja ohjaimen tila ohjelman sisäiseen muistiin. Tämän jälkeen tarkistetaan, onko aikaa kulunut tarpeeksi viime päivityksestä, että voi päivittää komponentteja uudestaan. Jos komponentteja päivitetään jokaisella syklillä, emulaatio olisi liian nopea. Ajoitukseltaan tämä toimii, kun tiedetään, että komponenttien päivitykseen ja ikkunan piirtämiseen kuluva aika on vähemmän kuin kohdeaika. Työssä päätettiin käyttää 60 Hz kohdeaikaa, mikä vastaa NES-alustan kuvaruudun päivitysnopeutta. Tämä tarkoittaa, että emulaattoria päivitetään aina, kun 1 s/ 60 s on kulunut, kuten kuvasta 3 voidaan tulkita.

```
while (running) {
    // update game pad and run while esc key is pressed
    running = keystate_update();

    currentTime = SDL_GetTicks();
    double delta = (double)(currentTime - lastTime) / 1000.0;

    if(delta > targetDelta) {
        lastTime = currentTime;
        /* Update emulation components */
    }
    debugger_update();
    draw();
}
```

Kuva 3. Ohjelman päivityskoodi

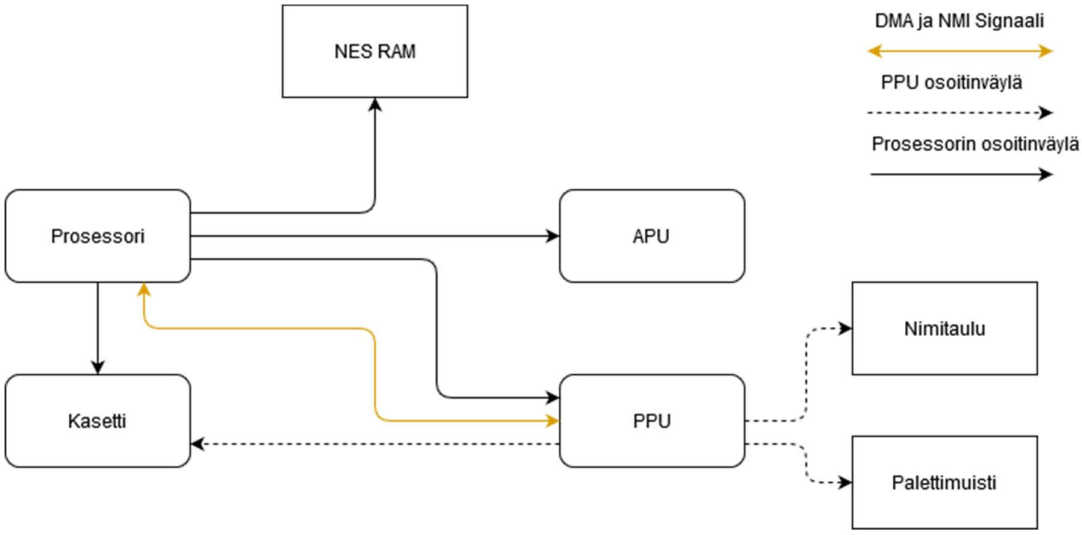
Tämä silmukka toimii pohjana emulaattorin päivittämiselle ja yleisesti sen logiikalle. Kun komponentteja tehdään lisää, voidaan tarvittava logiikka lisätä tähän.

4.3 Ohjelman resurssien vapautus

Ohjelman lopuksi vapautetaan ohjelmassa käytetyt resurssit ja suljetaan ikkuna. Tässä jokaiselta komponentilta kutsutaan komponentin siivousfunktiota. Siivousfunktion vastuuna on vapauttaa komponentin käyttämät resurssit takaisin käyttöjärjestelmälle. Tärkeimpänä vapautettavana resurssina on dynaamisesti varattu muisti. Moderneissa systeemeissä dynaamisesti varattu virtuaalimuisti vapautuu automaattisesti ohjelman suljettua, jos sitä ei ole vapautettu ohjelman suorituksen aikana. Päätin kuitenkin tehdä asianmukaisen muistin vapautuksen ohjelmalle, koska sen avulla voidaan paremmin seurata muistin varausta työkalujen avulla ja tämä auttaa muistivuotojen löytämisessä. Vaikka dynaamisesti varattu muisti vapautuu ohjelman suljettua, muistin vapautusta pidetään ohjelmoinnissa hyvänä käytäntönä [13].

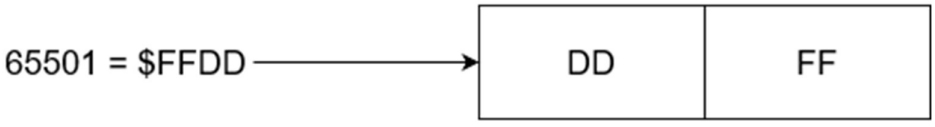
5 Osoiteväylät

Seuraavaksi projektissa toteutettiin osoiteväylät. Osoiteväylien avulla laitteisto yhdistää kaikki alustan komponentit. Osoiteväyliä käytetään datan siirtoon, lukemiseen ja komponenttien ohjaukseen. NES-alustalla on kaksi osoiteväylää, jotka emulaattorissa pitää toteuttaa, prosessorin osoiteväylä ja PPU:n osoiteväylä. Tämä nähdään myös kuvasta 4.



Kuva 4. Osoitinväylien toiminta. Pyöristetyt kulmat ovat funktionaalisia komponentteja.

NES-alustan osoiteväylät edustavat $0000–$FFFF osoiteavaruutta, mikä on 16-bittinen luku [14, s. 10]. Tämä muisti jaetaan 255 ($FF) muistisivuun, joista jokainen sisältää 255 ($FF) tavua [15]. Osoiteväylien avulla komponentit pystyvät lukemaan 8-bittisiä lukuja. Jos isompia lukuja pitää lukea, täytyy suorittaa monta lukua peräkkäin. NES-alustassa on kuvan 5 tapainen little endian tavujärjestys, missä vähiten merkitsevät tavut luetaan ensimmäiseksi.



Kuva 5. 16-bittinen luku little endian muodossa.

5.1 Prosessorin osoiteväylä

Prosessorin osoiteväylä yhdistää prosessorin laitteen sisäiseen RAM-muistiin, PPU:hun, APU:n, IO-rekistereihin sekä kasettiin, kuten taulukosta 1 voidaan tulkita [16]. Emulaattorissa tämä muodostaa NES-alustan rungon, jonka ympärille emulaattori on rakennettu. Tässä kohtaa työtä emulaattori on jaoteltu sen funktionaalisiin osiin: prosessoriin, PPU-komponenttiin ja kasettiin.

Osoitealue	Laite
$0000-$1FFF	2KB **RAM** mikä on peilattu 0x07FF osoitteen välein (ram_osoite = osoite & $07FF)
$2000-$3FFF	8 **PPU** rekisteriä joita käytetään PPU:n ohjaamiseen. Nämä rekisterit on peilattu jokaisen $0007 välein. (ppu_rekisteri = osoite & $0007)
$4000-$4015	**APU** rekisterit joita käytetään APU:n ohjaamiseen
$4016-$4017	Kaksi **IO**-rekisteriä, käytetään **ohjaimiin**
$4018-$401F	**Test mode** aktivointi rekisterit, ei emuloitu. Näitä ollaan käytetty APU:n testaukseen
$4020-$FFFF	**Kasetin muistiavaruus.** Tämä muistiavaruus toimivuus riippuu kasetin tyypistä.

Taulukko 1. Prosessorin osoitealueet [16].

Toteutukseltaan osoiteväylä on yksinkertainen ja muodostuu emuloinnin kannalta kahdesta funktiosta, jotka tulkkaavat prosessorin antaman osoitteen oikealle komponentille ja suorittavat lukemisen tai kirjoittamisen. RAM-muisti ei ole funktionaalinen osa, joten sille ei emulaattorissa ole omaa emulaattorikomponenttia ja yksinkertaistamisen takia tämä osoiteväylä hoitaa myös ohjainten shift-rekisterien toiminnan. Osoiteväylän toiminta näytetään kuvassa 6.

```
static u8
bus_read8(u16 addr) {

    u8 ret = 0;
    if(address_is_between(addr, CPU_MEMORY_START, CPU_MEMORY_SIZE)) {
        ret = ram[addr & CPU_MEMORY_MIRROR_RANGE];
    } else if(address_is_between(addr, PPU_MEMORY_START, PPU_MEMORY_END)) {
        ret = ppu_cpu_read(addr);
    } else if (addr == CONTROLLER1) {
        ret = (buttonState[0] & 0x80) > 0;
        buttonState[0] <<= 1;
    } else if (addr == CONTROLLER2) {
        ret = (buttonState[1] & 0x80) > 0;
        buttonState[1] <<= 1;
    } else if (address_is_between(addr, CARTRIDGE_MEMORY_START, CARTRIDGE_MEMORY_END)){
        ret = cartridge_cpu_read_rom(addr);
    } else {
        ret = 0;
    }

    return ret;
}

static void
bus_write8(u16 addr, u8 data) {

    if(address_is_between(addr, CPU_MEMORY_START, CPU_MEMORY_SIZE)) {
        ram[addr & CPU_MEMORY_MIRROR_RANGE] = data;
    } else if(address_is_between(addr, PPU_MEMORY_START, PPU_MEMORY_END)
            || addr == PPU_DMA_WRITE_ADDRESS) {
        ppu_cpu_write(addr, data);
    } else if (addr == CONTROLLER1) {
        buttonState[0] = internalButtonState[0];
    } else if (addr == CONTROLLER2) {
        buttonState[1] = internalButtonState[1];
    } else if (address_is_between(addr, CARTRIDGE_MEMORY_START, CARTRIDGE_MEMORY_END)){
        cartridge_cpu_write_rom(addr, data);
    }
}
```

Kuva 6. Prosessorin osoitinväylien ohjaus komponentteihin

Komponenttien tilan lukeminen saattaa muokata niiden tilaa. Esimerkiksi osoitteesta $4016 luetaan ohjaimen tila ja samalla muutetaan rekisterin arvo seuraavan napin arvoon. Tämän takia osoiteväylään on lisätty erillinen funktio, jolla voidaan ”kurkistaa” komponenttien muistia kuitenkaan muokkaamatta niiden tilaa. Tätä funktiota käytetään muistin visualisoinnissa.

5.2 PPU-osoiteväylä

PPU-komponentilla on oma osoiteväylä, joka yhdistää sen nimitauluun, palettimuistiin ja kasetissa sijaitsevaan kuviomuistiin. Tämä näkyy myös taulukossa 2. Lisäksi PPU-komponentilla on oma sisäinen muisti, jota prosessori pystyy manipuloimaan osoiteväylänsä kautta. Tämä osoiteväylä täyttää prosessorin osoiteväylästä poiketen vain $0000–$3FFF alueen. [17.]

Tämä osoiteväylä ei sisällä kompleksisia komponentteja, joten emulaattorissa tämä on toteutettu PPU-komponentin sisällä. Toteutus on samanlainen prosessorin osoiteväylän kanssa sisältäen luku- ja kirjoitusfunktion, jossa osoite tulkataan oikeaan komponenttiin ja data käsitellään asianmukaisesti. Komponenttien toiminnallisuutta kuvataan tarkemmin PPU-luvussa.

Osoitealue	
$0000-$1FFF	**Kuviotaulu** on kasetissa sijaitseva muistialue, joka koostuu vasemmasta ja oikeasta 256B kokoisesta muistista, jossa sijaitsee kaikki pelin piirrettävät kuvat.
$2000-$3EFF	Nimitaulu on 1024 tavun kokoinen alue, joka määrittelee, mikä kuvio kuviotaulusta piirretään ja millä värillä se piirretään. Tätä tietoa käytetään taustojen piirtämisessä.
$3F00-$3FFF	Palettimusti koostuu neljästä taustaväripaletista ja neljästä sprite-väripaletista. Jokainen paletti koostuu komesta tavusta jotka kuvastavat kolmea väriä, ja yhdestä tavusta joka kuvastaa läpinäkyvyyttä. Tämän muisti on peilattu 32 bitin välein (väri = osoite & $001F)

Taulukko 2. PPU:n osoitealueet [17].

6 Kasetti ja muistikartoittajat

Seuraavaksi projektissa tehtiin pohja kasetin lataamiselle sekä luotiin ensimmäinen muistikartoittaja. NES-alusta on suunniteltu siten, että siihen pystytään pelin tarpeiden mukaan liittämään ylimääräisiä komponentteja kasettikohtaisesti. Tämän suunnittelun avulla Nintendo pystyi varmistamaan pidemmän elinajan konsolille ja parantamaan konsolin ominaisuuksia vuosien varrella. [14, s. 27.] Koska on olemassa monia erilaisia NES-kasettityyppejä, niille on tehty vastaavat muistikartoittajat. Muistikartoittajien avulla ohjataan kasetin luku- ja kirjoitustoimintaa.

Kasetti toteuttaa oman alustusfunktion, jota kutsutaan komponenttien alustuksen yhteydessä ja jäsentää annetun .nes-tiedoston ja sitä vastaavan muistikartoittajan. Lisäksi kasetti toteuttaa siivousfunktion, joka kutsuu vastaavasti kartoittajan siivousfunktiota. Kartoittajan siivousfunktio taas vapauttaa varatun ohjelma- ja merkkimuistin.

Projektissa toteutettiin kasetti kartoittajien toimintarajapintana. Kasettiin on toteutettu prosessorin ja PPU:n osoiteväylille datan kirjoitus- ja lukemisfunktiot. Kasetti ohjaa tämän funktiokutsun oikeaan kartoittajaan funktio-osoittimien avulla kuvan 7 kuvaamalla tavalla.

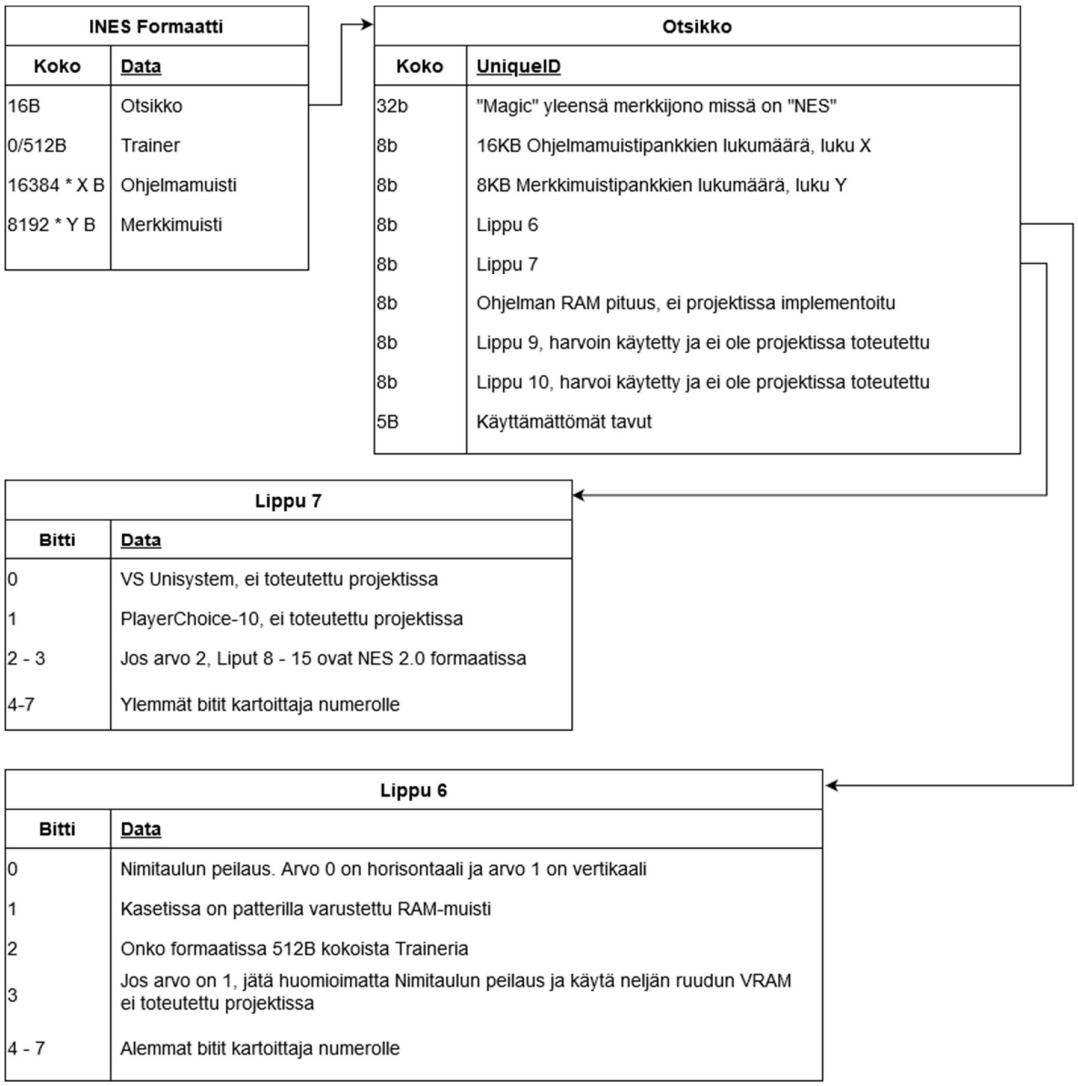
```
static inline u8
cartridge_cpu_read_rom(u16 addr) {

    return mapper.cpu_read_cartridge(&mapper.data, addr);
}

static inline void
cartridge_cpu_write_rom(u16 addr, u8 val) {

    mapper.cpu_write_cartridge(&mapper.data, addr, val);
}
```

Kuva 7. Prosessorin osoitinväylien ohjaus kartoittajaan

6.1 INES-formaatti

INES-formaatti kehitettiin varhaiselle iNES NES -emulaattorille. Formaatista saadaan selville, kuinka monta merkki- ja ohjelmamuistipankkia kasetista löytyy. Lisäksi formaatti määrittelee muistikartoittajan tyypin, nimitaulun peilauksen ja sen, onko kasetissa alun perin ollut paristolla varustettu ram-muisti. Formaatti koostuu kuvan 8 näyttämistä kohdista: otsikosta, trainerista,

ohjelmamuistin datasta ja merkkimuistista. [14, s. 27–29.] Trainer-osaa formaatista ei yleisesti käytetä, koska se sisältää vanhojen NES-emulaattoreiden käyttämää dataa.



INES Formaatti

Koko	Data
16B	Otsikko
0/512B	Trainer
16384 * X B	Ohjelmamuisti
8192 * Y B	Merkkimuisti

Otsikko

Koko	UniqueID
32b	"Magic" yleensä merkkijono missä on "NES"
8b	16KB Ohjelmamuistipankkien lukumäärä, luku X
8b	8KB Merkkimuistipankkien lukumäärä, luku Y
8b	Lippu 6
8b	Lippu 7
8b	Ohjelman RAM pituus, ei projektissa implementoitu
8b	Lippu 9, harvoin käytetty ja ei ole projektissa toteutettu
8b	Lippu 10, harvoi käytetty ja ei ole projektissa toteutettu
5B	Käyttämättömät tavut

Lippu 7

Bitti	Data
0	VS Unisystem, ei toteutettu projektissa
1	PlayerChoice-10, ei toteutettu projektissa
2 - 3	Jos arvo 2, Liput 8 - 15 ovat NES 2.0 formaatissa
4-7	Ylemmät bitit kartoittaja numerolle

Lippu 6

Bitti	Data
0	Nimitaulun peilaus. Arvo 0 on horisontaali ja arvo 1 on vertikaali
1	Kasetissa on patterilla varustettu RAM-muisti
2	Onko formaatissa 512B kokoista Traineria
3	Jos arvo on 1, jätä huomioimatta Nimitaulun peilaus ja käytä neljän ruudun VRAM ei toteutettu projektissa
4 - 7	Alemmat bitit kartoittaja numerolle

Kuva 8. INES formaatin kuvaus

6.2 NROM - Ensimmäinen kartoittaja

NROM on ensimmäinen ja yksinkertaisin kartoittaja. NROM sisältää pelkästään ROM-muistia. NROM sisältää joko 16 KB tai 32 KB ohjelmamuistipankin sekä 8 KB merkkimuistipankin. [18.] Kartoittajan alustuksessa varataan muisti ohjelma- ja merkkimuistille ja siivousfunktiossa muisti puolestaan vapautetaan kuvan 9 mukaisesti.

```
typedef struct Mapper0Data {
    u8* characterMemory;
    u8* programMemory;
} Mapper0Data;

void
mapper0_init(Mapper0Data* data, u8* progMem, u8* charMem) {

    data->programMemory = calloc(cartridge.numProgramRoms, PROG_ROM_SINGLE_SIZE);
    data->characterMemory = calloc(cartridge.numCharacterRoms, CHAR_ROM_SINGLE_SIZE);

    memcpy(data->programMemory, progMem,
            cartridge.numProgramRoms * PROG_ROM_SINGLE_SIZE);
    memcpy(data->characterMemory, charMem,
            cartridge.numCharacterRoms * CHAR_ROM_SINGLE_SIZE);
}

void
mapper0_dispose(Mapper0Data* data) {

    free(data->programMemory);
    free(data->characterMemory);
}
```

Kuva 9. NROM-kartoittaja, alustus ja siivous

NROM-kartoittaja kattaa prosessorin osoiteväylässä osoitteet $8000 - $FFFF osoitealueen [18]. Kuten taulukosta 1 voidaan tulkita, prosessorin osoiteväylässä sijaitsevan kasetin muistiavaruus on alueella $4020–$FFFF. Kasetin lukiessa $4020–$7FFF aluetta NROM-kartoittajalla, ei mikään komponentti ole vastuussa muistialueesta. Tämä ongelma on tyypillinen tietokonearkkitehtuureille, jolla pystyy lukemaan muistiosoitteita ilman käyttöjärjestelmän validointia. Tätä kutsutaan ”open bus behavioriksi”. [19.] Projektissa päätettiin $4020–$7FFF alueiden lukemisesta palauttaa arvon 0 kuvan 10 mukaisesti. Tämä saattaa heikentää yhteensopivuutta emuloitavan alustan kanssa, koska oikeassa alustassa väärän muistiosoitteen lukemisen logiikka on muistiosoitteesta vastaavasta komponentista riippuvainen. Usein tämä on viimeksi osoiteväylästä haettu arvo. [19.] Tätä toiminnallisuutta ei ole toteutettu projektiin sen monimutkaisuuden takia.

Yhden ohjelmamuistipankin koko on 16 KB ja kasetti voi sisältää kartoittajan mukaan eri lukumääriä pankkeja. NROM-kartoittajassa ohjelmamuistipankkeja on yksi tai kaksi, eli se ei täytä koko NROM-kartoittajan vastuulla olevaa $8000–$FFFF aluetta. Tämän takia muisti toistaa itseään koko kartoittajan osoitealueella kuvan 10 mukaisesti. Eli esimerkiksi jos on yksi muistipankki, osoitteesta $8000 ja $C000 lukeminen tuottaa saman arvon. [18.]

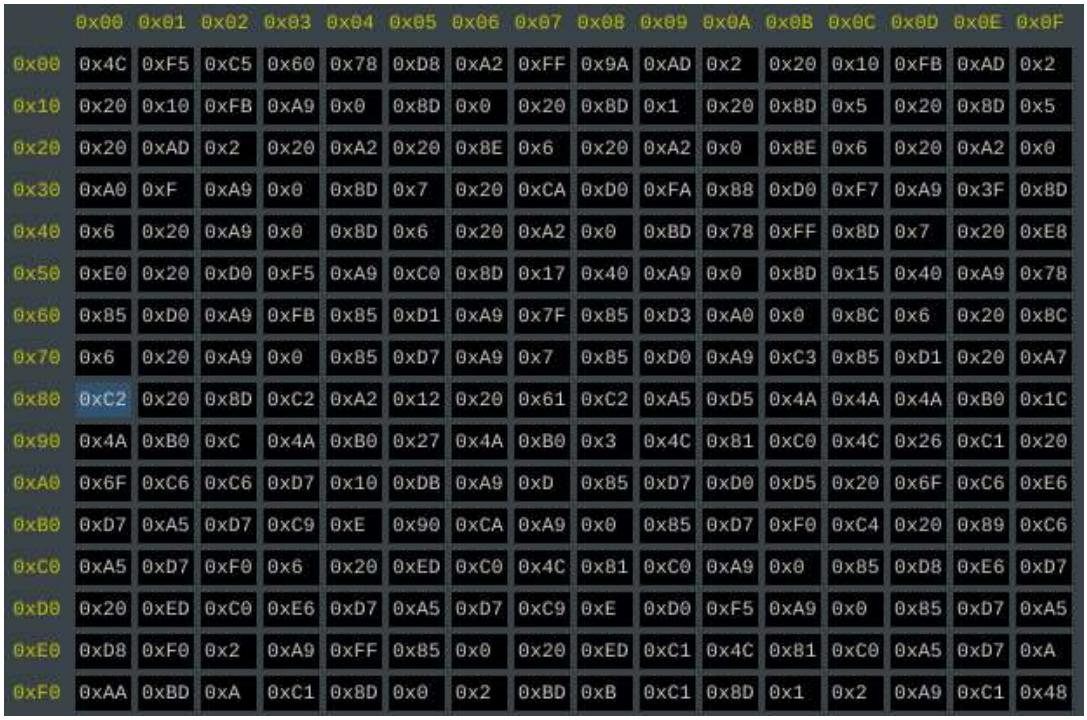
```
u8
mapper0_cpu_read(Mapper0Data* data, u16 addr) {

    if(!address_is_between(addr, MAP0_START, MAP0_END)) {
        return 0;
    }

    if(cartridge.numProgramRoms == 1)
        addr &= 0x3FFF; // if 1 rom capasity is 16K
    else
        addr &= 0x7FFF; // if 2 rom capasity is 32K

    return data->programMemory[addr];
}
```

Kuva 10. NROM-kartoittajan ohjelmamuistin lukeminen

Kartoittajien avulla lukemisen ja kirjoittamisen lisäksi myös kartoittaja toteuttaa muistin kurkistusfunktion. Tämä ei vaikuta emulaatioon ja sitä käytetään visualisointityökaluissa. Kurkistamisella voidaan selvittää, onko osoite oikealla ohjelmamuistialueella ja voidaanko se visualisoida kuitenkaan muokkaamatta sitä.

6.3 Muistin visualisointi

Seuraavaksi projektissa toteutettiin ensimmäinen visualisointikomponentti. Tämän komponentin tärkein funktio on tarkastella RAM– ja kasettimuistin arvoja. 16-bittinen muistiavaruus on jaettu 255 ($FF) muistisivuun, joista jokainen on 255 ($FF) tavun kokoinen. Tämän takia visualisointikin päätettiin toteuttaa jokaiselle muistisivulle erikseen kuvan 11 mukaisesti. Kuvassa väritetty alkio kuvaa tällä hetkellä kuvattua muistisivua. Visualisoinnissa käytetään avuksi prosessorin osoiteväylän kurkistusfunktiota, että emulaattorin tilaa ei muuteta. Kuvassa 11 on visualisoitu NROM-kartoittajaa käyttävän kasetin ensimmäistä muistisivua. Tämä muistisivu sijaitsee osoitteessa $8000 eli sivulla $80.

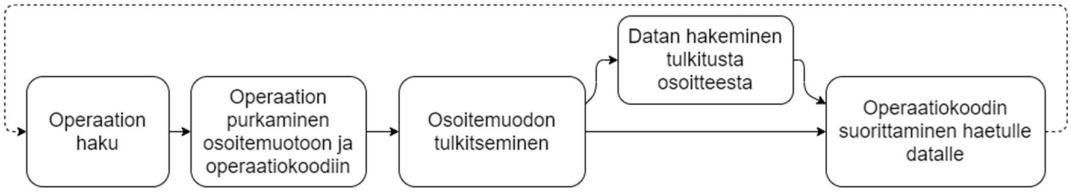


	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0x00	0x4C	0xF5	0xC5	0x60	0x78	0xD8	0xA2	0xFF	0x9A	0xAD	0x2	0x20	0x10	0xFB	0xAD	0x2
0x10	0x20	0x10	0xFB	0xA9	0x0	0x8D	0x0	0x20	0x8D	0x1	0x20	0x8D	0x5	0x20	0x8D	0x5
0x20	0x20	0xAD	0x2	0x20	0xA2	0x20	0x8E	0x6	0x20	0xA2	0x0	0x8E	0x6	0x20	0xA2	0x0
0x30	0xA0	0xF	0xA9	0x0	0x8D	0x7	0x20	0xCA	0xD0	0xFA	0x88	0xD0	0xF7	0xA9	0x3F	0x8D
0x40	0x6	0x20	0xA9	0x0	0x8D	0x6	0x20	0xA2	0x0	0xBD	0x78	0xFF	0x8D	0x7	0x20	0xE8
0x50	0xE0	0x20	0xD0	0xF5	0xA9	0xC0	0x8D	0x17	0x40	0xA9	0x0	0x8D	0x15	0x40	0xA9	0x78
0x60	0x85	0xD0	0xA9	0xFB	0x85	0xD1	0xA9	0x7F	0x85	0xD3	0xA0	0x0	0x8C	0x6	0x20	0x8C
0x70	0x6	0x20	0xA9	0x0	0x85	0xD7	0xA9	0x7	0x85	0xD0	0xA9	0xC3	0x85	0xD1	0x20	0xA7
0x80	0xC2	0x20	0x8D	0xC2	0xA2	0x12	0x20	0x61	0xC2	0xA5	0xD5	0x4A	0x4A	0x4A	0xB0	0x1C
0x90	0x4A	0xB0	0xC	0x4A	0xB0	0x27	0x4A	0xB0	0x3	0x4C	0x81	0xC0	0x4C	0x26	0xC1	0x20
0xA0	0x6F	0xC6	0xC6	0xD7	0x10	0xDB	0xA9	0xD	0x85	0xD7	0xD0	0xD5	0x20	0x6F	0xC6	0xE6
0xB0	0xD7	0xA5	0xD7	0xC9	0xE	0x90	0xCA	0xA9	0x0	0x85	0xD7	0xF0	0xC4	0x20	0x89	0xC6
0xC0	0xA5	0xD7	0xF0	0x6	0x20	0xED	0xC0	0x4C	0x81	0xC0	0xA9	0x0	0x85	0xD8	0xE6	0xD7
0xD0	0x20	0xED	0xC0	0xE6	0xD7	0xA5	0xD7	0xC9	0xE	0xD0	0xF5	0xA9	0x0	0x85	0xD7	0xA5
0xE0	0xD8	0xF0	0x2	0xA9	0xFF	0x85	0x0	0x20	0xED	0xC1	0x4C	0x81	0xC0	0xA5	0xD7	0xA
0xF0	0xAA	0xBD	0xA	0xC1	0x8D	0x0	0x2	0xBD	0xB	0xC1	0x8D	0x1	0x2	0xA9	0xC1	0x48

Kuva 11. Sivun $80 visualisointi

7 Prosessori

Kasetin toteutuksessa ladattiin ohjelmamuisti prosessorin osoiteväylälle. Seuraavaksi toteutetaan komponentti, joka tulkitsee tätä dataa ja suorittaa ladattuja operaatiokoodeja. NES-alusta käyttää MOS Technology 6502 -prosessorin varianttia, josta puuttuu kokonaan desimaalimuoto ja sisältää APU:n [14, s. 9]. Tässä työssä APU-komponenttia ei ole toteutettuna. Prosessorin toiminta emulaattorissa voidaan tiivistää viiteen toimintoon kuvan 12 mukaisesti.



Kuva 12. Prosessorin toiminta

7.1 Prosessorin muisti

MOS Technology 6502 -prosessori ei sisällä omaa muistia, johon ohjelma voisi muistiosoitteilla referoida. Prosessori luottaa täysin siihen yhdistettyihin laitteisiin muistin hallinnassa. Prosessorilla on kuitenkin kuusi rekisteriä, joita käytetään prosessorin tilan hallintaan ja operaatioiden suorittamiseen. Näistä kolmea rekisteriä käytetään operaatioiden suorittamiseen ja kolmea prosessorin tilan säilyttämiseen. Lähes kaikki prosessorin operaatiot hyödyntävät rekistereitä joko operaation tuloksen tallennuksessa tai tulokseen pääsyssä. Rekisterien avulla voidaan vähentää osoiteväylästä haettavien arvojen määrää ja näin nopeuttaa ohjelmaa.

Akkumulaattorirekisteri on 8-bittinen yleiseen tarkoitukseen tarkoitettu rekisteri. Tätä rekisteriä käyttävät aritmeettiset ja loogiset operaatiot, kuten haarautumisehdot. [20, s. 4.]

X- ja Y-rekisterit ovat 8-bittisiä indeksirekistereitä. Näitä rekistereitä käytetään dynaamisiin osoitteiden muutoksiin. Esimerkiksi osoitemuodon tulkinnassa halutun osoitteen saamiseksi indeksirekistereitä käytetään lisäämällä rekisterin arvo haettuun osoitteeseen. [20, s. 4.]

Prosessorilla on myös 16-bittinen ohjelmalaskurirekisteri, joka toimii osoittajana seuraavaan operaatioon. Mikäli haettu operaatio ei muokkaa ohjelmalaskuria, niin se muutetaan seuraavaan

operaatioon. [20, s. 4.] Ohjelmalaskuria manipuloimalla ohjelma pystyy suorittamaan operaatioita mistä tahansa prosessorin muistialueesta. Ohjelmalaskuria käytetään avuksi muun muassa koodin hyppimisessä ja aliohjelmien toteutuksessa.

Pino-osoitin on 8-bittinen rekisteri, jolla osoitetaan viimeisimpään pinomuistin arvoon. Pino-osoittimen voidaan kuvitella olevan korttikasa, jonka päälle voi laittaa tai poistaa arvoja. Pino-tietorakenteessa näitä kutsutaan ”push”- ja ”pop”-operaatioiksi. Pinomuisti sijaitsee prosessorin osoiteväylässä alueella $01FF–$0100. 6502-prosessorissa pino on laskeva, eli pino alkaa arvosta $FF ja laskee arvoon $00. Pinomuistia käytetään muun muassa aliohjelmissa ja keskeytyksissä. Aliohjelma voidaan toteuttaa työntämällä nykyinen ohjelmalaskuri pinon päälle, vaihtamalla ohjelmalaskuri aliohjelman alkuun ja aliohjelman lopussa vaihtaa ohjelmalaskuri pinon korkeimpaan arvoon. [20, s. 4.]

Viimeisenä rekisterinä on 7-bittinen tilalippu. Tilalipuilla merkitään operaatioiden tulosten ominaisuuksia. Esimerkiksi INC-ohje lisää muistiosoitteen arvoon yhden, minkä jälkeen ”Zero”-lippu laitetaan päälle arvon ollessa nolla. Näitä lippuja voidaan käyttää muun muassa tietyissä haarautumisehdoissa, joissa tehdään lipun tilan perusteella päätös haarautumisesta. [20, s. 8–9.] Emulaattorissa tämä rekisteri on toteutettu 8-bittisenä lukuna, kuten kuvasta 13 voidaan tulkita. Lippujen käyttöä helpottaa se, että lipuista vain viisi on emuloinnin kannalta tärkeitä. Desimaalimuoto- ja Break-lippu on toteutettu emulaattoriin vain, että prosessoria analysoivat testikasetit ei ilmoita virheitä.

C Carry	V Overflow	U Unused	B Break	D Decimal Mode	I Disable Interrups	Z Zero	N Negative

Kuva 13. Prosessorin lippurekisteri 8-bittisenä lukuna

Nämä muistirekisterit kokoavat emulaattorissakin prosessorin rakenteen kuvan 14 tavoin. Näiden arvojen lisäksi prosessori pitää yllä laskuria, joka merkitsee, kuinka monta kellosykliä tämänhetkinen operaatio kestää. Lisäksi on osoitin, jolla merkitsetään ohjelman keskeytyspaikka. Tätä voidaan käyttää kasettien virheenetsinnässä hyödyksi.

```
typedef enum CpuStatus {
    Carry               = (1 << 0),
    Zero                = (1 << 1),
    DisableInterrupts   = (1 << 2),
    DecimalMode         = (1 << 3),
    // only PHP and BRK sets this flag on push to the stack
    Break               = (1 << 4),
    // Always set on
    Unused              = (1 << 5),
    Overflow            = (1 << 6),
    Negative            = (1 << 7),
} CpuStatus;

typedef struct cpu {
    //  reqisters
    u8  Xreq;
    u8  Yreq;
    u8  accumReq;
    //  Processor status
    u8  flags;
    //  Program counter
    u16 pc;
    u8  stackPointer;
    // Number of cycles to execute
    u32 cycles;
    // Address where program halts
    U32 breakpoint;
} cpu;
```

Kuva 14. Emulaattorin prosessorin rakenne

7.2 6502-kieli

NES-alustan prosessoria ohjelmoidaan 6502-kielellä. 6502-kieli koostuu 56 operaatiosta sekä 13 osoitemuodosta. Nämä muodostavat 151 virallista operaatiota, jotka prosessorilla voidaan suorittaa. [14, s. 14.]

Osoitemuodot kertovat, miten operaatiolle annettua osoitetta tulkitaan. Osoitemuotoja voidaan käyttää muun muassa datan haun optimisointiin. 6502-prosessorissa yksi prosessorin osoiteväylän lukeminen kestää yhden syklin. Prosessori pystyykin vähentämään muistin lukemista esimerkiksi ”Zero page addressing” muodolla, jolla lukee RAM-muistin ensimmäistä sivua. Ensimmäistä sivua luettaessa tarvitaan vain yksi 8-bittinen luku osoiteväylästä, joka voidaan yhdellä syklillä hakea. [14, s. 39.] Osoitemuotoja käyttämällä voidaan myös tehdä operaatioita akkumulaattorirekisterillä ja käyttää X- ja Y-rekisteriä muistin indeksointia varten, kuten taulukosta 3 voidaan tulkita.

Osoitemuoto	Lyhenne	Selitys	koko
Accumulator	ACCUM	Operaatio akkumulaattorin kanssa	0
Implied	-	Operaatio ei tarvitse haettua muistia, esimerkiksi lipun asetus	0
Immidiate	IMM	Seuraava tavu toimii datana, ei osoitteena	8b
Zero page	ZP	Osoite ensimmäiseen muistisivuun	8b
Zero page X	ZP X	Haettuun osoitteeseen lisätään X-rekisteri	8b
Zero page y	ZP Y	Haettuun osoitteeseen lisätään Y-rekisteri	8b
Relative	-	Ohjelmalaskuriin relatiivinen osoite, voi olla negatiivinen	8b
Absolute	ABS	$0000 - $FFFF alueella oleva osoite	16b
Absolute X	ABS X	$0000 - $FFFF alueella oleva osoite, johon lisätään X-rekisteri	16b
Absolute Y	ABS Y	$0000 - $FFFF alueella oleva osoite, johon lisätään Y-rekisteri	16b
Indirect	-	Osoite luetaan seuraavan kahden tavun kertomasta paikasta	16b
Indirect X	IND X	18b Osoite luetaan seuraavan tavun ja X-rekisterin summasta	8b
Indirect Y	IND Y	Osoite luetaan seuraavan kahden tavun kertomasta paikasta, johon summataan Y-rekisteri	8b

Taulukko 3. 6502-kielen osoitemuodot

Operaatioiden avulla voidaan tulkita, kirjoittaa tai muuttaa osoitemuodon tuottamaa dataa. Operaatioilla pystytään myös suoraan manipuloimaan rekisterien arvoja tai sijoittamaan niitä muistiin. Operaatiot voidaan jaotella niiden käyttökohteiden mukaan kategorioihin. Esimerkiksi LDA-, LDX- ja LDY-operaatiot lataavat akkumulaattorin, X- tai Y-rekisterin. 6502-kielessä on monia operaatiota, jotka eroavat ainoastaan niiden kohteiltaan, kuten taulukosta 4 voidaan tulkita.

Operaation nimi	Operaation kohde tai tyyppi	Operaation kuvaus	Kaava
LDA, LDX, LDY	Rekisteri	Rekisterin lataus	M -> R
TAX, TXA, TAY, TYA, TSX, TXS	Rekisteri	Rekisterin siirto	R1 -> R2
STA, STX, STY	Rekisteri	Muistiin sijoitus	R -> M
ASL, ROL	Luku, muokkaus ja kirjoitus	Siirto vasemmalle sekä muistiin sijoitus	M << 1 -> M
LSR, ROR	Luku, muokkaus ja kirjoitus	Siirto oikealle sekä muistiin sijoitus	M >> 1 -> M
DEC, DEX, DEY	Aritmeettinen, RMW	Arvon vähentäminen yhdellä	M - 1 -> M
INC, INX, INY	Aritmeettinen, RMW	Arvon lisääminen yhdellä	M + 1 -> M
PLA, PLP	Pino	Poista akkumulaattori tai tilaliput pinosta	P -> A
PLP, PHP	Pino	Työnnä akkumulaattori tai tilaliput pinoon	A -> P
AND, ORA, EOR	Akkumulaattori	Muokkaa akkumulaattoria bittioperatiolla	F(A, M) -> A
ADC	Akkumulaattori	Lisäys "Carry" lipun kanssa	A + M + C -> A
SBC	Akkumulaattori	Vähennys "Carry" lipun kanssa	A - M - (1 - C) -> A
CLV, CLC, CLD, CLI	Lippu	Asettaa lipun päälle	L = 1
SED, SEI, SEC	Lippu	Asettaa lipun pos päältä	L = 0
BPL, BVC, BNE, BCC	Haarautuminen	Haarautuu jos lippu ei ole päällä	
BMI, BVS, BEQ, BCS	Haarautuminen	Haarautuu jos lippu on päällä	
CMP, CPY, CPX	Lippu	Vertaa arvoja ja asettaa tilaliput	
JSR, RTS	Ohjelman ohjaus	Aliohjelma kutsu ja paluu	
RTI, BRK	Ohjelman ohjaus	"Break" keskytytys ja siitä paluu	
JMP	Ohjelman ohjaus	Hyppää haluttuun osoitteeseen	

M: Osoitemuodon kertoma muisti **Rx**: Rekisteri **P:** Pino **A:** Akkumaattori **F:** Funktio **C:** "Carry" lippu **L:** Lippu
RMW: "**R**ead **M**odify and **W**rite"

Taulukko 4 6502-kielien ohjeet ryhmitettynä

Kirjoitettuna 6502-syntaksi riippuu mitä assembleria käytetään. Periaatteessa prosessorille koodin kirjoittamiseen voidaan käyttää korkeamman kielen kääntäjää. Yksi suosittu korkeamman kielen kääntäjä on cc65, joka kääntää C-kielen lähdekoodin 6502-prosessorin binaarille [21]. Esimerkkinä 6502-koodista voidaan käyttää kuvaa 15, joka pyörii kolme kertaa silmukassa vähentäen X-rekisteriä. Kuvasta nähdään myös, että osoitemuotoja ei kaikissa assemblereissa kirjoiteta koodiin, vaan ne ovat tulkittuja operaation mukaan.

```
  LDX #$05       ; Lataa arvo 5 X-rekisteriin. IMM
decrement:
  DEX            ; Vähennä X-rekisteriä yhdellä. Implied
  STX $0202      ; Sijoita X-rekisteri 0202 osoitteeseen. ABS
  CPX #$03       ; Vertaa X-rekisteriä arvoon 3. IMM
  BNE decrement  ; jos 0 lippu ei ole päällä,
                 ; siirrä ohjelmalaskuri kohtaan "decrement". Relative
  STX $0209      ; Sijoita X-rekisteri 0209 osoitteeseen. ABS
  BRK            ; "Break" keskeytys. Implied
```

Kuva 15. Esimerkki 6502-koodista

7.3 Operaatio-ohjeiden tulkinta

Kirjoitettu 6502-koodi muutetaan assemblerissa binäärimuotoon, ja tämä luetaan prosessorin tai emulaattorin suoritettavaksi. Yksi operaatio on tiivistetty kahdeksaan bittiin, joka sisältää operaatiokoodin ja osoitemuodon. Kun seuraava operaatio tarvitaan, se ladataan ohjelmalaskurin osoittamasta paikasta. Tämä tavun kokoinen luku on indeksi $FF-kokoiseen operaatiomatriisiin, mistä saadaan selville, mikä operaatio ja osoitemuoto seuraavaksi suoritetaan. Nämä kaksi operaatiota koostavat kuvan 12 kaksi ensimmäistä vaihetta. Emulaattorissa operaatiomatriisi on toteutettu jonona kuvan 16 tapaisia tietueita, jotka vastaavat kuvan 17 alkioita.

```
typedef struct Instruction {
    u8 instructionCode;
    u8 addressMode;
    u8 cycles;
} Instruction;
```

Kuva 16. Operaatiomatriisin arvo emulaattorissa. Kaksi ensimmäistä arvoa ovat tyypiltään enumeraatioarvoja, ja sykli kertoo, kuinka monta kellosykliä operaation suorituksessa kestää.

Prosessori olisi voitu myös voinut toteuttaa suoraan tulkitsemalla ohjelmalaskurin osoittamaa arvoa ja toteuttaa 151 kohtainen ”Switch … case”-rakenne, joka mahdollisesti nopeuttaisi prosessorin toimintaa. Emulaattorissa valittu lähestymismuoto kuitenkin mahdollistaa helpon ohjekoodien lisäämisen matriisiin. Tämä on etenkin tärkeää, kun kaseteissa tulee vastaan epävirallisia ohjeita. Epäviralliset ohjeet ovat kuvassa 17 näkyvät tyhjät osat. Jokainen arvo $FF-alueella vastaa jotain operaatioita, ja ohjelmoijat ovat löytäneet tapoja käyttää näitä epävirallisiakin operaatioita hyödykseen.

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BRK Implied 1 7	ORA (IND, X) 2 6				ORA ZP 2 3	ASL ZP 2 5		PHP Implied 1 3	ORA IMM 2 2	ASL Accum 1 2			ORA ABS 3 4	ASL ABS 3 6		0
1	BPL Relative 2 2**	ORA (IND), Y 2 5*				ORA ZP, X 2 4	ASL ZP, X 2 6		CLC Implied 1 2	ORA ABS, Y 3 4*				ORA ABS, X 3 4*	ASL ABS, X 3 7		1
2	JSR Absolute 3 6	AND (IND, X) 2 6			BIT ZP 2 3	AND ZP 2 3	ROL ZP 2 5		PLP Implied 1 4	AND IMM 2 2	ROL Accum 1 2		BIT ABS 3 4	AND ABS 3 4	ROL ABS 3 6		2
3	BMI Relative 2 2**	AND (IND), Y 2 5*				AND ZP, X 2 4	ROL ZP, X 2 6		SEC Implied 1 2	AND ABS, Y 3 4*				AND ABS, X 3 4*	ROL ABS, X 3 7		3
4	RTI Implied 1 6	EOR (IND, X) 2 6				EOR ZP 2 3	LSR ZP 2 5		PHA Implied 1 3	EOR IMM 2 2	LSR Accum 1 2		JMP ABS 3 3	EOR ABS 3 4	LSR ABS 3 6		4
5	BVC Relative 2 2**	EOR (IND), Y 2 5*				EOR ZP, X 2 4	LSR ZP, X 2 6		CLI Implied 1 2	EOR ABS, Y 3 4*				EOR ABS, X 3 4*	LSR ABS, X 3 7		5
6	RTS Implied 1 6	ADC (IND, X) 2 6				ADC ZP 2 3	ROR ZP 2 5		PLA Implied 1 4	ADC IMM 2 2	ROR Accum 1 2		JMP Indirect 3 5	ADC ABS 3 4	ROR ABS 3 6		6
7	BVS Relative 2 2**	ADC (IND), Y 2 5*				ADC ZP, X 2 4	ROR ZP, X 2 6		SEI Implied 1 2	ADC ABS, Y 3 4*				ADC ABS, X 3 4*	ROR ABS, X 3 7		7
8		STA (IND, X) 2 6			STY ZP 2 3	STA ZP 2 3	STX ZP 2 3		DEY Implied 1 2		TXA Implied 1 2		STY ABS 3 4	STA ABS 3 4	STX ABS 3 4		8
9	BCC Relative 2 2**	STA (IND), Y 2 6			STY ZP, X 2 4	STA ZP, X 2 4	STX ZP, Y 2 4		TYA Implied 1 2	STA ABS, Y 3 5	TXS Implied 1 2			STA ABS, X 3 5			9
A	LDY IMM 2 2	LDA (IND, X) 2 6	LDX IMM 2 2		LDY ZP 2 3	LDA ZP 2 3	LDX ZP 2 3		TAY Implied 1 2	LDA IMM 2 2	TAX Implied 1 2		LDY ABS 3 4	LDA ABS 3 4	LDX ABS 3 4		A
B	BCS Relative 2 2**	LDA (IND), Y 2 5*			LDY ZP, X 2 4	LDA ZP, X 2 4	LDX ZP, Y 2 4		CLV Implied 1 2	LDA ABS, Y 3 4*	TSX Implied 1 2		LDY ABS, X 3 4*	LDA ABS, X 3 4*	LDX ABS, Y 3 4*		B
C	CPY IMM 2 2	CMP (IND, X) 2 6			CPY ZP 2 3	CMP ZP 2 3	DEC ZP 2 5		INY Implied 1 2	CMP IMM 2 2	DEX Implied 1 2		CPY ABS 3 4	CMP ABS 3 4	DEC ABS 3 6		C
D	BNE Relative 2 2**	CMP (IND), Y 2 5*				CMP ZP, X 2 4	DEC ZP, X 2 6		CLD Implied 1 2	CMP ABS, Y 3 4*				CMP ABS, X 3 4*	DEC ABS, X 3 7		D
E	CPX IMM 2 2	SBC (IND, X) 2 6			CPX ZP 2 3	SBC ZP 2 3	INC ZP 2 5		INX Implied 1 2	SBC IMM 2 2	NOP Implied 1 2		CPX ABS 3 4	SBC ABS 3 4	INC ABS 3 6		E
F	BEQ Relative 2 2**	SBC (IND), Y 2 5*				SBC ZP, X 2 4	INC ZP, X 2 6		SED Implied 1 2	SBC ABS, Y 3 4*				SBC ABS, X 3 4*	INC ABS, X 3 7		F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Kuva 17. 6502-prosessorin operaatioiden tulkintataulu, jossa ylempi arvo kertoo operaation, alempi osoitemuodon ja numerot syklimäärän [22, s. 10].

7.4 Prosessorikomponentin toiminta

Prosessorikomponentti alustetaan ja päivitetään ohjelman rungosta. Muista komponenteista poiketen prosessorissa ei ole toteutettua siivousfunktiota, koska prosessori ei varaa dynaamista muistia. Prosessorikomponentti on riippuvainen muiden komponenttien antamasta muistista.

7.4.1 Prosessorin alustus ja RESET signaali

Alustusfunktiossa X-, Y-, akkumulaattori- sekä lippurekisterit asetetaan arvoon 0. Pino-osoitin asetetaan arvoon $FF, mikä osoittaa pinon alhaisimpaan arvoon. Lisäksi prosessorin ohjelmalaskuri asetetaan paikasta $FFFC haettuun 16-bittiseen lukuun. Alustusfunktio toimii myös prosessorin ”RESET” toimintona. ”RESET” on signaali, minkä 6502-prosessori saa käynnistyksen tai nollauksen yhteydessä. Tälle signaalille on varattu $FFFC-osoitteesta kaksitavuinen luku, mistä luetaan ohjelman aloitusosoite. [14, s. 13.]

7.4.2 Prosessorin päivitys

Prosessorin päivitysfunktio vastaa yhtä prosessorin kellotusta. Normaalisti yhden operaatio-ohjeen suorittaminen prosessorilla kestää enemmän kuin yhden kellosyklin, esimerkiksi ABS-osoitemuodossa pelkästään ohjeen datan osoitteen haussa kestää kaksi kellosykliä. Tässä emulaattorissa ohjeiden suorittaminen suoritetaan kokonaan aina, kun viimeksi suoritetun ohjeen syklit on suoritettu. Tämä saattaa tuottaa alkuperäisestä alustasta poikkeavia ominaisuuksia, esimerkiksi jos PPU muuttaa rekisterien arvoja syklien välillä. Tämä tarkoittaa, että emulaation tarkkuus heikkenee, mutta tässä työssä päätettiin tyytyä yksinkertaisempaan ratkaisuun monimutkaisen sijaan. Tämä on hyvin yleinen ja tunnettu ratkaisu tunnettuun ongelmaan emulaattorikehityksessä. Vaihtoehtoinen ratkaisu olisi päivittää komponentteja aina, kun prosessori käsittelee osoitelinjaa. Tämän tyylinen komponenttien päivitys voisi tuottaa tarkempaa emulaatiota, mutta se on myös monimutkaisempi. Tätä tapaa on muun muassa käytetty MedNES-emulaattorissa, jonka tavoitteena on olla kellotarkka toteutus alustasta [23].

7.4.3 Osoitemuotojen toteutus

Kun seuraavaa ohjetta aletaan suorittamaan, haetaan osoitinväylästä ohjelmalaskurin osoittamasta paikasta ohje, joka puretaan operaatiomatriisin avulla. Operaatiomatriisista saadaan kuvan 16 tietue, jota käytetään kahdessa ”Switch … case”-rakenteessa. Ensimmäisessä toteutetaan osoitemuodot kuvan 18 tapaan. Kuvassa 18 on esimerkkinä kaksi osoitemuotoa. Ensimmäinen on ”Immediate”-osoitemuoto, missä ohjelmalaskurin jälkeinen tavu on operaation data. Toisena on ”Zero page”-osoitemuoto, missä ohjeesta seuraava tavu kertoo datan osoitteen. Kuvasta 18 nähdään myös, kuinka yksinkertaisia osoitemuodot pohjimmiltaan ovat. Kuva käsittelee ensimmäiset kolme vaihetta prosessorin toiminnasta kuvasta 12.

```
u8 opcode = bus_read8(cpu.pc);
// Increase pc to next data point
cpu.pc += 1;

Instruction instruction = instructionTable[opcode];
cpu.cycles = instruction.cycles;
u16 addr = 0;
// fetch required address or data
switch(instruction.addressMode) {
    case IMM:
        {
            addr = cpu.pc;
            cpu.pc += 1;
        } break;
    case ZP:
        {
            addr = bus_read8(cpu.pc);
            cpu.pc += 1;
        } break;
         /* Rest of the addressing modes */
    default:
        ABORT("Error addressing mode 0x%04X", instruction.addressMode);
}
```

Kuva 18. Prosessorin ohjeen haku ja esimerkkejä osoitemuotojen toteutuksesta

7.4.4 Operaatiokoodien toteutus

Kun ohjeen osoite on tulkittu, voidaan suorittaa operaatiokoodi. Operaatiokoodit suoritetaan samankaltaisessa ”Switch … case”-rakenteessa kuin osoitemuodot. Kuvassa 19 on esimerkkejä operaatioiden toteutuksista. Ensimmäisenä toteutuksena on AND-operaatio, mikä tekee AND-operaation akkumulaattorilla ja haetulla muistilla. Muistin hakeminen on prosessorin toiminta kuvan

12 mukaisesti valinnainen vaihe, jota kaikki operaatiot eivät suorita. Kun AND-operaatio on suoritettuna, saadulle tulokselle asetetaan ”Negative”-lippu päälle, jos tavun korkein bitti on päällä, ja ”Zero”-lippu, jos arvosta tulee nolla. Kaikki operaatiot eivät muokkaa kaikkia lippuja, kuten ”and”-operaatiosta voidaan tulkita. Operaatioiden liput on tarkistettu masswerk.at-verkkosivustolta, missä on dokumentoitu kaikkien operaatioiden lippujen käytöt [24]. Seuraava operaatio on ”CLC”, joka asettaa ”Clear”-lipun pois päältä. Tämä operaatio ei tarvitse dataa, joten tämä operaatio käyttää ”Implied”-osoitemuotoa taulukon 3 mukaisesti.

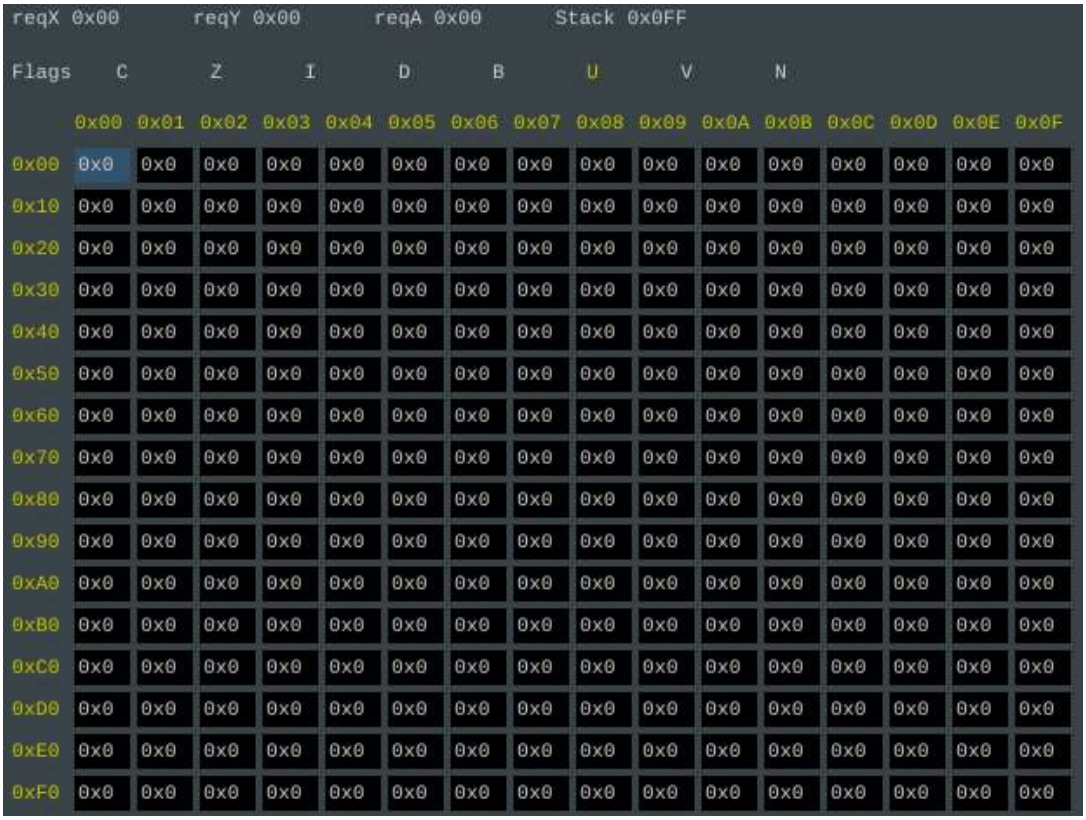
```
// perform the instruction (this is hopefully optimized to jumptable)
switch (instruction.instructionCode) {
        case AND: //and (with accumulator), A AND M -> A
        {
                fetched = fetch_data(addr);
                cpu.accumReq &= fetched;
                // Set flags
                cpu_set_flag(Negative, cpu.accumReq & 0x80);
                cpu_set_flag(Zero, cpu.accumReq == 0);
        } break;
        case CLC: //clear carry
        {
                cpu_set_flag(Carry, 0);
        } break;
        case CMP: //compare (with accumulator) A - M
        {
                fetched = fetch_data(addr);
                u16 temp = (u16)cpu.accumReq - (u16)fetched;
                // Set flags
                cpu_set_flag(Carry, cpu.accumReq >= fetched);
                cpu_set_flag(Negative, temp & 0x0080);
                cpu_set_flag(Zero, (temp & 0x00FF) == 0x0);
        } break;
        case LDA: //load accumulator, M -> A
        {
                fetched = fetch_data(addr);
                cpu.accumReq = fetched;
                // Set flags
                cpu_set_flag(Negative, cpu.accumReq & 0x0080);
                cpu_set_flag(Zero, cpu.accumReq == 0x0);
        } break;
        /* Rest of the instruction implementations */
        default:
                ABORT("Unknown instruction 0x%04X", instruction.instructionCode);
}
```

Kuva 19. Esimerkki prosessorin ohjeiden toteutuksesta

7.5 Prosessorin visualisointi

On hyödyllistä, että prosessorin rekisterejä ja suoritetun koodin tilaa voidaan visualisoida. Visualisoinnin lisäksi lisätään mahdollisuus astua prosessoria ohje kerrallaan sekä asettaa keskeytysarvo, johon prosessorin toiminta pysähtyy. Näitä ominaisuuksia voidaan käyttää kasettien tutkimiseen ja virheenetsintään. Rekisterien visualisointi lisätään samaan komponenttiin kuin muistin

visualisointi, kuten kuvassa 20 näkyy. Akkumulaattori-, pino-, X- ja Y-rekisterin on visualisoitu yksinkertaisen tekstin avulla, joka kertoo kunkin rekisterin arvon. Lippurekisterin jokainen arvo visualisoidaan erikseen, niitä vastaavilla kirjaimilla. Lipun ollessa päällä se väritetään muusta tekstistä erottuvalla värillä, kuvan 20 tavoin.



Kuva 20. muistin visualisointi prosessorin rekisterien kanssa

Askellusominaisuutta varten toteutettiin emulaattoriin ”virheenetsintätila”, joka korvaa kuvassa 3 näkyvän päivityssilmukan. Tämän ominaisuuden tavoitteena on antaa käyttäjän katsoa ja ohjata prosessorin toimintaa tarkemmin. Tätä käytetään vain, jos käyttäjä on aktivoinut virheenetsintätilan käyttöliittymästä. Tässä tilassa emulaatiota suoritetaan yhden ohjeen verran, kun käyttäjä painaa kuvassa 22 näkyvää ”Step”-nappia. Lisäksi toteutettiin keskeytysarvo, jonka käyttäjä kirjoittaa heksadesimaalina. Kun prosessorin seuraava suoritettava ohje on keskeytysarvoa vastaava, emulaattori vaihdetaan virheenetsintätilaan.

Seuraavaksi työssä toteutettiin ohjeiden visualisointi. Ohjeiden visualisointi toteutettiin osana kasettia mistä visualisointikomponentti voi kysyä onko muistiarvolle olemassa pätevää käännöstä.

Ohjeiden binaarimuodon muuttaminen takaisin lähdekoodimuotoon on varsinkin 6502-kielessä helppoa koska tarvitsee vain katsoa prosessorissa toteutetusta operaatiomatriisista oikea käännös koodille. Koska operaatio on 1–3 tavua leveä, osa tavuista jätetään purkamatta operaatioiksi. Kasetissa on tietty määrä 16 KB kokoisia ohjelmamuistipankkeja, joista jokainen puretaan erikseen taulukoihin. Purkaminen tapahtuu kartoittajan alustuksessa. Purkamisen jälkeen kartoittajan vastuulla on tuottaa sama muistiosoite, joka tulisi normaalista muistin lukemisesta. Tätä muistiosoitetta käytetään purettuun kooditaulukkoon.

Tässä toteutustavassa ohjeiden purkaminen voidaan tehdä etukäteen, mikä nopeuttaa tiedon käsittelyä emulaattorin ajon aikana. Jos ohjelma suorittaa koodia RAM-muistista, ei sopivaa tulkintaa voida antaa ohjeelle. 6502-prosessorille on täysin laillista suorittaa ohjeita mistä tahansa prosessorinosoiteväylästä, ja tätä ominaisuutta käytetään useasti dynaamisen koodin suorittamiseen [25]. Tämän takia seuraavaksi suoritettava ohje visualisoidaan erikseen tekstin avulla, kuten kuvassa 21 nähdään.

Kuvassa 21 nähdään, että ohjelman seuraavaksi suoritettava ohje näytetään selvästi eri värillä kuin muut operaatiot ja se on aina keskellä luotua taulukkoa. Lisäksi punaisella oleva ohje kertoo ohjelman keskeytys paikan arvon.

Kuvassa nähdään ”Continue”- ja ”Reset”-painike. ”Continue”-painike palauttaa ohjelman virheenetsintätilasta normaaliin tilaan ja ”Reset”-painike käyttää prosessorin ”Reset”-signaalia sekä palauttaa PPU:n aloitustilaan.

Kuvasta nähdään myös, kuinka ohjeiden leveys vastaa operaation osoitemuodon leveyttä. Tämä vastaa myös taulukossa 3 olevia arvoja. Esimerkiksi osoitteessa $C009 oleva ”absolute”-osoitemuoto tekee ohjeesta kolme tavua leveän ja $C008-osoitteessa oleva ”implied”-osoitemuoto tekee ohjeesta vain yhden tavun kokoisen.

Kuva 21. Prosessorin tilan visualisointi

8 Kuvankäsittely-yksikkö

Picture Prosessing Unit eli PPU on viimeinen työssä toteutettu komponentti. PPU:n vastuulla on tuottaa kuva prosessorin ohjaamalla tavalla. PPU:n piirtäminen voidaan jakaa kahteen osaan: taustakuviin ja dynaamisiin spritekuviin. PPU nimensä mukaan käsittelee vain kuvia, eikä NES alustana pysty piirtämään yksittäisiä pikseleitä tai mielivaltaisen kokoisia kuvia. Kaikki alustan kuvat ovat 8x8 pikselin kokoisia. Jos halutaan piirtää isompia kuvia, pitää kuvia yhdistää. Kuvien yhdistämistä käytetään esimerkiksi suositun *Super Mario Bros.*-pelin päähahmoa piirrettäessä, jossa käytetään kuuden kuvan yhdistelmää. PPU:ta ohjataan prosessorin osoiteväylän kautta kahdeksan rekisterin avulla. Lisäksi sillä on oma osoiteväylä mihin on yhdistetty PPU:n tarvitsemat muistit: kuviotaulut, nimitaulu ja palettimuisti. Nämä muodostavat 16 KB ($0–$3FFF) osoitealueen mitä PPU pystyy käsittelemään. [17.]

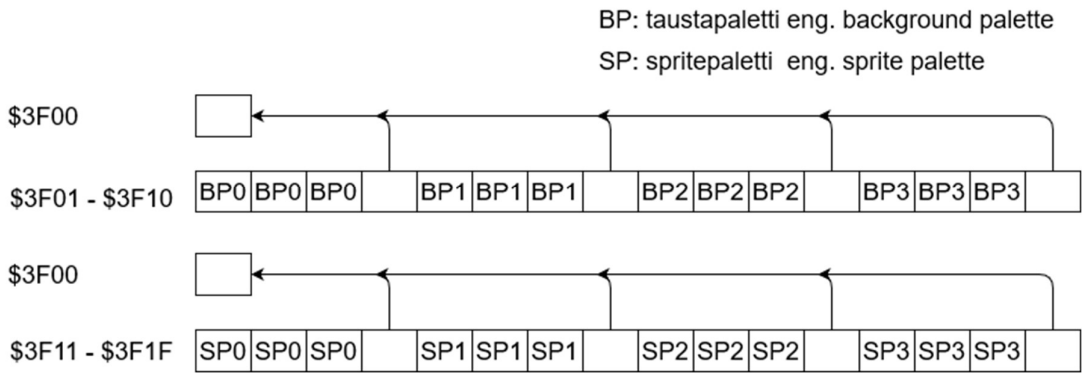
8.1 Palettimuisti

Palettimuisti sijaitsee PPU:n osoiteväylässä alueella $3F00–$3FFF. Tämä muisti on peilattuna 32-tavun välein. Tätä muistia prosessori pystyy kirjoittamaan PPU:n rekisterien avulla. NES-alustalla on yksi yleinen taustaväri, neljä taustapalettia ja neljä spritepalettia. PPU:lla on 64 ennalta määriteltyä väriä, joita paletit pystyvät käyttämään. Tämä perustuu NTCS-videoformaattiin mitä tämän ajan televisiot käyttivät. Paletissa jokaista väriä voidaan viitata yhdellä tavun kokoisella numerolla kuvan 22 näyttämästä taulukosta ja yksi paletti koostuu kolmesta väristä.

Kuva 22. Numeroarvoja vastaavat värit [26].

Kuvia piirrettäessä paletissa olevia värejä viitataan arvoilla 0–3. Taustaa piirtäessä automaattisesti käytetään taustapaletteja ja spitejä piirtäessä spritepaletteja. Koska kuville halutaan antaa

myös läpinäkyvyyttä, on annettu yksi yleinen väri minkä ohjelmoija voi asettaa PPU:n osoiteväylässä osoitteeseen $3F00. Kuvadatassa läpinäkyvyys vastaa arvoa 0. Muistia käsitellessä palettien neljäs tavu viittaa yleiseen taustaväriin, mikä on visualisoituna kuvassa 23. [27.]

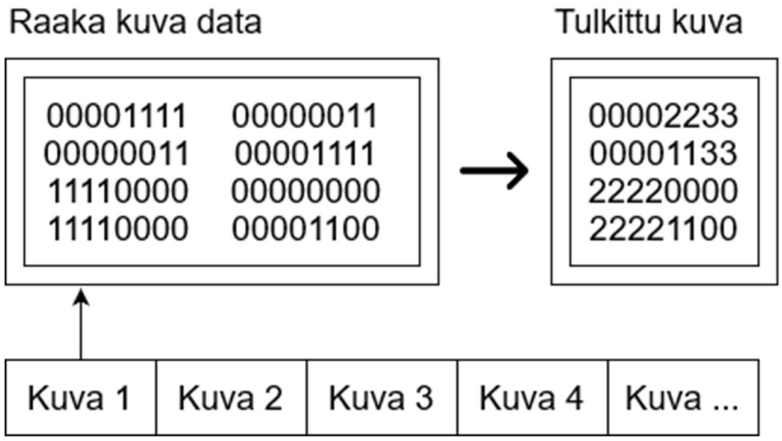


Kuva 23. Väripalettien visualisointi muistissa

8.2 Kuviotaulut

Kuviotaulut sijaitsevat PPU:n muistissa alueella $0–$1FFF. Fyysisesti tämä muisti sijaitsee kasetissa ja tätäkin muistia pitää käsitellä muistikartoittajien kautta. Kartoittajien avulla ohjelmoijat ovat pystyneet käyttämään enemmän muistia kuin osoitealueen tarjoaman 16 KB määrän. Muistin tarkoituksena on sisältää kaikki PPU:n piirrettävät kuvat. Tätä muistia visualisoi kuvassa 4 oleva viiva kasetti- ja PPU-komponentin välillä.

Kuviotaulut koostuvat 16x16 kuvasta, joista jokainen kuva on 16-tavun kokoinen. Muistissa kuvat sijaitsevat peräkkäin ja niitä on 512 kappaletta yhdessä merkkimuistipankissa. Kuvat koostuvat 8x8 kokoisesta matriisista 0–3 arvoja, missä 0 merkkaa läpinäkyvyyttä ja 1–3 vastaa valitun paletin värejä. Kuvia voi piirtää millä tahansa paletilla, mitä hyödynnetään useasti kuvien uudelleen käytössä. Kuva datassa yksi rivi (kahdeksan pikseliä), saadaan kun luetaan kaksi tavua ja yhdistetään alemman- ja ylemmän tavun vastaavat bitit numeroiksi. [28.] Tätä prosessia on visualisoitu kuvassa 24, jossa visualisoidaan kuvan lukemista.



Kuva 24. Kuvan purkaminen muistista

8.2.1 Kuvien visualisointi työssä

Kuviotaulua visualisoidessa toteutettiin yleinen kuvien piirtokomponentti työhön. Tämä tarvitaan koska käytännössä kaikki emuloitava piirtäminen hoidetaan ohjelmiston koodissa ja visualisointi tapahtuu näytönohjaimessa. Tämän tavoitteena on pystyä piirtämään ja ylläpitämään ohjelmassa tuotettua kuvadataa. Käytännössä tämä luo tietyn kokoisen OpenGL-tekstuurin, lataa sen näytönohjaimelle ja aina kun tekstuuria päivitetään, päivitetään myös näytönohjaimelle näkyvää tekstuuria.

OpenGL:n avulla pystytään hallitsemaan tekstuurin tulkinta tapaa. Tätä voidaan käyttää tekstuurin laadun parantamiseen eri tarkoituksien mukaisesti. Tätä operaatiota kutsutaan tekstuurin suodattamiseksi. Suodattaminen määrittelee mitä prosessia käytetään, kun tekstuurin näytettä haetaan. Työssä käytettiin ”GL_NEAREST”-parametria suodatuksessa, joka valitsee lähimmän mahdollisimman näytteen kuvaa piirtäessä [29]. Tällä tavalla piirretyt kuvat ovat mahdollisimman teräviä, kun ne piirretään 2D tasolle. Kuvien käyttö on tarkemmin kuvattuna kuvassa 25.

```
typedef struct ImageView {
        u8*     data;
        u32     w, h;
        GLuint  tex;
} ImageView;

static ImageView
imageview_create(u32 w, u32 h) {

        ImageView ret = { .w = w,.h = h,.data = calloc(w * h, sizeof(Color)) };
        GLCHECK(glGenTextures(1, &ret.tex));
        GLCHECK(glBindTexture(GL_TEXTURE_2D, ret.tex));

        GLCHECK(glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, w, h, 0, GL_RGBA,
                GL_UNSIGNED_BYTE, ret.data));

        GLCHECK(glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST));
        GLCHECK(glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST));

        GLCHECK(glBindTexture(GL_TEXTURE_2D, 0));

        return ret;
}

static void
imageview_update(ImageView* view) {

        GLCHECK(glBindTexture(GL_TEXTURE_2D, view->tex));
        GLCHECK(glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, view->w, view->h,
                GL_RGBA, GL_UNSIGNED_BYTE, view->data));
        GLCHECK(glBindTexture(GL_TEXTURE_2D, 0));
}
```

Kuva 25. Generoidun kuva datan piirtäminen

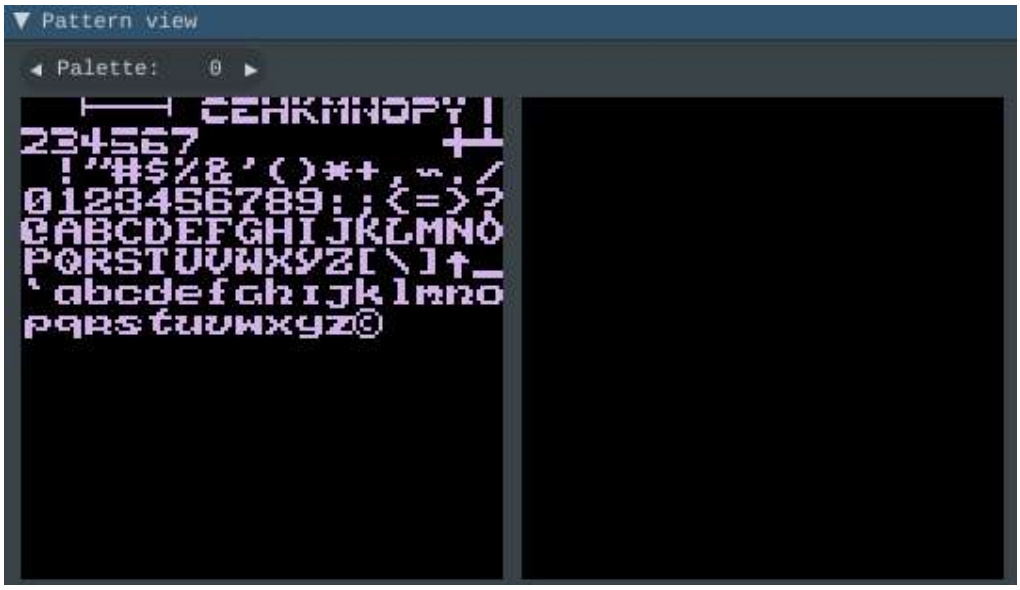
8.2.2 Kuviotaulujen visualisointi

Kuviotauluja on työssä visualisoitu kahdessa 16x16 paneelissa mitkä piirtävät tauluissa olevan datan ”ImageView”-tietueeseen. Kun kuvat on tuotettu, ne piirretään Nuklear-kirjaston avulla käyttöliittymään. Kuvien tuottaminen tapahtuu piirtämällä yksi kuviotaulun kuva kerrallaan ja hakemalla sen kuvan data rivi kerrallaan. Kun yksi kuvassa oleva rivi on saatu, voidaan pikselit piirtää kuvatietueeseen. Tämä on kuvassa 26 näytetty tarkemmin koodi esimerkkinä.

```
static void
ppu_render_patterntable(u8 index /*2 pattern tables so this is 0 or 1*/, u32 paletteIndex) {
    for(u16 tileY = 0; tileY < NUM_TILES; tileY++) { // FOR TILE Y

        for(u16 tileX = 0; tileX < NUM_TILES; tileX++) { // FOR TILE X

            u16 tileoffset = (tileY * NUM_TILES + tileX)
                * 16 // one tile size
                + index * 0x1000; // which patterntable
            for(u16 pixelY = 0; pixelY < TILE_DIM; pixelY++) { // FOR PIXEL Y

                u8 lsb = ppu_read(tileoffset + pixelY);
                u8 msb = ppu_read(tileoffset + 8 + pixelY);
                u16 imageY = tileY * TILE_DIM + pixelY;

                for(u16 pixelX = 0; pixelX < TILE_DIM; pixelX++) {  // FOR PIXEL X

                    u16 imageX = (tileX * TILE_DIM + (7 - pixelX));
                    u8 pixel = ((msb & 0x1) << 1) | (lsb & 0x1);
                    lsb >>= 1;
                    msb >>= 1;
                    // Set pixel in texture
                    Color color = ppu_palette_get_color(pixel, paletteIndex);
                    // Draw the pixel
                    memcpy(ppu.pattern[index].data +
                            (imageY * ppu.pattern[index].w + imageX) * sizeof(Color),
                            &color, sizeof(Color));
                }
            }
        }
    }
}
```

Kuva 26. Kuviotaulujen piirtäminen

Koska kuviotaulut eivät sisällä tietoa millä paletilla ne piirretään, on samalla annettu käyttäjälle mahdollisuus vaihtaa käytettyä väripalettia. Tämä toimii myös väripalettien visualisointina. Käyttäjä pystyy muuttamaan millä paletilla kuva piirretään painamalla kuviotaulujen päällä olevaa nuolta, kuten kuvassa 27 näkyy. Käyttäjälle on annettu mahdollisuus käyttää taustapaletteja 0–3 ja spritepaletteja 4–7.



Kuva 27. Kuviotaulun visualisointi työssä. Kuviotaulu on saatu ”nestest”-kasetista, jolla testataan emulaattorin toimintaa

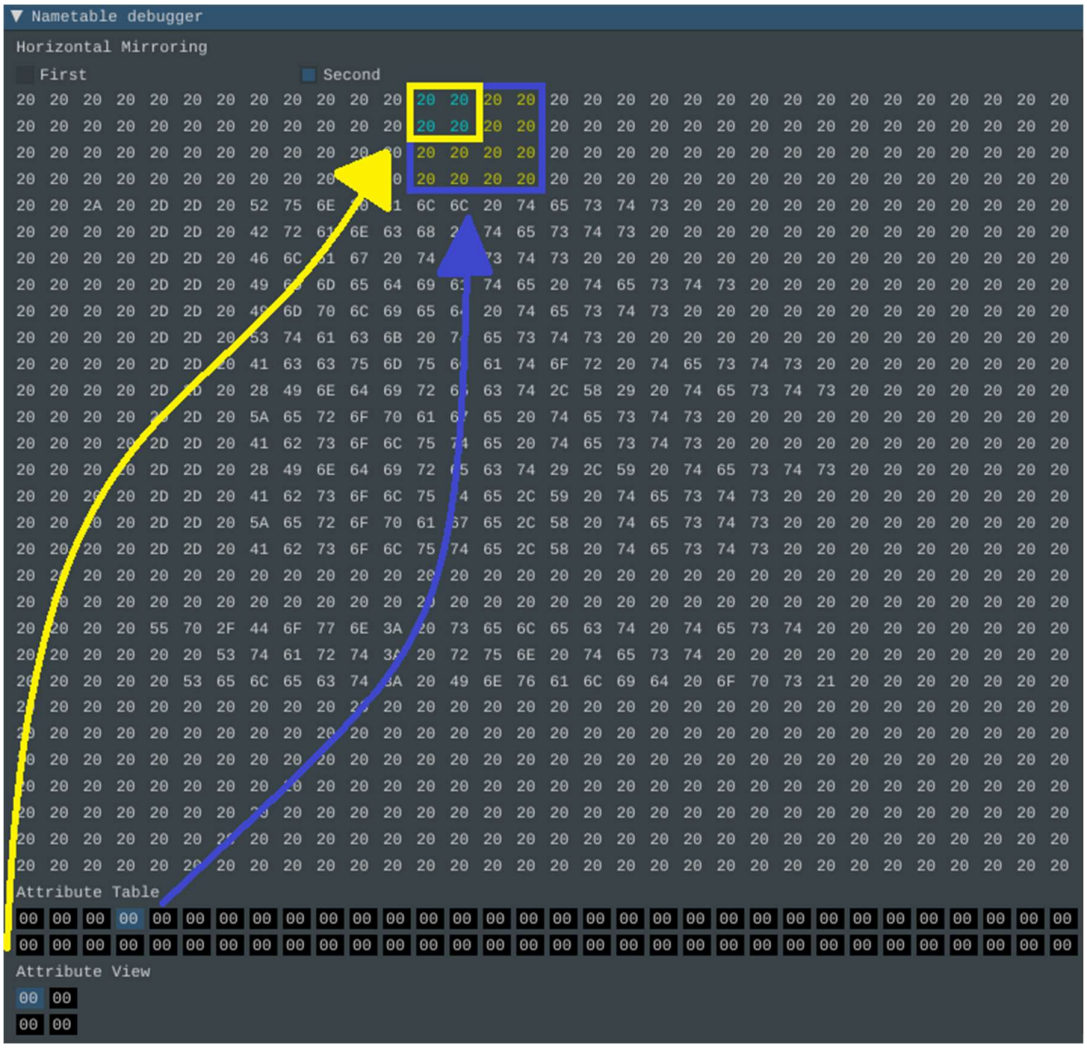
8.3 Nimitaulu

Nimitaulut määrittelevät taustakuvien asetelman. Tätä muistia prosessori pystyy kirjoittamaan PPU:n rekisterien kautta. PPU:lla on neljä nimitaulua PPU:n osoiteväylässä muistialueella $2000–$3EFF ja näistä kaksi on peilauksia toisistaan. Nimitaulujen peilauksen avulla voidaan saavuttaa esimerkiksi ”Super Mario Bros.” pelin ruudun oikealle liikkumisen tai ”Ice Climber” pelin ruudun ylöspäin liikkumisen. Tätä kutsutaan ruudun vierityksiksi. PPU:lla on eri peilaustyyppejä, mitkä määrittelevät kasetit. Peilaustyypit määrittelevät mihin nimitauluun osoitteet viittaavat, minkä avulla pystytään manipuloimaan taustakuvien asetelmia. INES-formaatissa ”lippu 6”:den ensimmäinen bitti määrittelee, onko peilaus vertikaalinen vai horisontaalinen. Lisäksi voidaan käyttää yhden ruudun peilausta, missä käytetään vain toista nimitaulua. Nämä yleisimmät peilaustyypit ovat tässä työssä toteutettuna, mutta näiden lisäksi on myös tietyissä kasettityypeissä käytettyjä peilaustyyppejä, kuten neljän ruudun peilaus pelissä ”Rad Racer II”. [30.]

Kuvadatan lisäksi nimitauluun kuuluvat paletit, jolla kuvat piirretään. Attribuuttitaulu on 64-tavuinen taulu, joka määrittelee millä paleteilla 4x4 taustakuvaa piirretään. Tämä sijaitsee molempien nimitaulujen lopussa. Attribuuttitaulussa yksi tavu kattaa 4x4 alueen nimitaulusta. Tämä voidaan vielä purkaa neljään osaan, mikä tuottaa kaksi bittisiä arvoja (0–3). Nämä arvot kertovat

millä taustapaletilla kuva piirretään. Tämä tuottaa 2x2 alueita, jotka piirretään samoilla väripaleteilla. [14, s. 19–21.]

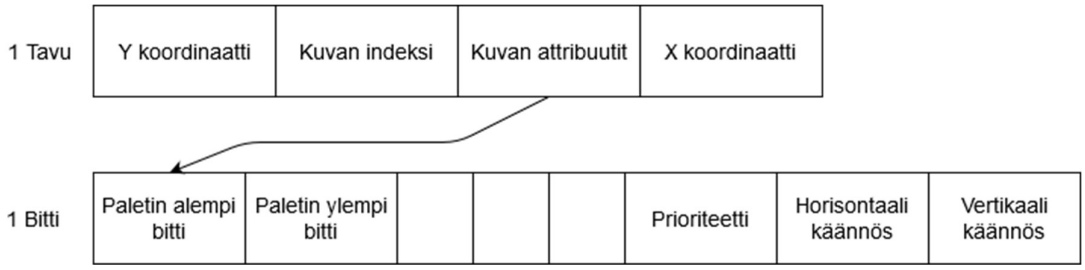
Tätä ominaisuutta on visualisoitu kuvassa 28, jossa näkyy työhön toteutettu nimitaulun visualisointityökalu. Työkalussa voidaan valita, kumpi nimitaulu halutaan visualisoida. Työkalussa olevat numerot vastaavat suoraan mitä ruudulle piirretään, jos ruutua ei ole vieritetty. Nimitaulun alapuolella on attribuuttitaulun visualisointi, joka näyttää jokaisen taulussa olevan arvon. Arvoja painamalla käyttäjä pystyy maalamaan sen osan nimitaulusta, jota attribuutit vastaavat. Tämä vastaa sinistä nuolta kuvassa 28. Näiden lisäksi käyttäjä voi valita attribuutin näyttämän 2x2 alueen arvon ja maalata sen nimitauluun, joka on kuvassa näytetty keltaisella nuolella.



Kuva 28. nimitaulun visualisointi ja attribuuttitaulun toiminta. Kuva on ”nestest”-kasetin tuottama taustakuva, jonka numerot vastaavat kuvan 28 kuviotaulun kuvia.

8.4 Objektin määritemuisti

OAM eli “Object Attribute Memory” on PPU:n sisällä sijaitseva muisti, joka sisältää listan piirrettäviä spritejä. OAM-muisti on PPU:n sisäinen muisti ja se ei sijaitse PPU-osoiteväylässä, joten sitä ei voi käsitellä suoraan muistiosoitteiden kautta. OAM:iin mahtuu 64 spriteä, joita voidaan käyttää yhden ruudun piirron aikana. Jokainen sprite tarvitsee neljä tavua dataa piirtämistä varten. Yhteen kuvaan tarvittava data on kuvattu kuvassa 29 kuvan attribuuttien kanssa. [31.]



Kuva 29. Sprite data ja kuvan attribuutit

8.4.1 Suora muistiosoitus

OAM-muistiin voidaan kirjoittaa joko PPU:n rekisterien kautta tai käyttämällä PPU:n DMA eli ”Direct Memory Access”-ominaisuutta, joka on näytetty kuvassa 4 prosessorin ja PPU:n välillä. DMA on yleisimmin käytetty tapa saada spritetiedot PPU:lle. DMA:n avulla voidaan kerralla kopioida kaikki spritedata PPU:hun. DMA aktivoidaan kirjoittamalla PPU:n $4014 osoitteessa sijaitsevaan rekisteriin. Rekisteriin kirjotettu tavu kertoo miltä prosessorin muistisivulta DMA hakee spritedatan. DMA:n ollessa aktiivinen prosessorin toiminta on kokonaan keskeytetty 512 prosessorin kellotuksen ajaksi. Lisäksi DMA pystyy alkamaan vain parittomilla prosessorin päivityksillä. [14, s. 18.] DMA on sunnilleen neljä kertaa nopeampaa, kuin yksikerrallaan kuvadatan kirjoittaminen rekisterien kautta [31].

8.4.2 OAM visualisointi

OAM visualisointi hoidettiin työhön toteutetulla yleisellä kuvakomponentilla. OAM-muistin kuvat piirretään 8x8 taulukossa. Toteutukseltaan tämä on hyvin samanlainen kuin kuviotaulujen visu-

alisointi. Kuvassa näkyy kokonaisuudessaan kaikki 64 spritekuvaa mitä voidaan piirtää. Kun ohjelma haluaa piirtää vähemmän kuin 64 spritekuvaa, pitää ohjelman asettaa ei haluttujen spritekuvien X- tai Y-arvo näytön ulkopuolella sijaitsevaan alueeseen. Kuvasta 31 näkyy, että ylhäällä vasemmalla on ainoa piirrettävä spritekuva. Tämä voidaan päätellä sen Y- ja X-arvojen perusteella, jotka ovat näytön sisällä. Tätä helpottaakseen työhön on myös visualisoitu kuvien attribuutti, kuvaindeksi, X- ja Y-arvot. Tämä toteutettiin tekstipohjaisena visualisointina, kuten kuvassa 30 näkyy.

Kuva 30. OAM visualisointi. Kuvassa on avoimen lähdekoodin ”Nova The Squirrel”-peli [32].

8.5 PPU:n rekisterit

Rekisterien kautta prosessori pystyy kirjoittamaan ja lukemaan tietoa PPU:sta, sekä kontrolloimaan sen toimintaa. Rekistereitä on yhteensä 9 kappaletta, mutta tärkeimpiä PPU:n toiminnan ymmärtämiselle ovat PPUSATUS, PPUSCROLL, PPUADDR ja PPUDATA.

PPUSTATUS sijaitsee osoitteessa $2002 ja se kertoo kolme eri tilaa PPU:n toiminnasta. Yksi tiloista ilmaisee, kun piirrettävän ruudun ensimmäinen sprite ollaan piirtämässä. Seuraava tila kertoo, kun PPU on odotustilassa piirtämään seuraavaa ruutua. Tätä kutsutaan englanniksi ”Vertical Blank”-nimellä. Viimeinen tila ilmaisee, jos enemmän kuin 8 spriteä on piirretty rivillä. Tätä rekisteriä pystyy pelkästään lukemaan ja se asetetaan PPU:n sisäisesti. Tämän rekisterin lukeminen tyhjentää PPUSCROLL- ja PPUADDR-rekisterien salvan. [33.]

PPUSCROLL sijaitsee osoitteessa $2005. Tämän rekisterin avulla ohjataan ruudun vieritystä. Tämä rekisteri on salparekisteri. Salparekisteriin kirjoittaessa joka toinen kirjoitus tekee eri asian. Ensimmäinen kirjoitus asettaa ruudun vierityksen X-arvon ja toinen Y-arvon. Ruudun rullauksen avulla voidaan toteuttaa liikkuvia taustoja peleihin. Tästä esimerkkinä on muun muassa ”Super Mario Bros.”. Tätä rekisteriä voidaan vain kirjoittaa. [33.]

PPUADDR sijaitsee osoitteessa $2006. Tämän rekisterin avulla prosessori pystyy käsitellä PPU:n osoiteväylässä olevia muisteja. Rekisteriä käytetään kertomaan mitä muistiositetta halutaan PPU:sta käsitellä. Tämä rekisteri on toinen salparekistereistä. Ensimmäinen kirjoitus tähän rekisteriin asettaa 16-bittisen osoitteen ylemmän tavun ja seuraava kirjoitus alemman tavun. Tätä rekisteriä voidaan vain kirjoittaa. [33.]

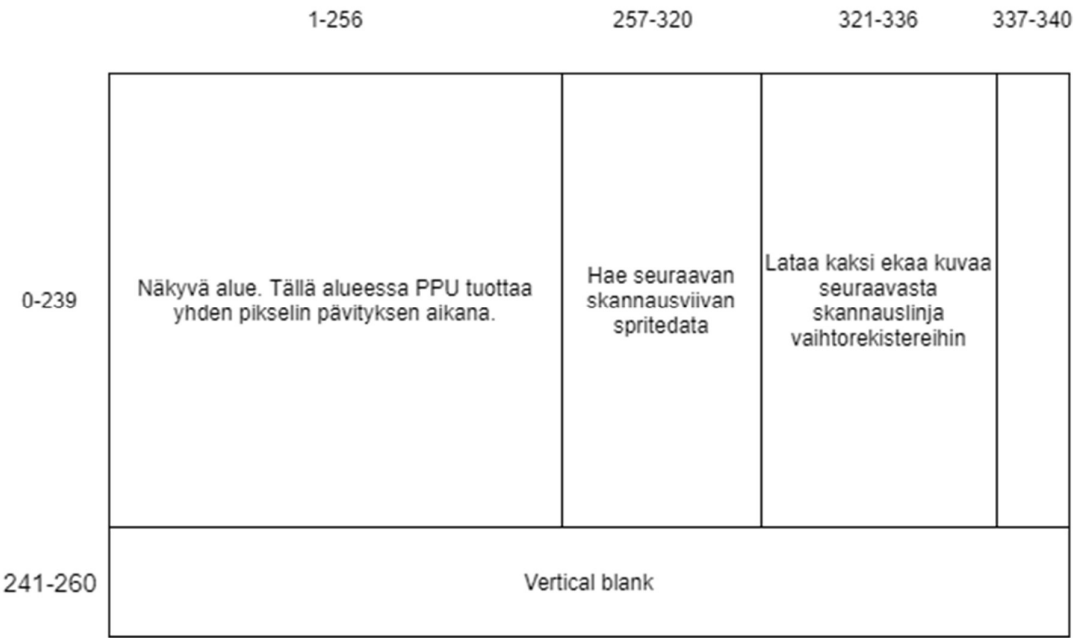
PPUDATA sijaitsee osoitteessa $2007. Tämän rekisterin avulla prosessori pystyy kirjoittamaan tai lukemaan PPU:n osoiteväylästä tietoa. Tiedon osoite määritellään PPUADDR-rekisterin avulla. Aina rekisterin lukemisen tai kirjoittamisen jälkeen kirjoittamisen osoitetta lisätään eteenpäin. Tämän avulla PPU:hun pystyy kirjoittamaan ja lukemaan tietoa monta kertaa peräkkäin muuttamatta välissä osoitetta rekisteriin PPUADDR. [33.]

Tähän on listattu vain olennaisimmat rekisterit ja niiden toiminta, että PPU:n toimintaa voidaan ymmärtää paremmin. Näiden rekisterien lisäksi on PPUCRTL, PPUMASK, OAMADDR ja OAMDATA rekisterit, jotka ohjaavat pienempiä osia PPU:sta [33]. Työssä on täysin toteutettu vain osa näistä rekistereistä.

8.6 PPU:n päivitys

Seuraavaksi työhön toteutettiin PPU-komponentin päivitys, mikä tuottaa yhden pikselin kuvaa kerrallaan. Piirtäminen toteutetaan yleiseen kuvakomponenttiin, jonka kuvadataa PPU muokkaa. PPU tuottaa 256x240 resoluutioisen kuvan. En täydellisesti selitä miten piirtäminen toimii koska se ei liity varsinaisesti alustan visualisointiin. Sen sijaan selitän suurimmat piirtämisen ominaisuudet, että PPU:n toimintaa voidaan ymmärtää. PPU:n piirtämisen eteneminen voidaan jakaa skannauslinjoihin ja sykleihin. Skannauslinjat vastaavat ylhäältä alas horisontaalista syvyyttä kuvassa. Skannauslinjoja on 262 kappaletta. Jokainen skannauslinja sisältää 341 sykliä, mikä vastaa vasemmalta oikealle vertikaalista syvyyttä. Jokaisessa PPU:n päivityksellä sykliä lisätään yhdellä, ja kun päästään skannauslinjan loppuun siirrytään seuraavalle viivalle. Päivittäessä PPU kulkee 341x262

kokoisen alueen, josta vain 256x240 on näkyvä osa. Lopuilla alueista on omat tarkoitukset, mitkä on näytetty kuvassa 31.



Kuva 31. PPU:n piirtämisen alueet sykli- ja skannauslinjanumeroiden kanssa.

8.6.1 Vertikaalinen tyhjä alue

”Vertical blank” on skannauslinjoissa 241–260 tyhjää signaalia generoiva alue. Tässä alueessa PPU ei muokkaa omaa tilaansa ja on tyhjäkäynnillä. Kun PPU etenee linjalle 241, se luo NMI-signaalin prosessorille. NMI-signaali eli ”No Mask Interrupt” puskee prosessorin ohjelmalaskurin ja tilaliput prosessorin pinoon ja asettaa ohjelmalaskurin osoitteessa $FFFA olevaan arvoon. [34.] Tähän signaaliin ei vaikuta prosessorin tilalippujen ”I”-lippu, joka päällä ollessaan estää muut keskeytykset prosessorissa (kuva 13). Kun PPU on ”Vertical Blank”-alueella, prosessori pystyy muokkaamaan ja lukemaan PPU:n muisteja. PPU:n muistiarvojen muokkaus tämän ulkopuolella aiheuttaa häiriöitä piirretyn kuvan tulokseen. [35.]

8.6.2 Taustojen piirtäminen

Piirrettäessä näkyvää osaa ruudusta PPU lataa kahdeksan päivityksen välein seuraavan piirrettävän taustakuvan datan kahteen 16-bittiseen vaihtorekisteriin. Kuvat valitaan PPU:n ruudun vieritysarvojen perusteella nimitaulusta. Vieritysarvot kertovat X- ja Y-koordinaatin mistä lukea nimitaulua. Jos ruudun vieritysarvot ovat nolla, ne vastaavat nimitaulussa olevia arvoja. Kaksi 16-bittistä vaihtorekisteriä sisältää yhdistettynä kahden kuvan datan. Nämä vaihtorekisterit puretaan samalla tavalla kuin kuviotaulu kuvassa 24. Näitä rekistereitä siirretään näkyvän alueen aikana jokaisella syklillä yhdellä korkeammalle ja yhdistettynä rekisterien korkeimmat bitit kertovat seuraavan pikselin arvon. Tämän takia syklillä 321–336 ladataan jo seuraavan skannauslinjan ensimmäiset taustakuvat, kuten kuvassa 31 näytetään. [34.]

8.6.3 Spritejen piirtäminen

Spritejä piirrettäessä OAM-muistista haetaan seuraavalle skannauslinjalle kuuluvat spritet, X-koordinaatit ja niiden attribuutit, joita voi olla kahdeksan kappaletta yhdellä linjalla. Spritejen haussa käytetään OAM-muistissa olevia Y-koordinaatteja. Spritejen hakeminen muistiin tapahtuu sykleillä 257–320. Spritejen haussa ne puretaan kuviotaulusta saataviin arvoihin. Jos spritet ovat kuvassa toistensa päällä, OAM-muistissa ensimmäisenä olevat spritet ovat suuremmassa prioriteetissa. Koska ruudun vieritys ei vaikuta spriten piirtämiseen, X-koordinaatista voidaan suoraan laskea millä syklillä se piirretään. [34.]

8.6.4 Taustojen ja spritejen yhdistäminen kuvaan.

Viimeisenä askeleena pikselin tuottamiseen on taustan ja spriten priorisoinnin selvittäminen. Ennen tätä vaihetta on selvitetty taustan ja spriten pikseliarvo. Jos molemmat arvoista ovat nollia, ruutuun saatu arvo on yleinen taustaväri palettimuistista. Kun vain toinen arvoista on erisuuri kuin nolla, käytetään sitä arvoa paletista haettavan värin saamiseksi. Jos molemmat arvoista ovat erisuuria kuin nolla, käytetään spriten attribuuttilippujen priorisointibittiä hyödyksi (kuva 29). Jos priorisointibitti on päällä, tausta piirretään. [34.] Tämän avulla PPU pystyy piirtämään spritejä myös taustojen taakse, mikä luo enemmän syvyyttä kuvaan. Tämä prosessi on näytetty kuvassa 32.

```
u8 finalPixel = 0, finalPalette = 0;
// determine if you render background or sprite
if (bgPixel == 0 && fgPixel == 0) {
	// Do nothing
} else if (bgPixel == 0 && fgPixel != 0) {
	// Draw foregroung
	finalPixel = fgPixel;
	finalPalette = fgPalette;
} else if (bgPixel != 0 && fgPixel == 0) {
	// Draw background
	finalPixel = bgPixel;
	finalPalette = bgPalette;
} else {
	// if both pixels are none zero check sprites priorisation flag
	if (fgPriority) {
		finalPixel = fgPixel;
		finalPalette = fgPalette;
	} else {
		finalPixel = bgPixel;
		finalPalette = bgPalette;
	}
}

Color color = ppu_palette_get_color(finalPixel, finalPalette);
// Copy the color to image
u32 imageDataIndex = (ppu.scanline * TEX_WIDTH + (ppu.cycle - 1)) * sizeof(Color);
memcpy(ppu.screen.data + imageDataIndex, &color, sizeof(Color));
```

Kuva 32. Kuvan lopullisen pikselin selvittäminen ja sen siirtäminen kuvaan.

9 Prosessorin ja PPU:n päivittäminen.

Työn prosessorin ja PPU:n toteutuksen jälkeen oli aika liittää komponentit ohjelman runkoon. Koska PPU ja prosessori ovat erillisiä fyysisiä komponentteja alkuperäisessä NES-alustassa, ne toimivat myös eri nopeuksilla. Prosessori toimii noin 1.79 MHz ja PPU noin 5.37 MHz nopeudella. Molemmat komponentit käyttävät samaa kellolähdettä niiden kellotukseen. Koska samaa kellolähdettä käytetään päivitykseen, komponentit päivittyvät aina samassa suhteessa ilman ajan ajautumista. Tämä suhde on NES-alustan NTSC-versioissa kolme PPU:n päivitystä yhteen prosessorin päivitykseen. [36.] PPU tuottaa yhden kuvan 1s/60s aikana, mikä vastaa myös ohjelman päivitysnopeutta. Ohjelman rungossa komponentteja päivitetään, kunnes PPU on tuottanut yhden kokonaisen kuvan. Ohjelman runko myös ohjaa PPU:n tuottaman NMI-signaalin tarvittaessa prosessorille ja suorittaa OAM muistin DMA:n. Kun PPU on piirtänyt yhden ruudun verran kuvaa, se piirretään käyttäjän ikkunalle käyttöliittymän elementtien kanssa. Tämä prosessi on kuvattu kuvassa 33.

```
// Update if target FPS is reached
if (delta > targetDelta) {
	lastTime = currentTime;
	do {
		ppu_clock();
		// Update PPU three times per CPU
		if (updateCounter % 3 == 0) {
			// If DMA isnt active, update CPU
			if (!ppu.oam.DMAactive) {
				cpu_clock();
			} else {
				ppu_dma_oam(updateCounter);
			}
		}
		if (ppu.NMIGenerated == true) {
			ppu.NMIGenerated = false;
			// redirect NMI from PPU
			cpu_no_mask_iterrupt();
		}
		updateCounter += 1;
	} while (ppu.frameComplete == 0 && debug == 1);
	ppu.frameComplete = 0;
}
```

Kuva 33. Ohjelman runko komponenttien päivitysten kanssa.

10 Ohjelman syötteet

NES-alustaa ohjataan sen mukana tulleessa kahdeksan painikkeisella ohjaimella. Alustaan on saanut myös muita ohjaimia, mutta näiden toiminta on ollut hyvin samankaltaista normaaliin ohjaimeen verrattuna. Prosessorin osoiteväylässä osoitteissa $4016 ja $4017 on kahden ohjaimen tilan lukemiseen tarkoitetut rekisterit. Kun näiden rekisterien tiloja lukee, ne palauttavat tavun korkeimman bitin ja siirtävät muita bittejä yhden korkeammalle. Jos rekisterejä kirjoitetaan, ne asetetaan ohjaimen alkuperäiseen asentoon. Näiden rekisterien toimintaa kutsutaan vaihtorekistereksi (eng. shift register). Jos suoritettava ohjelma haluaa lukea ”B” painikkeen tilan, sen pitää lukea kahdesti ohjaimen tila. Tämä vastaa kuvassa 34 olevia arvoja.

A	B	Select	Start	Ylös	Alas	Vasen	Oikea

Kuva 34. Ohjaimen tila 8-bittisenä lukuna

Emulaattorissa ohjainten tilan lukeminen on toteutettu SDL-kirjastolla. SDL-kirjaston avulla voidaan syötteisiin kirjoitettua koodia käyttää monella alustalla. Ohjainten tilaa päivitetään aina ikkunan piirtämisen jälkeen. SDL-kirjasto tarjoaa yksittäisten syötetapahtumien hakemisen tietylle ohjaimelle, minkä avulla toteutetaan molempien ohjainrekisterien päivitys. Päivittäessä ohjaimen tilaa tarkistetaan, onko syötetapahtuma ohjaimen painikkeen ylös tai alas painaminen sekä vastaako painetun painikkeen arvo ohjaimen painikkeita. SDL-kirjastossa suuntanäppäimet toimivat ohjaussauvan tavoin, minkä takia niiden tila tarkistetaan erikseen. Jos suuntapainikkeiden akselien arvo on erisuuri kuin nolla, uusi syöte rekisteröidään emulaattoriin.

11 Emulaattorin testaus

Iso aika työn tekemisestä meni emulointivirheiden etsimiseen ja korjaamiseen. Suurimmat virheet ovat kuitenkin helppo löytää emulaatiosta, koska ne tuottavat useasti näkyviä virheitä peliruudulle. Emulointiyhteisö on myös kehittänyt monia testaukseen tarkoitettuja kasetteja, jotka testaavat eri osia emulaattoreista. Emulointiyhteisön tekemiä testaukseen tarkoitettuja kasetteja on listattu wiki.nesdev.com sivustolla [37]. Tässä työssä testaukseen on käytetty ”kevtris” aliaksen tekemää nestest nimistä kasettia, jolla testataan 6502-prosessorin toimintaa. Tämän kasetin toiminta näkyy kuvassa 35. Tämä kasetti käyttää työhön toteutettua NROM-muistikartoittajaa. Kasetista pystyy ohjaimen avulla valitsemaan tiettyjä osia prosessorin toiminnasta mitä halutaan testata. Kasetti myös tukee prosessorin epävirallisten operaatio-ohjeiden testaamista, joita tässä työssä ei ole toteutettuna. Testaukseen käytettiin myös avoimen lähdekoodin pelejä, joita on esimerkiksi Nova The Squirrel ja Thwaite. Testaus on prosessina hyvin oleellinen osa etenkin emulaattorikehitystä. Testaamalla kasetteja, voidaan vähitellen korjata emulaation vikoja ja parantaa emulaatiotarkkuutta. Monia alustan toimintatapoja ei edes huomaa, ennen kuin ne tulevat oikeassa kasetissa vastaan.

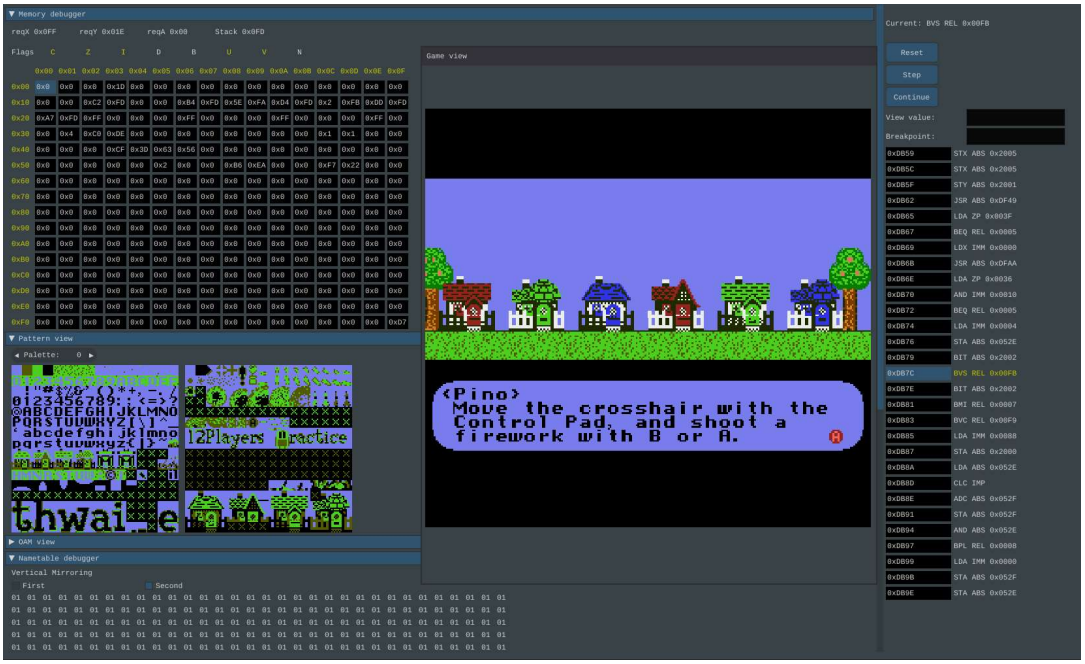
Kuva 35. Emuloitu Nestest-kasetti

12 Emulaattorin käyttöliittymä

Emulaattorin käyttöliittymä jaettu kolmeen osaan. Käyttöliittymäelementtien päällimmäisin elementti on liikuteltava ikkuna, johon kasetista tuotettu kuva piirretään. Ikkunassa oleva pelinäkymä skaalautuu, kun ikkunan kokoa kasvatetaan. Tämän avulla käyttäjä voi kontrolloida kuinka paljon näkymä peittää ikkunasta. Elementin toteutukseen käytettiin Nuklear-kirjaston kykyä piirtää käyttäjän luomia OpenGL tekstuureita. PPU:n tuottamaa pelitekstuuria päivitetään näytönohjaimelle aina ennen seuraavaa ikkunan piirtämistä. Tämä käyttöliittymäelementti näkyy kuvassa 37.

Toinen elementti on sovelluksen vasemmalle piirretty staattinen 6502-koodin visualisoija. Tämä komponentti vie aina 25% sovelluksen ikkunan koosta. Tämän komponentin tavoitteena on visualisoida ja hallita kasetin suoritusta. Elementti näkyy kuvassa 36 pelikuvan vasemmalla puolella.

Viimeinen ja suurin käyttöliittymäelementti on 75% sovelluksen ikkunan koosta vievä muistien visualisoija. Tässä elementissä on piilotettavia välilehtiä, joissa visualisoidaan eri alustan osia. Tätä osaa sovelluksesta käyttäjä pystyy vierittämään hiiren keskipainikkeella, mikä auttaa järjestämään välilehtiä käyttäjän tarpeiden mukaisesti. Visualisoitaviin osiin kuuluvat prosessorin osoiteväylän muistisivut, kuviotaulut, OAM ja nimitaulut.



Kuva 36. Emulaattoriympäristö kokonaisuutena. Kuvassa on avoimen lähdekoodin peli Thwaite [38].

13 Yhteenveto

Työssä toteutettiin Nintendo Entertainment System -emulaattori ja sen toiminnan visualisointi sovellus. Työllä pystyy vaatimuksien mukaisesti suorittamaan NES-kasetteja ja visualisoimaan ohjelman tilaa. Työ toteutettiin ohjelman siirtäminen mielessä ja suunnitellessa käytettiin vain ohjelmakirjastoja, joita voidaan käyttää mahdollisimman monella käyttöjärjestelmällä. Emulaattorista kuitenkin puuttuu monia alkuperäisessä alustassa olevia ominaisuuksia mitkä saatetaan myöhemmin lisätä. Näihin kuuluvat esimerkiksi äänentoisto ja monet kasetin muistinkartoittajat. Työhön toteutettujen ominaisuuksien määrää piti rajata työajan takia.

Työn toteutus tapahtui asteittain yksi komponentti kerrallaan. Aina kun komponentti oli tehty, sitä pyrittiin testaamaan ja todentamaan sen toimivuus. Työn aikana useasti virheet kuitenkin tulivat vastaan vasta paljon myöhemmin ja näiden löytäminen oli haastavaa. Työhön kehitetyt visualisointikomponentit auttoivat todentamaan missä komponentissa virhe voisi olla. Suurin apu työn kehityksessä on ollut avoimen lähdekoodin kasetit, joiden avulla pystyttiin todentamaan emulaation toiminta ja tarkkuus.

Työtä kehittäessä pystyin soveltamaan monia olennaisia ohjelmistokehitystaitoja, kuten grafiikkaohjelmointia, käyttöliittymäkehitystä, prosessorin toimintaa ja sovelluskehitystä. Emulaattorikehityksessä korostuu etenkin ohjelman testaus ja asteittainen kehitys. Lisäksi pääsin vahvistamaan ymmärrystäni emulaatiotekniikoista ja NES-alustasta.

Lähteet

(1) Wired – Nintendo Entertainment System Launches, saatavilla https://www.wired.com/2010/10/1018nintendo-nes-launches/ Viitattu 23/06/2020

(2) Wikipedia – C programming language. https://en.wikipedia.org/wiki/List_of_Nintendo_Entertainment_System_games Viitattu 15/11/2020

(3) Wikipedia – Homebrew video games. Saatavilla https://en.wikipedia.org/wiki/Homebrew_(video_games) Viitattu 15/08/2020

(4) NesDev – Mapper. https://wiki.nesdev.com/w/index.php/Mapper Viitattu 15/08/2020

(5) NesDev – NesDev verkkosivut http://nesdev.com/ Viitattu 15/11/2020

(6) Wiktionary – Emulaattori. https://fi.wiktionary.org/wiki/emulaattori Viitattu 15/08/2020

(7) iNES - NES/Famicon emulaattori, saatavilla: https://fms.komkon.org/iNES/ Viitattu 23/06/2020

(8) Gametechwiki – High/ Low level emulation, saatavilla: https://emulation.gametechwiki.com/index.php/High/Low_level_emulation Viitattu 23/06/2020

(9) Simple DirectMedia Layer – SDL Overview. Saatavilla https://wiki.libsdl.org/FrontPage Viitattu 25/06/2020

(10) OpenGL – OpenGL Overview. Saatavilla https://www.opengl.org/about/ Viitattu 25/06/2020.

(11) Wikipedia – C programming language. https://en.wikipedia.org/wiki/C_(programming_language) Viitattu 15/08/2020

(12) Nuklear – Nuklear Github repository. Saataville https://github.com/Immediate-Mode-UI/Nuklear Viitattu 30/06/2020.

(13) Geeksforgeeks - what happens when you don’t free memory after using malloc https://www.geeksforgeeks.org/what-happens-when-you-dont-free-memory-after-using-malloc/ Viitattu 15/08/2020

(14) Patric Diskin - Nintendo Entertainment System Documentation. Saatavilla http://nesdev.com/NESDoc.pdf Viitattu 04/07/2020.

(15) NesDev – Sample Ram Map. https://wiki.nesdev.com/w/index.php/Sample_RAM_map Viitattu 15/08/2020

(16) NesDev – CPU memory map. Saatavilla https://wiki.nesdev.com/w/index.php/CPU_memory_map Viitattu 04/07/2020.

(17) NesDev – PPU memory map. Saatavilla https://wiki.nesdev.com/w/index.php/PPU_memory_map Viitattu 04/07/2020.

(18) NesDev – NROM. Saatavilla https://wiki.nesdev.com/w/index.php/NROM Viitattu 06/08/2020.

(19) NesDev – Open bus behavior. Saatavilla https://wiki.nesdev.com/w/index.php/Open_bus_behavior Viitattu 10/07/2020.

(20) Rockwell – R6501Q Microprocessor. Saatavilla http://archive.6502.org/datasheets/rockwell_r6501q.pdf sivu 4. Viitattu 20/07/2020.

(21) CC65 – The 6502 C compiler. Saatavilla https://www.cc65.org Viitattu 22/07/2020.

(22) Rockwell – R650X and R651X Microprocessors. Saatavilla http://archive.6502.org/datasheets/rockwell_r650x_r651x.pdf sivu 4. Viitattu 20/07/2020.

(23) Wpmed62 – MedNES Github repository. Saatavilla https://github.com/wpmed92/MedNES Viitattu 30/07/2020.

(24) Masswerk – 6502 Instruction set. Saatavilla https://www.masswerk.at/6502/6502_instruction_set.html Viitattu 30/07/2020.

(25) Wilsonminesco - Self-modifying code on 65xx. https://wilsonminesco.com/SelfModCode/ Viitattu 15/11/2020

(26) NesDev – Savtool-swatches. Saatavilla https://wiki.nesdev.com/w/index.php/File:Savtool-swatches.png Viitattu 05/08/2020.

(27) NesDev – PPU palettes. Saatavilla https://wiki.nesdev.com/w/index.php/PPU_palettes Viitattu 05/08/2020.

(28) NesDev – PPU pattern tables. Saatavilla https://wiki.nesdev.com/w/index.php/PPU_pattern_tables Viitattu 06/08/2020.

(29) Khronos Group – Sampler Object. Saatavilla https://www.khronos.org/opengl/wiki/Sampler_Object Viitattu 06/08/2020.

(30) NesDev – PPU nametables https://wiki.nesdev.com/w/index.php/PPU_nametables Viitattu 15/08/2020

(31) NesDev – PPU OAM. Saatavilla https://wiki.nesdev.com/w/index.php/PPU_OAM

(32) Nove The Squirrel – Github repository. Saatavilla https://github.com/NovaSquirrel/NovatheSquirrel Viitattu 06/08/2020.

(33) NesDev – PPU reqisters. Saatavilla https://wiki.nesdev.com/w/index.php/PPU_registers Viitattu 13/08/2020.

(34) NesDev – PPU rendering. Saatavilla https://wiki.nesdev.com/w/index.php/PPU_rendering Viitattu 14/08/2020.

(35) Marat Fayzullin – Nintendo Entertainment System Architecture. Saatavilla https://fms.komkon.org/EMUL8/NES.html Viitattu 14/08/2020.

(36) NesDev – Clock rates. Saatavilla http://wiki.nesdev.com/w/index.php/Cycle_reference_chart#Clock_rates Viitattu 20/08/2020.

(37) NesDev – Emulator tests Saataville https://wiki.nesdev.com/w/index.php/Emulator_tests Viitattu 20/08/2020.

(38) Pinobatch – Twaite Github. Saatavilla https://github.com/pinobatch/thwaite-nes. Viitattu 20/08/2020.