Implementierung eines Mikrorechners in VHDL auf einem FPGA

Von Markus Schneider

Educational Processing Unit

EPU

Von Markus Schneider

Gliederung

- Beschreibung des Projektes
- Hardwareübersicht
- Umsetzung in VHDL
- EPU-Assembly
- Befehlsaufbau
- Funktionsweise
- Demo

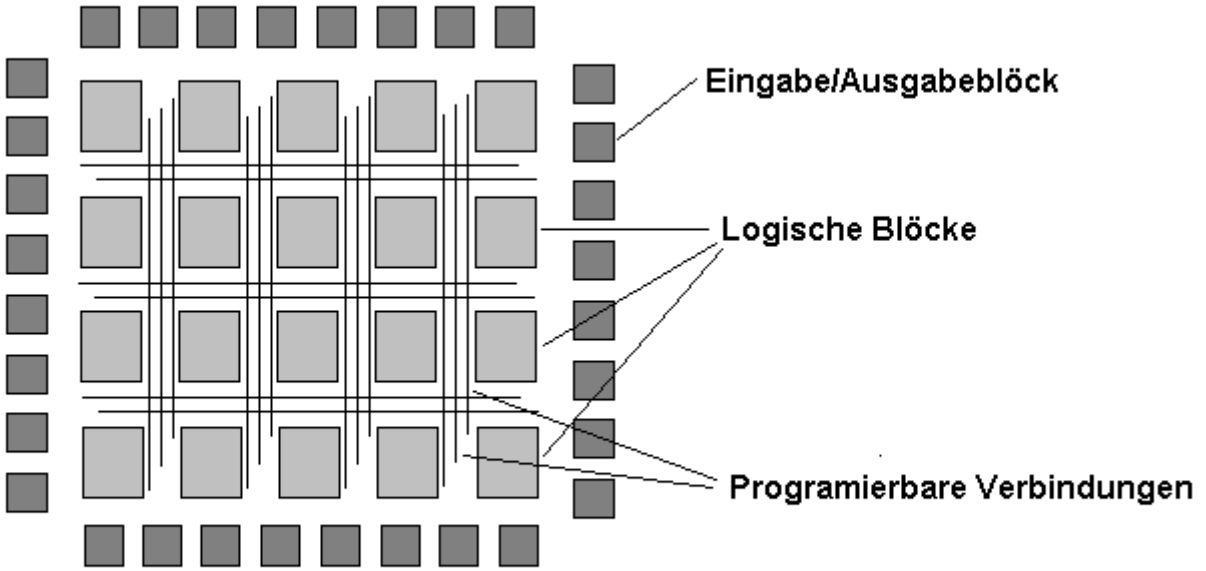
Beschreibung des Projektes

- **Ziel:**
  - Entwurf eines funktionsfähigen Mikrorechners
  - Vom Grund auf selbst erstellt
  - Umsetzung des Entwurfes in die Realität
- **Umsetzung:**
  - Hardware ist auf einem FPGA implementiert
  - Hardwarebeschreibungssprache ***VHDL*** zum Beschreiben der Hardware

Was ist ein FPGA?

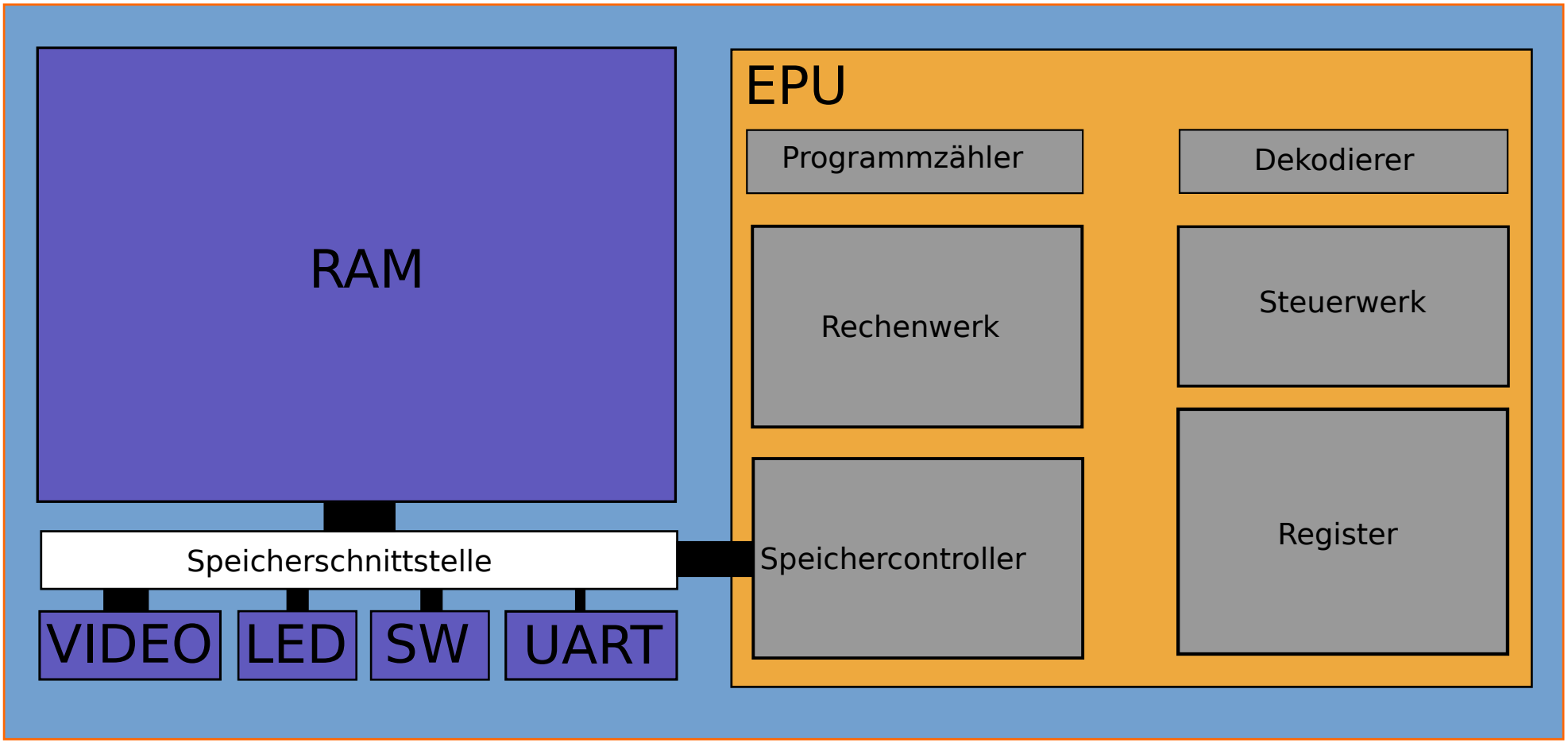
- **“Field Programmable Gate Array”**
- **Integrierter Schaltkreis**
- **Besteht aus Funktionsblöcken**
- **Funktionsblöcke können beliebig zusammengefügt**
- **Durch Programmierung entsteht die gewünschte Schaltung**

Was ist ein FPGA?

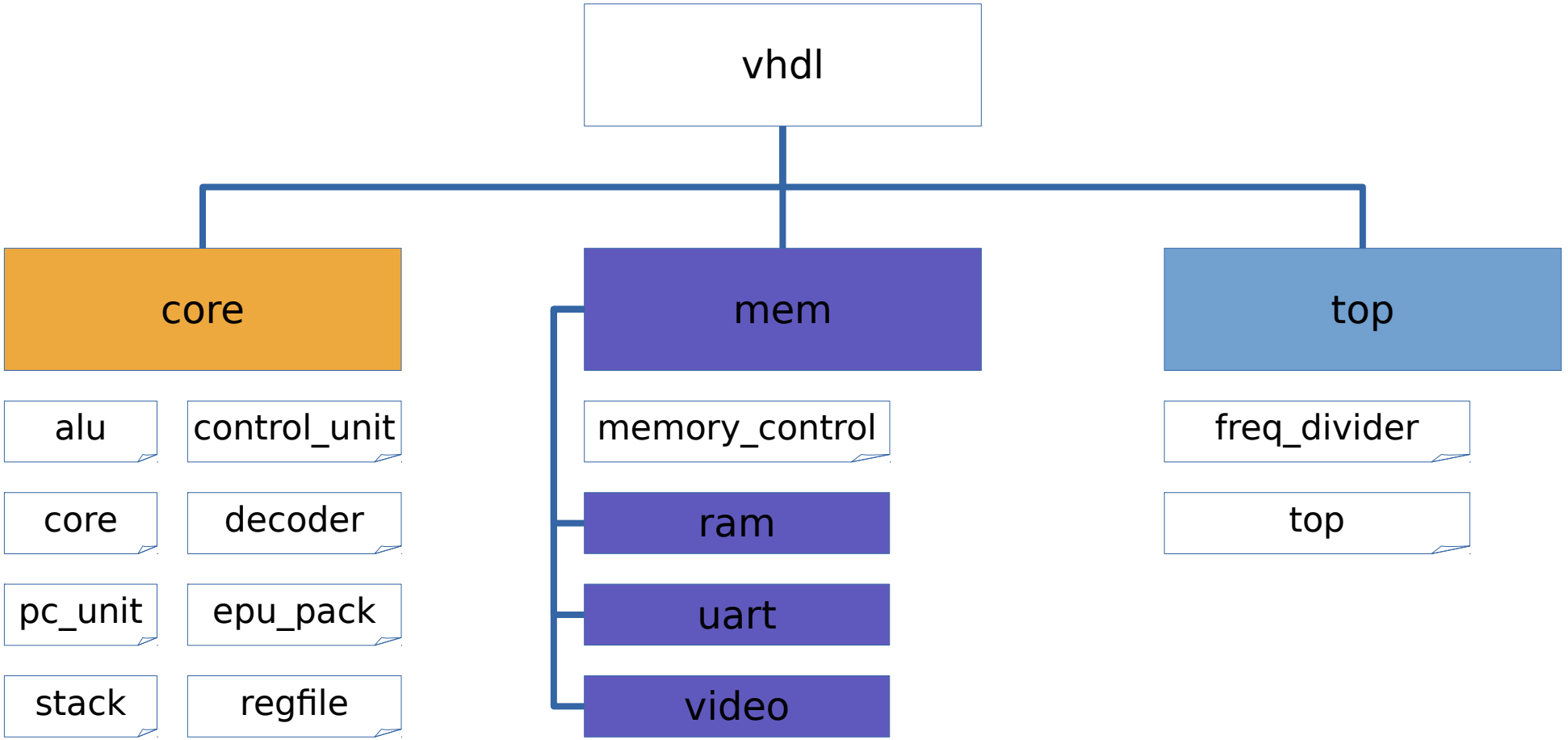


Quelle: http://www.chemgapedia.de/vsengine/media/vsc/de/ch/11/cmt/media/gif/fpga.gif

Hardwareübersicht



Umsetzung in VHDL



Umsetzung in VHDL

Definition des ‘core’-Moduls

```
entity core is
    port(
        -- Eingaenge
        I_CORE_Clk          : in std_logic;
        I_CORE_Reset        : in std_logic;

        I_MEM_Ready         : in std_logic;
        I_MEM_Data          : in std_logic_vector(7 downto 0);

        -- Ausgaenge
        O_CORE_HLT          : out std_logic;

        O_MEM_Reset         : out std_logic;
        O_MEM_En            : out std_logic;
        O_MEM_We            : out std_logic;
        O_MEM_Data          : out std_logic_vector(7 downto 0);
        O_MEM_Addr          : out std_logic_vector(15 downto 0);

        O_LED               : out std_logic_vector(7 downto 0)
    );
end core;
```

Umsetzung in VHDL

Deklaration des ‘core’-Moduls

```
architecture behav_core of core is
    -- Komponentendeklaration
    component pc_unit
        port(
            -- Eingaenge
            I_Clk       : in std_logic;                     -- Takteingang
            I_Op        : in std_logic_vector(1 downto 0);  -- PC-Operation
            I_PC        : in std_logic_vector(15 downto 0); -- Sprungadresse

            -- Ausgaenge
            O_PC        : out std_logic_vector(15 downto 0) -- Ausgabe
        );
    end component;
    [...]

    -- Signale
    signal State        : std_logic_vector(6 downto 0) := (others => '0');
    [...]
begin
```

Umsetzung in VHDL

Deklaration des ‘core’-Moduls

```
    [...]
    PCOp <= PC_OP_RESET when I_CORE_Reset = '1' else
        PC_OP_ASSIGN when SB = '1' and En_RegWrite = '1' else
        PC_OP_INC when En_Decode = '1' and DecDone = '0' else
        PC_OP_NOP;

    PCIn <= StackData when En_RegWrite = '1'
        and AluOp(IFO_REL_OPCODE_BEGIN downto IFO_REL_OPCODE_END) = OPCODE_RET
        else ADDR_INT when En_RegWrite = '1'
        and AluOp(IFO_REL_OPCODE_BEGIN downto IFO_REL_OPCODE_END) = OPCODE_INT
        else Res;

    uut_pc_unit : pc_unit port map (
        I_Clk => I_CORE_Clk,
        I_Op => PCOp,
        I_PC => PCIn,
        O_PC => PC
    );
end behav_core;
```

EPU-Assembly

Befehlsaufbau

$$Mnemonic[.Option] \ [Operand1][,Operand2][,Operand3]$$

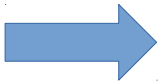
Beispiele

add.u r2, r3, r4	*R2 = R3 + R4*
call.i $test	*Rufe Funktion mit dem Label **test** auf.*
load r3, 0xDEAD	*R3 = 0xDEAD*

EPU-Assembly

Assembly

```
start:
        load r4, 0x41
        load r5, 0x90
        cmp r4, r5
        jge.i $start
```



Maschinencode

```
0000: 3B40 0041

0004: 3B50 0090

0008: 6AE4 50

000B: 7F4E 0000
```

Befehlsaufbau

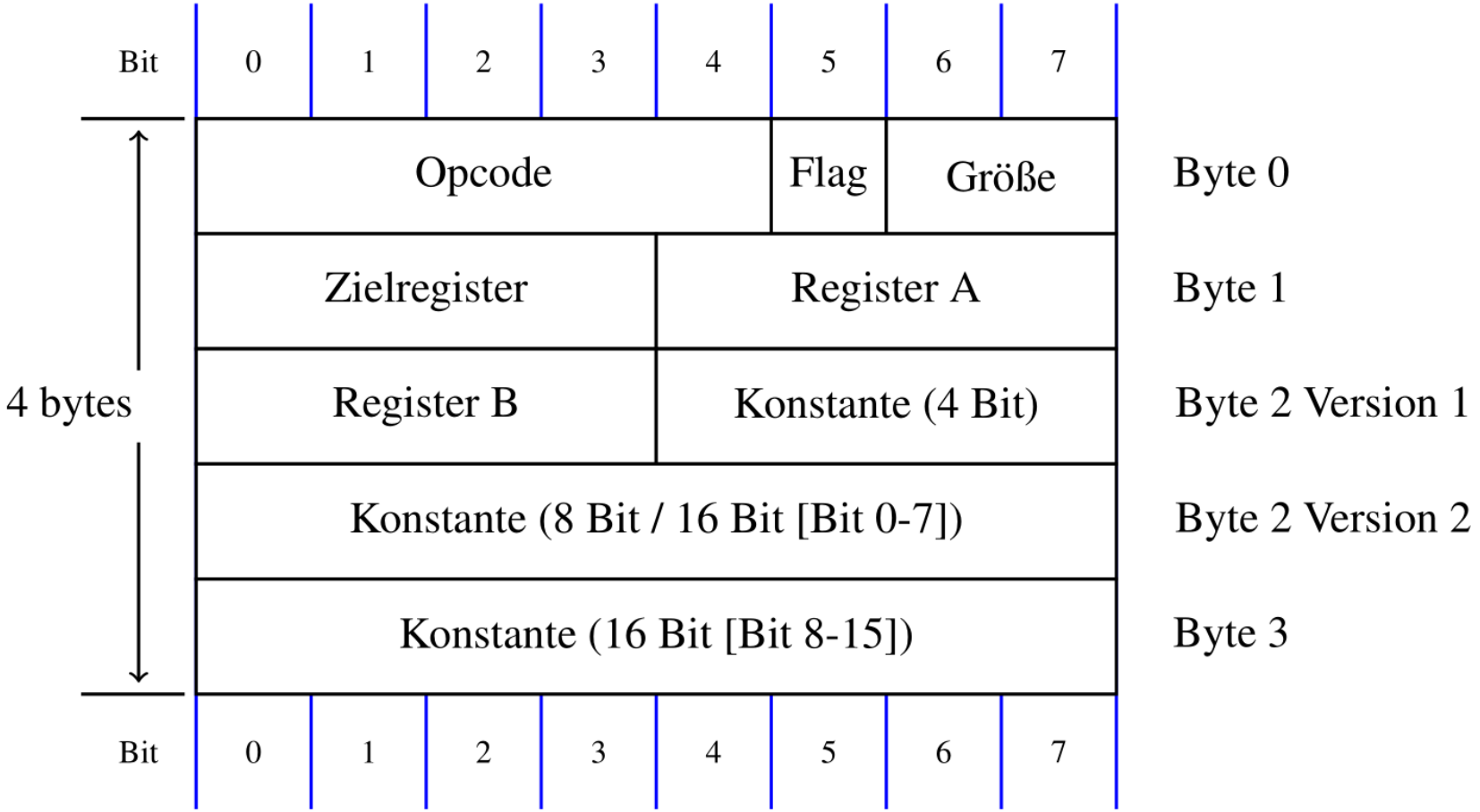
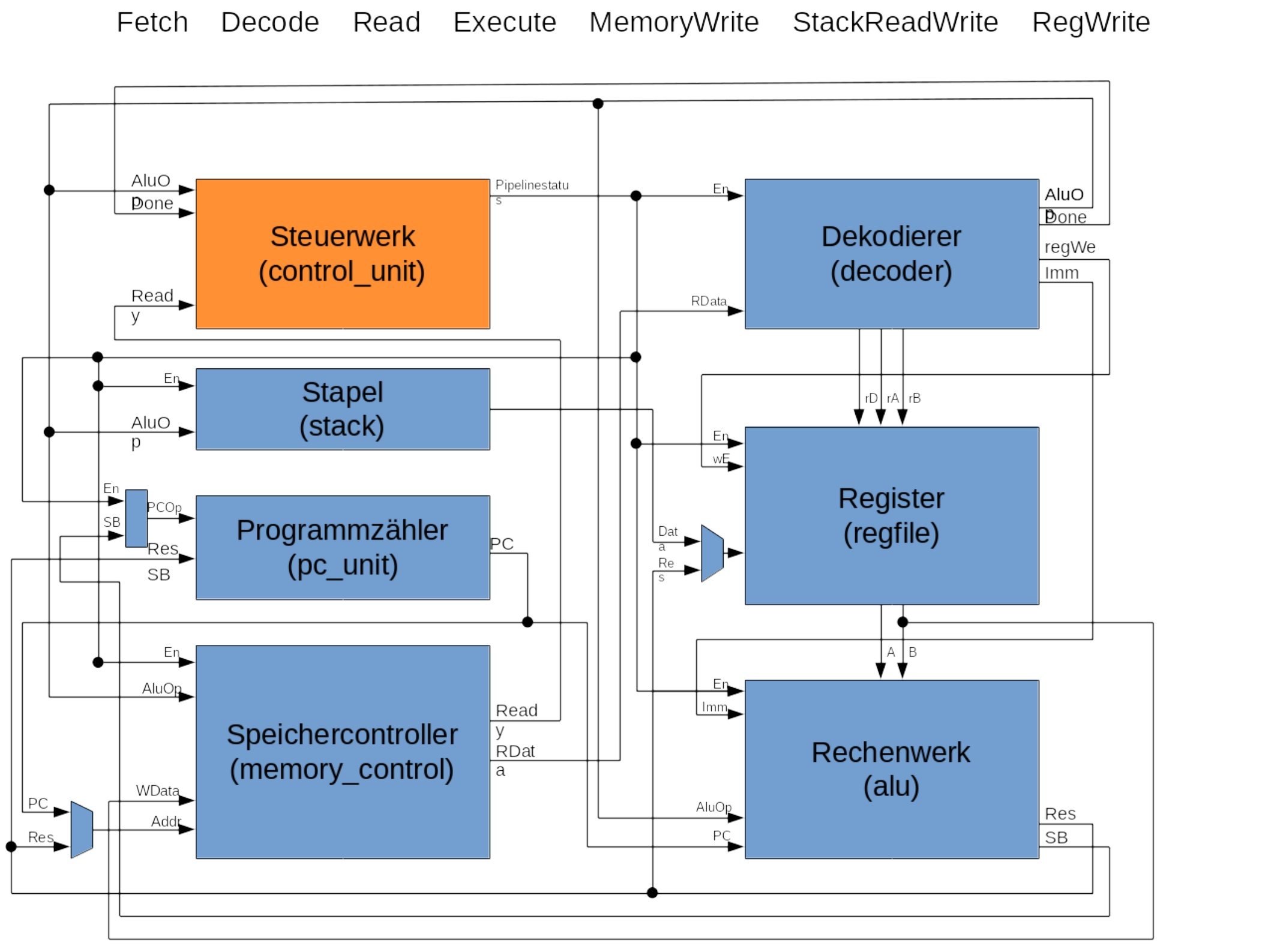
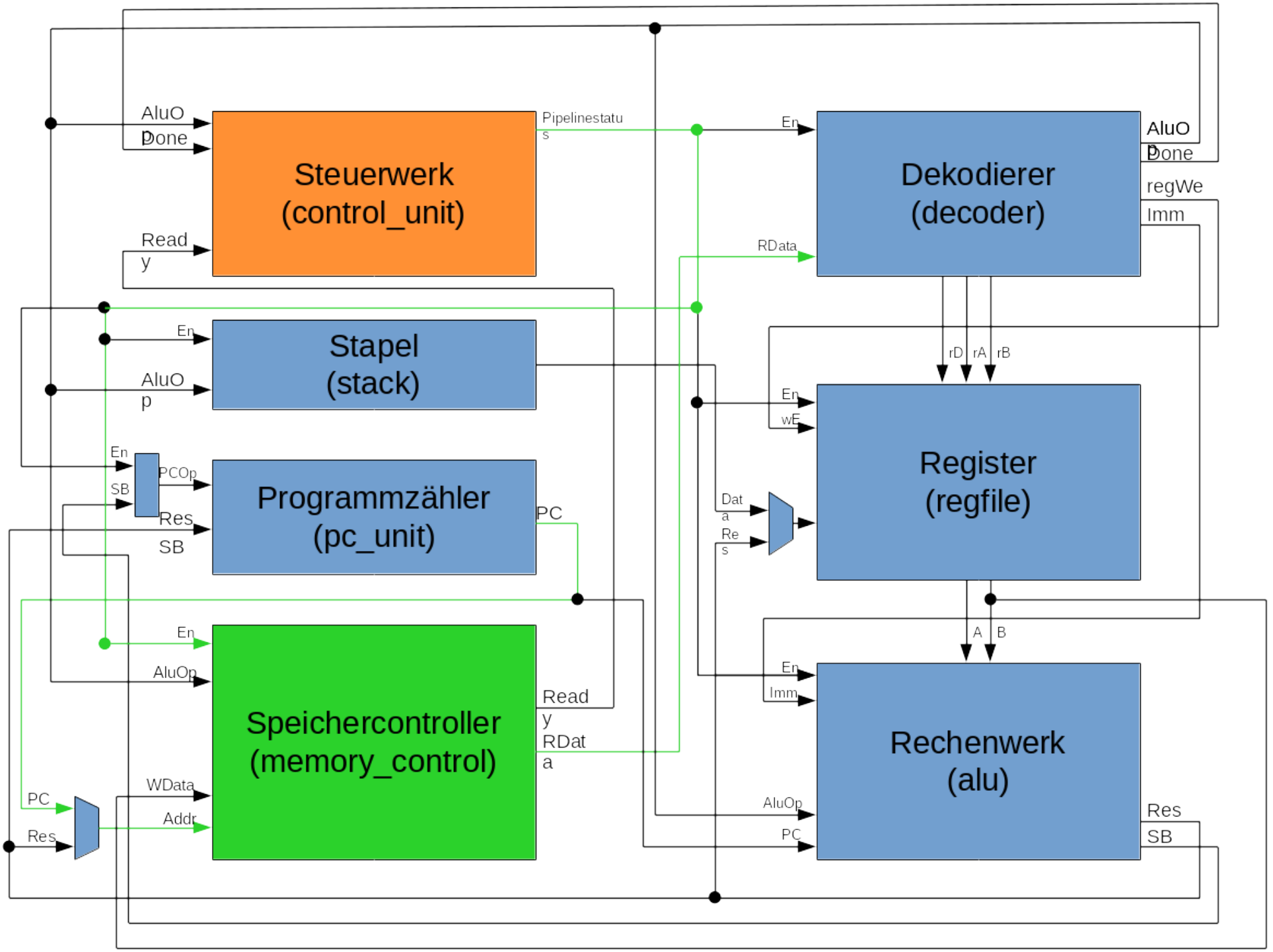


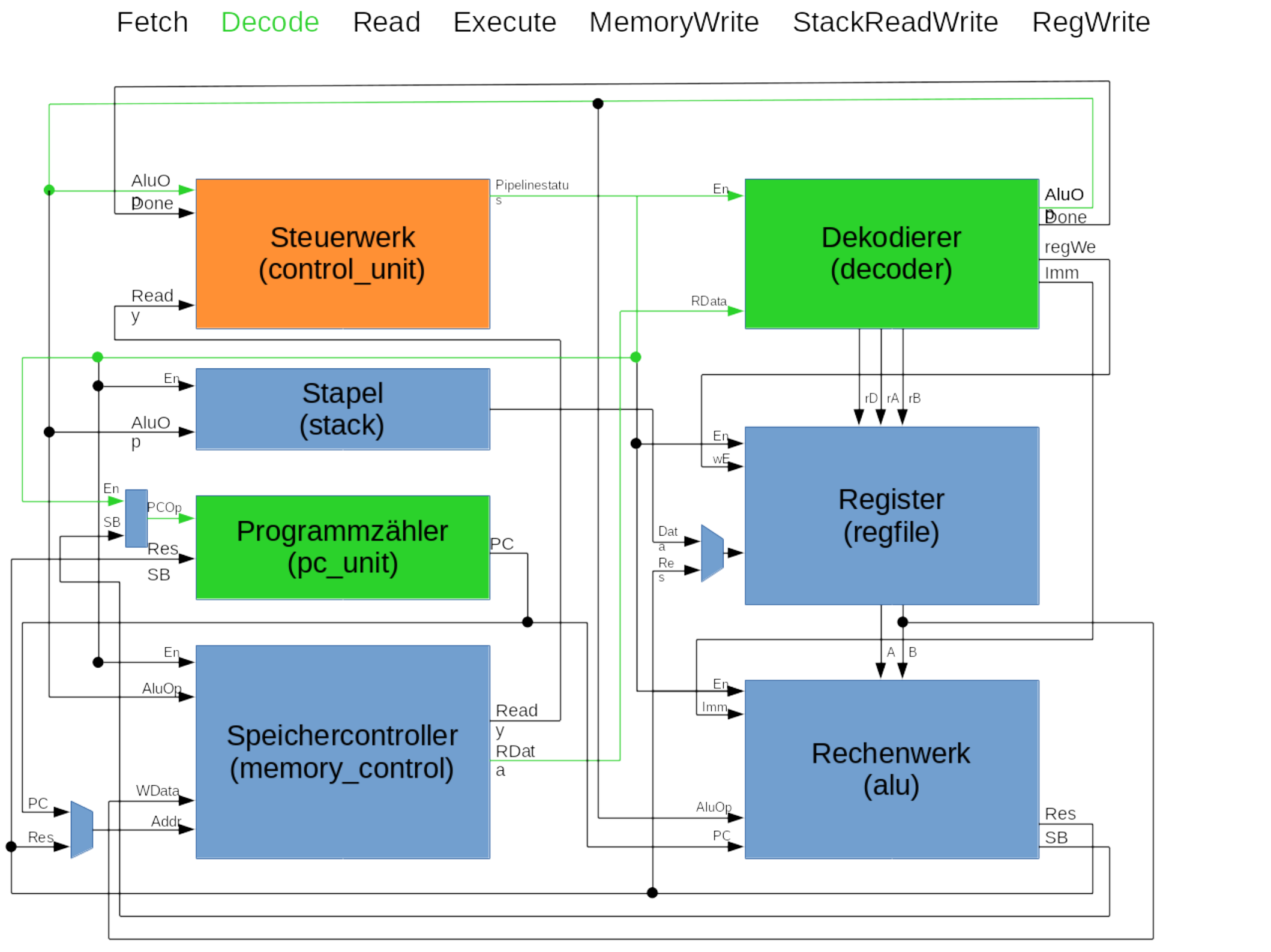
Abbildung 2.3.: Befehlsaufbau

Funktionsweise
add.u r2, r3, r4

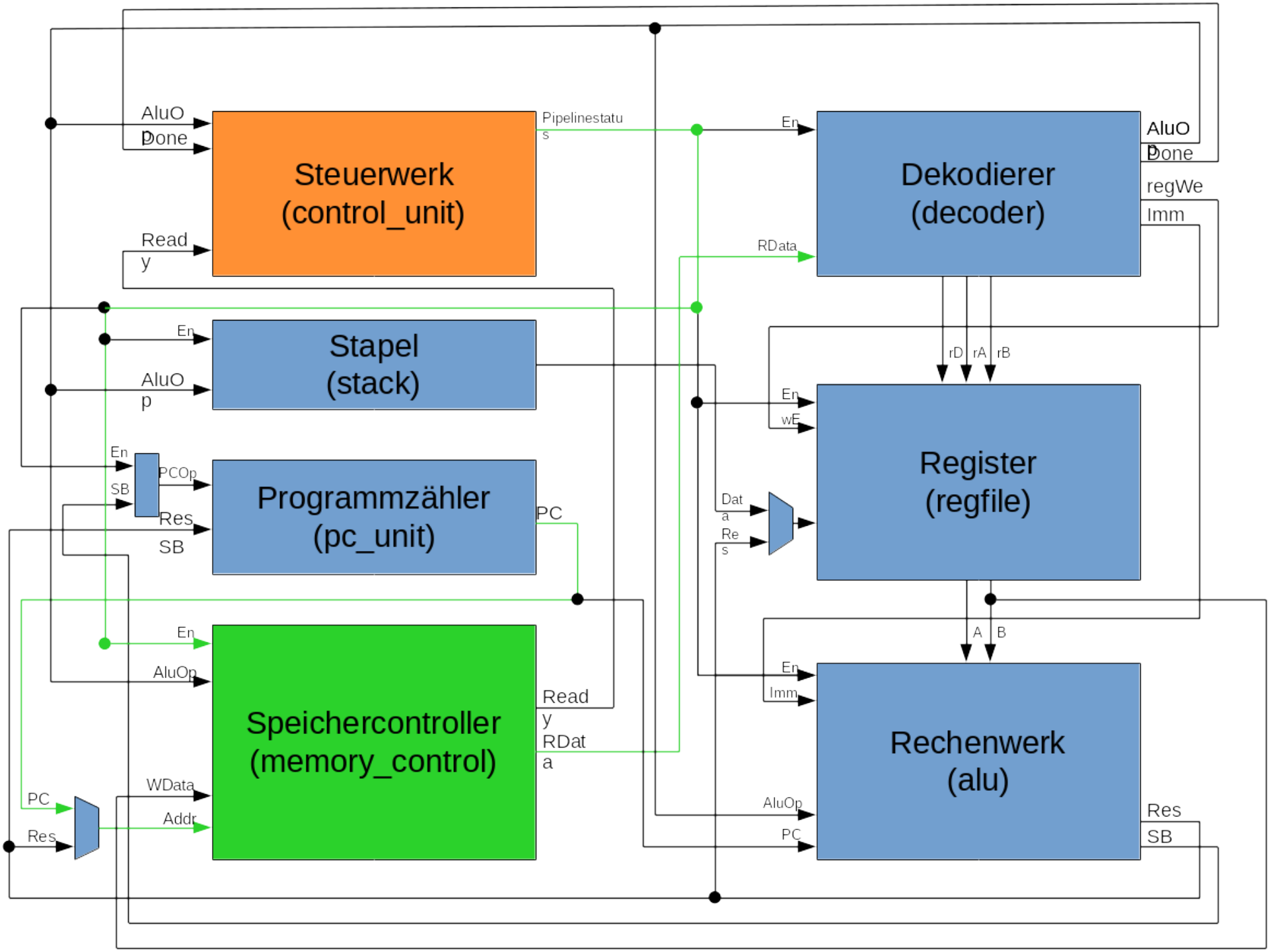

Fetch
Decode
Read
Execute
MemoryWrite
StackReadWrite
RegWrite
Steuerwerk
(control_unit)
AluOp
Done
Pipelinestatus
Ready
Dekodierer
(decoder)
En
AluOp
Done
regWe
Imm
RData
Stapel
(stack)
En
AluOp
rD
rA
rB
Register
(regfile)
En
wE
Programmzähler
(pc_unit)
En
SB
PCOp
Res
SB
PC
Data
Res
A
B
Speichercontroller
(memory_control)
En
AluOp
Read
Ready
RData
Rechenwerk
(alu)
En
Imm
WData
Addr
PC
Res
AluOp
PC
Res
SB

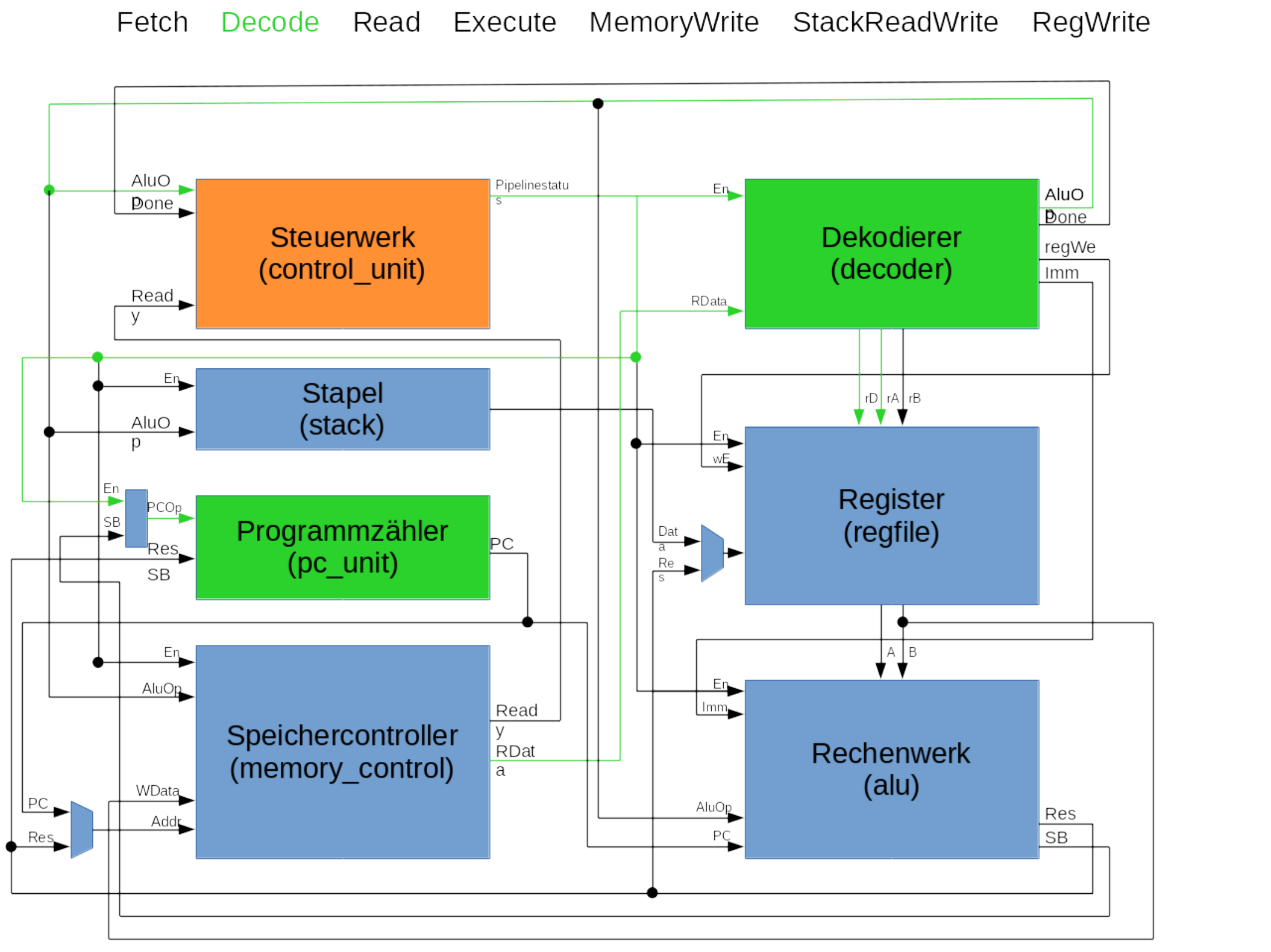
Fetch Decode Read Execute MemoryWrite StackReadWrite RegWrite

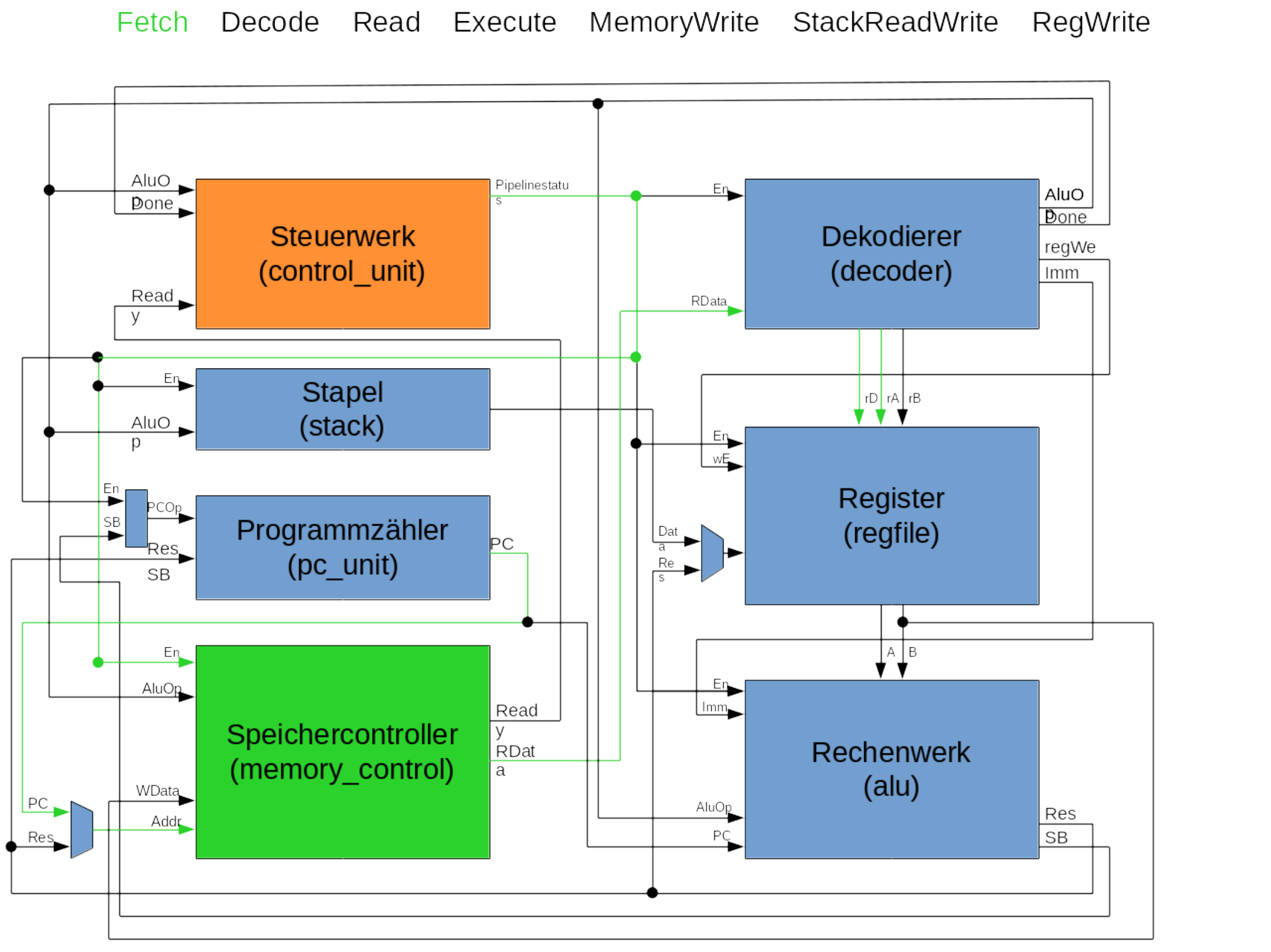



Fetch
Decode
Read
Execute
MemoryWrite
StackReadWrite
RegWrite
Steuerwerk
(control_unit)
AluOp
Done
Read
y
Pipelinestatus
Dekodierer
(decoder)
En
RData
AluOp
Done
regWe
Imm
Stapel
(stack)
En
AluOp
rD
rA
rB
Programmzähler
(pc_unit)
En
SB
PCOp
Res
SB
PC
Register
(regfile)
En
wE
Data
Res
A
B
Speichercontroller
(memory_control)
En
AluOp
Read
y
RData
WData
Addr
PC
Res
Rechenwerk
(alu)
En
Imm
AluOp
PC
Res
SB

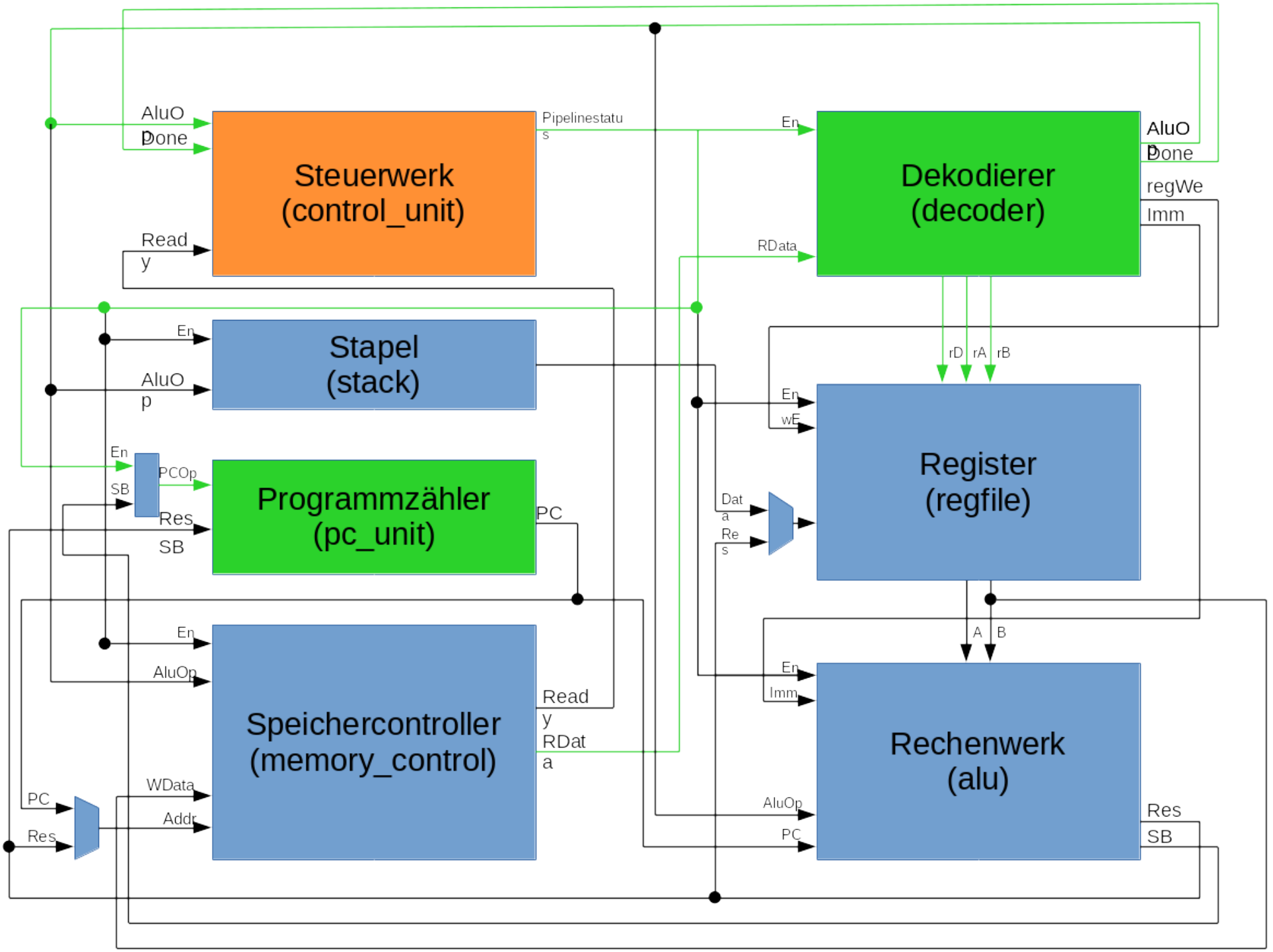
Fetch Decode Read Execute MemoryWrite StackReadWrite RegWrite



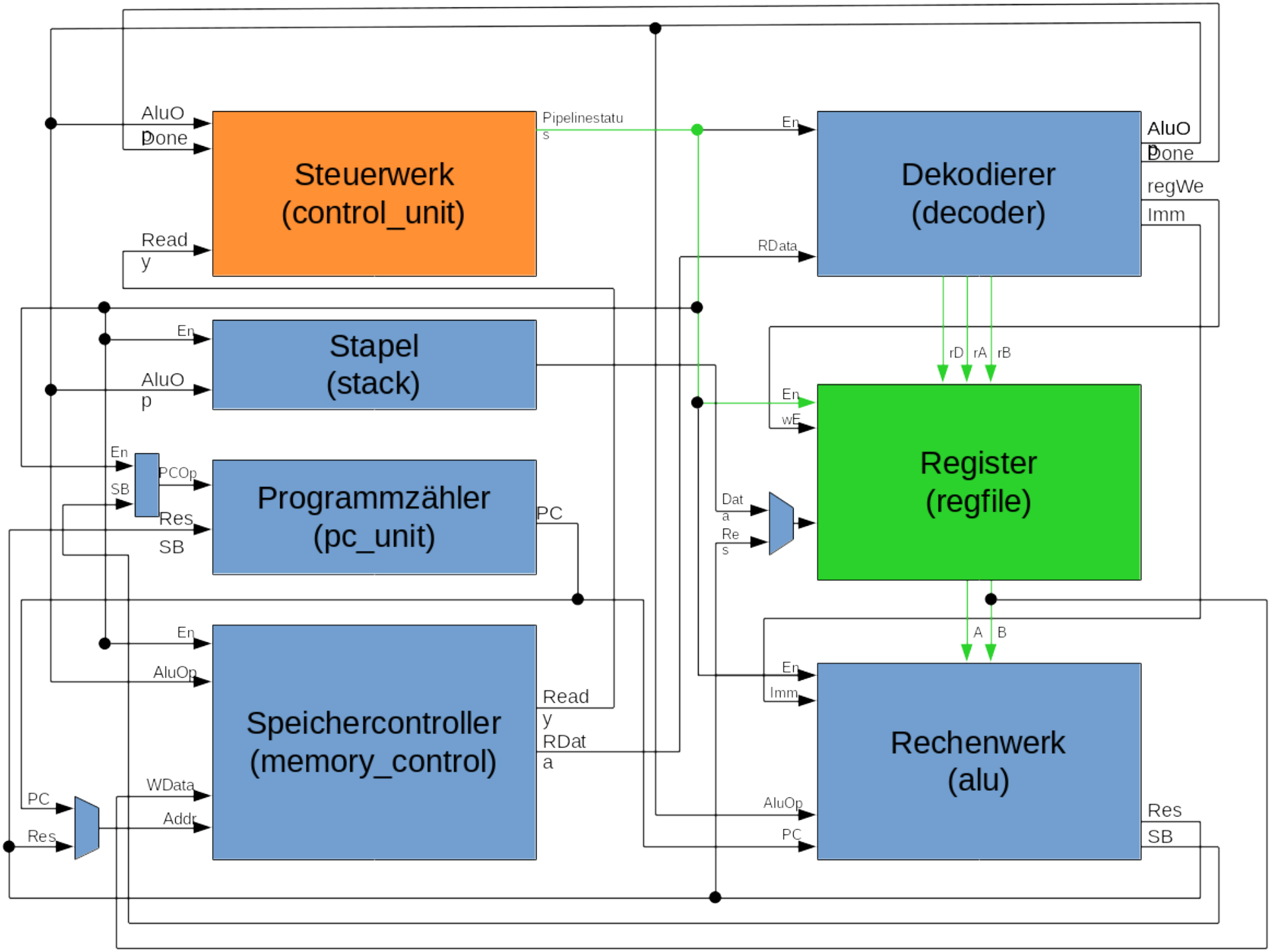

Fetch
Decode
Read
Execute
MemoryWrite
StackReadWrite
RegWrite
Steuerwerk
(control_unit)
AluOp
Done
Read
y
Pipelinestatus
Dekodierer
(decoder)
En
RData
AluOp
Done
regWe
Imm
Stapel
(stack)
En
AluOp
rD
rA
rB
Register
(regfile)
En
wE
Programmzähler
(pc_unit)
En
SB
PCOp
Res
SB
PC
Data
Res
A
B
Speichercontroller
(memory_control)
En
AluOp
Read
y
RData
Rechenwerk
(alu)
En
Imm
WData
Addr
PC
Res
AluOp
PC
Res
SB


Fetch
Decode
Read
Execute
MemoryWrite
StackReadWrite
RegWrite
Steuerwerk
(control_unit)
AluOp
Done
Read
y
Pipelinestatus
Dekodierer
(decoder)
En
RData
AluOp
Done
regWe
Imm
rD
rA
rB
Stapel
(stack)
En
AluOp
Programmzähler
(pc_unit)
En
SB
PCOp
Res
SB
PC
Register
(regfile)
En
wE
Data
Res
A
B
Speichercontroller
(memory_control)
En
AluOp
Read
y
RData
WData
Addr
PC
Res
Rechenwerk
(alu)
En
Imm
AluOp
PC
Res
SB

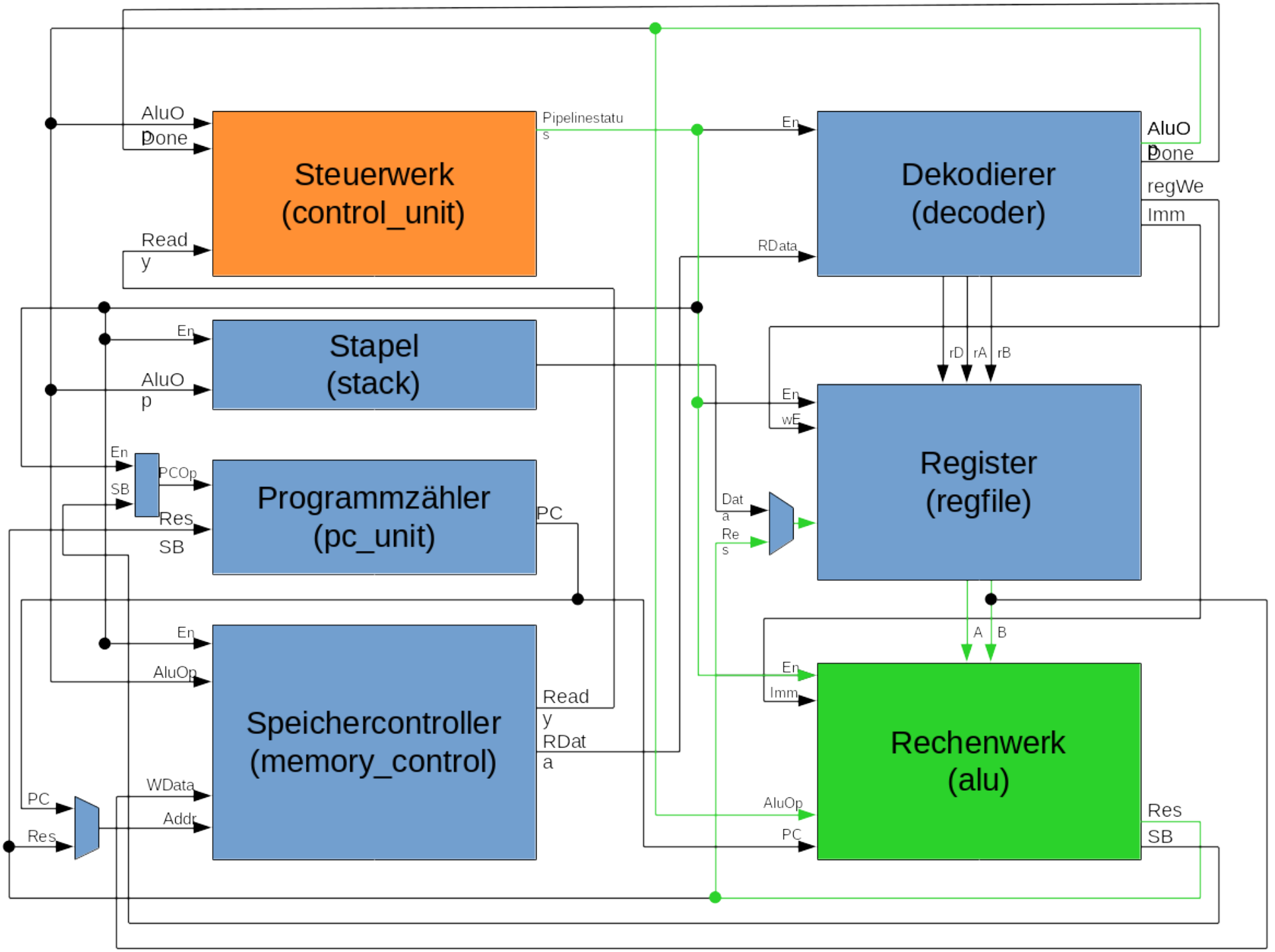
Fetch Decode Read Execute MemoryWrite StackReadWrite RegWrite



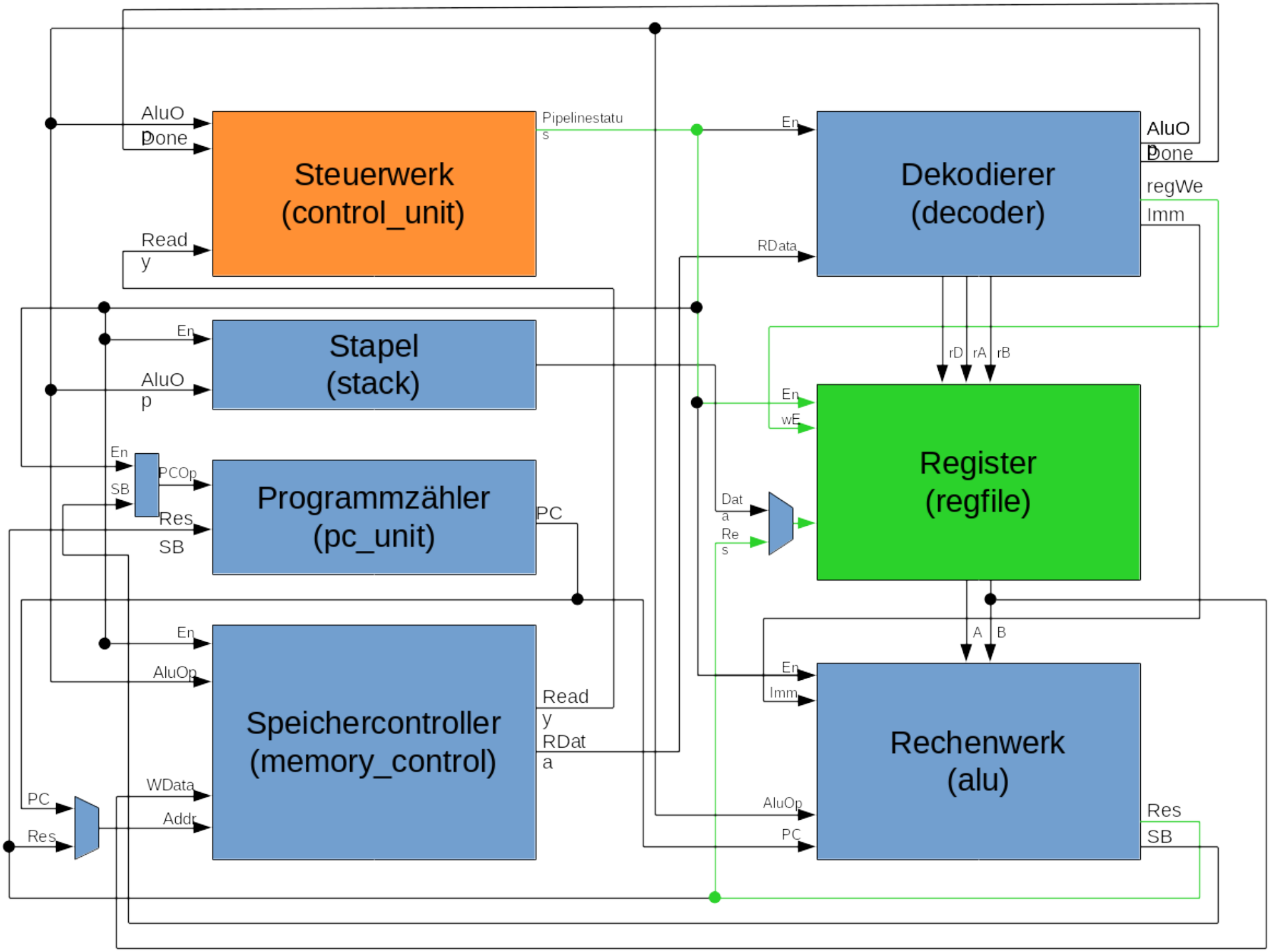
Fetch Decode Read Execute MemoryWrite StackReadWrite RegWrite



Fetch Decode Read Execute MemoryWrite StackReadWrite RegWrite



Fetch Decode Read Execute MemoryWrite StackReadWrite RegWrite



Demo

Vielen Dank für ihre Aufmerksamkeit