

# LAB 2 – CPU simulaator

# Vajalik tarkvara I

- JRE (Java Runtime Environment)

[www.java.com](http://www.java.com)

LogiSim on Java rakendus ja vajab toimimiseks minimaalselt JRE pakki.

- LogiSim
- CPU mudeli failid

# Vajalik tarkvara II

- CPU-mudel ja LogiSim-i pakitud arhiivi leiad siit:

<http://www.tud.ttu.ee/im/Elmet.Orasson/IAX0043/Labor2CPUmudel/CPU-to-send.zip>

Lae see omale alla ja paki lahti oma kodukausta.

- LogiSim (<https://sourceforge.net/projects/circuit/>)

Siit leiad kõige värskema LogiSim rakenduse

# CPU simulaator

Simulaatori autori originaal kasutusjuhendi leiata kas  
<http://ati.ttu.ee/~lemva/praktikumi-materjal-lyhem-ver08.pdf>

või

CPU mudeli arhiivist (vt link eespool)

või

<http://www.tud.ttu.ee/im/Elmet.Orasson/IAX0043/Labor2CPUmudel/CPU%20kirjeldus.pdf>

# Simulaatori käivitamine I

- Mine oma lahtipakitud CPU simulaatori failide kausta
- Käivita seal LogiSim (tõenäoliselt ainuke .jar fail kaustas)
- Ava LogiSim programmiga **pohiskeem.circ**  
(File -> Open ->...)
- LogiSim bug – avamisel tõenäoliselt küsitakse CPU moodulfailide nimesid üle. Näidake siis kõige sarnasema nimega .circ failide nimed talle kätte.

# Simulaatori käivitamine II

- Vajalikuks võib osutada CPU juhtosa mikroprogrammi laadimine. Parem hiireklõps skeemil **CU** moodulil -> View control...

Seejärel CU vaates uuesti parem hiireklõps ainukesel tabeli moodi objektile -> Load image. Valige **protsessoriROMsisu.txt**

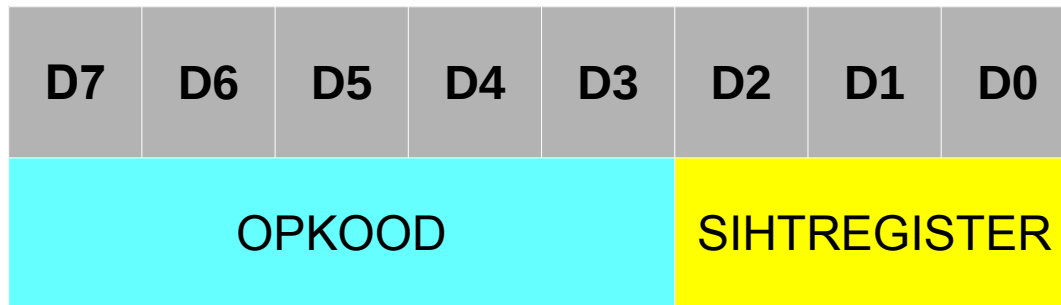
- Teeme topelkliki **Processor scheme** valikus, sellega liigume tagasi põhiskeemile.
- Paneme paika sünkrosignaali generaatori, selleks klikime läbi:  
Simulate -> Tick frequency -> 1Hz  
Simulate -> Ticks enabled (et oleks linnuke kirjas)

# Programmi sisestus ja jooksumine

- Masinkoodis käsud sisestame põhiakna ainukesse tabelisse (näitab vaid 16 hex numbrit). Parema hiireklikiga saab avada **Edit Contents**, mille abil pääseb vajadusel suurema hulga baitide ligi.
- Soovitav on esmalt klikkida RES (lähtestab CPU)
- Seejärel saab programmi täitmist jälgida kas MCLK sisendit klikkides (iga klikk on üks taktimpulss) või ACLK lubades (Ticker sagedusega sünkro).

# L-CPU käsustik

- Käsud on 8-bitised, millest 5 bitti (D7-D3) on opkood (seega max 32) ja ülejäänud 3 bitti (D2-D0) adresseerivad vajadusel erinevaid registreid:





# L-CPU käsustik

| opkood | mnemkood | opkood | mnemkood | opkood | mnemkood             | opkood | mnemkood            |
|--------|----------|--------|----------|--------|----------------------|--------|---------------------|
| 00000  | NOP      | 01000  | C -> R   | 10000  | A&B->R               | 11000  | ror A-> R           |
| 00001  | LD R,(M) | 01001  | M -> R   | 10001  | A B->R               | 11001  | ror B-> R           |
| 00010  | LD R #   | 01010  | NOP      | 10010  | $\overline{A B}$ ->R | 11010  | rol A-> R           |
| 00011  | NOP      | 01011  | JmpZ #   | 10011  | A xor B              | 11011  | rol B-> R           |
| 00100  | ST C,(M) | 01100  | JmpC #   | 10100  | A+B -> R             | 11100  | $\overline{A}$ -> R |
| 00101  | ST C #   | 01101  | JmpE #   | 10101  | A-B -> R             | 11101  | $\overline{B}$ -> R |
| 00110  | A -> R   | 01110  | JMP #    | 10110  | A cmp B              | 11110  | B-1 -> R            |
| 00111  | B -> R   | 01111  | NOP      | 10111  | A+B+c -> R           | 11111  | B+1 -> R            |

# L-CPU registrid

| registri spetsifikaator<br>käsu koosseisus | käsu sihtregister |
|--|-------------------|
| ?????000                                   | -                 |
| ?????001                                   | A                 |
| ?????010                                   | B                 |
| ?????011                                   | C                 |
| ?????100                                   | MAR               |
| ?????101                                   | MDR               |
| ?????110                                   | PC                |
| ?????111                                   | M                 |

# Näidislahendus I

**Ülesanne:** laadida 0x02 ja 0x5B väärtusega operandid põhimälust, sooritada nendega XOR tehe ja salvestada tulemus uuesti mälli, kasutades selleks **otsest adresseerimist**.

# Näidilahendus II

Paneme maha põhiplaani. Arvestame et iga vahetus adresseerimist kasutav operatsioon on 2 baiti. Esialgu me veel ei tea, kuipikk programm tuleb ning seepärast ei tea operandide täpseid mäluaadresse (ehk nn first pass).

Kirjutame aadressideks fiktiivsed addr#1, addr#2 jne.

# Näidislahendus III

| Mnemokood    | pikkus baitides | binaarkood | hex   | hex   |
|--------------|-----------------|------------|-------|-------|
| LD M #addr1  | 2               | 00010111   | '0x17 | '0x0c |
| LD A,(M)     | 1               | 00001001   | '0x09 |       |
| LD M #addr2  | 2               | 00010111   | '0x17 | '0x0d |
| LD B,(M)     | 1               | 00001010   | '0x0a |       |
| A xor B -> C | 1               | 10011011   | '0x9b |       |
| LD M #addr3  | 2               | 00010111   | '0x17 | '0x0e |
| ST C,(M)     | 1               | 00100011   | '0x23 |       |
| JMP 0x00     | 2               | 01110000   | '0x70 | 0x00  |
|              | 12              |            |       |       |

# Näiteülesanne – vahetu adresseerimine

Näidisülesanne: laadida vahetu adresseerimisega 3 liidetavat operandi 4, 5, 6  
Teeme liitmistehted, salvestame tulemuse samuti vahetu adresseerimisega

| Mnemkood | Binary   | Hex                         | Hex2 | Pikkus    |   |
|----------|----------|-----------------------------|------|-----------|---|
| LD A #4  | 00010001 | 0x11                        | 0x04 | 2         | Laeme arvu 4 A reg.                                       |
| LD B #5  | 00010010 | 0x12                        | 0x05 | 2         | Laeme arvu 5 B reg.                                       |
| A+B ->B  | 10100010 | 0xA2                        |      | 1         | Liidame A ja B, tulemus B reg.                            |
| LD A #6  | 00010001 | 0x11                        | 0x06 | 2         | Laeme arvu 6 A reg.-sse                                   |
| A+B ->C  | 10100011 | 0xA3                        |      | 1         | Liidame A ja B, tulemus C reg.-sse                        |
| ST C #   | 00101011 | 0x2B                        | 0x00 | 2         | Salvestame C vahetult (so. käsule järgnevasse mälupeassa) |
| JMP 0x00 | 01110000 | 0x70                        | 0x00 | 2         | Siirdume tagasi programmi algusse (tsükkel)               |
|          |          | <b>Programmi kogupikkus</b> |      | <b>12</b> |   |