

Workflow für Von-Neumann-Architektur in VHDL

Kevin Grygosch, 1530895 Kevin Höfle, 1615343

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [Straße Hausnummer] [PLZ Ort] |  | T.[Telefon]  F.[Fax] |  | [E-Mail-Adresse]  [Internetadresse] |

Inhaltsverzeichnis

1. [Zusammenfassung 2](#_Toc447794794)
2. [Aufgabenbeschreibung 2](#_Toc447794795)

[Speichwerk](#_Toc447794796)

[Speichwerk](#_Toc447794797)

[\_Toc447794798](#_Toc447794798)

# Zusammenfassung

# Aufgabenbeschreibung

Spezifikationen des Prozessors:

Datenbus: 16 Bit  
Adressbus: 8 Bit

**Befehlsausführung im Leitwerk:**

1. Befehl aus dem Speicher ins Befehlsregister laden (FETCH CYCLE)
2. Befehlsentschlüsselung (DECODE CYCLE)
3. Laden der Operanden (Operand FETCH)
4. Befehlsausführung (Execution Cycle)
5. Rückschreiben der Ergebnisse (Write Back)

I

II

III

IV

IV

## Speichwerk

Da der Speicher essentiell ist zur Programmierung einer Von-Neumann-Architektur wird dies der erste Punkt sein.

Der Speicher besteht aus:

1. Speicher-Adress-Register (SAR)
2. Adress-Decoder 1:N
3. Speicherregister, in Worten aufgeteilt
4. Steuerlogik
5. Leseverstärker (Tri-State)

Er besitzt folgende Schnittstellen:

1. Anbindung an den Adressbus (über SAR)
2. Anbindung an den Datenbus (über Leseverstärker)
3. Chip-Select Eingang
4. WRITE-Enable

TODO: Bild einfügen (Skript Seite 10)

## Speichwerk

Da der Fetch Cycle (I) direkt das Befehlsregister benötigt, welches sich im Leitwerk befindet wird jenes im nächsten Arbeitsabschnitt realisiert.

#### Befehlsregister

Das Befehlsregister setzt sich aus den folgenden Teilen zusammen:

1. OP-CODE
2. HIGH-BYTE
3. LOW-BYTE

Das Register ist 16 Bit lang, entsprechend der Spezifikationen des Datenbusses.

Dabei muss die Codierung des OP-Codes intelligent gewählt werden um eine möglichst einfache Identifikation des auszuführenden Befehls zu erlangen. Die Länge des OP-Codes beträgt 8 Bit, 1 Byte – und bietet daher 256 mögliche Befehle, die jedoch in diesem Setup nicht ausgenutzt werden.

##### Befehlssatz

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Code | Klasse | Erklärung |
| AND | 0x80 | Logisch |  |
| OR | 0x81 | Logisch |  |
| XOR | 0x82 | Logisch |  |
| NOT | 0x03 | Logisch |  |
| ADD | 0xA0 | Arithmetisch |  |
| SHIFT | 0x21 | Arithmetisch |  |
| MOV | 0xC0 | Sprung |  |
| JMP | 0x41 | Sprung |  |
| NOP | 0x60 , 0xE0 | Speziell |  |
| END | 0x61, 0xE1 | Speziell |  |

## 

Abbildung 2‑1 Op-Code-Codierung

7

6

5

4

3

2

1

0

00: Logische Operationen

01: Arithmetische Operationen

10: Sprungbefehle  
11: Speziell

0: Benötigt 1 Operand

1: Benötigt 2 Operanden

Eindeutige Identifikation des Befehles innerhalb der Gruppierung

Undefiniert, noch nicht in Verwendung. daher immer „00“

#### Befehlsdecoder

Der Befehlsdecoder verwendet die Informationen Abbildung 2‑1, dies wird vermutlich über eine LUT realisiert.   
Eventuell kann jedoch eine intelligente Schaltung die sich signifikante Bits zuerst anschaut realisiert werden. Eine Möglichkeit wäre hierzu das siebte Bit. Falls dies gesetzt ist, müssen schnellst möglich die Operanden geladen werden, da zwei davon benötigt werden. Weiterhin gibt das **Steuerwerk** essentielle Informationen weiter an die ALU des Rechenwerks.

Hiermit ist der Zyklus bei Schritt III angekommen, der FETCH- und DECODE-Cycle wurde somit abgeschlossen.  
Es folgt somit der OPERAND FETCH.

Offene Fragen:

- Input / Output wird wie realisiert ? 🡪 wird das Program in den Speicher geschrieben? Oder reicht es einen einzelnen Befehl auszuführen im Terminal. 🡪 Fragen: Vettermann  
 🡪 Speicher komplett abhängig 🡪 würde dann keine richtige Von-Neumann-Architektur sein  
 🡪 Befehlszähler wird redundant wenn nur ein Befehl eingelesen wird (?)

MUX im Leitwerk 🡪 wird benötigt um zu unterscheiden ob Programm oder Daten auf Datenbus geladen werden ?

Befehlsregister 🡪 hat 16 bit ? (=Adressleitung Breite) 🡪 Op-Code mit 8 Bit, Low-Byte: 4 Bit, High-Byte: 4 Bit 🡪 Bei Befehlen mit 2 Operanden wird Low-Byte und High-Byte jeweils verwendet für die Operanden ? 🡪 Kann nicht sein, da Datenbuss 8 Bit breit