8-Bit CPU Design

↘ Autor: Andre Stratz, Samuel Hessberger

↘ Datum: 04/07/2023

Hochschule Fulda
University of Applied Sciences

Inhaltsverzeichnis

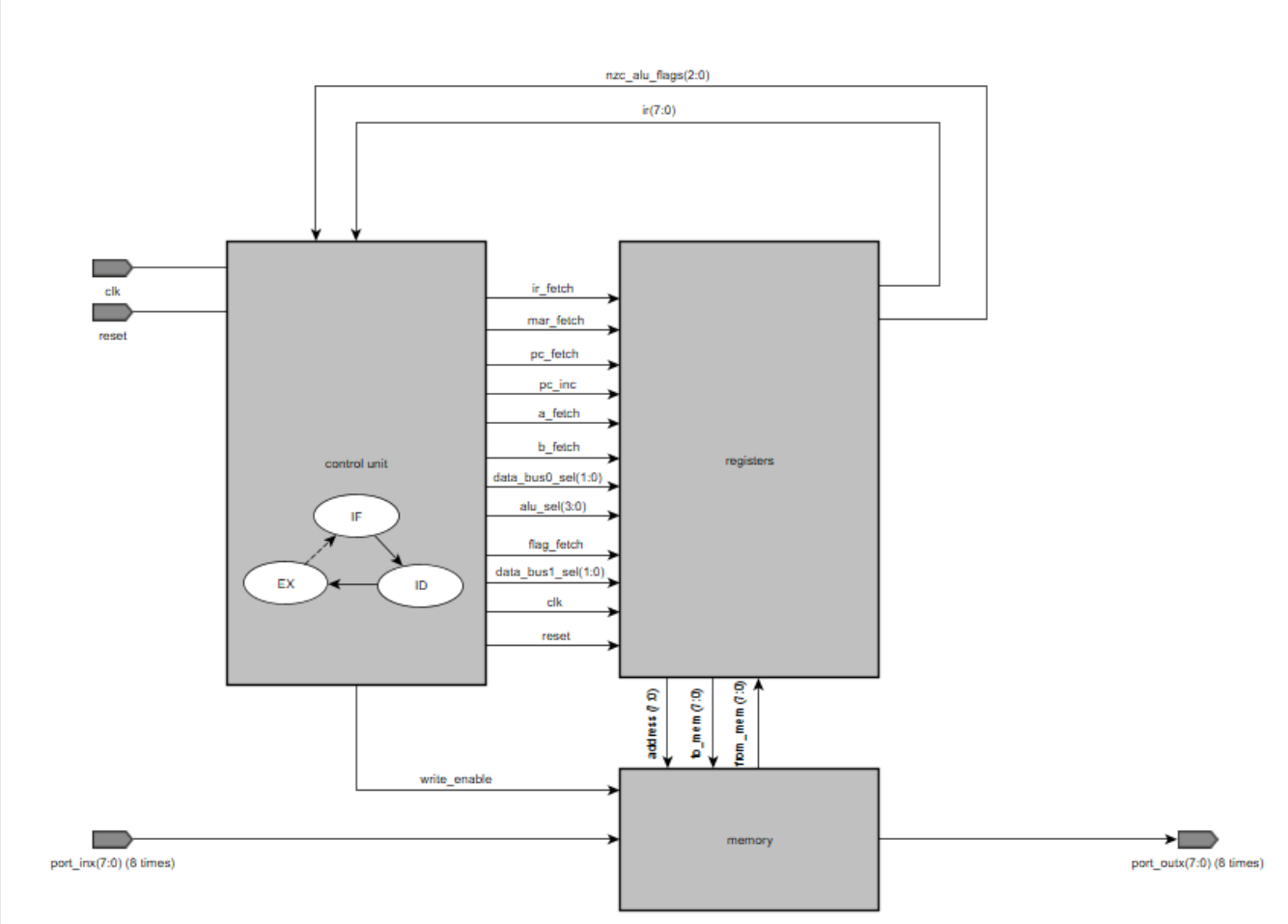
CPU

Drei Komponenten:
- Control unit
- Registers
- Memory

Control unit:
- Speicherung des Zustandes der Maschine (IF -> ID -> EX -> IF)
- Wählen des nächsten Zustandes

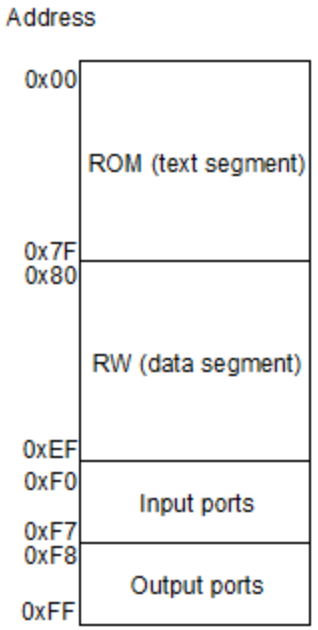
Registers:
- Zwischenspeicher
- Arithmetische logische Einheit (ALU)

Memory:
- Non volatile memory (NVM) (Programmspeicher)
- Volatile memory (Read-Write-Memory)

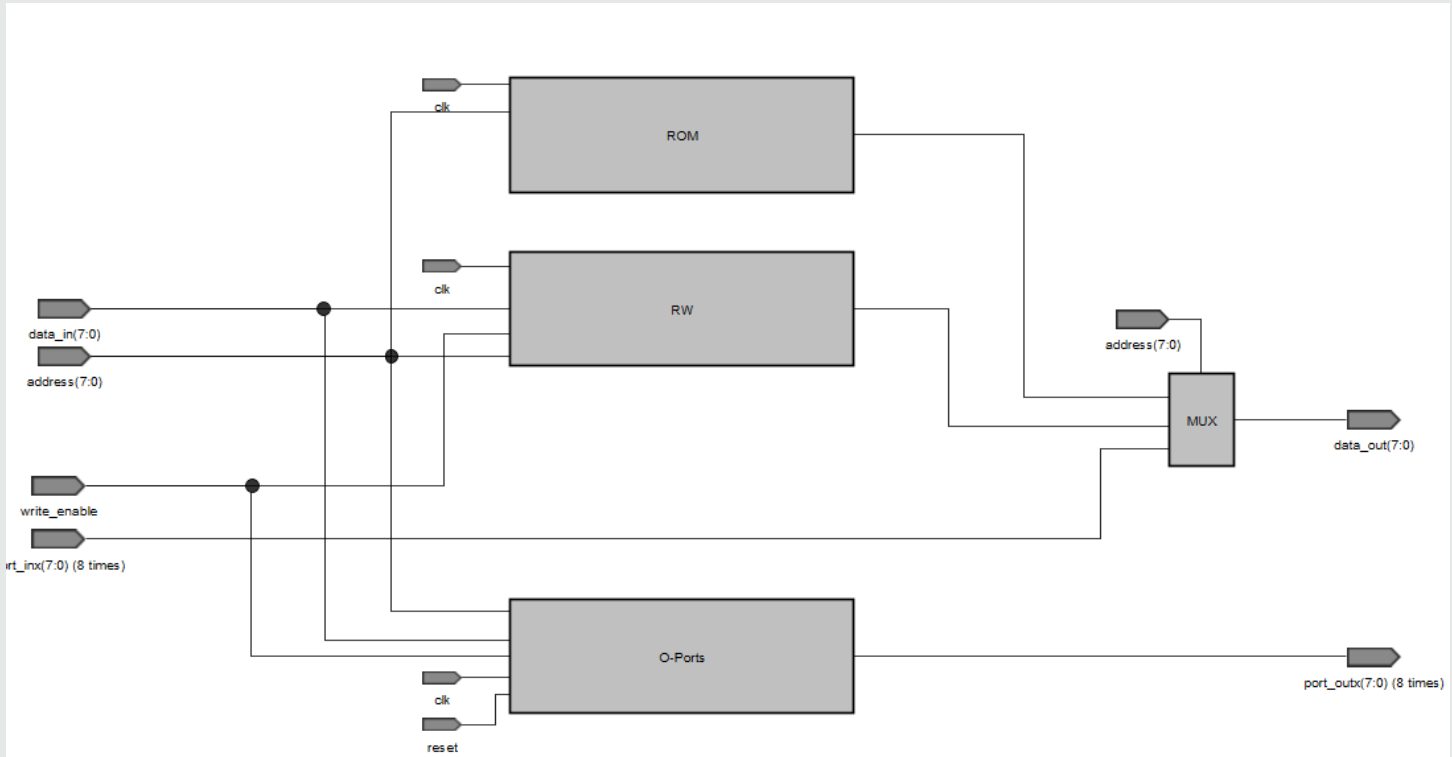


Speicher

Speicheraufteilung:



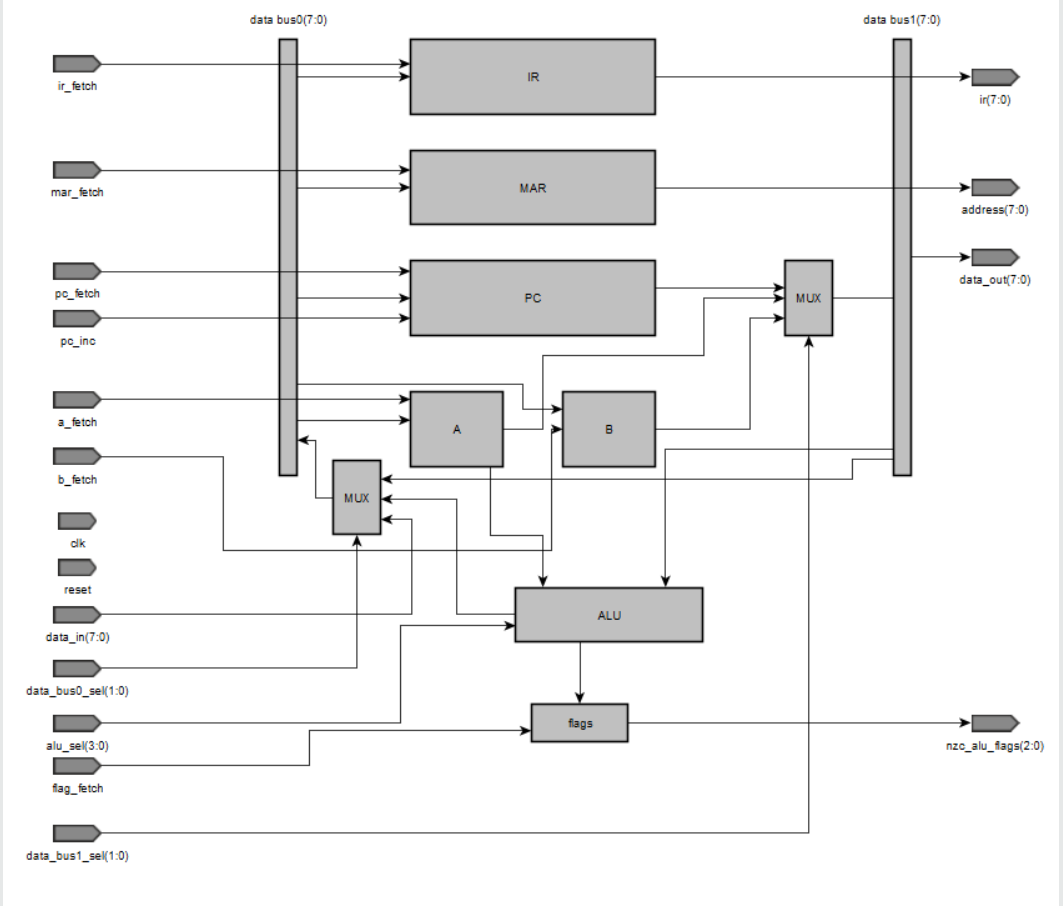
Speicherdesign:



Register

ALU (Artithmetic Logic Unit):

- Kein Register
- Führt logische und arithmetische Befehle aus
- Das Ergebnis wird in A gespeichert



Register

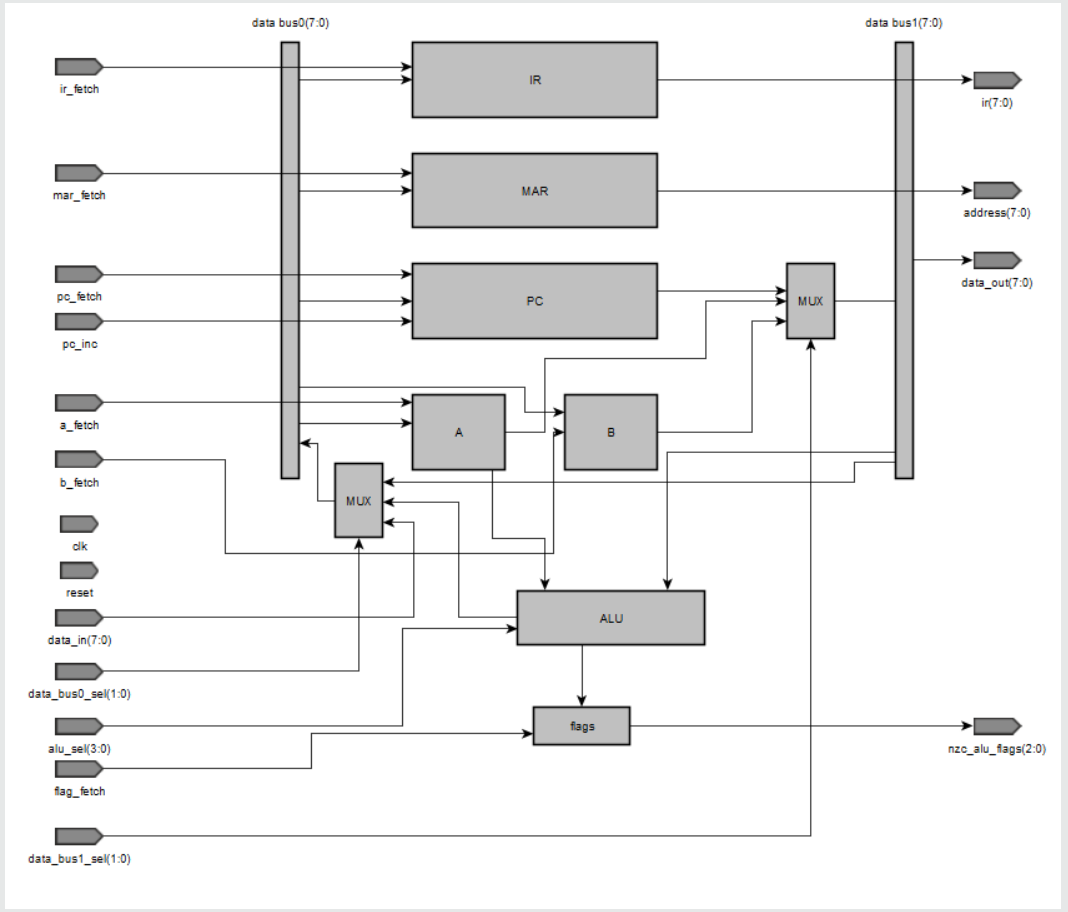
IR (Instruction Register):
- Speicherung der Instruktion

MAR (Memory Adress Register)
- Speicherung der Zugriffsadresse zum Speicher

PC (Program Counter)
- Speicherung der Adresse der nächsten Instruktion

A/B (General Purpose Register):
- A: Eingabe für die ALU
- B: Mögliche Eingabe für die ALU

Flags
- Negativ (Eins, wenn das achte Bit eins ist)
- Zero (Eins, wenn alle Bits Null sind)
- Carry (Eins, wenn das theoretische neunte Bit gesetzt wird)



Befehle

```
library ieee;
use ieee.std_logic_1164.all;
package INSTRUCTIONS is
--LOAD and STORE Instructions--
constant LDA_IMM : std_logic_vector(7 downto 0) := x"00"; -- LDA_IMM <value>: A = value --
constant LDA_DIR : std_logic_vector(7 downto 0) := x"01"; -- LDA_DIR <address>: A = mem[address]--
constant LDB_IMM : std_logic_vector(7 downto 0) := x"02"; -- LDB_IMM <value>: B = value --
constant LDB_DIR : std_logic_vector(7 downto 0) := x"03"; -- LDB_DIR <address>: B = mem[address]--
constant STA_DIR : std_logic_vector(7 downto 0) := x"04"; -- STA_DIR <address>: mem[address] = A --
constant STB_DIR : std_logic_vector(7 downto 0) := x"05"; -- STB_DIR <address>: mem[address] = B --
--Arithmetic and Logic--
constant ADD_AB : std_logic_vector(7 downto 0) := x"55"; --ADD_AB: A = A + B--
constant SUB_AB : std_logic_vector(7 downto 0) := x"56"; --SUB_AB: A = A - B--
constant AND_AB : std_logic_vector(7 downto 0) := x"57"; --AND_AB: A = A & B--
constant OR_AB : std_logic_vector(7 downto 0) := x"58"; -- OR_AB: A = A | B--
constant INC_A : std_logic_vector(7 downto 0) := x"59"; -- INC_A: A = A + 1--
constant INC_B : std_logic_vector(7 downto 0) := x"5A"; -- INC_B: B = B + 1--
constant DEC_A : std_logic_vector(7 downto 0) := x"5B"; -- DEC_A: A = A - 1--
constant DEC_B : std_logic_vector(7 downto 0) := x"5C"; -- DEC_B: B = B - 1--
--Branches--
constant JMP : std_logic_vector(7 downto 0) := x"AA"; --JMP <address> jumps to address--
constant JMP_IN : std_logic_vector(7 downto 0) := x"AB"; --JMP_IN <address> jumps to address, if negativ flag is set (N=1)--
constant JMP_NN : std_logic_vector(7 downto 0) := x"AC"; --JMP_NN <address> jumps to address, if negativ flag isn't set (N=0)--
constant JMP_IZ : std_logic_vector(7 downto 0) := x"AD"; --JMP_IZ <address> jumps to address, if zero flag is set (Z=1)--
constant JMP_NZ : std_logic_vector(7 downto 0) := x"AE"; --JMP_NZ <address> jumps to address, if zero flag isn't set (Z=0)--
constant JMP_IC : std_logic_vector(7 downto 0) := x"AF"; --JMP_IC <address> jumps to address, if carry flag is set (C=1)--
constant JMP_NC : std_logic_vector(7 downto 0) := x"B0"; --JMP_NC <address> jumps to address, if carry is'nt carry set (C=0)--
end INSTRUCTIONS;
```

Adressierungsarten:

- Immediate (Konstante ist Teil des Befehls)
- Direkt (Effektive Adresse ist Teil des Befehls)

Control Unit

Implementierung eines Moore-Schaltwerks

NSD:

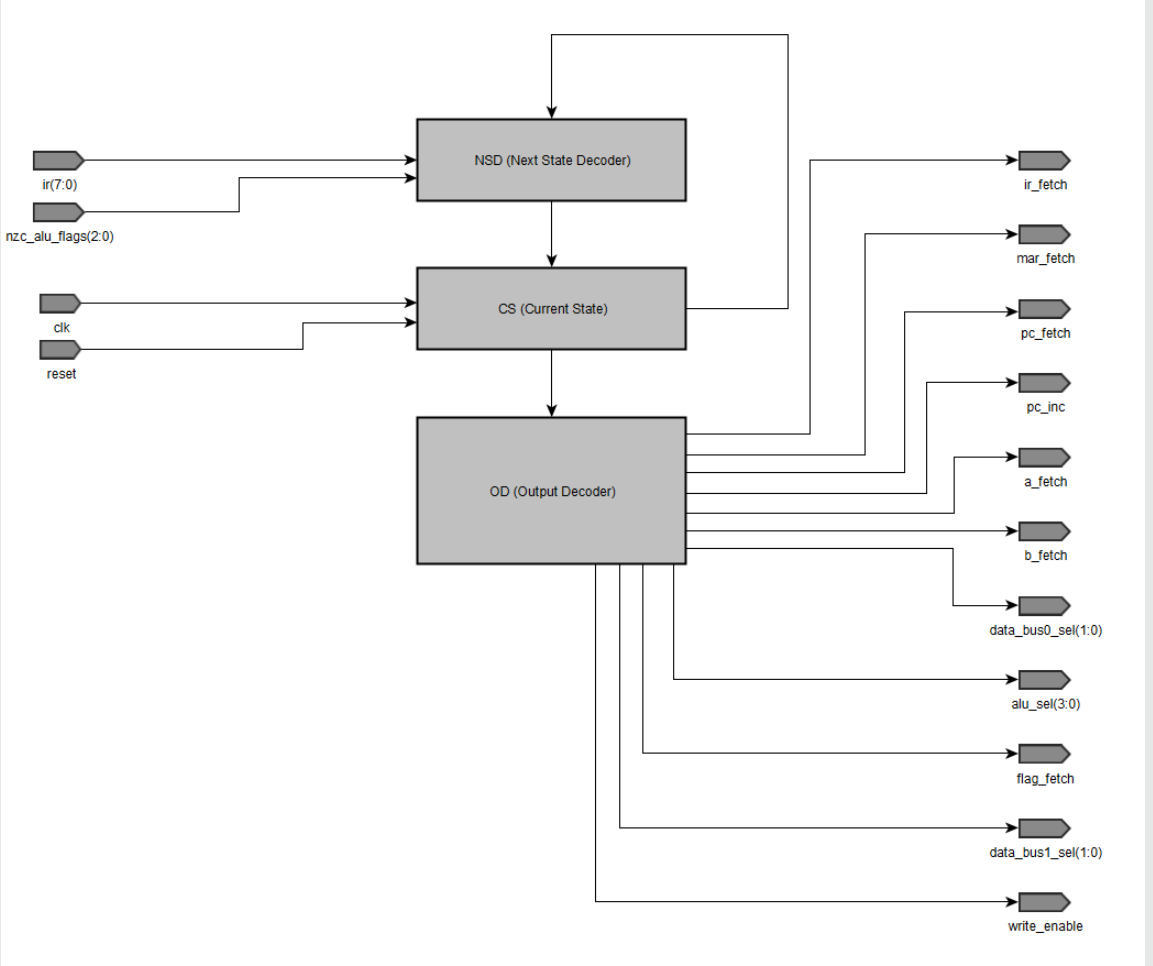
- Z(n+1) = f(Z(n), E)

CS:

- Register (Speicherung des aktuellen Zustands)

OD:

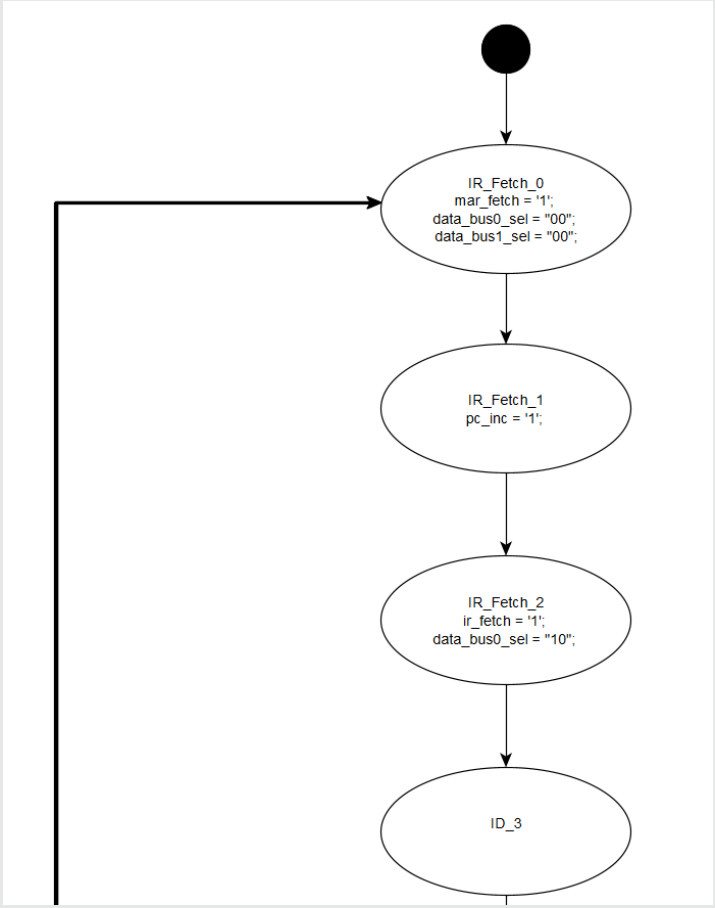
- A = f(Z(n))



Control Unit

Zustandsdiagramm:

- In jedem Taktzyklus ändert sich der Zustand
- Das gewünschte Ergebnis (e.g. Registermanipulation) wird im folge Zustand erreicht
- Lesen aus dem Speicher dauert ein Taktzyklus



Beispielprogramm

Programm im ROM:

```
constant my_rom : rom := (
0 => LDA_IMM,
1 => x"01",
2 => STA_DIR,
3 => x"F8",
4 => JMP,
5 => x"64",
6 => STA_DIR,
7 => x"80",
8 => LDB_DIR,
9 => x"80",
10 => ADD_AB,
11 => JMP_IC,
12 => x"00",
13 => JMP,
14 => x"02",
100 => LDB_IMM,
101 => x"02",
102 => DEC_B,
103 => JMP_NZ,
104 => x"66",
105 => JMP,
106 => x"06",
others => x"00" );
```

Mögliche Implementierung in C:

```
while(1)
{
  uint8_t* output = (uint8_t*)0xF8;
  uint8_t outval = 0x01;
  while(outval > 0x00)
 {
   *output = outval;
    for(uint8_t i = 0x2; i > 0; --i); //sleep
    outval = outval << 1;
  }
}
```

Der Wert des Ausgabeports wird in jeder Iteration um eins nach links geschiftet. Zwischen jeder Überschreibung wird geschlafen.

Danke für Ihre Aufmerksamkeit

Autor

E-Mail: samuel-lukas.hessberger@et.hs-fulda.de
E-Mail: andre-georg.stratz@et.hs-fulda.de

Code und Dokumentation

https://github.com/free43/8Bit-CPU-Design