TI2 9.5.12

weiter zu F3_10

Blöcke (mit Aufgaben)

- Prozessor
  - **Verarbeitung von Operanden in Abhängigkeit von Maschinenbefehlen**
  - Auswahl und Lesen von Maschinenbefehlen und Auswahl der aktuellen Maschinenbefehle (Steuerung der Befehsreihenfolge, Maschinenbefehl ist ein Befehl, den der Prozessor direkt abarbeiten kann)
  - Auswahl und Lesen und Schreiben von Operanden (in den Maschinendatentypen)
- Programmspeicher (Befehlsspeicher, Program Memory)
  - physischer Träger der Maschinenbefehle
  - Lesen von ausgewählten Maschinenbefehlen (Lesen: Speicher nach Prozessor)
- Datenspeicher (Data Memory)
  - physischer Träger der Operanden
  - Lesen und Schreiben von ausgewählten Operanden (Prozessor nach Speicher, Schreiben)
- Ein-/Ausgabe (EA, Input/Output, IO)
  - physische Schnittstelle für Eingangs- und Ausgangsoperanden
  - Lesen und Schreiben von ausgewählten EA-Schnittstellen

Auswahl erfolgt durch Adressierung (Grundprinzip F3_20)

Adressen sind vorzeichenlose ganze Zahlen (0 bis ((2 hoch m)-1))
m typ 16, 32, 64, 128

Adressen werden eindeutig auf Dateneinträge abgebildet (Maschinenbefehle bzw. Operanden in möglichen Maschinendatenformaten), sind evtl zeitabhängig auf den Aressen unterschiedlich (durch Schreiben)
F3_20 Bsp. Lesen: Prozessor gibt 2i (Binärzahl), Speicher bzw. EA ordnet den Dateneintrag zu und gibt ihn zum Prozessor

Verbindungsstruktur

- bei Harvardarchitekturvariante
  Blöcke Befehlspeicher, Datenspeicher und EA haben unterschiedliche Verbindungen zum Prozessor
- bei Princetonarchitektur
  Blöcke BS, DS und EA haben eine gemeinsame Verbindung zum Prozessor (Systembus)

Vorteile (entspricht einem Nachteil der anderen Architektur negiert):

- Harward: bei Befehlen die einen gleichzeitigen Zugriff (Oberbegriff zu Lesen und Schreiben) besitzen bis zu 3 mal schneller (auf unterschiedliche Blöcke)
- Priceton: einfachere Verbindungsschnittstelle nach außen (-> Systembus)

Bus: öffentliches Nahverkehrsmittel

Bus (im Rechner): öffentliches (verschiedene Teilnehmer) Nah- (rechnerintern) verkehrsmittel (Transportmittel für Befehle und Operanden), mit verschieden Haltestellen (Speicher- bzw. EA-

Schnittstelle, Prozessorschnittstelle), zeitgeteile Nutzung (typ. genau zu einem Zeitpunkt ein Befehl bzw. ein Operand)

➔ Taxi (Fahrer ist Busverwaltung, Ziel ist die ausgewählte Schnittstelle, Zielvorgaben erzeugt nur der Prozessor)

F3_30:

Ziel- (bzw. Quellvorgabe): durch Aussenden einer Adresse (s.o.) vom Prozessor auf den Adressbus) (unidirektional: Pr.->SP/EA)

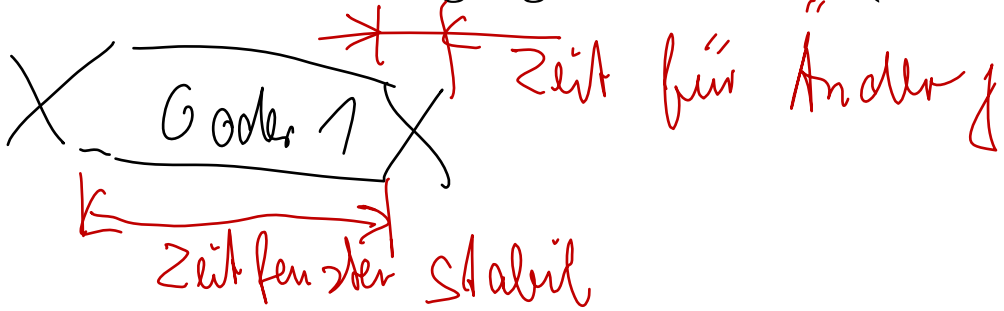
Datentransfer: Austausch eines Operanden über den Datenbus (s.o.) (vom Pr. auf den D-Bus (Schreiben) bzw. vom Sp. zum D-Bus) (bidirektional (Pr.->S exclusive or S->Pr. zu einem Zeitpunkt)

Steuerung: Einzelsignale zur zeitlich-logischen Steuerung des Datentrasfers (Speicher-Lesen als Binärsignal (Memory Read, MRD)

F3_40 zwei Darstellungsarten von (parallelen Busse, werden inTI2 zumeist verwendet (seriell Busse werden explizit benannt)

oben: Einzelsignaldarstellung:

i.a. ist die konkrete Belegung nicht bekannt (mit 0 oder 1)



Signale sind eine bestimmte Zeit stabil (d. h. ändern sich nicht): Zeitfenster

Zusammenfassung von einzelnen Bussignalen zum Gesamtbus (m=n) Signale, DB gesamt)

3.1. Prozessorgrundstruktur

➔ F3_50, simulierbar im elearnig (Vorführung im folgenden)

ALE (Arithmetik-Logik-Einheit, ALU, Arithmetic Logic Unit)

- ➔ Verknüpfung von bis zu 2 Eingängen zu einem Ausgang (Eingangsoperanden auf Ausgangsoperand)
- ➔ kombinatorische Logik (siehe TI1)
- ➔ Operation wird durch Maschinenbefehl eingestellt („Maschine“ : Rechnerhardware)
- ➔ Operationen sind im Befehlssatz festgelegt
- ➔ Status der Operation ist zusätzlicher Ausgang, gebildet in Abhängigkeit der Operation

Register in der Umgebung der ALE:

OR1, 2: Operandenregister 1 und 2:

- ➔ vor der Operation die bis zu 2 E-Operanden
- ➔ nach der Operation: OR1: Ergebnis (Akkumulatorprinzip), OR2 unverändert

PSR: Prozessorstatusregister:

- ➔ nach Operation deren Status
- ➔ typ. bitweise organisiert,

bits :2 verschiedene Zahlenüberläufe (siehe auch TI1 und Arbeitsblätter) cy (Carry-Flag) und o (Overflow-Flag) Nullergebnis z oder Zero-Flag, s (Sign-Flag)

Zahlenüberlauf zur evtl. Korrektur
z, s:

z=1 -> bei Vergleich E-Op1 = E-Op2
s=1 -> positives Ergebnis E-Op1 größer gleich E-Op2
s=0 -> negatives Ergebnis E-Op1 kleicher E-Op2

es ex. weitere Flags (PSR-bits) Prozessorspezifisch (siehe Bsp. Arbeitsblätter)

Register für Befehlsbehandlung:
BA: Befehlsadressregister: Adresse des aktuellen Befehls
BR: Maschinekode des aktuellen Befehls

Maschinenkode -> Abbildung der Maschinenbefehle auf eine Menge von Binärkombinationen im Maschinenbefehlsformat (z.B. 32 bit), eineindeutig (evtl. nicht alle Binärkombinationen in Benutzung)

fiktives Bsp.:

ADD 0000..00
SUB 0000..01
MUL 0000..10
DIV   0000..11
…

BD: Befehlsdekoder:
dekodiert aus dem Maschinenkode die für die Ausführung notwendigen Steuersignale zur ALE und zur AST (Befehlsvariante bei AST), kombinatorische Logik (siehe TI1)

AST: Ablaufsteuerung (sequentielle Logik), steuert logisch zeitlichen Ablauf der Befehlsvariante entsprechend Maschinenbefehl, Verbindung zu allen anderen Blöcken

OA: Operandenadressregister, enthält die Adresse eines aktuellen Operanden

AT, DT, ST: Treiber für die einzelnen Signale
- DT: bidirektional
- AT: unidirektional
- ST: unidirektional, aber je nach Signal unterschiedliche Richtung

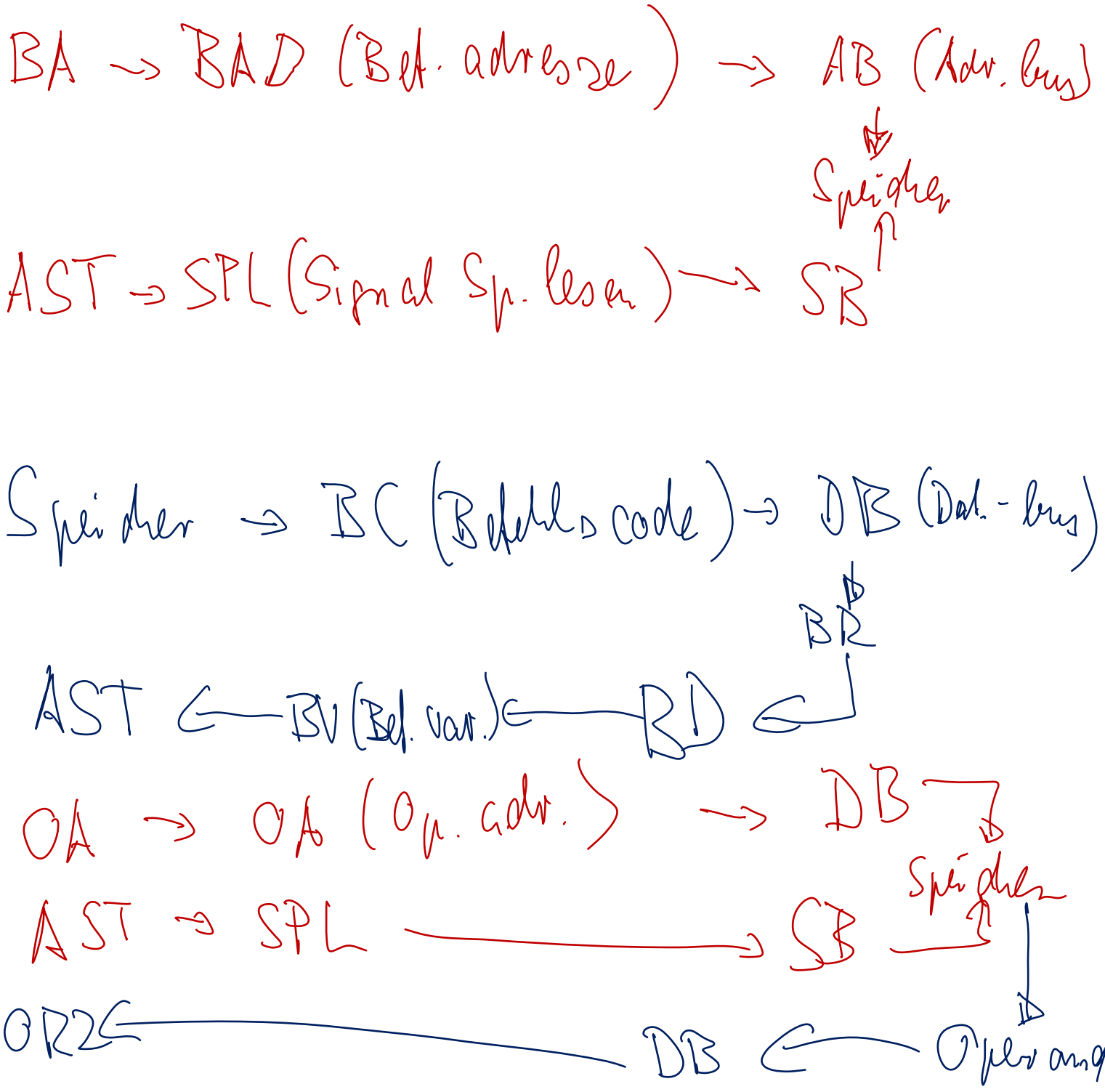
Zusammenwirken der Blöcke der Grundstruktur (siehe elearnig)

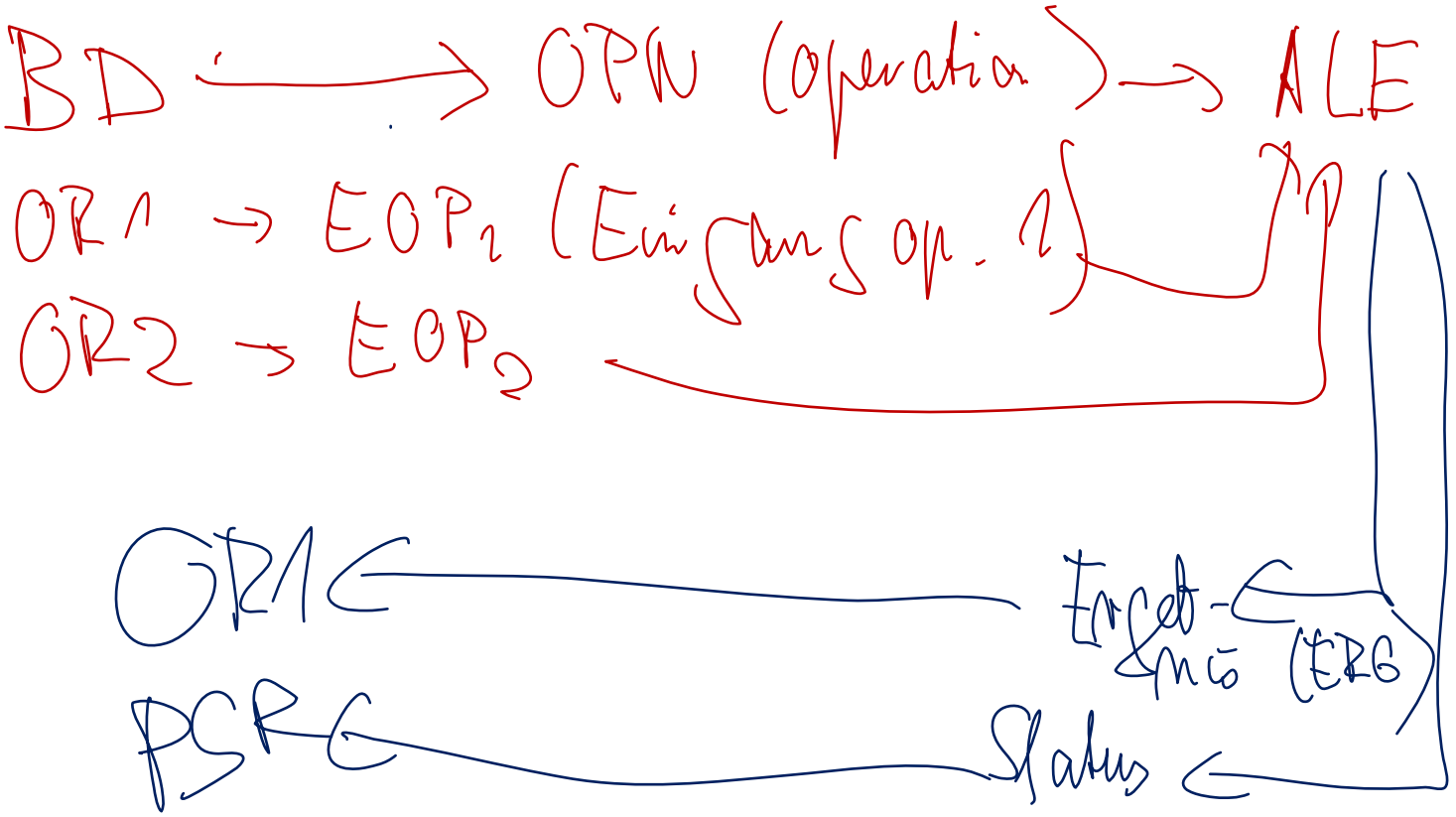
Befehlablauf (links Grundstruktur als Blöcke (F3_50), rechst PN für Ablauf (F3_60))

Bsp. Subtraktion: hier Befehlscode 3 dez bzw. 000…11 binär

OR1: 1. E-Op. = 16dez.
OA: Adr. des 2. E-Op. im Speicher 100H
BA: Adr. des akt. Befehl (SUB) 200H

Ablauf (PN, Struktur)





Als Vorbereitung für den nächsten Befehl folgt Erzeugen der nächsten Befehlsadr.

da Verarbeitungsbefehl: Adresse auf den im Speicher nächsten folgenden Befehl eingestellt

3.2. Erweiterungen zur Grundstruktur

Struktur nach F3_50: minimal, kann sehr eingeschränkt nur arbeiten

➔ Erweiterungen dieser Struktur

- mehrere Operandenregister
  Ziel: möglichst viele Operanden im Prozessor und nicht von außen lesen/schreiben -> Operationen schneller
  OR1 bis n (n z.B. 16 bis 256, Reg. werden im Befehl adressiert) wahlweise als E-OP1 oder 2 oder Ergebnis, auch mehrfach das gleiche Reg. mgl.

Erweiterte Strukturen in der Adressierung

- 3_80/3_90:

Basisadressierung

Grundidee:Typ. sind gleichzeitig im Prozessor mehrere Teilprogramme (TP) aktiv.
Die Teilprogramme brauchen zumeist unterschiedliche Operanden.

➔ Behandlung der Operandenbereiche unterschiedlicher TP separat (d.h. in getrennten Speicherbereichen) zum gegenseitigen Schutz, i.a. kürzere Adressen mgl. und beim Programmladen sind die Speicherbereich im Hauptspeicher lokalisierbar

Speicherbereich für ein TP ein Segment (Programm als auch Daten i.a. getrennt)

➔ segmentierte Adressierung

Realisierung im Prozessor durch „segmentierte Adressierung“ (3_80, links)

- zwei Adressregister je Operandenadresse, Gesamtadresse:= Basis-OA + Offset-OA

TP-Segment für Operanden hat eine Basisadresse, die beim Laden festgelegt wird (oder bei kleineren Systemen bei der Programmentwicklung festgelegt), für die Dauer der ununterbrochenen (bis zum nächsten Laden) Abarbeitung konstant.

Im Segment adressiert das TP über die variable Offsetadresse.

Segmentierte Adr. heißt auch Basisadressierung.

➔ Indexadressierung

Grundidee: Gedacht für Datensprukturen, die aus mehreren Elementen gleicher Länge bestehen, auf die zumeist sequentiell zugegriffen wird (d.h. auf Element i folgt Element i+1 usw.).
Bsp. Vektoren, Matritzen, Bilder, Textfelder, …

effektiver Zugriff (schnell, sicher) durch Unterstützung im Prozessor:

- Indexbasis enthält Adresse von erstes oder letztes Element der Datenstruktur, Indexoffset ist der aktuelle Abstand zu Randelement (n*i, n Länge eines Element im Speicher, i Abstand des aktuellen Elements zum Randelement)
- Beim Zugriff auf das aktuelle Element wird nach Zugriff der Offset um n erhöht bzw. verringert -> Adresse des nächsten Elements nach oben oder unten

F_3_80
Prizipiell wie segmentierte Operandenadressierung ist segmentierte Befehlsadressierung möglich

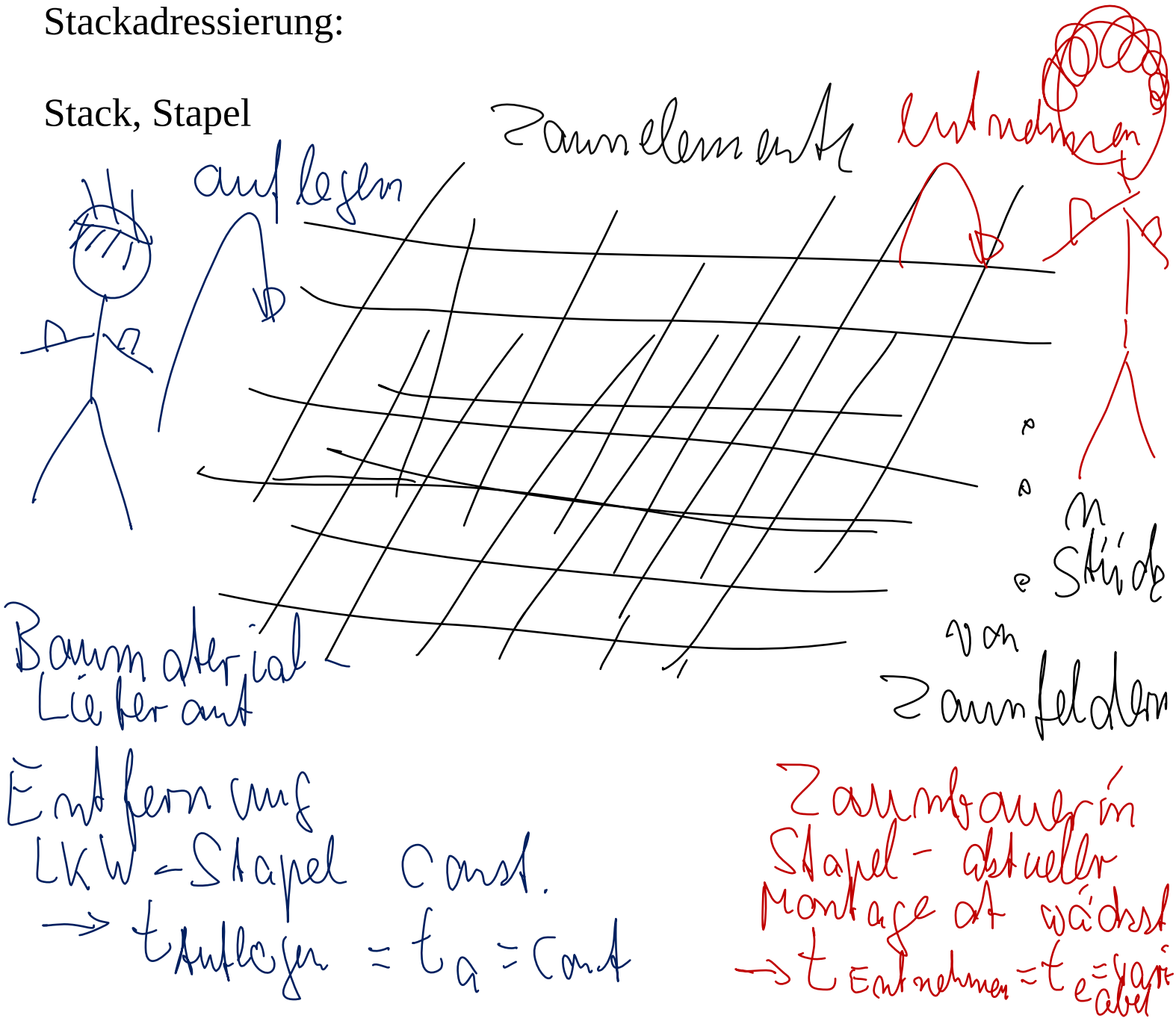
➔ F_3_100, F_3_90 hier DM durch BW (Befehlswort ersetzen)

Wechsel zwischen Teilprogrammen:

Basis-OA und Basis-BA neu laden,
Offset-OA und Offset-BA vom letzten mal übernehmen (falls TP-Fortsetzung, sonst auf Start-OA laden)

Stackadressierung:

Stack, Stapel



Strategie: LIFO (last in first out), d.h. Es wird immer das letzte aufgelegte Element entnommen.

Sinngemäße Übertragung auf den Speicher:

Unterstützung durch Adressierung: spezielles OA-Register, bezeichnet mit SP (Stack Pointer, Stapel-Zeiger)

mit der Adresse aus SP Lesen: danach Adresse inkrementieren (+1) -> nächnstes Lesen vom Folgeelement

mit der Adresse beim Schreiben SP erst dekrementieren, danach Schreiben

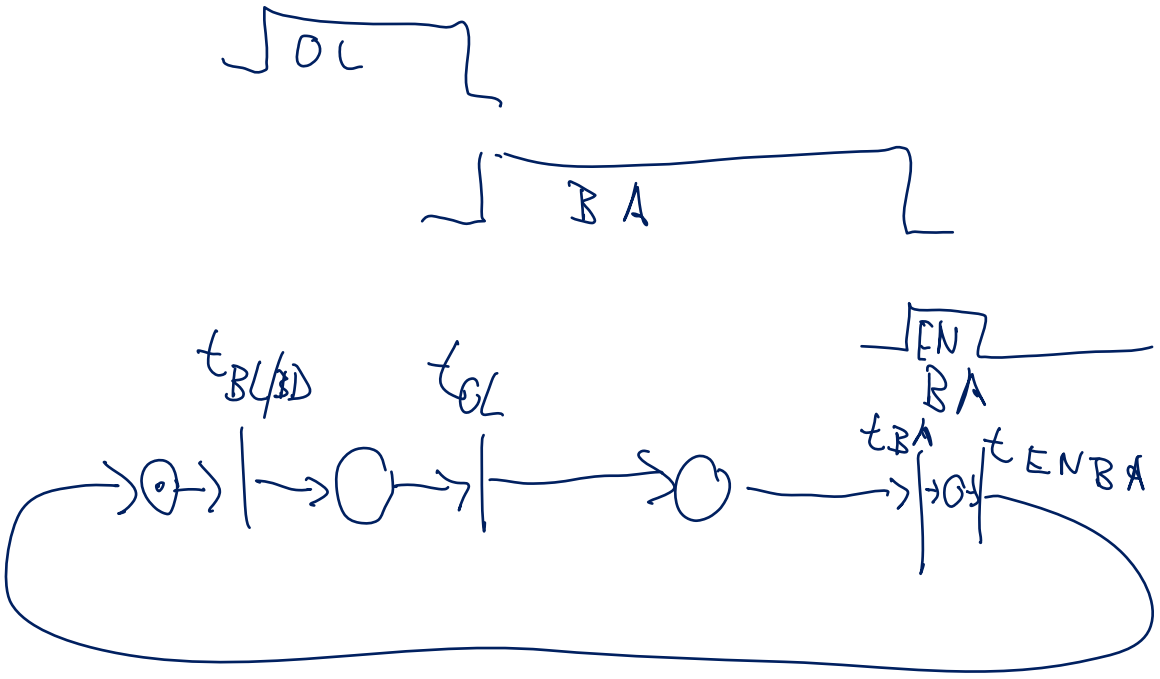
Folge zum Schreib-Lesebeispiel in F_3_110: Schreiben: erster Wert, zweiter Wert, danach Lesen: zweiter Wert, erster Wert (Last in First out)

Mehrere ALE’s

Möglicher Zeitverlauf von PN in 3_60 (Bsp. Befehlablauf)

Zeit: 

Befehls-Phasen:



Zeit verbraucht Befehl (Bsp. bef.)

$t_{ges} = t_{B/BD} + t_{OL} + t_{BA} + t_{ENBA}$

hier dominiert BA (typ. für komplexe Befehle)

Befehlsausführung für einen Befehl typ. nicht verringerbar -> Ausführung für mehrere (auf einander folgende) Befehle gleichzeitig, dadurch wird das Programm insgesamt schneller.

2 identische ALE's. typ doppelter Logikaufwand

2 spezialisierte unterschiedliche ALE’s für unterschiedliche Befehlsgruppen typ wenig Zusatzaufwand in der Logik, aber nur Befehle aus unterschiedlichen Gruppen parallel möglich.

Verallgemeinerung dieses Prinzips: Multicore-Architektur

➔ weiter mit F_3_130