Izgradnja emulatora za Game Boy Advance igraću konzolu

Puljizević, Orsat

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1441

IZGRADNJA EMULATORA ZA GAME BOY ADVANCE IGRAĆU KONZOLU

Orsat Puljizević

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1441

IZGRADNJA EMULATORA ZA GAME BOY ADVANCE IGRAĆU KONZOLU

Orsat Puljizević

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1441

Pristupnik:	**Orsat Puljizević (0036544195)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Klemo Vladimir

Zadatak: **Izgradnja emulatora za Game Boy Advance igraću konzolu**

Opis zadatka:

Proučiti i opisati rad Game Boy Advance (GBA) igraće konzole i ARM7tdmi procesora. Dodatno, proučiti i navesti osnovna načela emulacije igraćih konzola te pronaći i navesti postojeće emulatore za GBA konzolu. Izgraditi pokazni GBA emulator koji interpretira GBA ROM datoteke i simulira komponente konzole izvedene u sklopovlju. Emulator treba obuhvatiti emulaciju ARM7tmdi procesora, upravljanje dijelovima memorije poput RAM-a, VRAM-a i ROM-a, grafičko renderiranje, upravljanje zvukom, upravljanje korisničkim ulazom te pohranjivanje i učitavanje stanja emulatora. Opisati arhitekturu ostvarenog sustava, objasniti načine nadogradnje, opisati korištenje te pokazati rad na nekoliko primjera.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

Uvod

Game Boy Advance je prenosiva konzola koju je razvila kompanija Nintendo i koja se prvi put pojavila na tržištu 2001. godine. Ova konzola, poznata po svojoj bogatoj biblioteci igara i tehničkim mogućnostima, ostavila je trajan utjecaj na industriju videoigara. Cilj ovog rada je prikazati izradu Game Boy Advance (GBA) emulatora.

Prvo ćemo razmotriti osnove o emulaciji, navesti neke česte primjene emulatora i dati par primjera.

Nakon toga ćemo dati pregled GBA hardvera s posebnim fokusom na njegov ARM7TDMI procesor i prikazivanje grafike.

Konačno dat ćemo opširniji opis GBA emulatora sa svim potrebnim komponentama i navesti glavne izazove izrade ovog emulatora.

1. Emulacija

Emulator je softverski ili hardverski alat koji omogućuje jednom računalnom sustavu da oponaša ili imitira funkcionalnost drugog sustava. Drugim riječima, emulator omogućuje pokretanje softvera ili izvršavanje programa koji su izvorno dizajnirani za jedan operativni sustav ili platformu na drugom sustavu ili platformi.

Emulatori rade tako da čitaju strojne instrukcije i interpretiraju ih nalik originalnom sustavu. Preciznost je bitna za emulatore no većina emulatora nisu u potpunosti precizni jer ne trebaju biti ili ne mogu biti jer ne postoji potpuna dokumentacija sustava. Na primjer kod GBA neke dijelove memorije nije moguće čitati po četiri bajta, ali to nije bitno forsirati u emulatoru jer ni jedna igra neće to raditi i postoje registri za koje se ne zna njihova svrha, dakle niti jedan GBA emulator nije u potpunosti precizan. Trebaju biti dovoljno precizni da funkcionira softver koji želimo koristit.

Emulatori se mogu koristiti za testiranje softvera prije nego što se stavi na hardver. Na primjer kod razvoja Android aplikacija lakše ih je testirati na računalu. Također se koristi za testirane softvera kod ugradbenih računalnih sustava.

Još se koriste za rješavanje problema kompatibilnosti softvera između različitih sustava. Jedan od prvih emulatora je IBM 7094 emulator razvijen za IBM System/360 1964. Već od tada je bila potreba za korištenje starijeg softvera. Još jedan primjer su printeri koji podržavaju emulaciju HP LaserJet printera jer je mnogo softvera napisano za HP-ove printera.

Iako emulatori imaju mnoge praktične svrhe većina ljudi ih koriste za zabavu tako što emuliraju konzole za video igre ili operativne sustave starih računala poput DOS-a da igraju video igre za te sustave. Prvi takvi emulatori su nastali 1990-ih i emulirali su NES (Nintendo Entertainment System) i arkadne igre poput Asteroidsa i Pac-Mana.

Danas gotovo za svaku konzolu postoji emulator na Windowsu i Linuxu. Pogotovo su popularni emulatori Nintendovih i Sonyjevih konzola poput GBA o kojem ćemo više reći kasnije.

Dok je emuliranje konzola legalno, piratstvo video igara za te konzole nije. Većina ljudi koji koriste emulatore za konzole ne kupe CD ili kasetu igre nego je piratiraju ROM igre. Zbog toga Nintendo i Sony su pokušali zaustavit emuliranje njihovih konzola sudskim procesima, ali do sad nisu bili uspješni jer su emulatori nastali obrnutim inženjerstvom originalnog sustava.

2. Game Boy Advance

2.1. Osnovne informacije

GBA ima dva procesora: ARM7TDMI i Sharp SM83. Sharp SM83 je 8-bitni procesor koji se koristi samo za pokretanje starijih Game Boy i Game Boy Color igrica. Pošto je ovaj rad o emulaciji GBA Sharp SM83 više ne ćemo spominjati. O ARM7TDMI procesoru ćemo detaljnije pisati kasnije.

Konzola ima deset gumba od kojih su četiri gumbovi za smjer, a ostalih šest su: "A", "B", "L", "R", "Start" i "Select". Što se tiče izlaza ima ekran rezolucije 240x160 koji podržava 32768 boja i mono zvučnik.

Igrice se nalaze na ROM-u u kaseti. Maksimalna veličina ROM-a je 32 MiB. Kasete imaju i stalnu memoriju u obliku SRAM-a, Flash ROM-a ili EEPROM-a koja se koristi za čuvanje podataka. Kaste mogu imati i unutarnje uređaje poput sata kojima se pristupa na određenoj memorijskoj lokaciji.

Dva do četiri GBA se mogu spojiti posebnim kabelom što omogućava video igre s više igraća.

Slika 1: Kaseta za GBA [7]

Slika 2: Game Boy Advance [7]

2.2. ARM7TDMI

GBA-ov procesor je 16.78 MHz ARM7TDMI 32-bitni RISC procesor. Podržava dva skupa instrukcija: ARM i Thumb način rada. U ARM načinu rada koristi 32-bitni ARMv4 skup instrukcija, a u Thumb načinu rada koristi 16-bitni skup naredbi koji se mapira na ARMv4 skup instrukcija. GBA igrice su većinom u Thumb načinu rada jer je priključak do kasete samo 16 bita pa bi dohvat instrukcija bio sporiji u ARM načinu rada i više instrukcija stane u istu količinu memorije nego u ARM načinu rada.

Instrukcije se izvršavaju u protočnoj strukturi s tri koraka: dohvati, dekodiraj i izvrši. Rezultat toga je programsko brojilo je uvijek pokazuje na instrukciju dvije nakon trenutne instrukcije koja se izvršava.

Procesor koristi 15 registara opće namjene. Registar 15 služi kao programsko brojilo. Registri 13 i 14 se razlikuju od ostalih registara opće namjene. U registar 14 se

sprema povratna adresa nakon prekida i BL (branch and link) naredbe. Registar 13 se koristi kao pokazivač na vrh stoga.

Poseban je CPSR (current program status register) koji služi kao statusni register. U njemu se nalaze zastavice N, Z, C i V koje označavaju je li rezultat aritmetičke ili logičke operacije negativan ili nula i jeli se prenosi jedinica ili dogodio se preljev. U CPSR-u je još zapisan trenutni način rada, jesu li prekidi dopušteni i je li se trenutno izvršavaju Thumb ili ARM instrukcije.

ARM7TDMI ima sedam različitih načina rada, a to su: user, system, interrupt, fast interrupt, supervisor, undefined i abort.

User način rada je jedini ne privilegirani način rada.

System način koristi sve iste registre kao i user način rada, ali je za razliku od user načina rada privilegiran.

U Interrupt način rade se ulazi kad se dogodi prekid. Na ulasku u interrupt način rada programsko brojilo se stavi na 0x18 i onemoguće se prekidi.

U fast interrupt način rade se ulazi kad se dogodi "brzi" prekid, ali GBA nikad ne generira takve prekide.

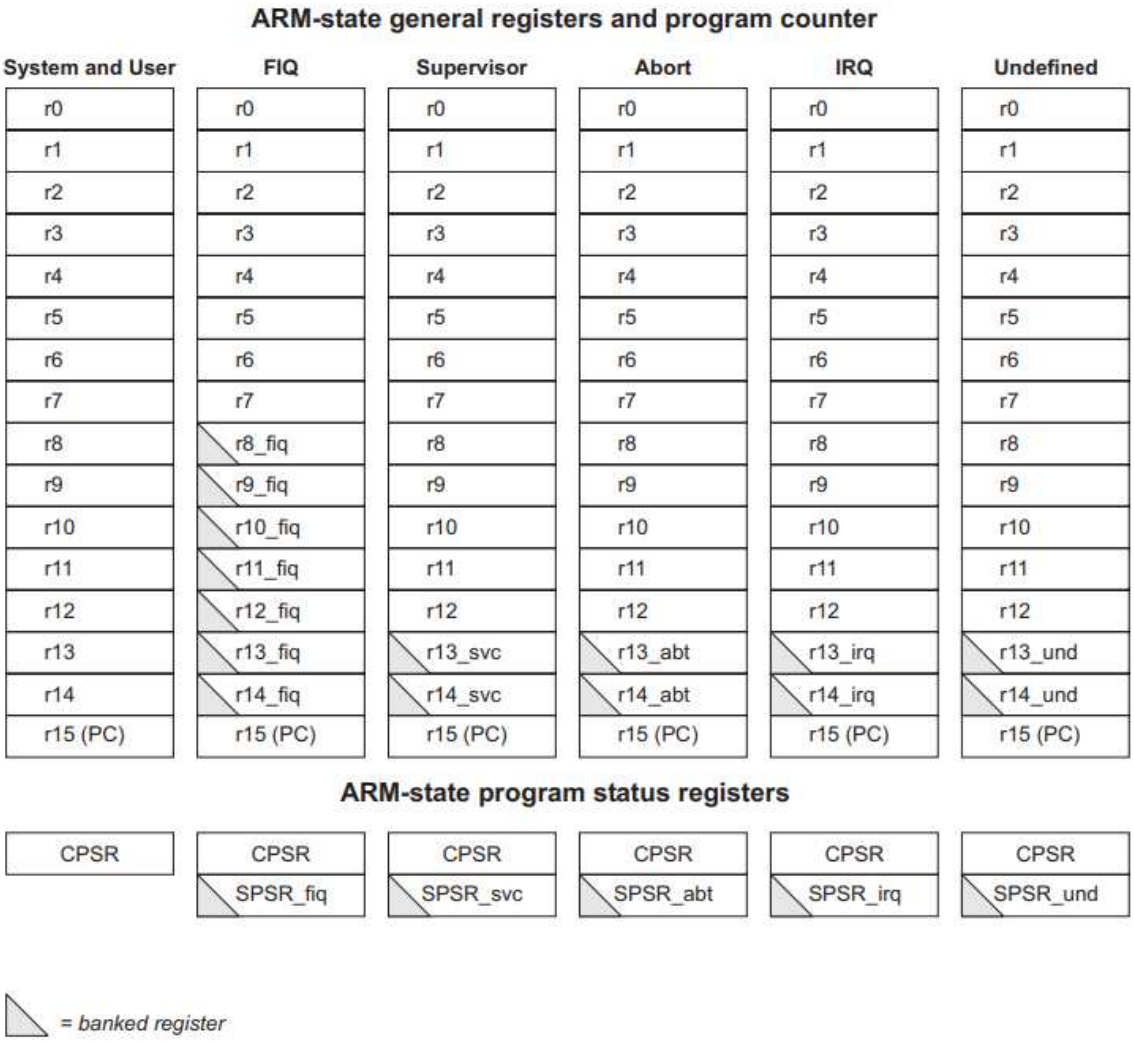
U supervisor načina rada se ulazi kad se dogodi softverski prekid naredbom SWI. Na ulazu programsko brojilo se stavi na 0x08.

U abort načina rada se su ulazi ako se ne uspije dohvatiti instrukcija. Na ulazu programsko brojilo se stavi na 0x0C.

U undefined načina rada se ulazi ako se izvrši ne definirana instrukcija. Na ulazu programsko brojilo se stavi na 0x04

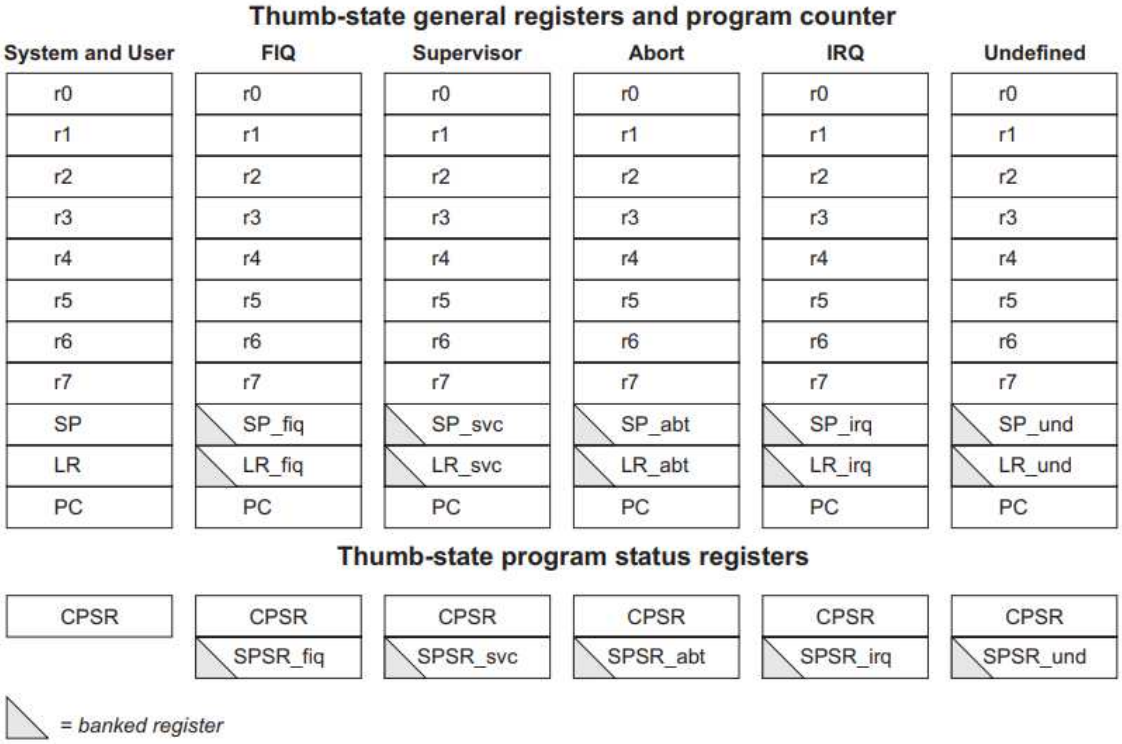
Na ulazu u neki od privilegiranih načina rada (osim system) CPSR se pohranjuje u SPSR (saved program status register) od tog načina rada. To služi da možemo nakon nekog prekida vratit program u prijašnje stanje.

Svi načini rada imaju svoje različite registre 13 (pokazivač na stog) i 14 (povratna adresa) osim user i system koji koriste iste i fast interrupt koji još ima i svoje 8 do 12 registre. Na slici 3 "banked" registri su oni koje se koriste u svojim načinima rada.



Slika 3: registri u ARM načinu rada [1]

S Thumb setom instrukcija nije moguće pristupiti registrima od 8 do 12 jer su instrukcije široke samo 16 bitova pa za označavanje registara se koriste 3 bita. Na slici 4 SP (stack pointer, pokazivač na stog), LR(link register, povratni registar) i PC (program counter, programsko brojilo) su zapravo registri 13, 14 i 15. U thumb načinu rada se tim registrima pristupa posebnim naredbama.



Slika 4: registri u Thumb načinu rada [1]

2.3. Memorija

Memorija GBA je podijeljena na devet sekcija: system ROM, EWRAM, IWRAM, IO RAM, palette RAM, VRAM, OAM, game pak ROM i cart RAM. Za svaku sekciju ćemo dati osnovne informacije i kratki opis.

System ROM započinje na adresi 0x00000000 i završava na adresi 0x00003FFF (16 KiB). Priključak je širok 32 bita. Nije moguće pisati na system ROM i čitanje je samo moguće s instrukcijama sa system ROM-a. U ovom djelu memorije se nalazi BIOS.

EWRAM započinje na adresi 0x02000000 i završava na adresi 0x0203FFFF (256 KiB). Memorija se ponavlja se do 0x02FFFFFF. Na primjer adrese 0x02000001 i 0x02040001 pokazuju na istu fizičku memorijsku lokaciju. Priključak je širok 16 bitova. Ova sekcija memorije se koristi kao RAM opće namjene.

IWRAM započinje na adresi 0x03000000 i završava na adresi 0x03007FFF (32 KiB) ponavljajući se sve do 0x03FFFFFF. Ovo je najbrži dio memorije i pošto je priključak širine 32 bita ovdje se često kopira ARM kod s game pak ROM-a (kaseta s igrom) kojem je priključak širine 16 bitova.

IO RAM započinje na adresi 0x04000000 i završava na adresi 0x04010000 ponavljajući samo word na 0x04000800 se sve do 0x04FFFFFF svakih 0x10000 bajtova. Priključak je širok 32 bita. Na ovu sekciju memorije su mapirani registri drugih uređaja na sabirnici poput DMA uređaja, tajmera, zvučnika itd.

Palette (hrv. paleta) RAM započinje na adresi 0x05000000 i završava na adresi 0x050003FF (1 KiB) ponavljajući se sve do 0x05FFFFFF. Priključak je širok 16 bitova. U ovoj sekciji memorije se pohranjuju 16 bitne RGB boje koje se koriste za grafičko procesiranje.

VRAM započinje na adresi 0x06000000 i završava na adresi 0x06017FFF (96 KiB). Bajtovi od 0x06010000 do 0x06017FFF se ponavljaju sve do 0x0601FFFF pa se onda tih 0x00020000 bajtova ponavlja sve do 0x06FFFFFF. Priključak je širok 16 bitova. U VRAM-u se pohranjuju informacije potrebne za grafičko prikazivanje.

OAM započinje na adresi 0x07000000 i završava na adresi 0x070003FF (1 KiB) ponavljajući se sve do 0x07FFFFFF. Priključak je širine 32 bita. U OAM-u se pohranjuju podaci o slikama, npr. lokacija podataka o pikselima i koordinati slike.

Game pak ROM započinje na adresi 0x08000000 i završava na adresi 0x09FFFFFF (32 MiB) ponavljajući se na 0x0A000000 i 0x0C000000. Priključak je širine 16 bita. Na ovoj sekciji memorije su podaci kasete igrice. Nakon što se izvrši BIOS izvršavanje naredbi se nastavi na 0x08000000.

Cart RAM započinje na adresi 0x0E000000 i maksimalna veličina mu je 64 KiB. Priključak je širine 8 bitova. Na ovoj lokaciji je SRAM, EEPROM ili flash ROM kasete koji služi za stalno pohranjivanje podataka o igrici.

2.4. Uređaji na sabirnici

Registri za upravljanje uređajima na sabirnici su mapirani na IO RAM sekciju memorije.

DMA (direct memory access) je uređaj koji kopira podatke s jednog djela memorije na drugi. GBA ima četiri DMA uređaja. Svaki od njih ima registre za izvornu adresu, odredišnu adresu i kontrolni registar koji odlučuje stvari poput uvjeta za pokretanje DMA prijenosa, hoće li se dogoditi prekid nakon prijenosa, količina podataka za prenijeti itd. DMA0 (prvi DMA) ima najveći prioritet na sabirnici, drugi slijede po redu. DMA1 i DMA2 se mogu koristiti za prijenos podataka o zvuku registrima zvučnika. DMA3 je jedini DMA koji može pristupiti game pak ROM-u.

GBA ima četiri brojača. Svaki ima registar koji sadrži trenutni broj, registar koji sadrži početnu vrijednost i kontrolni registar. Kontrolni registar određuje frekvenciju uvećavanja brojača i hoće li generirati prekid kad se dogodi preljev.

Zvučnik ima četiri analogna i dva digitalna zvučna kanala. Analogni kanali se koriste za generiranje zvučnih efekata poput šuma i specifičnih valova. Digitalni kanali se koriste za puštanje zvuka s game pak ROM-a. DMA1 i DMA2 kopiraju bajtove zvučnih podataka u registre zvučnih kanala. Stopa uzorkovanja se određuje s pomoću brojača 0 ili 1 (ovisno o konfiguraciji) dakle svaki put kad se brojač prelije dogodi se jedan DMA prijenos na zvučni registar.

Svih deset gumbi su spojeni na isti registar koji naravno služi za očitavanje pritisnutih gumba. Moguće je generirati prekid ako je određena kombinacija gumbi pritisnuta ili ako je barem jedan od određenih gumbi pritisnut.

Postoji još par specijalnih registara u IO RAM-u. Registar u kojeg kad se piše zaustavi procesor dok ne dođe do prekida i registri za upravljanje stanjem čekanja (eng. wait state) koji određuju koliko ciklusa trebamo čekati odgovor od memorije.

Ostali registri u IO RAM-u upravljaju grafikom i prekidima ili im se još nije otkrila svrha.

2.5. Grafika

Grafiku prikazuje uređaj na sabirnici koji se zove PPU (picture processing unit) koji je nalik jednostavnijem GPU-u. PPU osvježava piksele red po red s lijeva na desno. Svaki redak se osvježava 1004 ciklusa i taj vremenski period se naziva H-draw. Nakon H-draw slijedi H-blank koji traje 228 ciklusa u kojem se pikseli ne osvježavaju. Kad završi H-blank počne se osvježavati sljedeći redak piksela na isti način sve dok se ne dođe do zadnjega tj. 160. Dok se osvježavaju redci traje V-draw period te nakon što se osvježi 160. redak slijedi V-blank period koji traje 68 redaka tj. 83776 ciklusa. V-draw i V-blank zajedno traju 280896 ciklusa, a pošto je frekvencija procesora 16.78 MHz ekran se osvježi 59.74 puta u sekundi. V-blank i H-blank su bitni jer je tada poželjno mijenjati sadržaj registara i memorije koja utječe na grafiku. Mijenjanje grafičkih podataka tijekom H-draw i V-draw može uzrokovati ne predvidljive grafičke artefakte.

Palette RAM sadrži boje u RGB555 (5 bitova za svaku komponentu boje) zapisu. U VRAM-u su podaci o pozadinama (eng. background) i slikama (eng. sprite) koji indeksiraju boju u palette RAM-u i tako sačuvaju memoriju jer je dovoljan jedan bajt za indeksirati palette RAM, a boja je opisana s 2 bajta. Pošto jedan bajt može indeksirati 256 boja samo toliko ih možemo prikazati u isto vrijeme s pozadinama i slikama (osim bitmap pozadina, o tome uskoro). Palette RAM je podijeljen na dva dijela prvih 512 bajtova indeksiraju pozadine, a drugih 512 indeksiraju slike. Indeks 0 je uvijek prozirni piksel za pozadine i slike neovisno koja je boja zapisana na toj lokaciji. Boja na indeksu 0 prvog djela palette RAM-a je boja ekrana prije nego što se išta prikaže na njemu (eng. backdrop) dakle to je pozadina pozadinama i slikama (na engleskom se riječi razlikuju).

Pozadine (osin bitmap pozadina) i slike se sastavljaju od takozvanih tile-ova (hrv. pločica). Tile je dio teksture dimenzija 8x8 piksela. Informacije o tileovima se nalaze u VRAM-u. Jedan tile može zauzimati 64 ili 32 bajta memorije. Kada zauzima 64 svali bajt predstavlja 1 piksel tilea, a kada zauzima 32 svaki bajt predstavlja 2 piksela. Jedan bajt može predstavljati 2 piksela tako što pozadina ili slika izabere jednu od 16 paleta (podjeli palette RAM na 16 dijelova) pa 4 bita indeksiraju jednu od 16 boja.

GBA može prikazati tri vrste pozadina: tekst, rotacijske i bitmap pozadine.

Postoji 4 registara u IO RAM-u za informacije o pozadinama. Moguće je prikazati 4 tekst pozadine u isto vrijeme. U svakom registru je zapisan prioritet pozadine, lokacija tileova, dimenzija pozadine i veličina tilea (32 ili 64 bajta). Podatci o tekst pozadini u VRAM-u je lista indeksa tileova i za svaki tile se zapiše je li horizontalno i/ili vertikalno okrenuta i koju paletu koristi ako su tileove 32 bajta (1 od 16).

Rotacijske pozadine su slične tekst pozadinama jer rade na isti način osim sto se rotacijske pozadine mogu rotirati i skalirati. Moguće je prikazati dvije rotacijske pozadine u isto vrijeme i mogu koristiti samo tileove od 64 bajta.

Bitmap pozadine ne koriste tileove nego imaju zapisane boje u VRAM-u za svaki piksel ili zapisane indekse za svaki piksel. Može se prikazati samo jedna bitmap pozadina u isto vrijeme.

PPU ima šest načina rada. U prvom su dostupne 4 tekst pozadine, u drugom su dvije tekst i jedna rotacijska pozadina, a u trećem dvije rotacijske pozadine. Četvrti način rada prikazuje bitmap pozadinu kojoj su boje zapisane u VRAM-u, a Peti i šesti način rada prikazuju bitmap pozadinu indeksirajući palette RAM.

PPU prikazuje i slike čiji se atributi nalaze u OAM sekciji memorije. Ti atributi su koordinati slike, njene dimenzije, skaliranje, rotacija itd. Može se maksimalno prikazati 128 slika u isto vrijeme.

Specijalni efekti se mogu primjenjivati na pozadine i slike. Ti efekti uključuju mozaik efekt, posvjetljenje, potamnjenje i djelomičnu prozirnost.

GBA ima mogućnost stvaranja prozora. Moguća su dva pravokutna prozora kojima su dimenzije zapisane u posebnim registrima i prozor proizvoljnog oblika oblikovanim slikama u OAM-u koji imaju poseban parametar za definiranje prozora. Za svaki prozor i prostor izvan prozora može se odrediti što će se prikazivati a što ne u posebnim registrima.

2.6. BIOS

GBA BIOS započinje na 0x00000000 lokaciji. Prije nego što pokrene igru učini par zadataka. Ti zadaci su inicijalizacija stogova za interrupt, supervisor i user način rada,

postavljanje cijele memorije na 0, dovođenje napona na zvučnik i prikazivanje uvodne animacije.

U BIOS-u su 42 funkcije koje se pozivaju softverskim prekidom (naredba SWI). Nakon softverskog prekida ulazi se u supervisor način rada i programsko brojilo se stavlja na 0x08 nakon čega, oviseći o argumentu SWI naredbe, se izvršava jedna od 42 funkcije i vraća se nazad. Te 42 funkcije obuhvaćaju dijeljenje, korjenovanje, arkus tangens, čekaj V-blank itd.

2.7. Prekidi

Postoji 14 hardverskih prekida na GBA-u. Svaki od njih zasebno se može onemogućiti ili omogućiti. V-blank i H-blank prekidi se dogode na ulazu u V-blank i H-blank. V-count prekid se dogodi kada se počne osvježavati odabrani redak ekrana. Sva četiri brojača mogu izazvati vlastiti prekid kad se preliju i sva četiri DMA mogu izazvati prekid kad završe s radom. Kombinacije pritisaka gumba mogu izazvati prekid kao što je prije navedeno.

Postoje još i prekid kad se stavi ili izvadi kaseta i prekid kad se GBA spoji kabelom s drugim GBA, ali ovi prekidi nisu biti za svrhu ovog rada.

3. Emulacija Game Boy Advance-a

3.1. Emulacija ARM7TDMI procesora

Najvažniji dio za emulirati je procesor. Dalje možemo na procesor nadodavati druge dijelove.

Primjer glavne petlje emulatora (u pseudo kodu nalik C-u):

```
očitaj_tipkovnicu();
for (int i = 0; i < 280896; i++) {
      uvećaj_Hblank();
      uvećaj_brojače();
      if (čekanje) čekanje--;
      if (DMA_aktivan()) {
            DMA_prijenosi();
      }
      else if (čekanje == 0 && !cpu_stop) {
            if (ARM_način_rada) {
                  instrukcija = čitaj(R[15], 4);
                  R[15] += 4;
            }
            else if (Thumb_način_rada) {
                  instrukcija = čitaj(R[15], 2);
                  R[15] += 2;
            }
            izvrši( instrukcija );
      }
      if (prekidi_omogućeni && dogodio_se_prekid) {
            SPSR_irq = CPSR;
            CPSR = 0x00000092;
            zamjeni_registre();
            R[14] = R[15];
            R[15] = 0x18;
            cpu_stop = 0;
```

```
        }
    }
```

Kôd 1.1 – Glavna petlja emulatora

Očitavamo ulaze tipkovnice svakih 280896 ciklusa procesora jer je čitanje tipkovnice vremenski skupa funkcija i jer je dovoljno očitavati svako jedno osvježenje ekrana. Svaki ciklus uvećavamo aktivne brojače i H-blank brojač. Kada H-blank brojač dođe do 1004 trebamo prikazati taj red piksela, ali više o tome kasnije. Varijabla čekanje služi da emuliramo različite vremenske razlike između naredbi npr. instrukcija B (grananje, eng. branch) isprazni protočnu strukturu jer mijenja programsko brojilo pa instrukcija traje 3 ciklusa dok instrukcije poput ADD (zbrajanje, eng. addition) traju samo 1 ciklus procesora (ako ne spremaju rezultat u programsko brojilo). Provjeravamo za aktivne DMA jer oni imaju prioritet nad procesorom. Dok je DMA aktivan procesor ne izvršava naredbe. Procesor je moguće zaustaviti sve dok ne dobije prekid; tome služi "cpu_stop" varijabla. Ako je procesor u ARM načinu rada čitamo 4 bajta za jednu instrukciju i povećavamo programsko brojilo za 4, a u Thumb načinu rada čitamo 2 bajta i povećavamo programsko brojilo za 2. Nakon što izvršimo instrukciju provjeravamo je li se dogodio neki prekid. Ako se dogodio prekid spremamo CPSR u SPSR interrupt načina rada i stavljamo procesor u interrupt način rada pa zato mijenjamo registre 13 i 14 za registre 13 i 14 interrupt načina rada. U registar s povratnom adresom stavljamo adresu sljedeće instrukcije i mijenjamo programsko brojilo na 0x18. Sve ovo se ponavlja dok emulator radi.

U "izvrši" funkciji se dekodira i izvršava instrukcija. Određeni bitovi u instrukciji određuju o kakvoj se instrukciji radi, a ostali su onda argumenti te instrukcije. Ne isplati se objašnjavati izvedba svake ARM instrukcije jer nisu pretjerano komplicirane i ima ih puno pa ćemo navest samo jedan primjer s instrukcijom B (grananje, eng. branch).

```
if ((instrukcija & 0x0e000000) == 0x0a000000) {
      Sint32 nn = (instruckcija & 0x00ffffff);
      if (nn & 0x00800000) nn = nn | 0xff000000;
      if (instrukcija & 0x01000000) {
            R[14] = R[15];
      }
      R[15] = R[15] + 4 + (nn << 2);
      čekanje += 2;
```

```
}
```

Kôd 1.2 – Izvedba B/BL instrukcije

Prvo se provjerava je li ta instrukcija B. Argument naredbi je broj dvojnog komplementa u nižih 24 bita instrukcije. Ako je postavljen određeni bit radi se zapravo o BL (grananje s povratkom, eng. branch and link) instrukciji i u registar 14 se stavlja povratna adresa. Programsko brojilo se mijenja i dogodio se skok. Zbraja se broj 4 zbog protočne strukture procesora. Na čekanje se dodaje 2 jer se mijenjanjem programskog brojila ispraznila protočna struktura. Prije je spomenuto da B instrukcija traje 3 ciklusa, tu se dodaje 2 jer se 1 već pribrojio na dohvatu instrukcije.

3.2. Upravljane memorijom

Na pokretanju emulatora potrebno je inicijalizirati sve sekcije memorije. U system ROM kopiramo BIOS iz datoteke. To radimo i za game pak ROM (datoteka s igricom) i cart RAM (sačuvani podaci).

U primjeru s kodom 1.1 koristimo "čitaj" funkciju za dohvat instrukcije i u objašnjenju primjera 1.2 spominjemo da se u "čitaj" funkciji povećava varijabla čekaj. Funkcije za čitanje i pisanje nam trebaju iz više razloga. Prvi razlog je činjenica da je procesor little-endian pa kada čitamo više bajtova moramo im obrnuti redoslijed. Drugi razlog je veliki broj sekcija memorije pa s pomoću funkcija za čitanje i pisanje odabiremo točan blok memorije za operacije. Treći razlog je više adresa pokazuju na istu fizičku memorijsku lokaciju (adrese koje se ponavljaju), a to možemo riješiti modulo operatorom. Četvrti razlog je razlika u širini priključaka različitih sekcija memorije i registri za upravljanje stanja čekanja pa oviseći kojoj sekciji pristupamo povećavamo varijablu "čekanje" za različitu količinu npr. dohvat ARM naredbe (32 bita) s game pak ROM-a (širina 16 bitova) traje 2 ciklusa. Peti razlog su specijalni registri poput registra koji sadrži informacije o zahtjevima za prekid koji se briše tako da se u njega pišu jedinice. Neke dijelove memorije nije moguće čitati i u neke nije moguće pisati. Sve navedene problem rješavamo funkcijama "čitaj" i "piši". Nakon toga izvedba naredbi koje čitaju i pišu (LDR, STR, LDM, STM) je značajno jednostavnija. Sama izvedba funkcija za čitanje i pisanje

nije kompliciranа jer samo mora paziti na little-endian svojstvo, odrediti kojoj memorijskoj lokaciji se pristupa, koliko ciklusa pristup traje i za specifične registre popuniti uvjete poput maskiranja pročitanog ili upisanog registra.

3.3. Uređaji na sabirnici

Svaki DMA mora imati svoje varijable za pohranu izvorne adrese, odredišne adrese i broj preostalih prijenosa. Kada se DMA uključi registri koji sadrže izvornu i odredišnu adresu i broj prijenosa se kopiraju na unutarnje registre DMA-a koje procesor ne može vidjeti. Ti unutrašnji registri se onda mijenjaju dok DMA radi. Za čitanje i pisanje koristimo prijašnje opisane funkcije.

Za brojače moramo imati tri unutarnja registara tj. varijable. Prvi je djelitelj frekvencije i služi za mijenjanje frekvencije. Brojačima je moguće zadati frekvenciju manju od procesorove pa moramo oduzimati od djelitelja frekvencije dok ne postane 0 te povećamo brojač za jedan i resetiramo vrijednost djelitelja frekvencije. Drugi varijabla sadrži trenutnu vrijednost brojača, a treća vrijednost na koju se brojač resetira povodom preljeva. Kada čitamo s registra brojača u IO RAM-u dobivamo trenutnu vrijednost brojača, ali kada pišemo zapisujemo vrijednost resetiranja.

Kada se brojači 0 i/ili 1 preliju i ako su konfigurirani da uzorkuju zvučne podatke pokrećemo DMA1 i/ili DMA2 koji kopiraju podatke u posebne zvučne registre. U funkciju za pisanje stavljamo uvjet ako piše na te lokacije da zvučnik pušta zadanu frekvenciju.

Za analogne kanale trebamo očitati konfiguracijske registre i izračunati frekvenciju koju trebamo pustiti. Kod emulatora koji pušta zvuk treba biti na drugoj dretvi i raditi paralelno s glavnom petljom jer ako je na istoj dretvi sve drugo bi se zaustavilo dok zvučnik pušta zvuk pa sve bi se usporilo.

3.4. Grafika

U kodu 1.1 spominjemo funkciju "uvećaj_Hblank()". Pošto jednom retku treba 1004 ciklusa da se prikaže mi kod za prikazivanje grafike nadovezujemo na tu funkciju.

```
uvećaj_Hblank() {
      h_brojač++;
      if (h_brojač == 1004) {
            h_blank = istina;
            if (!v_blank) pripremi_redak()
      }
      if (h_brojač == 1232) {
            h_brojač = 0;
            h_blank = laž;
            v_brojač++;
            if (v_brojač == v_prekid) izazovi_prekid();
            if (v_brojač == 160) {
                  osvježi_ekran();
                  v_blank = istina;
            }
            if (v_brojač == 228) {
                  v_blank = laž;
                  v_brojač = 0;
            }
      }
}
```

Kôd 1.3 – Pokretanje grafičkog prikazivanja

Svaki ciklus procesora ova funkcija se pozove i povećavamo `h_brojač`. Nakon 1004 ciklusa trebamo redak pripremiti za prikazivanje ako smo u periodu V-draw, a ne prikazati jer prikazujemo sve retke odjednom jer nema potrebe 160 puta po frame-u slati zahtjev grafičkoj kartici. Kada `h_brojač` dođe do 1232 završi H-blank period i poveća se brojač redaka `v_brojač`. Prije smo spomenuli da je moguće izazvati prekid kada se počne osvježavati specifični redak pa u ovoj funkciji provjeravamo je li došlo do prekida i postavlja se zastavica za tu vrstu prekida u posebnom registru. Na 160. retku osvježavamo ekran tako što prikažemo sve retke koje smo pripremili. Pošto ekran ima samo 160 redaka

od `v_brojač` vrijednosti 160 do 228 traje V-blank period i redci se ne osvježavaju. U ovom periodu igrica stigne pripremiti sljedeću stvar za prikazati.

U kodu 1.3 koristimo funkciju `pripremi_redak()` koja koristeći palette RAM i VRAM dobije informacije of izgledu retka. Prvo treba provjeriti sve pozadine i ako se pojavljuju u traženom retku treba pročitati samo one podatke koji su u tom retku. Za svaki piksel u retku su zapisani svi pikseli pozadina i slika koji se preklapaju, njihovi prioriteti i kojem prozoru pripadaju pa za svaki piksel odaberemo onaj koji se prikazuje i primijenimo specijalne efekte koji mogu biti mozaik, posvjetljenje, potamnjenje i djelomična prozirnost. U slučaju djelomične prozirnosti boja se miješa s pikselom sljedećeg po prioritetu.

3.5. BIOS

BIOS nije potrebno emulirati jer je kod BIOS-a dostupan. Većina emulatora ipak emulira BIOS jer nije lagano dostupan na legalan način pošto Nintendo nikad nije objavio svoj BIOS niti je dopustio piratstvo BIOS-a. Postoji besplatna i otvorena alternativa Nintendovom BIOS-u, ali nije skroz funkcionalna ([2]).

Za emulirati BIOS potrebno je napisati sve 42 BIOS funkcije softverskih prekid (SWI naredba). Za dodatnu preciznost treba znati koliko ciklusa traje svaka od tih funkcija, ali to ne predstavlja problem jer su to ljudi koji su dokumentirali hardver istražili.

Na pokretanju emulatora bez BIOS-a treba ga ručno inicijalizirati.

```
init_bez_bios() {
      resetiraj_memoriju()
      R_sys_usr[13] = 0x03007F00;
      R13_irq = 0x03007FA0;
      R13_svc = 0x03007FE0;
      R[15] = 0x08000000;
}
```

Kôd 1.4 – Inicijalizacija bez BIOS-a

Prvo postavimo cijelu memoriju na 0. Inicijaliziramo pokazivač na vrh stoga za system, user, interrupt i supervisor načine rada. Konačno stavljamo programsko brojilo na 0x08000000 jer tu počinju instrukcije igrice (game pak ROM).

3.6. Kaseta

Kasete mogu imati unutarnje uređaje poput sata kao što smo prije spomenuli. Naravno sve moguće uređaje treba emulirati. Mogući uređaji su: sat, svjetlosni senzor, senzor nagiba i motor za vibraciju. Informacije o korištenim uređajima su u zaglavlju ROM-a uvijek na istoj lokaciji. Uređajima se upravlja preko GPIO priključka u kaseti igrice. Registri za upravljanjem GPIO-a su uvijek na lokaciji 0x080000C4. Emulator može očitati koji uređaji se koriste i emulirati potrebne uređaje i GPIO priključak.

Za čuvanje podataka nakon što se GBA izgasi koristi se stalna memorija u obliku EEPROM-a, SRAM-a i flash ROM-a. Pri gašenju emulatora sačuvamo cart RAM u datoteku koju sljedeći put učitavamo u cart RAM pri inicijalizaciji emulatora. Sva tri oblika stalne memorije kasete se koriste na drugačiji način pa trebamo emulirati cart RAM na drugačiji način ovisno o korištenoj memoriji. Vrsta memorije nije zapisana u zaglavlju ROM-a kao što je slučaj kod uređaja kasete već sami moramo otkriti koju vrstu memorije emulirati. Vrsta memorije se otkriva s pomoću takozvanih identifikacijskih string-ova koji mogu biti: "EEPROM_V", "SRAM_V", "FLASH_V", "FLASH512_V" ili "FLASH1M_V" (nakon "V" ide verzija biblioteke, ali to nije bitno za emulator). Nintendove biblioteke i alati automatski stavljaju te string-ove negdje u ROM ne uvijek na istu lokaciju pa treba pri inicijalizaciji emulatora skenirati cijeli ROM i otkriti o kojoj vrsti stalne memorije se radi. Glavni problem je EEPROM koji može biti jedan od dvije moguće veličine, ali ima samo jedan identifikacijski string i flash ROM koji se ponaša drugačije ovisno o proizvođaču. Najbolje rješenje detekcije tipa memorije je imati datoteku s listom identifikacijskih kodova svake igre (nalazi se u zaglavlju) i memorije koju koriste.

SRAM memorija se najlakše izvodi jer se može pristupiti na normalni način kao i svaka druga lokacija u memoriji za razliku od EEPROM i flash ROM-a s kojima se komunicira preko posebnih registara u cart RAM ili game pak ROM sekciji memorije.

Jedine restrikcije na SRAM su 8 bitno pisanje i čitanje to jest samo se može pristupiti naredbama LDRB i STRB (učitaj bajt i pohrani bajt).

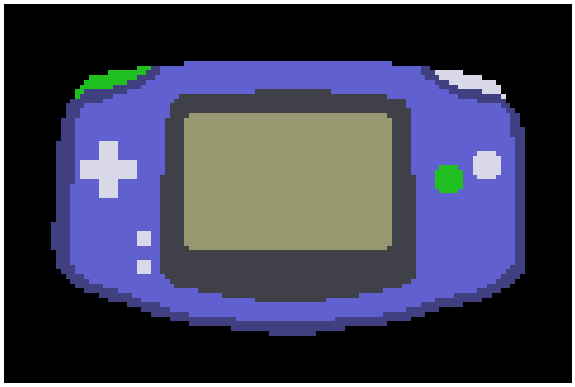
Još jedan način sačuvanja podataka je pohraniti cijelu memoriju i relevantne varijable emulatora u datoteku i učitati je sljedeći put. Većina emulatora može to učiniti i brže je nego emuliranje stalne memorije.

3.7. Testiranje

Važan dio razvoja bilo kojeg softvera je redovito testiranje. Postoji mnogo napisanih testnih ROM-ova za GBA koji se koriste za testiranje preciznosti emulatora ([8]). Testni ROM-ovi imaju više testnih slučajeva i svi naravno prolaze na pravom hardveru. Rade tako da nakon svakog testnog slučaja provjere korištene procesorove registre tako što ih usporede s točnim vrijednostima. Ako neki registar nije jednak točnoj vrijednosti to znači da emulator ima grešku na tom testnom slučaju i izvršavanje daljnjih slučajeva se zaustavi te se na ekran prikaže koji slučaj je pao. Za grafičko prikazivanje se koristi bitmap pozadina koja je najjednostavniji način rada PPU-a tako da ne mora raditi cijelo grafičko prikazivanje da testiramo emulator (slika 5). Ako prikazivanje grafike uopće ne radi testni ROM također spremi broj palog slučaja u specifični registar.

Slika 5: Testni slučaj je pao

Za testiranje je dobro i koristit jednostavne ROM-ove koji demonstriraju jednu funkcionalnost GBA. Tonc tutorial za razvoj GBA igara ([5]) je vrlo koristan jer sadrži mnogo takvih primjera.



Slika 6: Tonc primjer

Nas slici 6 je ROM s Tonc tutoriala koji prikazuje sliku GBA s pritisnutim gumbima obojenim u zeleno. S tim ROM-om bi mogli testirati i grafiku i rad gumbiju. Nakon što većina testnih slučajeva i Tonc primjera rade možemo biti zadovoljni s radom emulatora.

Zaključak

Ovim radom smo se bolje upoznali s emulacijom i GBA konzolom. Objasnili smo metode iza emulacije glavnih komponenti GBA konzole poput CPU-a, grafičkog prikaza, zvuka, memorije i ostalih uređaja na sabirnici.

Sa saznanjima o GBA i emulaciji iz ovog rada moguće je napraviti poprilično precizan emulator. Glavni nedostatci su ne mogućnost igranja s više igrača i nije moguće postići veliku brzinu emulacije.

Za emuliranje kabela koji spaja GBA-ove potrebno bi bilo omogućiti emulatoru da se poveže s drugim preko interneta pa onda sinkronizirati te emulatore. Ovo se moguća buduća nadogradnja emulatora.

Nije moguće postići veliku brzinu emulacije jer u sklopu rada nismo spominjali dinamičku rekompilaciju kojom se kod igre optimizira dok se izvršava. Moderna računala su mnogo brža nego GBA pa nije potrebno implementirati dinamičko rekompajliranje za postizanje dobrih brzina, ali ako zbog nekog razloga trebamo ubrzati izvođenje igrice na velike brzine nećemo moći.

Postoji još sitnica koje nismo spominjali u radu koje se mogu implementirati da se poboljša preciznost emulacije, ali većina takvih detalja utječe samo na igre koje su loše napisane i izazivaju greške npr. čitanje ne valjanih memorijskih adresa kod mnogo emulatora vraća nuli, ali u realnosti vraća zadnje očitane podatke.

Cilj ovog rada je bio prikazati izradu emulatora i to smo i postigli.

Literatura

[1] ARM-ova službena dokumentacija ARM7TDMI procesora, https://documentation-service.arm.com/static/5f4786a179ff4c392c0ff819, Pristupljeno: 2024.

[2] GBA BIOS od Cult of GBA, https://github.com/Cult-of-GBA/BIOS, Pristupljeno: 2024.

[3] Tom Happ, Unofficial documentation for the CowBite GBA emulator, 4. 8. 2002., Pristupljeno: 2024.

[4] Martin Korth, GBATEK, https://mgba-emu.github.io/gbatek/, 2014., Pristupljeno: 2024.

[5] Jasper Vijn, Tonc, https://www.coranac.com/tonc/text/intro.htm, 24. 3. 2013., Pristupljeno: 2024.

[6] Emulator, https://en.wikipedia.org/wiki/Emulator, Pristupljeno: 2024.

[7] Game Boy Advance, https://en.wikipedia.org/wiki/Game_Boy_Advance, Pristupljeno: 2024.

[8] Game Boy Test Roms, https://github.com/c-sp/game-boy-test-roms, Pristupljeno: 2024.

Sažetak

Izgradnja emulatora za Game Boy Advance igraću konzolu

Orsat Puljizević

Ovaj rad istražuje proces izrade emulatora za Game Boy Advance (GBA), prijenosnu konzolu kompanije Nintendo. Rad započinje s kratkim uvodom u emulaciju pa pregledom arhitekture GBA konzole, uključujući njezin procesor (ARM7TDMI), grafički podsustav, memoriju i ostale komponente. Glavni dio rada je usmjeren na tehničke izazove u razvoju emulatora, poput emuliranja CPU-a, grafičkog procesiranja i upravljanja memorijom.

Ključne riječi: emulacija, Game Boy Advance, GBA, ARM, GBA emulator

Summary

Emulator development for the Game Boy Advance game console

Orsat Puljizević

This paper investigates the process of creating an emulator for the Game Boy Advance (GBA), a portable cnsole from Nintendo. The paper begins with a brief introduction to emulation, followed by an overview of the architecture of the GBA console, including its processor (ARM7TDMI), graphics subsystem, memory and other components. The main part of the work is focused on technical challenges in emulator development, such as CPU emulation, graphics processing and memory management.

Key words: emulation, Game Boy Advance, GBA, ARM, GBA emulator

Skraćenice

GBA	*Game Boy Advance*	asinkroni način prijenosa
ROM	r*ead only memory*	memorija iz koje se podatci mogu
	samo čitati	
RAM	*random access memory*	memorija s nasumičnim pristupom
SRAM	*static RAM*	statični RAM
EEPROM	e*lectrically erasable programmable*	električno izbrisiva programibilna
	read-Only Memory	ispisna memorija
RISC	r*educed Instruction Set Computer*	procesor sa smanjenim skupom
		naredaba
CPSR	*current program status register*	statusni registar trenutnog
	programa	
SPSR	*saved program status register*	pohranjeni statusni registar
	programa	
EWRAM	*external work RAM*	vanjski radni RAM
IWRAM	*internal work RAM*	unutarnji radni RAM
IO RAM	*input output RAM*	ulazno izlazni RAM
VRAM	*video RAM*	video RAM
OAM	*object attribute memory*	memorija objektnih atributa
BIOS	*basic* input/*output system*	osnovni ulazno/izlazni sustav
DMA	*direct memory access*	izravni pristup memoriji
GPU	*graphics processing unit*	grafiči procesor
PPU	*picture processing unit*	slikovni procesor