Kako napraviti (GameBoy) emulator?

(iliti, odbrana maturskog rada v2.0)

Petar Veličković

Matematička gimnazija, Nedelja informatike

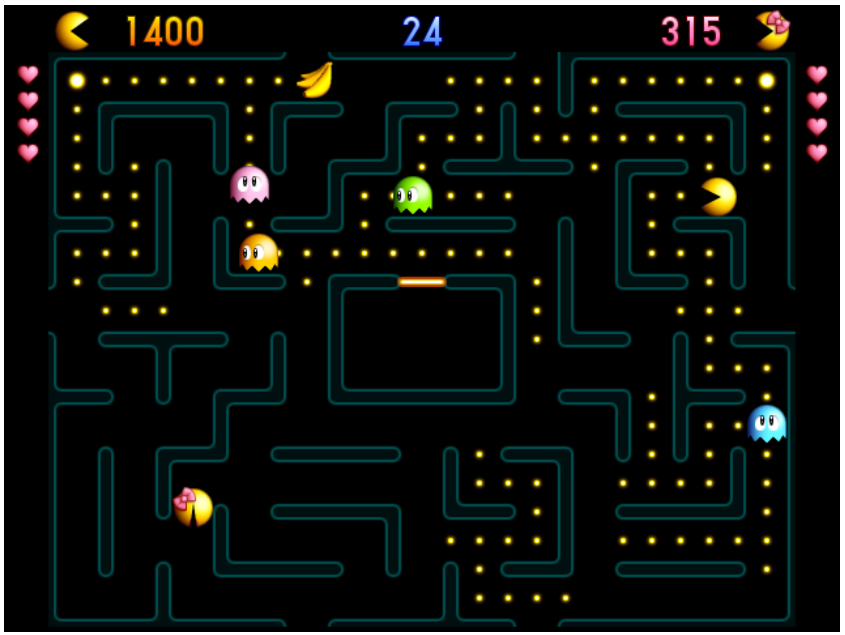
31. mart 2015.

Emulacija

- **Emulacija** (*emulation*) podrazumeva *imitiranje jednog računarskog sistema na drugom.*
- Uporediti sa **simulacijom**, koja podrazumeva samo oponašanje neke specifične funkcionalnosti (igre).
- Emulatori mogu biti **hardverski** i **softverski**; u ovom predavanju razmatraćemo samo softverske emulatore.

Emulator vs. simulator

PSX **emulator**



Pac-man **simulator**

Emulator izvršava originalni ROM, dok je simulator potpuno nova igrica.

Potrebna znanja

Da bismo napravili emulator, potrebno je:

1. Generalno iskustvo u programiranju i odabranom programskom jeziku;
2. Poznavanje osnovne teorije računarskog dizajna i arhitekture (znanje assemblera je poželjno);
3. Obilje informacija o sistemu koji nas interesuje (dobijenih od strane proizvođača ili obrnutim inženjeringom);
4. Mnogo stprljenja. :-)

U redovnom programu MG-a se mogu naučiti prve dve stavke, a zahvaljujući internetu stavka 3 uglavnom nije problem—ostaje samo četvrta, ali... *"the theoreticians have proven that this is unsolvable, but there's three of us, and we're smart..."* – Arthur Norman.

Uvod u predavanje

- Moj maturski rad (šk. godina 2011/12) dokumentuje kako implementirati osnovne elemente GameBoy emulatora. Možete ga preuzeti sa:

  `http://mg.edu.rs/images/stories/dokumenta/maturski/izrada%20emulatora%20za%20gameboy%20konzolu%20-%20petar%20velikovi.pdf`

- Korišćen C# jezik; svi isečci koda će takođe biti u tom jeziku.

- Za svaku razmatranu komponentu sistema napravićemo generalnu kao i GameBoy-specifičnu diskusiju.

GameBoy

- Priručna (*handheld*) igračka konzola razvijena od strane japanske kompanije Nintendo, 1989. godine.

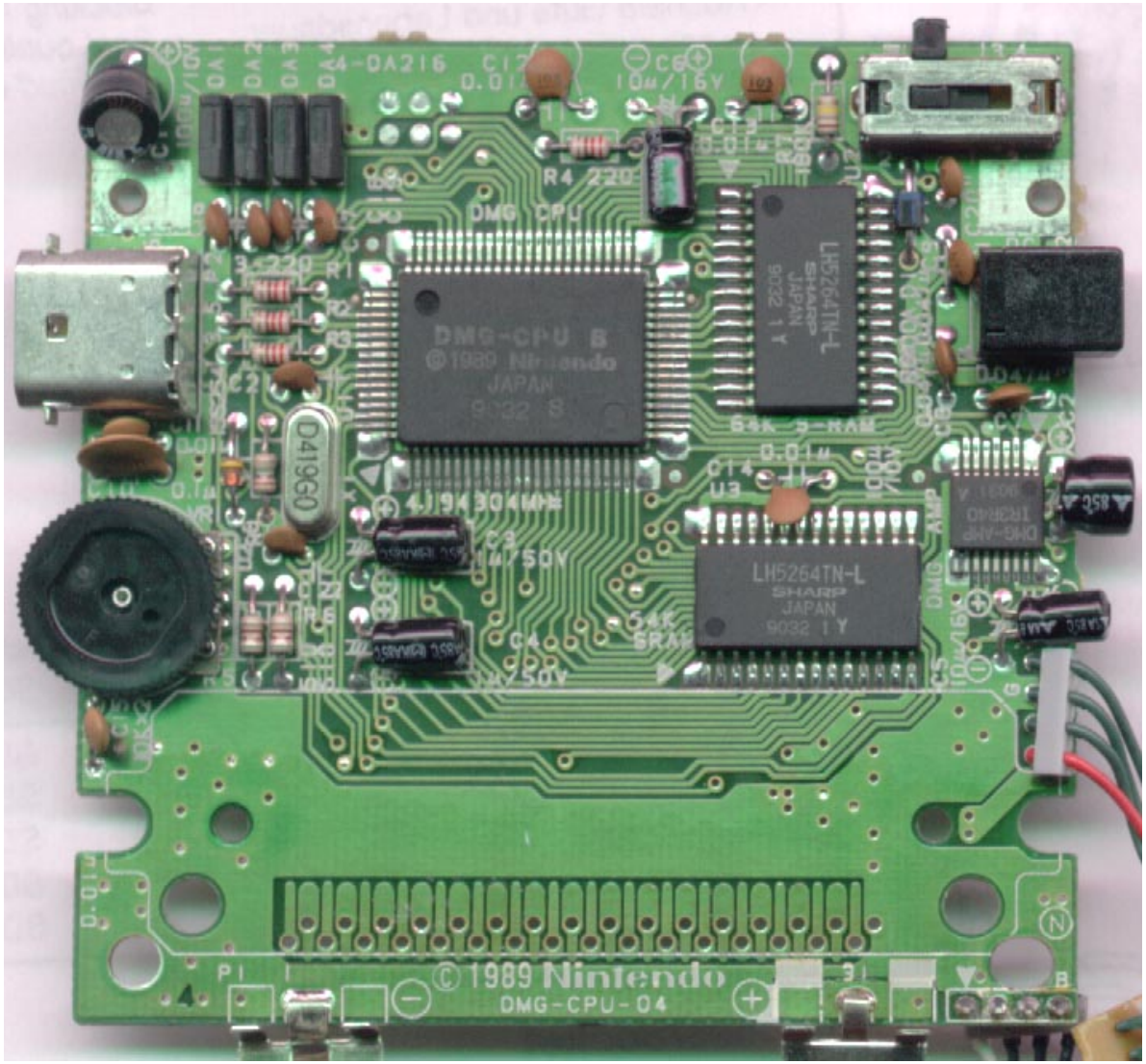
- Prva šire zapamćena *handheld* konzola, uprkos tehnološki superiornijoj konkurenciji:
  - Zajedno sa naslednikom (Game Boy Color) prodato **preko 119 miliona primeraka** (ubedljivo najprodavanija konzola ovog tipa sve do pojave Nintendo DS-a).
  - Samo kertridž od igre Tetris prodat u 35 miliona primeraka!

Spoljašnji izgled konzole

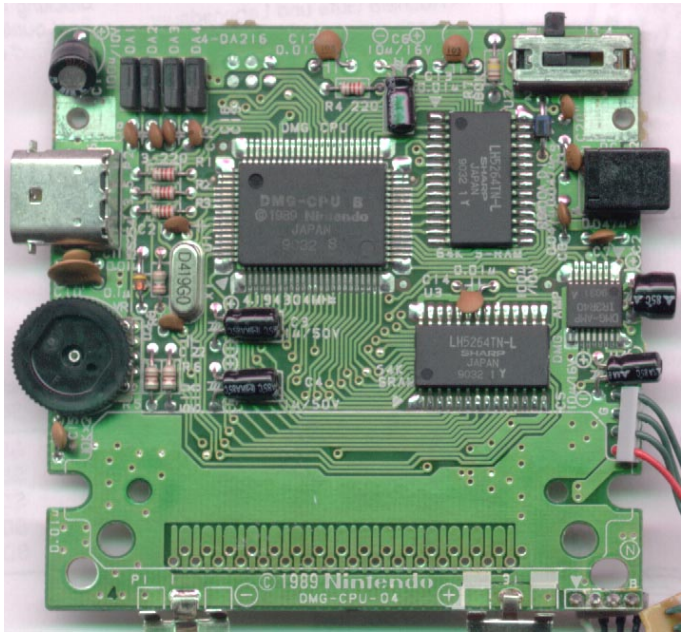
Spoljašnji izgled konzole

- Prednja strana: D-Pad, A/B/SELECT/START, LCD (160x144), zvučnik;
- Gornja strana: slot za kertridž, prekidač za paljenje;
- Desna strana: podešavač zvuka, konektor za link kabl;
- Leva strana: podešavač kontrasta;
- Donja strana: priključak za slušalice;
- Zadnja strana: četiri AA baterije.

Unutrašnji izgled konzole



Unutrašnji izgled konzole



- Procesor: 8-bitni modifikovani Sharp LR35902, frekvencije 4.19 MHz;
- Memorija: 16 KB statičkog RAM-a (8 KB radna + 8 KB grafička);
- ROM: 256 bajtova BIOS-a, kertridži 256 Kb – 8 Mb;
- Paleta: monohromatska (4 nijanse sive);
- Zvuk: četiri kanala sa stereo izlazom;
- Napajanje: jednosmerna struja, 6 V, 150 mA.

Modularni pristup

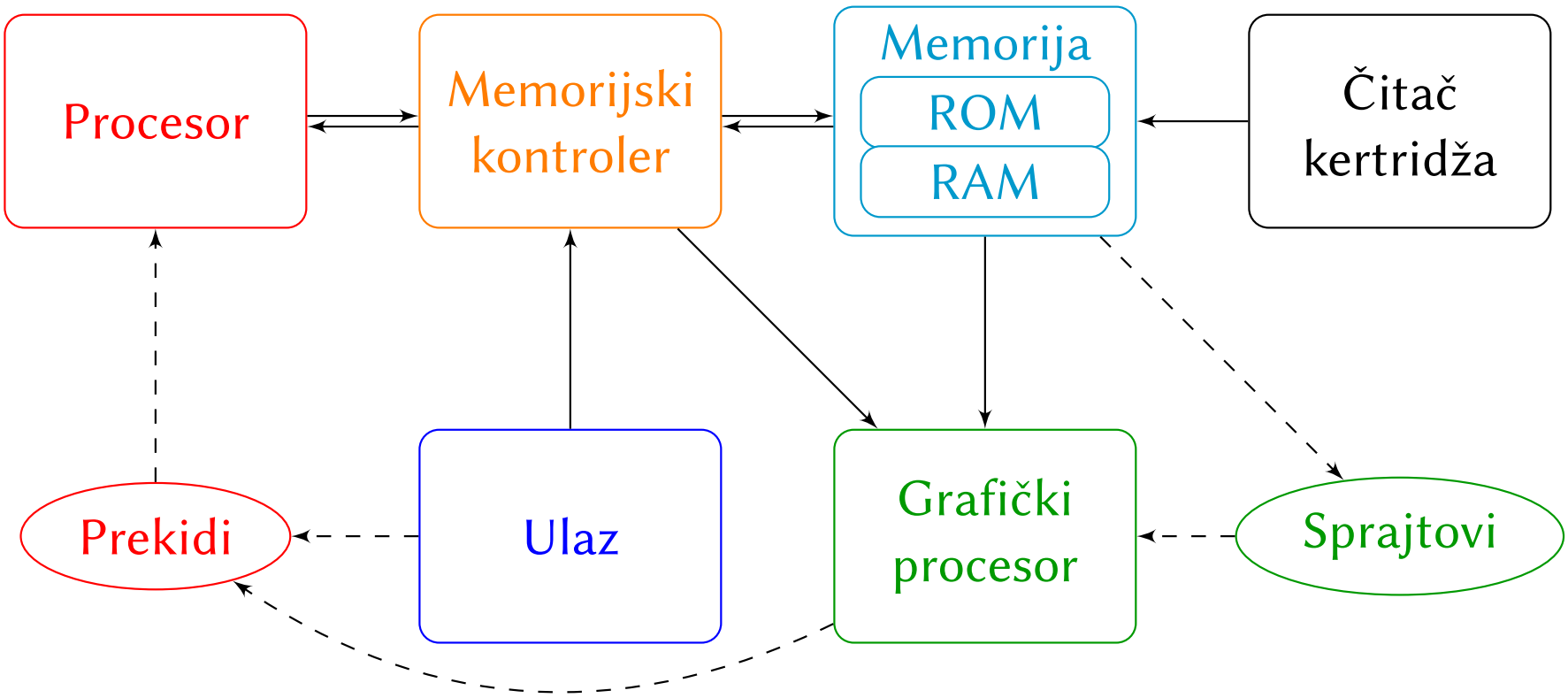
- Koristićemo *modularni* pristup – emulator će da se sastoji iz gomile **odvojenih** implementacija svih njegovih podsistema ($\sim$ modula)!

- Zašto modularni pristup?
  - Debagovanje mnogo lakše, jer je efekat grešaka lokalizovan;
  - Moguće je ponovo iskoristiti neki modul u drugom projektu – hardver je *standardizovan*.
  - Magistrale se implementiraju u vidu metoda i promenljivih jednog modula koje su vidljive drugom ($\sim$ objektno-orijentisani model: **modul == klasa**).

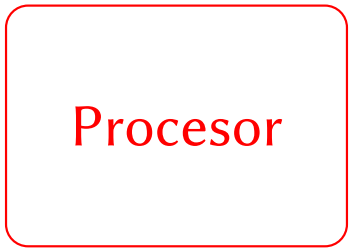
Potrebne komponente

- Za osnovni GameBoy softverski emulator nam je potrebna implementacija sledećih podsistema:
  - Procesor;
  - Memorija;
  - Grafički procesor.
- Korisno implementirati i neke dodatne podsisteme:
  - Čitač kertridža;
  - Ulaz;
  - Sprajtovi;
  - Prekidi;
- Neki od izostavljenih podsistema:
  - Zvuk;
  - Tajmeri;
  - Memorijsko bankiranje;
  - ...

Skica emulatora



Procesor



- Uglavnom najzahtevniji deo emulatora (zbog čega programeri često pribagavaju korišćenju "prepakovanih" implementacija procesora).
- Dve osnovne metode:
  - **Interpreter:** prolaz kroz program i direktno simuliranje svake instrukcije koristeći lokalni procesor;
  - **Rekompilacija:** (statički ili dinamički) direktno prevođenje programa u mašinski kod lokalnog procesora.

Interpreterski emulator

- Interpreteri uglavnom simuliraju gorenavedenu (*fetch-decode-execute*) petlju, kojom se rukovodi većina procesora današnjice:
  - **IF** *Instruction Fetch* – dopremi tekuću instrukciju procesoru;
  - **DC** *Decode* – odredi o kom tipu instrukcije je reč;
  - **EX** *Execute* – izvrši tekuću instrukciju.
- Ovako možemo emulirati samo sisteme sa procesorima **nekoliko redova veličine** sporijih od našeg.

Instruction Fetch

IF

Uzmi instrukciju

- *Gde* se nalazi program???
- Kod GameBoy-a, i program i podaci se skladište u memoriji sistema ($\sim$ fon Nojmanova arhitektura).
- $\Longrightarrow$ Potrebne su nam metode za čitanje i pisanje iz memorije.

Memorijski interfejs

- Pošto je GameBoy-ev procesor 8-bitni, sva čitanja i pisanja iz memorije, kao i izračunavanja, se obavljaju bajt po bajt. Stoga su nam potrebne sledeće funkcije:

```
1 public int ReadByte(int address);
2 public void WriteByte(int address, int value);
```

- Za sada je za procesor najbitnije da ove metode postoje – njihovim konkretnim implementacijama ćemo se baviti kasnije.

Memorijski interfejs, *cont'd*

- Česta radnja je takođe čitanje/pisanje 16-bitnih vrednosti (dva susedna bajta odjednom); ovo se lako implementira koristeći prethodne metode:

```
1  public int ReadWord(int address)
2  {
3     int low = ReadByte(address);
4     int high = ReadByte(address + 1);
5     return (high << 8) | low;
6  }
7  public void WriteWord(int address, int value)
8  {
9     WriteByte(address, value & 0xFF);
10    WriteByte(address + 1, value >> 8);
11 }
```

- Zašto prvo low pa high?

Kratko skretanje – *endianness*

- Ukoliko imamo mašinu koja podržava rad sa tipovima koji zauzimaju više od jednog bajta, postavlja se pitanje: *kojim redosledom poređati te bajtove u memoriji*?
  - **Big-endian:** prvo više značajni bajtovi
    `0x1234DEAD` ⟶

12	34	DE	AD
  - **Little-endian:** prvo manje značajni bajtovi
    `0x1234DEAD` ⟶

AD	DE	34	12
- GameBoy koristi procesor koji je little-endian, stoga u metodama `ReadWord` i `WriteWord` prvo pristupamo nižem pa višem bajtu.
- Problematične situacije nastaju pri komunikaciji između sistema: internet je **big-endian**, većina modernih procesorskih arhitektura **little-endian**...

Decode

DC

Dešifruj instrukciju

- Nakon što smo učitali tekući bajt programa, potrebno je na neki način odrediti koju instrukciju on predstavlja (i samim tim da li moramo čitati dodatne parametre iz memorije).
- Ovo zahteva poznavanje **arhitekture** GameBoy-evog procesora (koja je jako slična Z80 arhitekturi).

GameBoy arhitektura

- Pošto smo učitali jedan bajt programa kao instrukciju, to nam daje $2^8 = 256$ različitih instrukcija.
- Potrebno je implementirati jednostavan `switch-case` blok, kojim prepoznajemo svaku instrukciju odvojeno u zavisnosti od vrednosti ovog bajta.
- Nekoliko začkoljica...

GameBoy arhitektura, *cont'd*

- Prikaz prve 32 instrukcije u zavisnosti od učitanog bajta:

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
NOP	LD BC,nn	LD (BC),A	INC BC	INC B	DEC B	LD B,n	RLC A
0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
LD (nn),SP	ADD HL,BC	LD A,(BC)	DEC BC	INC C	DEC C	LD C,n	RRC A
0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17
STOP	LD DE,nn	LD (DE),A	INC DE	INC D	DEC D	LD D,n	RL A
0x18	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F
JR n	ADD HL,DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,n	RR A

Začkoljice (*Gotchas*)

- Neki brojevi nemaju definisanu instrukciju – tretiraćemo ih kao `NOP` instrukcije ($X = X$).
- Postoje **dve** tabele koje mapiraju bajtove u odgovarajuće instrukcije (stoga ima ukupno ~ 512 instrukcija); specijalan bajt `0xCB` signalizira da će sledeći bajt predstavljati instrukciju iz druge tabele ("CB-instrukciju").
- Ovo znači da ćemo unutar '`case 0xCB:`' imati novo čitanje bajta i zatim `switch-case` po tom bajtu.

Execute

EX

Izvrši instrukciju

- Svaka instrukcija ima potencijalni efekat na procesorovo interno stanje ($\sim$ **registre**) kao i na stanje memorije.
- Takođe, svaka instrukcija zahteva određen (fiksni) broj procesorskih ciklusa (**vrlo važno za kasnije!**)
- Efekte na memoriju smo već prešli (`WriteByte`, `WriteWord`).
- Da bismo mogli da implementiramo svaku instrukciju zasebno, potrebno je implementirati procesorovo interno stanje.

Procesorsko stanje

Brojevi koji se nalaze u sledećim 16-bitnim registrima potpuno određuju trenutno procesorovo stanje:

`PC` *Program Counter* — adresa sledeće instrukcije u memoriji;

`SP` *Stack Pointer* — memorijska adresa vrha steka;

`AF` *Accumulator & Flags* — akumulator i flegovi;

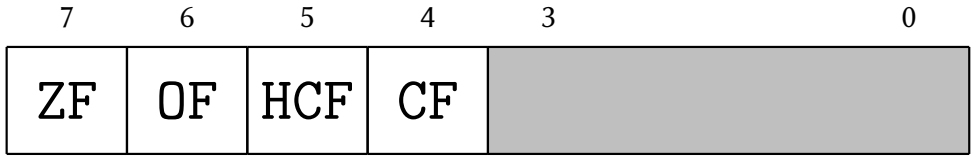
`BC/DE` Registri opšte namene;

`HL` Memorijska adresa opšte namene.

- Možemo pristupati i 8-bitnim delovima nekih registara: (`A`, `B`, `C`, `D`, `E`, `H`, `L`).
- Takođe ćemo čuvati i broj proteklih procesorskih ciklusa (*ticks*) od početka rada.

Flegovi

- Flegovi su logičke (`true`/`false`) promenljive koje pomažu procesoru da odredi efekte prethodno izvršene instrukcije.
- Kod GameBoy-a, registar sa flegovima izgleda ovako:



- Gorenavedeni flegovi su:
  - `ZF` *Zero Flag* — označen ukoliko je rezultat operacije nula;
  - `OF` *Operation Flag* — označen ukoliko je operacija bila oduzimanje;
  - `HCF` *Half-Carry Flag* — označen ukoliko je pri računanju rezultata donjih 4 bita premašilo 15 (pri sabiranju) ili 0 (pri oduzimanju);
  - `CF` *Carry Flag* — označen ukoliko je rezultat operacije veći od 255 (pri sabiranju) ili manji od 0 (pri oduzimanju);

Procesorsko stanje, *cont'd*

```
1 public class Z80
2 {
3   // registri
4   private int A, B, C, D, E, H, L, PC, SP;
5   // flegovi
6   private bool ZF, OF, HCF, CF;
7   // protekli broj ciklusa
8   public int ticks;
9 }
```

Implementacije instrukcija: aritmetika

```
1  private void Add(int b) // dodaj broj b akumulatoru (A)
2  {
3    // Halfcarry Flag: da li je donja polovina prekoracila 15?
4    HCF = (A & 0x0F) + (b & 0x0F) > 0x0F;
5    A += b;
6    CF = A > 0xFF; // Carry Flag: da li je A prekoracio 255?
7    A &= 0xFF; // A mora ostati 8-bitan
8    OF = false; // Operation Flag: operacija nije oduzimanje
9    ZF = A == 0; // Zero Flag; da li je A postao 0?
10   ticks += 4;
11 }
12
13 private void Compare(int b) // uporedi b sa akumulatorom (A)
14 {
15   // Compare se vrsi oduzimanjem broja b od A
16   HCF = (A & 0x0F) < (b & 0x0F);
17   CF = b > A;
18   OF = true;
19   ZF = A == b;
20   ticks += 4;
21 }
```

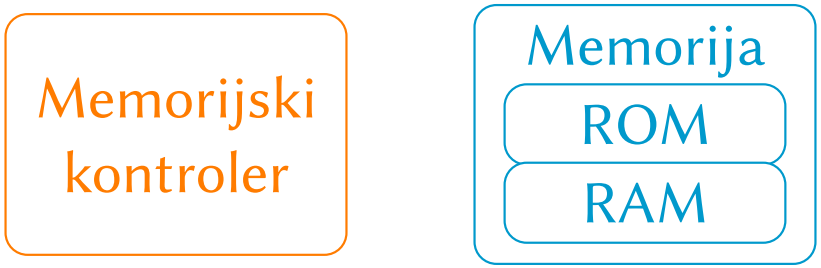
Implementacije instrukcija: memorija

```
1  private void Push(int rh, int rl) // ubaciti rh||rl na stek
2  {
3    WriteByte(--SP, rh);
4    WriteByte(--SP, rl);
5    ticks += 16;
6  }
7
8  // izbaciti rh||rl sa vrha steka
9  private void Pop(ref int rh, ref int rl)
10 {
11   rl = ReadByte(SP++);
12   rh = ReadByte(SP++);
13   ticks += 12;
14 }
15
16 // ucitaj vrednost na trenutnoj lokaciji PC-a u r
17 private void LoadImmediate(ref int r)
18 {
19   r = ReadByte(PC++);
20   ticks += 8;
21 }
```

Iteracija procesorskog ciklusa

```
1  public void Step()
2  {
3     int inst = ReadByte(PC++); // ucitaj instrukciju
4     switch (inst)
5     {
6        case 0x00: // NOP
7           NoOperation(); break;
8        case 0x01: // LD BC, (nn)
9           LoadImmediate(ref B, ref C); break;
10       case 0x02: // LD (BC), A
11          WriteByte((B << 8) | C, A); break;
12       . . .
13       case 0xCB: // CB-instrukcije
14          switch(ReadByte(PC++)) // ucitati jos jedan bajt
15          {
16             case 0x00: // RLC B
17                RotateLeft(ref B); break;
18             . . .
19          } break;
20       . . .
21    }
22 }
```

Memorija



- Neophodne su korektne implementacije metoda `ReadByte` i `WriteByte`: potrebno je emulirati **memorijski kontroler**, tj. memorijsku upravljačku jedinicu (*MMU*).

Naivna implementacija

- Pošto PC ima 16 bitova, memorijski sistem GameBoy-a podržava $2^{16} = 65536$ adresa.

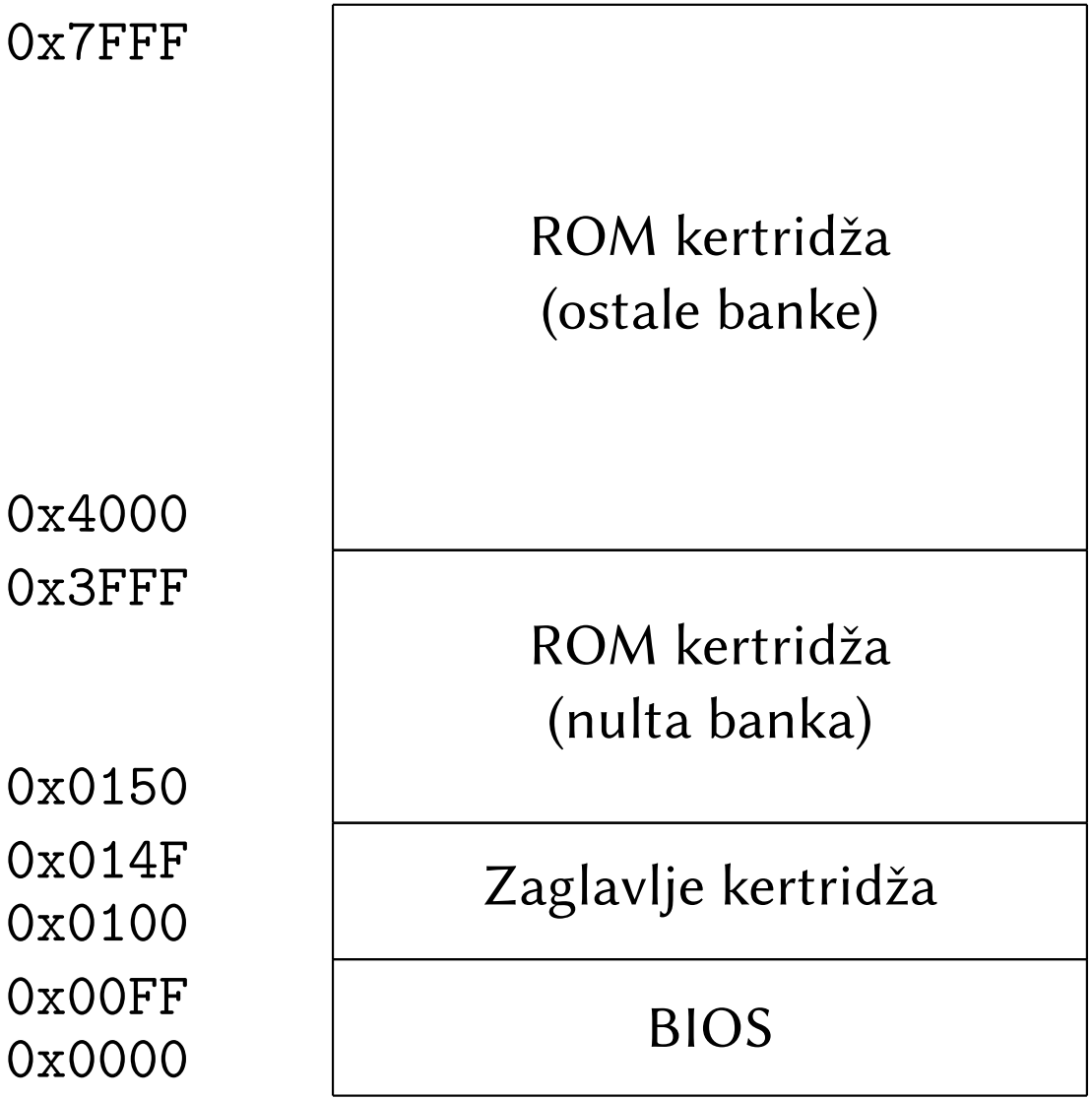
- Naivna metoda vrlo jednostavna...

```
1 int memo[65536];
2 public int ReadByte(int address)
3 {
4   return memo[address];
5 }
6 public void WriteByte(int address, int value)
7 {
8   memo[address] = value;
9 }
```

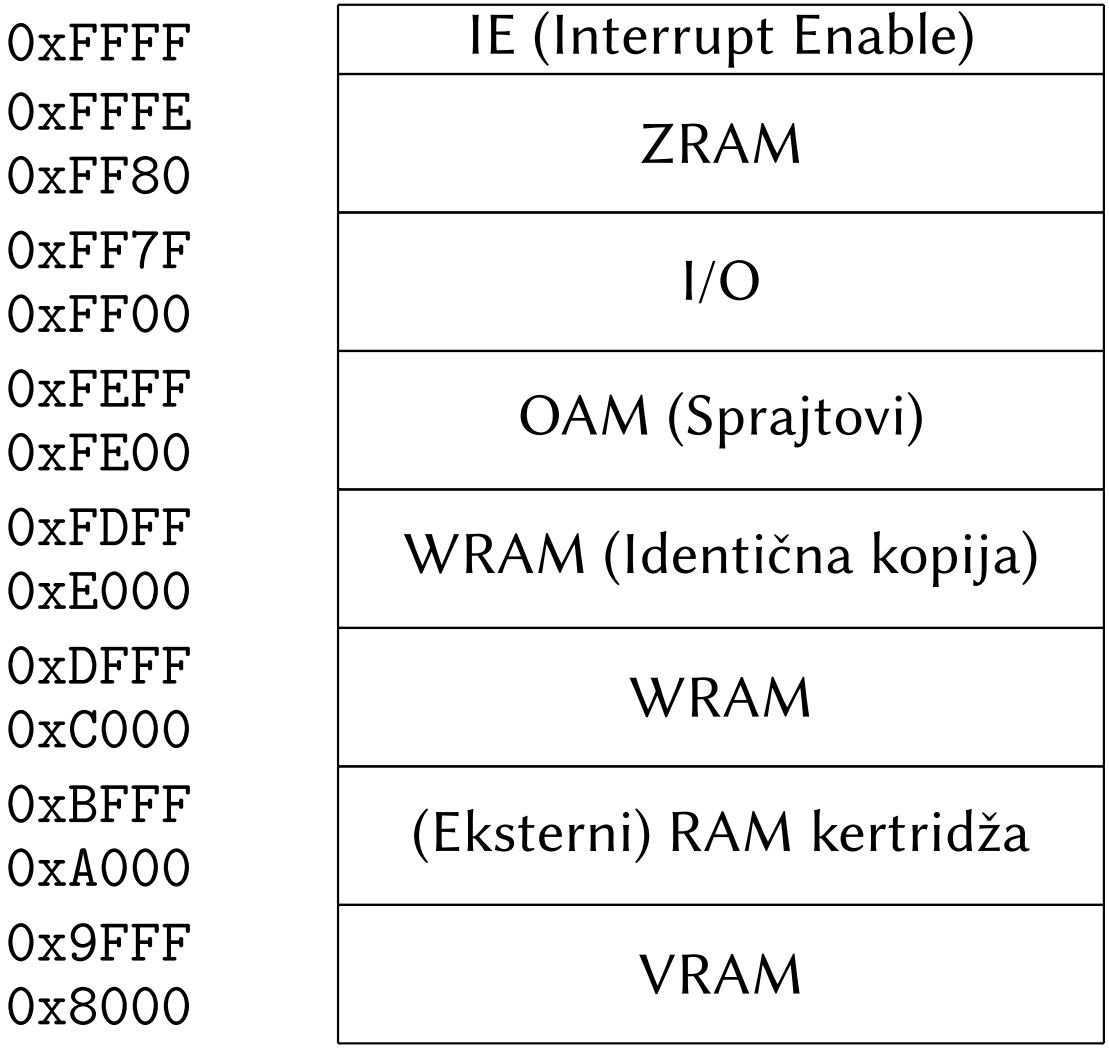
Naivna implementacija, *cont'd*

- Naivna implementacija previđa nekoliko ključnih stvari:
  - Ne možemo pisati u sve adrese (ROM...);
  - Ne predstavljaju sve adrese radnu memoriju (npr. kontrolni registri ulazno/izlaznih uređaja imaju svoje adrese...);
  - Mogu postojati dve različite adrese koje se mapiraju u istu fizičku lokaciju.
  - ...
- Kod GameBoy-a, **sve gorenavedene stavke važe!**
- Treba detaljnije pogledati šta adrese zaista predstavljaju.

Memorijska mapa GameBoy-a: ROM



Memorijska mapa GameBoy-a: RAM



ReadByte

```
1  private byte[] ZRAM = new byte[128];
2  private byte[] VRAM = new byte[8 * 1024];
3  private byte[] WRAM = new byte[8 * 1024];
4  public byte[] OAM = new byte[256];
5
6  public int ReadByte(int address)
7  {
8    if (address <= 0x7FFF) || address >= 0xA000 && address <=
       0xBFFF)
9      return cartridge.ReadByte(address);
10   else if (address >= 0x8000 && address <= 0x9FFF)
11     return VRAM[address - 0x8000];
12   else if (address >= 0xC000 && address <= 0xDEFF)
13     return WRAM[address - 0xC000];
14   else if (address >= 0xE000 && address <= 0xFDFF)
15     return WRAM[address - 0xE000];
16   else if (address >= 0xFE00 && address <= 0xFEFF)
17     return OAM[address - 0xFE00];
18   else if (address >= 0xFF80 && address <= 0xFFFE)
19     return ZRAM[address - 0xFF80];
20   else ... // I/O region, za sada izostavljen
21 }
```

Grafički procesor

Grafički procesor

- Dosadašnjim radom smo uspeli da napravimo emulator koji može, korak po korak, izvršiti bilo koji program za GameBoy.
- Sada želimo da vidimo rezultate toga na grafičkom izlazu.

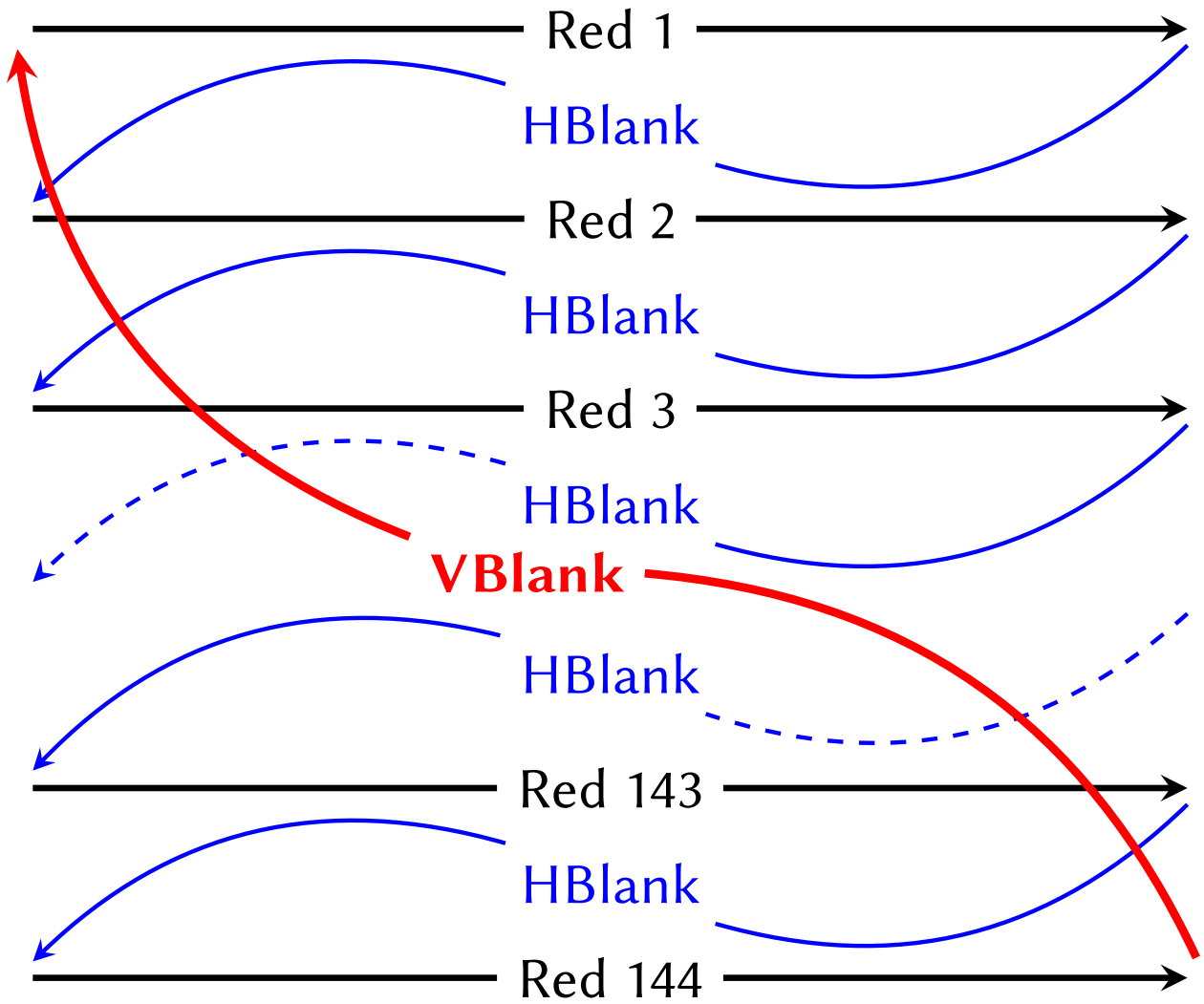
GameBoy grafika

- GameBoy-ev izlaz je LCD rezolucije 160x144 piksela;
  $\Longrightarrow$ možemo deklarisati niz
  `private uint pixels[] = new uint[160 * 144];`
  i vezati ga za objekat klase `PictureBox`.

- Vrednosti u ovom nizu su sledećeg oblika:
  `0xAARRGGBB` (Alpha–Red–Green–Blue)

- GameBoy podržava četiri nijanse sive, tako da će sve vrednosti ovog niza biti jedna od sledeće četiri:
  - `0xFF000000` (BLACK)
  - `0xFF555555` (DARK_GRAY)
  - `0xFFAAAAAA` (LIGHT_GRAY)
  - `0xFFFFFFFF` (WHITE)

Iscrtavanje jednog frejma (∼ *CRT*)



Kada crtati novi frejm?

- Svaka od faza sa prethodne slike traje tačno određen broj procesorskih ciklusa:

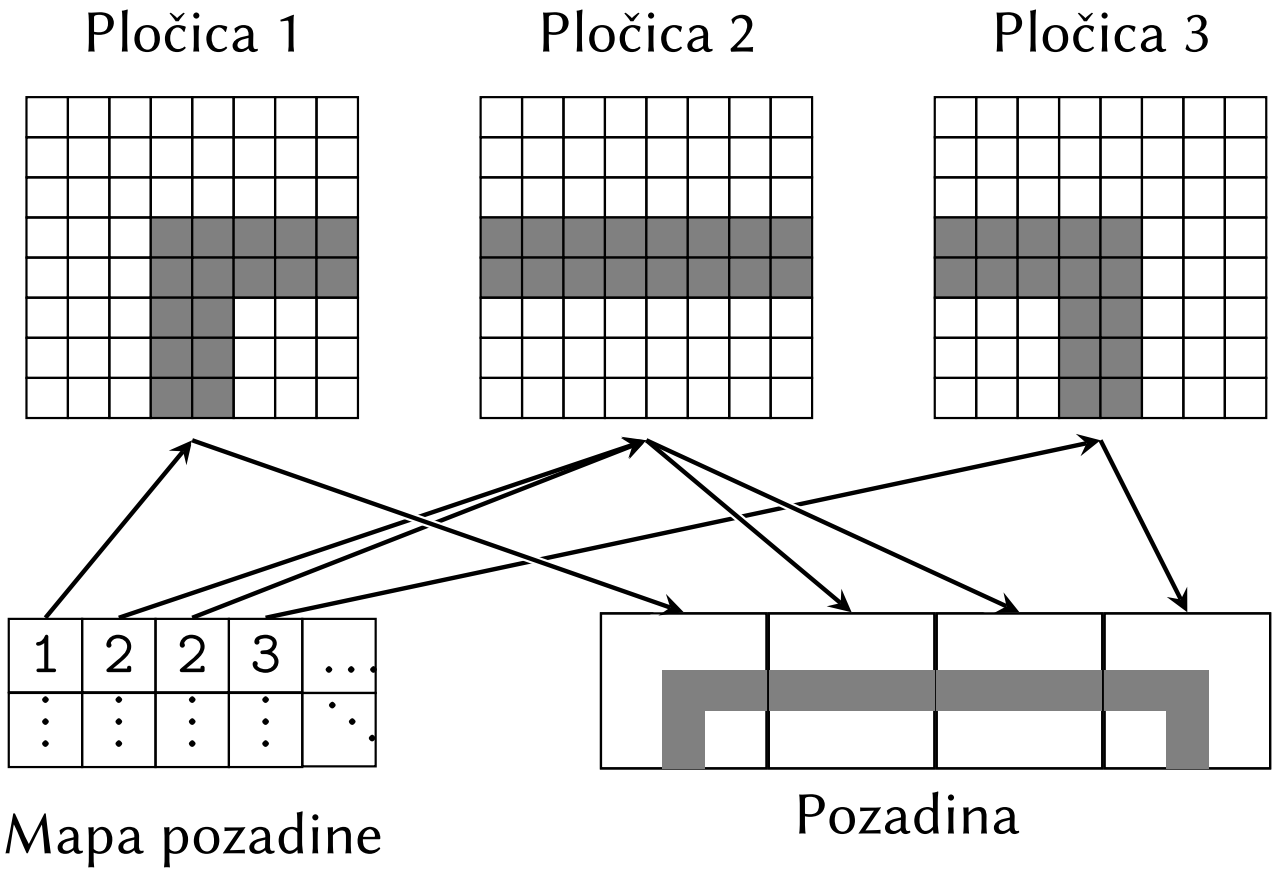
Faza	**Br. ciklusa**
Red (pozadina)	172
Red (sprajtovi)	80
Horizontalni prelaz	204
Jedan red	456
Vertikalni prelaz	4560
Ceo frejm	70224

- Za merenje vremena, možemo koristiti (prethodno definisan) **procesorski clock** (`int ticks`)!

Sistem pločica

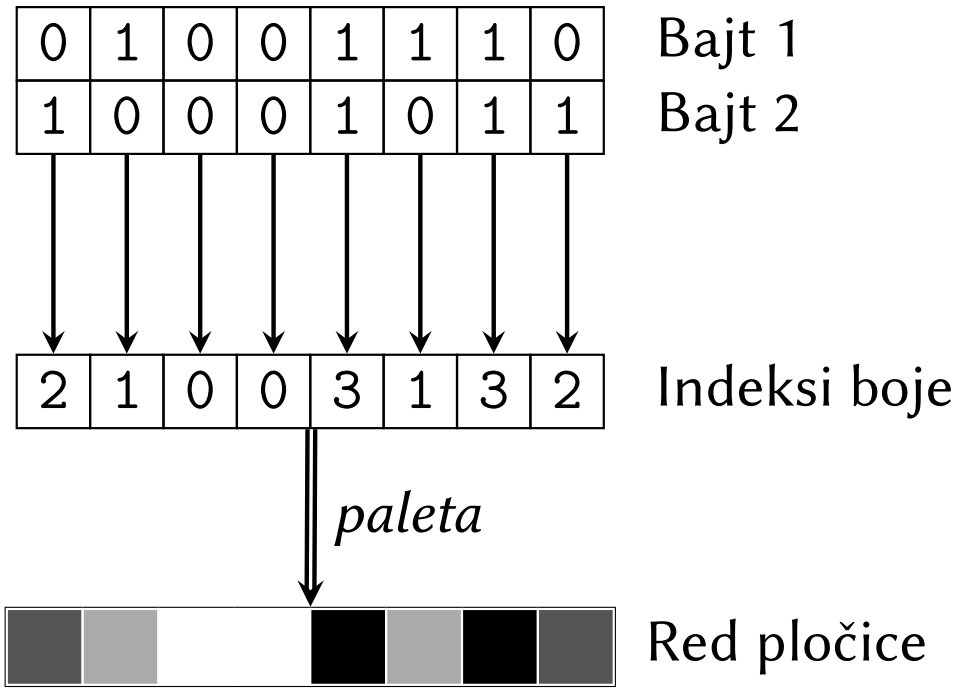
- GameBoy nema dovoljno memorije da sačuva celu matricu piksela; pribegava se *tiling* sistemu.
- U VRAM-u se čuvaju "pločice" veličine 8x8, a pozadina se gradi kao 32x32 matrica pokazivača na pločice.
- Primetimo da ovo daje ukupnu rezoluciju od 256x256, a GameBoy-ev LCD je 160x144!
  - Dva specijalna registra, `Scroll-X` i `Scroll-Y`, sadrže koordinate tačke na pozadini koja će se pokazati u gornjem levom uglu LCD-a.

Sistem pločica, *cont'd*



Gotcha: Organizacija podataka u pločici

- Pošto podržavamo 4 boje, trebaju nam dva bita po polju; svaki red pločice se zapravo sastoji iz dva susedna bajta u VRAM-u:



- *Palete*: par (indeks, boja) nije konstantan!

Ulaz

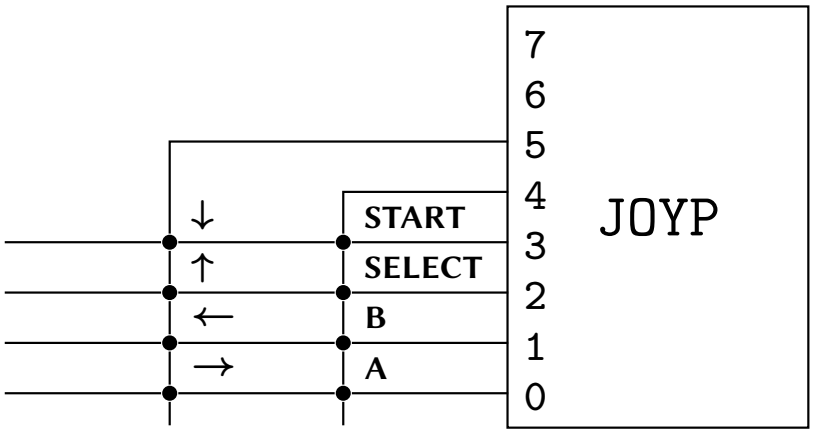
Ulaz

- Sa implementiranim grafičkim procesorom, emulator je sposoban da prikaže grafički izlaz, međutim korisnik nema nikakvu kontrolu nad igricom.

- Sledeći korak: emulacija korisnikovog ulaza.

Ulaz

- Ulaz se procesira pomoću specijalnog JOYP registra, kome procesor može pristupiti na adresi 0xFF00. Žice koje vode do registra su povezane na sledeći način:



- Procesor najpre upiše 0x10 ili 0x20 u registar, da bi aktivirao jednu od dve kolone; zatim posle nekoliko ciklusa može očitati donja 4 bita da bi saznao koji tasteri su pritisnuti.

Implementacija ulaza : front-end

```
1  public bool up, down, left, right;
2  public bool a, b, select, start;
3  public bool col1, col2; //kolone
4
5  private void Form1_KeyDown(object sender, KeyEventArgs e)
6  {
7    switch (e.KeyCode)
8    {
9      case Keys.D: b = true; break;
10     case Keys.F: a = true; break;
11     . . .
12   }
13 }
14
15 private void Form1_KeyUp(object sender, KeyEventArgs e)
16 {
17   switch (e.KeyCode)
18   {
19     case Keys.D: b = false; break;
20     . . .
21   }
22 }
```

Implementacija ulaza : back-end

```
1  public void WriteByte(int address, int value)
2  {
3     . . .
4     else if (address == 0xFF00)
5     {
6        col1 = (value & 0x01) != 0x01; // active low!
7        col2 = (value & 0x02) != 0x02; // active low!
8     }
9     . . .
10 }
```

Implementacija ulaza : back-end, *cont'd*

```
1
2  public void ReadByte(int address)
3  {
4     . . .
5     else if (address == 0xFF00)
6     {
7        int ret = 0;
8        if (col1)
9        {
10          if (!down) ret |= 0x08; // active low!
11          if (!up) ret |= 0x04;  // active low!
12          if (!left) ret |= 0x02; // active low!
13          if (!right) ret |= 0x01; // active low!
14          return ret;
15       }
16       else if (col2)
17       {
18          . . .
19       }
20    }
21    . . .
22 }
```

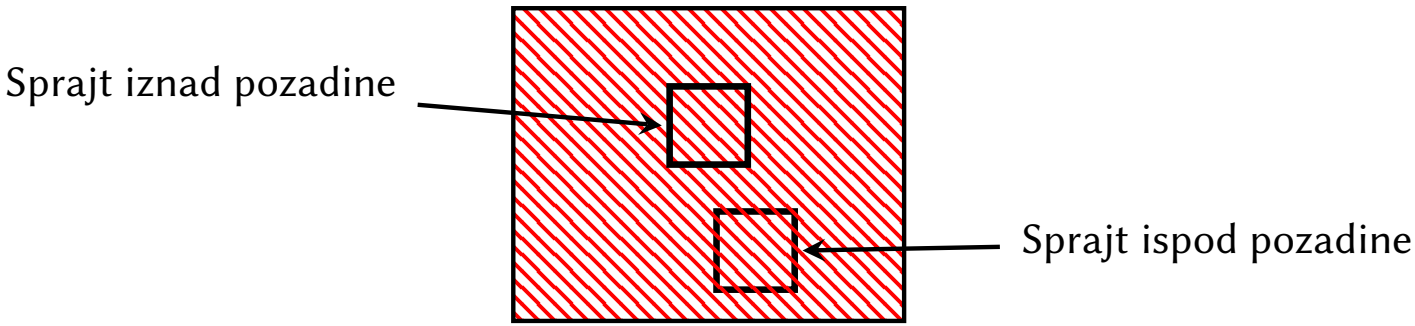
Sprajtovi

- Sa procesiranim ulazom, mogu se igrati neke igrice – međutim gotovo sve se moraju igrati naslepo (ne vidi se indikator koji govori igraču koja mu je pozicija).

- Ovaj problem (kao i mnogi drugi) se rešavaju upotrebom **sprajtova** – pločica koje se mogu crtati i pomerati odvojeno od pozadine.

Sprajtovi: dodatni detalji

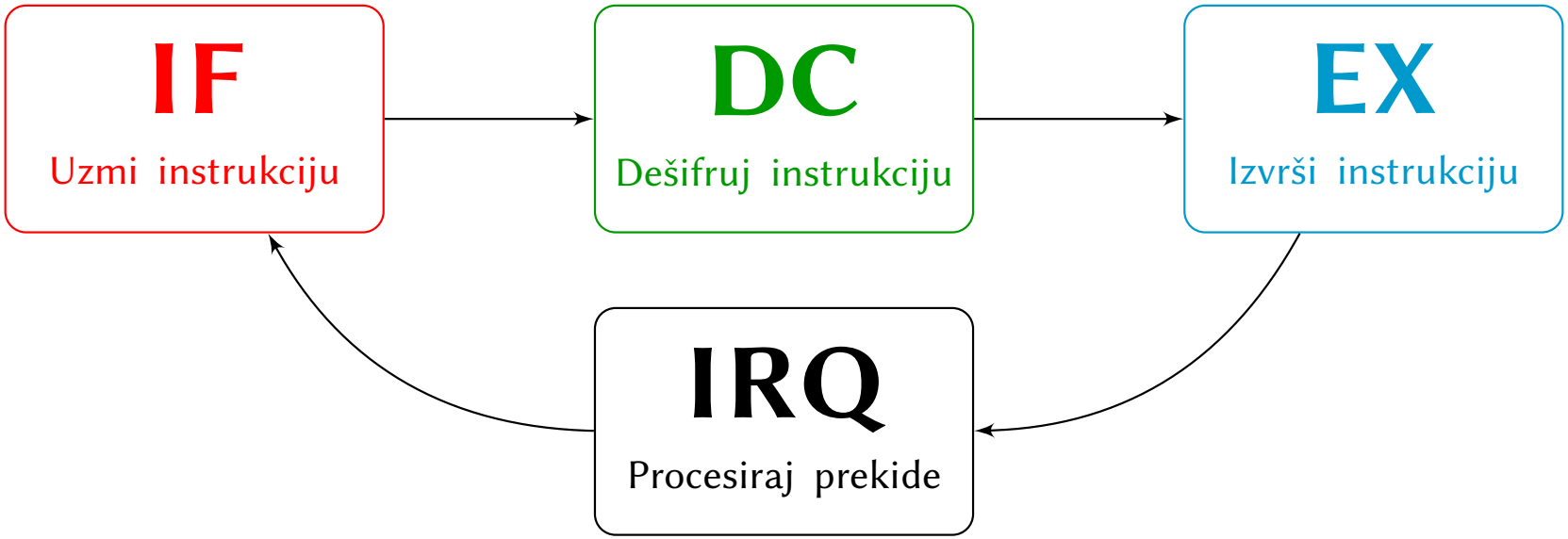
- Sprajtovi koriste potpuno isti tip 8x8 pločica kao i pozadina.
- Specifični parametri koji se čuvaju za svaki sprajt u OAM-u:
  - (x, y) koordinate gornjeg levog temena;
  - *prioritet:* da li je iznad ili ispod pozadine;
  - *obrtanje:* horizontalno ili vertikalno;
  - *veličina:* mogu se svi sprajtovi odjednom povećati na 8x16.
- Algoritam za crtanje frejma: prvo nacrtati pozadinu, onda zaključivati na osnovu boja i prioriteta sprajtova da li precrtavati sprajt preko pozadine.



Prekidi

- Uz podršku za sprajtove, neke manje igrice će raditi u potpunosti.
- Mnogim igricama fali jedan bitan detalj: stavljanje do znanja procesoru **kada** može generisati sledeći frejm.
- Najpogodniji period za menjanje frejma je **tokom VBlank-a**, zato što tokom prelaza nazad u prvi red grafički procesor ne pristupa ni VRAM-u ni OAM-u.
- Najbolji (u smislu potrošnje resursa) metod za obaveštavanje procesora o nekom važnom događaju kao što je ulaz u VBlank je upravo **prekid** *(interrupt)*.

Procesorski ciklus, ponovo posećen



- Posle svake izvršene instrukcije, procesor proverava da li se desio neki prekid; ukoliko je prepoznat prekid, procesor:
  - Zapamti svoje trenutno stanje;
  - Skače na adresu *razrešivača* za prepoznati prekid.
  - Izvršava kod razrešivača, koji se završava specijalnom `RETI` *(Return From Interrupt)* instrukcijom.

Prekidi: GameBoy specifičnosti

- Procesor ima pristup *Interrupt Enable (IE)* registru na adresi `0xFFFF`, gde može precizirati koje prekide (od njih ukupno 5) želi u sadašnjem momentu da procesira.
- Procesor takođe ima *Interrupt Master Enable (IME)* prekidač preko kog može potpuno deaktivirati procesiranje prekida (ovo se radi npr. tokom procesiranja jednog, jer ne možemo procesirati dva prekida odjednom).
- Da li se neki prekid desio se može očitati iz *Interrupt Request (IRQ)* registra, na adresi `0xFF0F`, mada su ove informacije direktno dostupne procesoru, bez potrebe da čita memoriju.

GameBoy prekidi

Prioritet	Prekid	Adresa razrešivača
0	*VBlank*	0x0040
1	*LCD Status*	0x0048
2	*Timer*	0x0050
3	*Serial*	0x0058
4	*Joypad*	0x0060

```
1 public bool interruptsEnabled;
2 public bool vBlankInterruptEnabled,vBlankInterruptRequested;
3 . . .
```

Implementacija INTR i RETI instrukcija

```
1  private void Return() // vracanje na prvu adresu sa steka
2  {
3    Pop(ref PC);
4  }
5
6  // skoci na razresivac na adresi address
7  private void Interrupt(int address)
8  {
9    // dok procesiramo jedan prekid ne mozemo druge
10   interruptsEnabled = false;
11   Push(PC); // pamcenje trenutne pozicije na stek
12   PC = address; // skok do adrese
13 }
14
15 private void ReturnFromInterrupt() // povratak iz prekida
16 {
17   interruptsEnabled = true;
18   Return();
19   ticks += 16; // operacija zahteva 16 procesorskih tickova
20 }
```

Implementacija u procesorskom ciklusu

```
1  public void Step()
2  {
3    if (interruptsEnabled)
4    {
5      if (vBlankInterruptEnabled && vBlankInterruptRequested)
6      {
7        vBlankInterruptRequested = false;
8        Interrupt(0x0040); // razresivac za VBlank prekid
9      }
10     else if . . . // ostali prekidi
11   }
12   . . . // ostatak procesorskog ciklusa
13 }
```

Kraj!

- Sa dodatkom samo VBlank prekida, emulator je sposoban da pokrene *Tetris*. :)

- Sve detalje oko funkcionalnosti GameBoy, kao i Game Boy Color konzole možete naći na sledećem (jako korisnom) referentnom materijalu:
  `http://problemkaputt.de/pandocs.htm`