Rechnerarchitektur

Prof. Dr.-Ing. Thomas Beierlein
Hochschule Mittweida, Fak. CB

2024

Vorstellung

Prof. Dr.-Ing. Thomas Beierlein

- HS Mittweida, Fak. CB
- Lehrgebiete
  - Mikroprozessortechnik/Rechnerarchitektur
  - Echtzeitbetriebssysteme
  - Schaltungsentwurf mit VHDL
  - Eingebettete Systeme
  - Systemprogrammierung
- Forschung
  - Vernetzung im industr. Umfeld
  - Mikrocontroller-Steuerungen
  - Einsatz von embedded Linux
  - VHDL-Design

Kontakt

- Haus 8, Raum 301
- Tel.: 58 1043, Email: `tb@hs-mittweida.de`

Rechnerarchitektur
Motivation und Ziele

. . . umfaßt die Analyse, die Funktionsweise und den Entwurf von Rechnern und Rechnerkomponenten.

Motivation

- Verständnis der Arbeitsweise eines Rechners ist wichtig für
  - effektive Programmierung (auch in Hochsprache)
  - Problemanalyse (Debugging / Forensik)
- zunehmender Einsatz alternativer Prozessoren
  - erweitertes Wissen über Arbeitsweise notwendig
  - Basis für fundierte Auswahl und Bewertung

Inhalte und Ablauf
inhaltliche Schwerpunkte

- technische Realisierung maschineller Informationsverarbeitung
  - Aufbau und Funktionsweise von Computersystemen
  - grundlegende Begriffe
  - Grundlagen ihrer Programmierung
- *Hochsprache* $\Leftrightarrow$ *Umsetzung auf Maschinenebene*
- moderne Prozessorarchitekturen
  - Pipelining
  - superskalare Prozessoren
  - Multi-Core Systeme

Inhalte und Ablauf

- wöchentliche Vorlesung (Di) und 14 tägig Seminar (Fr, Wo1)
  - selbständige Arbeit zur Vertiefung des Stoffes notwendig
- Seminar
  - bereitet Praktikum vor
  - vermittelt dazu notwendiges ergänzendes Wissen
- wöchentliches Praktikum
  - Bearbeitung verschiedener Programmieraufgaben
  - selbständige Vorbereitung zu Hause
- Modul-Abschluss
  - 90’ Klausur
  - Arbeitsmaterialien und 1 Blatt A4 handschriftlich
  - Testat als Prüfungsvorleistung erforderlich (s. Webseite)

Literaturhinweise

- Wüst; *Mikroprozessortechnik*; 4. Auflage, Vieweg, 2011
- Herrmann; *Rechnerarchitektur – Aufbau, Organisation und Implementierung*; 4. Auflage, Vieweg, 2010
- Hoffmann; *Grundlagen der Technischen Informatik*; 4. Auflage, C. Hanser, 2012
- Hennessy, Patterson; *Rechnerorganisation und -entwurf — Die Hardware/Software-Schnittstelle*; Spektrum, 2005
- Erdweg; *Assembler Programmierung*; Vieweg, 1992
- Beierlein, Hagenbruch; *Taschenbuch Mikroprozessortechnik*; Fachbuchverlag Leipzig, 4. Auflage, 2011

Organisatorisches

HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

ToDo:

- Lehrmaterial herunterladen und ausdrucken
  - Befehlsliste für Intel 8086
  - *Assemblerprogrammierung mit dem SBC-86*
  - Foliensatz
- evtl. zusätzliche Literatur beschaffen
- Entwicklungsumgebung (*i8086*-Emulator) vorbereiten

Arbeitsmaterialien und lehrbegleitende Informationen:

- Vorlesungsfolien, allgemeine Unterlagen, Hinweise ...
  - *http://www.hs-mittweida.de/tb* $\longrightarrow$ Rechnerarchitektur

Kontaktadresse:

- Prof. Dr.-Ing. Thomas Beierlein, `tb@hs-mittweida.de`, Haus 8, Raum 301, Tel. 58 1043

HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

Einführung

Rechnerarchitektur

Definition

. . . umfaßt die Analyse, die Funktionsweise und den Entwurf von Rechnern und Rechnerkomponenten.

1. Unter Hardwarearchitektur versteht man die technischen Aspekte des Aufbaus, der Funktionsweise und des Zusammenspiels der einzelnen Komponenten eines Rechners.
2. Softwarearchitektur beschreibt die dem Programmierer sichtbaren Bestandteile, also *Befehlssatz*, *Befehlsformate*, *Adressierungsarten*, sowie die ansprechbaren *Register* und Speicherplätze.

Kriterien des Rechnerentwurfs

Je nach Anwendungsfall stehen unterschiedliche Kriterien beim Entwurf eines Rechners im Mittelpunkt:

- hohe Verarbeitungsleistung
  - kurze Ausführungszeit des Programms
  - großer Datendurchsatz
- Stabilität des Rechnersystems
- determinierte Reaktionszeiten
- geringer Umfang des erzeugten Programms
  - hohe *Codedichte*
- geringe Kosten des Rechnersystems

geschichtliche Entwicklung

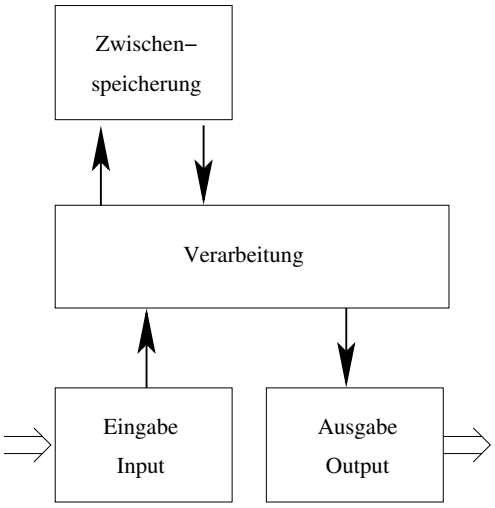
HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

30er Jahre	▶ Anfänge digitaler Realisierung des binären Rechnens
um 1950	▶ Konzept des gespeicherten Programms ⇒ informatorische Steuerung ▶ Entwicklung des Transistors
60er Jahre	▶ Entwicklung leistungsfähiger Großrechner ▶ Dezentralisierung der Rechentechnik ⇒ Minirechner ▶ zunehmende Integration von Bauelementen auf einem Chip
ab 1970	▶ Entstehung der Informatik • als Denkweise • als Lehre vom Bau und Gebrauch von Computern ▶ erster Mikroprozessor entsteht (1974)
80er Jahre	▶ breiter Einsatz von Mikroprozessoren für • dezentrale, personengebundene Computer, • industrielle Steuerungen und • Heim- und Konsumgüterelektonik.
heute	▶ Computertechnik erschließt vollkommen neue Einsatzgebiete • Hochleistungs-PC's und Workstations • digitale Verarbeitung von Signalen (Klänge und Bilder) • embedded systems, *IoT* – Internet of Things

Grundprinzip der Datenverarbeitung

Eingabe – Verarbeitung – Ausgabe

HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES



Input Einlesen von Informationen

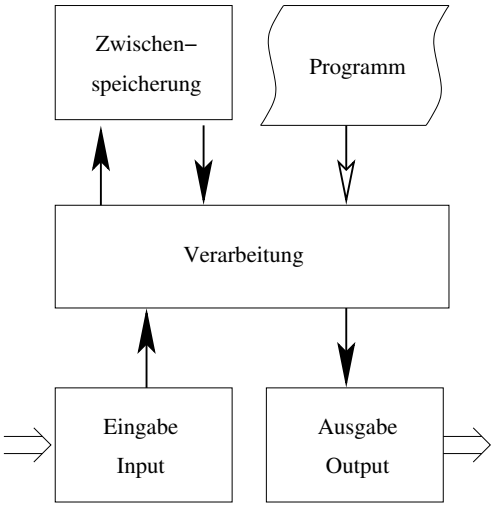
Verarbeitung der Informationen
– logische Entscheidungen
– numerische Operationen
– Symbolverarbeitung

Output Ausgabe der Ergebnisse

Speicherung Verarbeitung auf Basis aktueller und vergangener Informationen
⇒ gedächtnisbehaftetes System

Grundprinzip der Datenverarbeitung

informationsgesteuerte Datenverarbeitung



Input Einlesen von Informationen

Verarbeitung der Informationen
- logische Entscheidungen
- numerische Operationen
- Symbolverarbeitung

Output Ausgabe der Ergebnisse

Speicherung Verarbeitung auf Basis aktueller und vergangener Informationen
⇒ gedächtnisbehaftetes System

Programm

Liste von in Folge sequentiell abzuarbeitender Elementaroperationen

- konkrete Verarbeitungsvorschrift
- anderes Programm ⇒ anderer Algorithmus

Begriffe

CPU Central Processing Unit / Zentrale Verarbeitungseinheit
⇒ *Prozessor*

Daten durch die CPU zu verarbeitende Informationen

Adressen numerische Zahlenwerte
⇒ eindeutige Angabe eines Speicherplatzes

Befehl binärer Wert einer Elementaroperation, die vom Prozessor verstanden wird
⇒ *Maschinenbefehl / Code*

Mikroprozessor-Systeme

Einsatzfelder

- Personal Computer, Workstations
- *integrierte* Steuerrechner in Geräten, Anlagen, Sensoren, ...
- digitale Verarbeitung analoger Signale

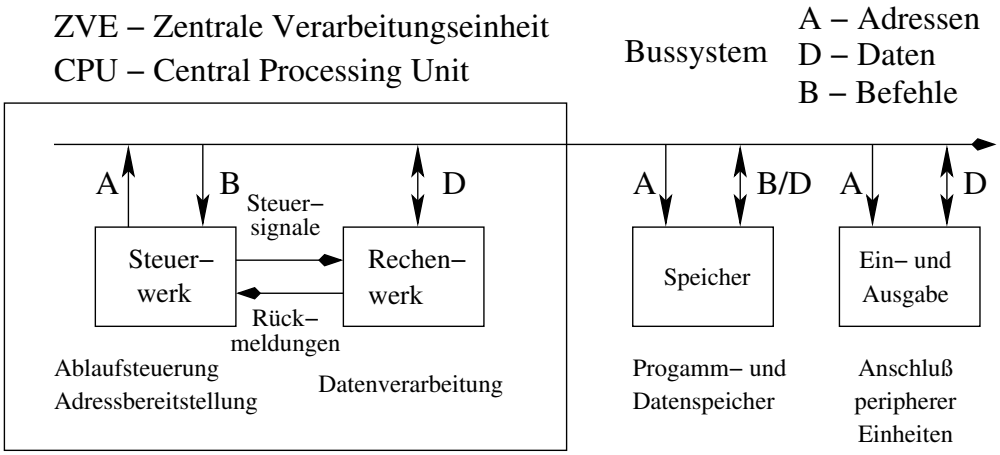
Vorteile der Anwendung

- ermöglicht Umsetzung neuartiger Funktionsprinzipien
- Funktionalität ist programmierbar (und leicht änderbar)
- kompakt auch bei großer Komplexität
- Geräte sind vernetzbar und zentral steuerbar
- Selbsttest / Eigendiagnose möglich

$\Rightarrow$ breiter Einsatz von Mikroprozessorlösungen (ca. $5 * 10^8$/a)

Rechnersystem

mit von Neumann Architektur

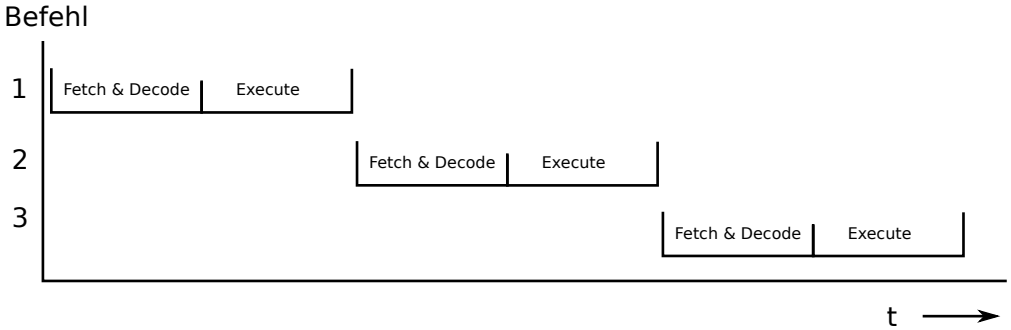


- nach John von Neumann benannt
- grundlegendes Modell eines Universalrechners
- arbeitet nach Master-Slave-Prinzip

von Neumann Architektur

charakteristische Merkmale

- Ein gemeinsamer Bus verbindet CPU, Speicher und Peripherie
- Gemeinsamer Speicher (feste Wortlänge) für Programm und Daten
- Numerische Adressen zur Auswahl von Speicherplätzen
- *von-Neumann* Zyklus $\Rightarrow$ 1 Befehl/Zeiteinheit in Bearbeitung



- ++ einfaches Design
- ++ Befehle und Daten austauschbar
- -- schlechte Performance
- -- Engpass sind Bus und Speicher

von Neumann Zyklus

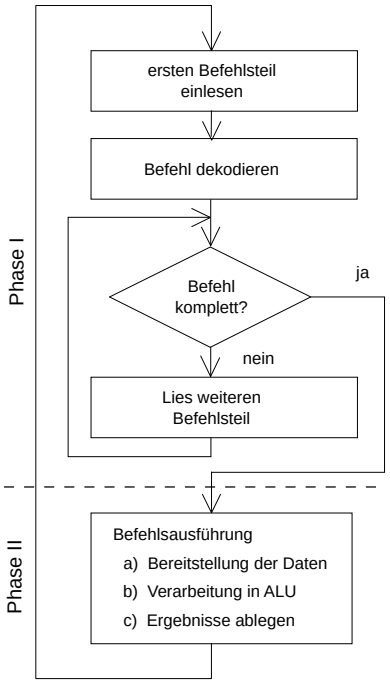
Ablauf der Befehlsbearbeitung

- Befehlsablauf ist endloser Zyklus
- sequentielle Folge von Teilschritten:

  Phase I
  1. Instruction Fetch IF
  2. Instruction Decode ID

  Phase II
  3. Data Fetch DF
  4. Execute EX
  5. Write Back WB

- Ein Befehlszeiger
  (Program Counter *PC* / Instruction Pointer *IP*)
  - adressiert auszuführenden Befehl
  - wird beim Einlesen (Phase 1) auf Folgebefehl gestellt



Arbeitsprinzipien

HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

sequentielle Befehlsbearbeitung

- CPU
  - liest einen Befehl nach dem anderen aus Speicher
  - dekodiert Befehl und führt ihn aus
- Befehlswirkung
  - Datentransporte
  - Verarbeitung der Daten
  - Ablaufsteuerung

Master-Slave-Prinzip

- CPU ist Master im System
  - übernimmt Befehlsbearbeitung
  - steuert alle Abläufe im System
- Speicher und E/A arbeiten als Slaves
  - werden nur auf Aufforderung des Masters hin aktiv

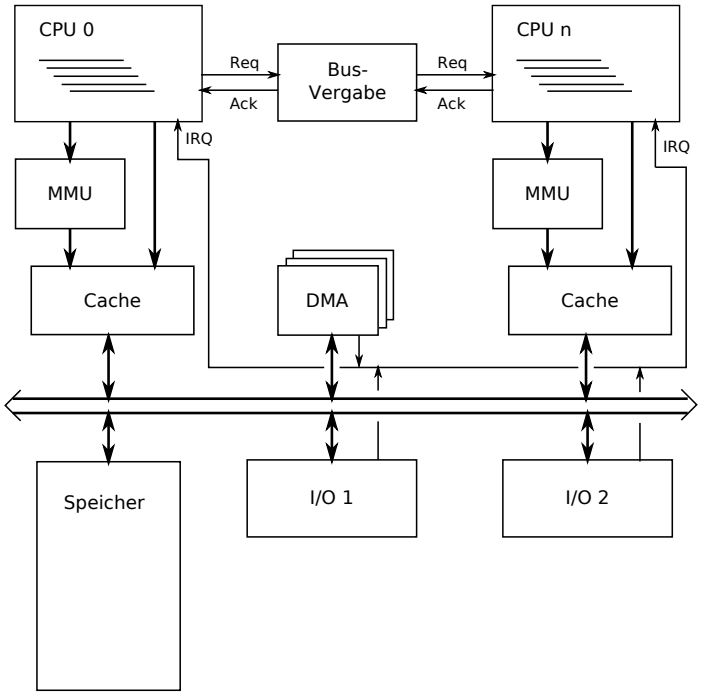
typische Erweiterungen

hauptsächlich zur Leistungssteigerung

HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

- Interrupt-Prinzip
  - Peripherie kann Bedienforderungen stellen
  - CPU reagiert an 'passender' Stelle
- Entlastung von I/O-Transporten
  - 'intelligente' I/O-Module mit Vorverarbeitung und Handshaking
  - Direct Memory Access – Datentransport im CPU-Auftrag
- Beschleunigung der Datenverarbeitung
  - mehrere CPU's $\Rightarrow$ Multiprozesing
  - CPU interne Parallelisierung
    - Pipelining, RISC, Superskalare Architektur, VLIW
  - Beschleunigte Speicherzugriffe $\Rightarrow$ Cache
- Memory Management Unit zur Speicherverwaltung
  - Überwachung Speichernutzung
  - Paging, Segmentierung, virtueller Speicher

Ein modernes Rechnersystem



grobe Inhaltsübersicht

- Einführung
- Softwarearchitektur – Programmierschnittstelle
  - Registersätze, ALU
  - Speicher und Adressräume
  - Adressmodelle, Adressierungsarten
  - Ablaufsteuerung
  - Interrupts und Exceptions
  - Schnittstellen zur Hochsprache
- Hardwarearchitektur
  - Wege zu höherer Verarbeitungsleistung
    - Pipelining
    - Superskalare Prozessoren
    - VLIW
    - Parallelverarbeitung
  - Bus- und Speichersystem, Cache
  - Betriebssystem-Support

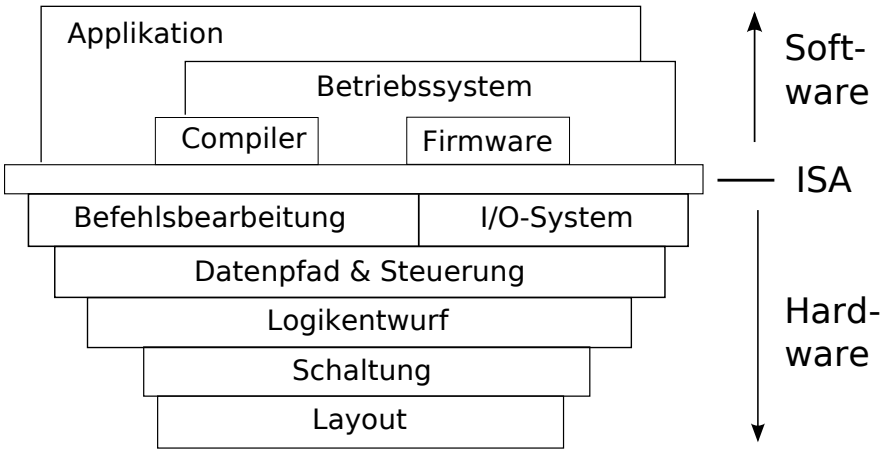
Softwarearchitektur

. . . beschreibt die dem Programmierer sichtbaren Bestandteile, also *Befehlssatz*, *Befehlsformate*, *Adressierungsarten*, sowie die ansprechbaren *Register* und Speicherplätze.

Instruction Set Architecture

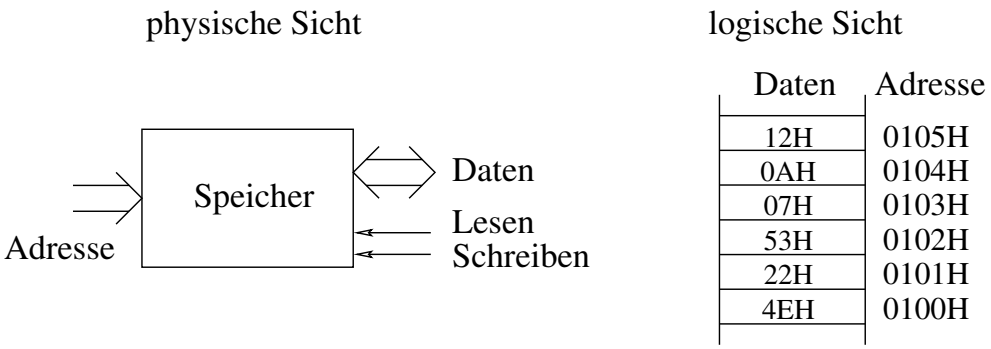
Befehlssatzarchitektur

. . . beschreibt die dem Programmierer sichtbare Schnittstelle zwischen Hardware und übergeordneten Software-Werkzeugen



Speicher

- 1-dim. Array von Speicherstellen mit fester *Datenbreite* (*n* Bits)
- eineindeutige Auswahl mittels *binärer Adresse*

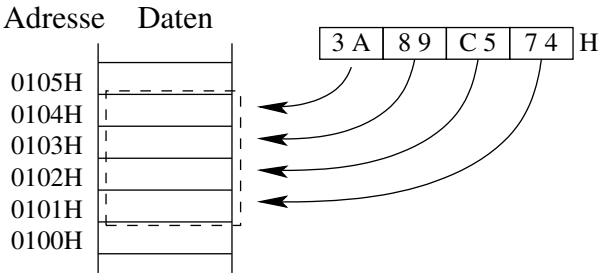


- Adresse — konkrete Kombination *m* binärer Adreßbits $A_0 \ldots A_{m-1}$
- Adreßraum — Menge aller möglichen Adressen.
- Datenbreite — Anzahl binärer Einheiten je Speicherstelle $D_0 \ldots D_{n-1}$.

Ablage größerer Dateneinheiten

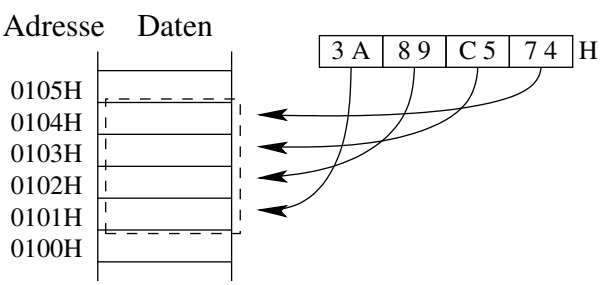
- Aufspaltung in Teilkomponenten — passen je in eine Speicherzelle
- Ablage in aufeinanderfolgende Speicherstellen
- Adresse der ersten Speicherstelle ist Adresse der gesamten Einheit
- Es existieren zwei gleichberechtigte Varianten:

Little Endian Format



Der *niederwertige Teil* findet sich auf der *niederen* Adresse.

Big Endian Format



Der *höherwertige Teil* findet sich auf der *niederen* Adresse.

Ein-/Ausgabe-Schnittstellen

typische Aufgaben

- vermitteln Datenaustausch mit Außenwelt
- koppeln periphere Geräte an Bussystem
- Signalanpassung zwischen internen und externen Signalgrößen
  - Pegel
  - zeitlicher Ablauf
  - Datenbreite
  - Übertragungsprotokoll
- entlasten CPU von Routineaufgaben
  - Pufferung
  - Prüfsummenberechnung
  - Steuersignale für Handshaking bzw. Synchronisation

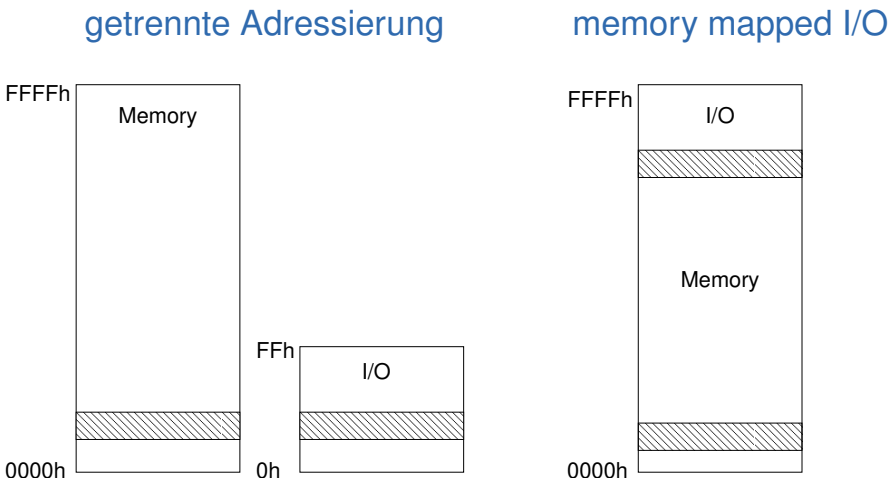
Ein-/Ausgabe — Input/Output

Ansteuerung peripherer Geräte

- 1-dim. Array von *I/O-Ports* (Port = Tor zur Außenwelt)
- *Port* $\equiv$ physikalisches Register
- Ports enthalten
  - Konfiguration der Arbeitsweise
  - Status
  - aktuelle I/O-Daten

  des Gerätes
- Datenbreite entspricht Speicherbreite
- Auswahl über numerische Adresse

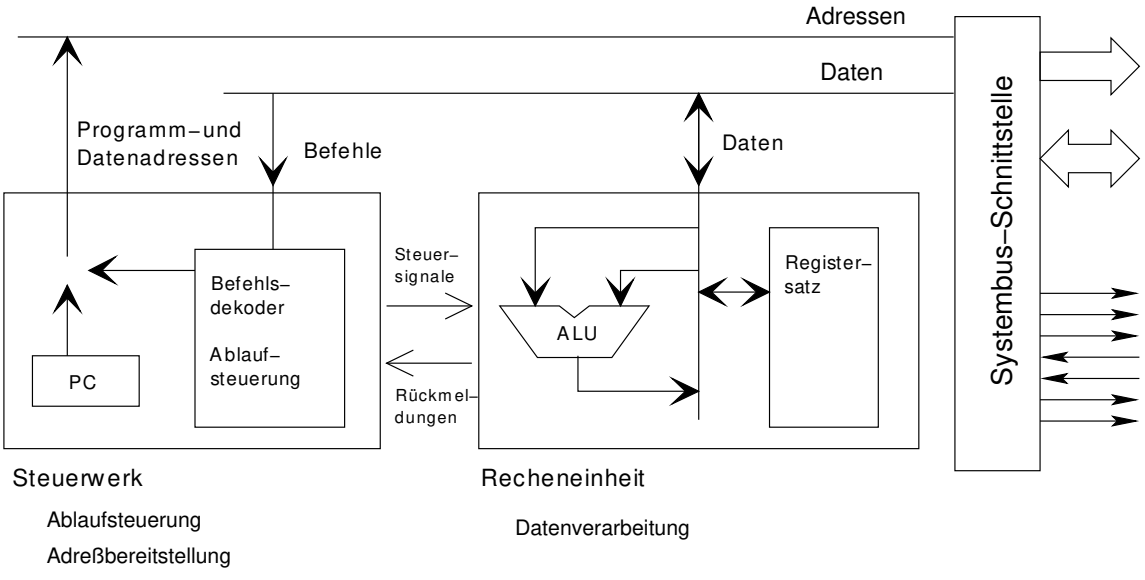
Varianten der I/O-Adressierung



Unterscheidung von I/O- und Speicher-Adressen

- getrennte Adressierung – spezielle Befehle (*IN/OUT*) für I/O
- *memory mapped I/O* – unterschiedliche Adressen

prinzipieller Aufbau einer CPU



Registersatz
Was sind Register?

- CPU-interne Datenablageplätze fixer Größe
  - 4 . . . 32 Register
  - 8, 16, 32 od. 64 Bit breit
  - kurze Zugriffszeit (0 Takte Overhead)
- enthalten:
  - Kopien oft benötigter Speicherinhalte
  - Zwischenergebnisse, Statusinformationen
- Inhalte der Register $\Rightarrow$ momentaner Arbeitszustand CPU
  - s.g. *Kontext*
  - bei Task-/Prozess-Wechsel zu sichern

Registersatz
typische Gruppen

Universalregister — *General Purpose Register*

- dienen der Datenablage
- sind Quelle und Ziel arithmetischer und logischer Operationen
- teilweise für Adreßrechnung benutzt

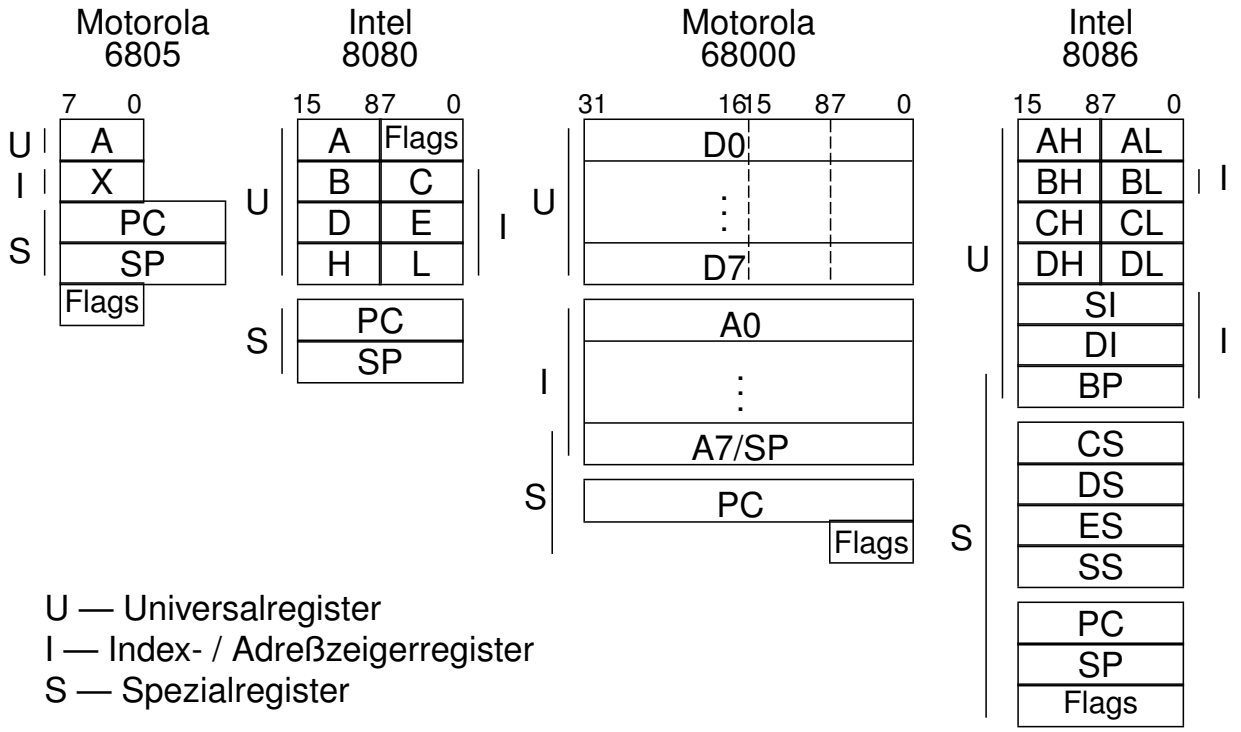
Adreßregister — *Pointer Register, Index Register*

- Adreßinformationen für Zugriffe auf Speicher und E/A
- ermöglichen die Berechnung von Operandenadressen
- eingeschränkt zur Datenablage geeignet

Spezialregister

Program Counter	PC	adressiert den nächsten Befehl
Stack Pointer	SP	adressiert temporäre Ablageplätze
Flagregister	Flags	Sonderfälle der letzten ALU-Operation

Beispiele für Registersätze

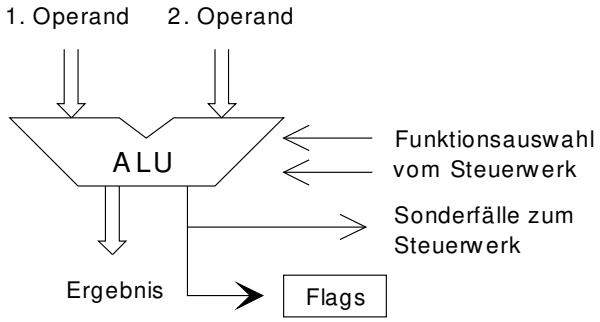


U — Universalregister
I — Index- / Adreßzeigerregister
S — Spezialregister

Arithmetic Logical Unit — ALU

Verknüpft i.d.R. zwei Operanden im binären Festkommaformat.

- Komplexe Operationen ⇒ aus Folgen einfacher Aktionen zusammensetzen



unterstützte Funktionen:

- Arithmetische Operationen.
- Logische Verknüpfungen.
- Verschiebung um eine oder mehrere Bitstellen nach links bzw. rechts.

Flags – speichern besondere Ergebniszustände zur späteren Auswertung z.B.:

Zero	Ergebnis ist genau Null.	**Carry**	Übertrag ist aufgetreten.
Sign	Ergebnis ist negativ.	**Overflow**	Vorzeichenbehafteter Zahlenbereich überschritten.

Befehlssatz – *Instruction Set*

HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

. . . umfaßt alle ausführbaren binären Maschinenbefehle.

- vom Hersteller bei der Fertigung implementiert.
- in seinem Aufbau an die Struktur der CPU angepaßt
- ermöglicht die Ansprache aller vorhandenen Ressourcen
- Zeitverhalten jedes Befehls durch internen Bearbeitungsablauf fest bestimmt $\Rightarrow$ *Clocks per Instruction*

typische Befehlsgruppen

HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

Transportbefehle `MOV, PUSH, POP`
Datentransfer zwischen Registern, Speicherzellen und Peripherie.

Arithmetikbefehle `ADD, SUB`
numerische Verarbeitung von Daten.

logische und Bitmanipulationsbefehle
– logische Verknüpfung von Bitgruppen. `AND, OR, XOR`
– Verschiebung von Bitstellen. `SHL, SHR`

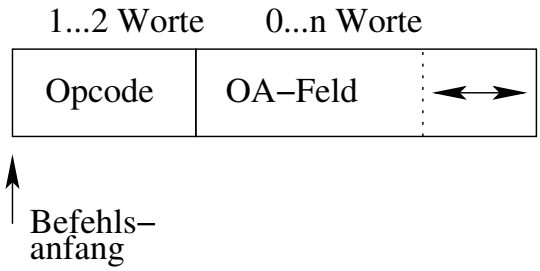
Programmsteuerbefehle `JMP, Jcc, CALL, RET`
Änderung des Programmablaufes.

Prozessorsteuerbefehle `NOP, STI, CLI`
Ansteuerung von Prozessorkomponenten
(direkte Beeinflussung von Flags, Steuerung Interruptsystem, . . .)

Aufbau des Befehlscodes

Maschinenbefehle bestehen aus binären Mustern, die für den Prozessor eine definierte Bedeutung haben.

Inhaltliche Zweiteilung



Operationcode (*Opcode*)
enthält Informationen über
- auszuführende Operation,
- Größe der Operanden u.
- Adressierungsart.

Operanden- u. Adreßfeld (*OA-Feld*)
enthält Informationen
- über zu verarbeitende Daten,
- zum Auffinden der Daten bzw.
- Programmfortsetzungsadressen.

Adressierungsarten

Überblick

Adressierungsart

- Algorithmus zur Bestimmung der Ablageorte von Informationen
  - können in Registern, Speicherzellen oder Ports stehen
- ist im Befehlscode verschlüsselt

Adressierungsart	Beispiel
implizit	`NOP`
immediate	`MOV CL,100`
direkt	`MOV AH,[0104H]`
indirekt	`MOV [BX],AL`
relativ	`JR Z,200H`

Beachte: Abweichende Bezeichnungen einzelne Hersteller

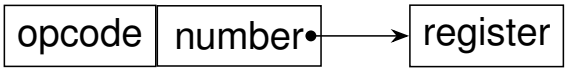
Adressierungsarten

implizite Adressierung

- Operand durch Befehl eindeutig festgelegt, keine separate Angabe
  - SCF Set Carry Flag
  - NOP No operation

unmittelbare Adressierung (immediate addressing)

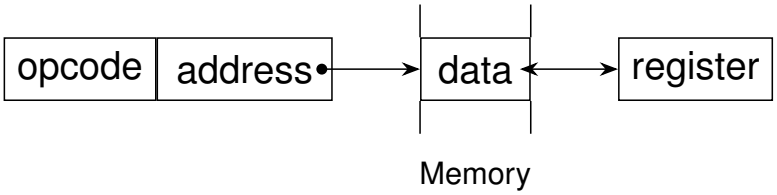
- Befehle enthält *unmittelbar* einer *konstanten* Operanden
  - MOV CX, 1234H Lade CX mit 1234H
  - MOV BH, 99H Lade BH mit 99H



Adressierungsarten

direkte Adressierung (direct addressing)

- Befehlscode enthält *direkt* die Adresse eines Speicher- bzw. I/O-Operanden
  - MOV DX,[1000H] Lade DX mit Inhalt der Speicherzelle 1000H
  - OUT 20H,AL Ausgabe AL auf I/O-Port 20H



Beachte:

Zwischen Bitbreite der Adresse und der des Datenwertes besteht kein Zusammenhang!

Adressierungsarten

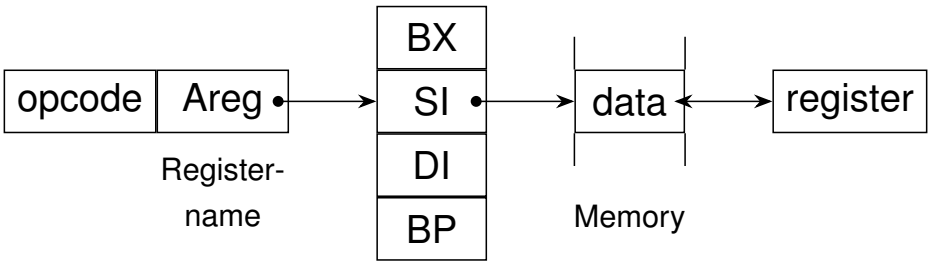
HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

indirekte Adressierung (indirect addressing)

- Befehlscode enthält Information zur *Bestimmung* der Operandenadresse

Basisvariante

- Befehl benennt ein Adressregister (BX, SI, DI, BP)
- Adresse steht in diesem Registerpaar
  - MOV CL,[BX] Lade CL mit Inhalt der Speicherzelle, deren Adresse in BX steht



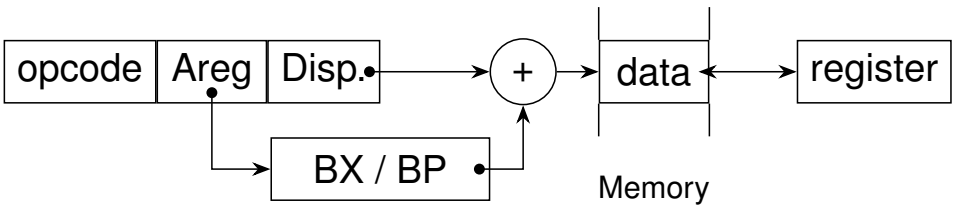
Adressierungsarten

HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

komplexe Varianten

(auch *indirekt mit Offset*)
(Details s. Befehlsliste, S.14)

- Adresse ist Summe aus Inhalt 16-Bit Adressregister (BX od. BP), einem 16-Bit Indexregister (SI od. DI) und einem konstanten *Displacement/Offset*
  - MOV BX,1000H
    MOV [BX+2],32H Schreib 32H in Speicherzelle 1002H



Adressierungsarten

abschließende Bemerkungen

- indirekte Adressierung bestimmt Operandenandresse zur Laufzeit
  - Adressrechnung liefert eine *effektive Adresse* $\Rightarrow$ *EA*
- Sprungziele verwenden
  - absolute 16-Bit Adresse (direkte Adressierung)
  - 8-Bit Distanz zum PC (relative Adressierung)
- relative Sprünge
  - Sprungziel ist Summe aus PC und Distanz
  - Sprungweite von -126. . . +129 Adressen
  - erlaubt kompaktere Programme

Kontrollstrukturen — I

IF and ELSE

IF *condition true*
THEN if_block;

```
        Test condition
        Jump if false to M1
        ... if_block ...
M1:     ...
```

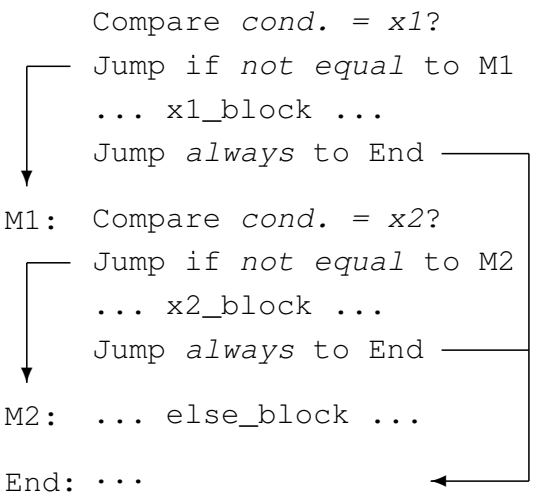
IF *condition true*
THEN if_block
ELSE else_block;

```
        Test condition
        Jump if false to M1
        ... if_block ...
        Jump always to End
M1:     ... else_block ...
End:    ...
```

Kontrollstrukturen — II

CASE/SWITCH

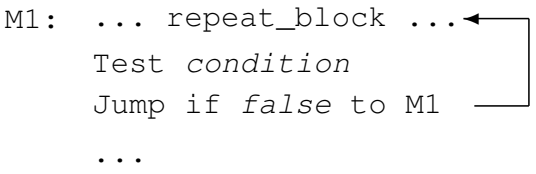
CASE *condition* OF
  x1: x1_block;
  x2: x2_block;
  ELSE else_block;
END



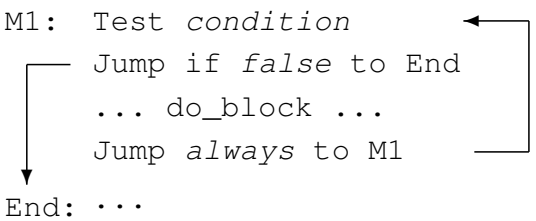
Kontrollstrukturen — III

unabgezählte Schleifen

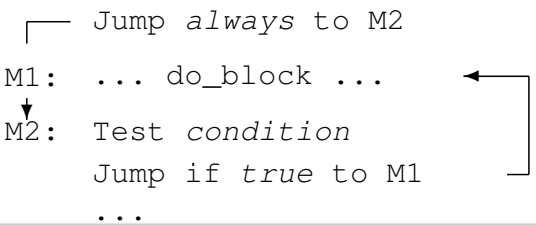
REPEAT repeat_block
  UNTIL *condition true*;



WHILE *condition true*
  DO do_block;



oder besser:



Kontrollstrukturen — IV

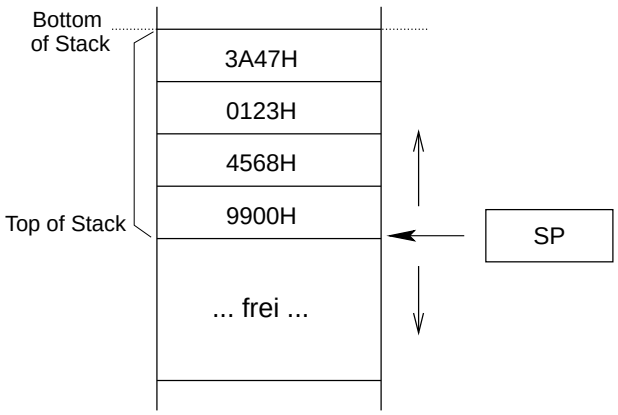
abgezählte Schleifen

FOR *Wert=Start* TO *End*
DO do_block;

```
        Wert = Start
        Jump always to M2
M1:     ... do_block ...
        inkrement Wert
M2:     Compare Wert = End?
        Jump if not equal to M1
        ...
```


Kellerspeicher – *Stack*

Reservierter Speicherbereich zur Ablage von Daten und Adressen nach dem LIFO-Prinzip.



- LIFO – Last In First Out
- Stack Pointer SP zeigt auf Top of Stack
- Zugriff mittels zweier Operationen:

Push	Datenelement hinzufügen
Pull / **Pop**	Datenelement entnehmen

Verwendung:

- temporäre Datenablage
- Speicherung von Rückkehradressen
- Parameterübergabe an Unterprogramme

Ablauf von *PUSH* und *POP*

s.a. Animation in Stack.pdf

HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

PUSH zz Rette 16-Bit Registerpaar *zz* auf Stack

1. SP := SP - 2
2. [SP] := Low(zz)
3. [SP+1] := High(zz)

POP zz Hole obersten Stackeintrag in Registerpaar zz

1. Low(zz) := [SP]
2. High(zz) := [SP+1]
3. SP := SP + 2

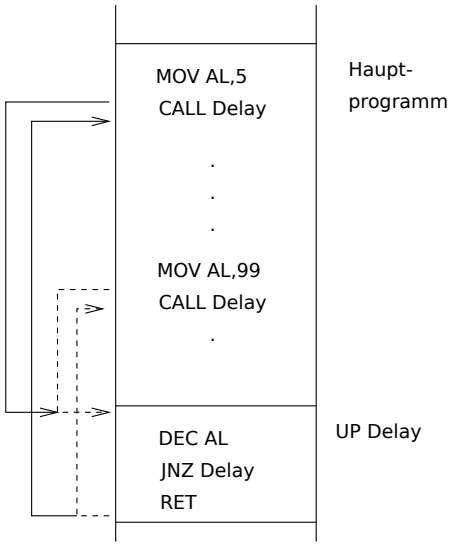
Bemerkungen:

- Stackpointer zeigt auf zuletzt abgelegtes Element (Top of Stack)
- Stack *wächst* in Richtung kleinerer Adressen

Unterprogramme

HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

Befehlssequenz mit abgeschlossener Funktionalität, welche mehrfach im Programmablauf aufgerufen wird.



- Code des UP ist nur einmal im Speicher
  - geringere Programmgröße
  - bessere Wartbarkeit
- Aufruf von beliebiger Stelle möglich (CALL – Call to Subroutine)
- Rückkehr zum folgenden Befehl nach UP-Abschluß
  RET – Return from Subroutine)
- mehrfach verschachtelte Aufrufe möglich

Ablauf von *CALL* und *RET*

s.a. Animation in Stack.pdf

CALL mn Aufruf Unterprogramm auf Adresse mn

1. SP := SP - 2
2. (SP) := PC ; sichere Rückkehradresse
3. PC := mn ; springe zu Unterprogramm

RET Abschluß Unterprogramm / Rückkehr zum Aufrufer

1. PC := (SP) ; lies Rückkehradresse zurück
2. SP := SP + 2

Bemerkungen:

- UP sind schachtelbar; auch rekursiv
- UP sollten frei von Nebenwirkungen sein
- Stackoperationen *müssen* paarweise erfolgen (Push/Pop, Call/Ret)

Aufruf-Konventionen

Calling Conventions

- *standardisierte* Methoden zur Parameterübergabe an UP in Hochsprachen
- regeln
  - Übergabeort der Parameter/Rückgabewerte (Register/Stack)
  - Reihenfolge der Parameter
  - Verantwortung für Entfernung der Aufrufparameter
  - Reservierung von Registern
- Beachte! Die Aufrufkonvention muß für alle Teilmodule eines Programmes gleich sein!

Aufruf-Konventionen

typische Vertreter in x86 Systemen

HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

cdecl – 32-Bit C Programme

- Parameter von rechts nach links auf Stack übergeben
- Rückgabe in Register EAX
- Parameter vom Aufrufer vom Stack entfernt

Pascal – 32-Bit Pascal Programme

- Parameter von links nach rechts auf Stack übergeben
- Rückgabe in Register EAX
- Parameter vom UP vom Stack entfernt

x86-64 – 64-Bit C Programme

- Parameter von rechts nach links in Registern RDI, RSI, RDX, RCX, R8, R9
- Rückgabewert in Register RAX
- Stackframe für weitere Parameter und lokale Variable

Das Interruptsystem

Unterbrechungs-Anforderungen

HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES

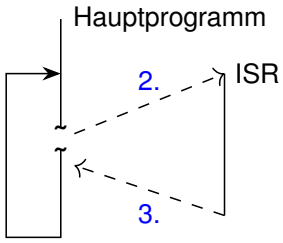
- Einführung
  - Interrupt-Konzept
  - Sonderformen von Programmunterbrechungen
- Aufbau einer ISR
- Annahme eines Interrupt Requests
- Interrupt-Vektor und Interrupt-Vektor-Tabelle
- Freigabe und Sperrung

Interrupts

Das dahinter liegende Konzept

- Erweiterung des Master-Slave-Prinzips
- periphere Einheiten dürfen Bedienforderung stellen
  - *Service Request*
  - *Interrupt Request*
- CPU (Master)
  - entscheidet über passenden Zeitpunkt der Behandlung
  - startet passende Behandlungsroutine
    ⇒ *Interrupt Service Routine* — ISR

prinzipieller Ablauf



1. *möglichst schnelle*, koordinierte Unterbrechung des Programm
2. Start ISR zur Problembehandlung
3. Fortsetzung unterbrochenes Programm

Sonderformen von Unterbrechungen

- kritische Ereignisse im Programmablauf `/0, illegaler Befehl`
  ⇒ *Ausnahmen, Exceptions*
- Befehle zur Auslösung Interruptbehandlung
  ⇒ *Software Interrupts*

Gegenüberstellung:

Merkmal	Interrupt	Exception	Software-Interrupt
Auslösung	extern (HW)	intern (Bef.ausführung)	intern (per Befehl)
Ablauf	asynchron	synchron	synchron
Behandlung	an Befehlsgrenzen	während Befehl	als Befehlswirkung
sperrbar	ja	nein	nein

Interrupt Service Routine

prinzipieller Aufbau

- Struktur analog Unterprogramm
- muß Register und Flags retten (`PUSH/POP`)
  - Vermeidung von Nebeneffekten für aufrufendes Programm
- spezielle Abschlußsequenz nötig (`IRET`)

typische Variante:

```
My_ISR: PUSH AX     ; Retten benutzter Register
        PUSH BX
        ...         ; Funktionsrumpf zur
        ...         ; Problembehandlung
        POP BX      ; Register restaurieren
        POP AX
        IRET        ; Abschlusssequenz
```


Annahme eines *Interrupt Request*

detaillierter Ablauf

1. Prozessor registriert Anforderung und beendet aktuellen Befehl
2. Falls Interrupt-Bearbeitung gesperrt, gehe zu Punkt 8
3. lösche gemerkte Interrupt-Anforderung
4. Sichere Kontext auf Stack (Flags, PC)
5. Sperre weitere Interrupts
6. Bestimme Startadresse der ISR (HW, fester Platz, Tabelle)
7. Lade PC mit Startadresse
8. Lies nächsten Befehl ein

Ergänzungen:

- `IRET`-Befehl am Ende der ISR gibt Interrupts wieder frei
  lädt Rückkehradresse vom Stack in PC

Interrupt-Vektortabelle — IVT

Zuordnung Interruptquelle zu Startadresse der ISR

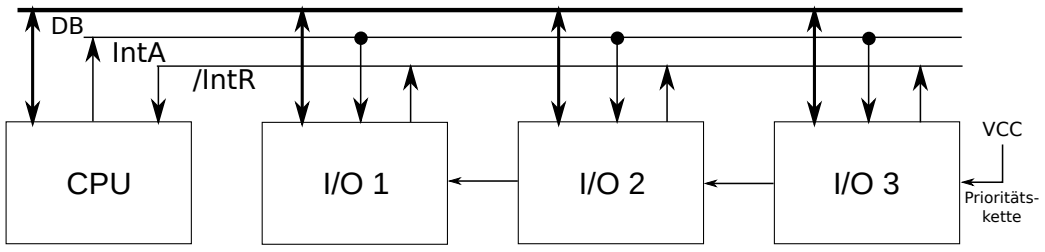
- jede Interrupt-/Exception-Quelle besitzt eine Interrupt-Vektornummer
  - Exception – fest verdrahtet
  - Interrupt – programmiert
- die Interrupt-Vektor-Tabelle enthält
  - Startadresse der jeweiligen ISR — Interrupt Vektor (IV)
- *Vektornummer* dient als Index in die *Vektortabelle*
  - Vektor# ist Nummer des Tabellenplatzes
  - wählt zugehörige ISR aus

8086:

- IVT enthält 256 Einträge zu je 4 Byte (für 20-Bit Startadressen)
- Lage der Vektortabelle im Speicher: Adresse 0000H...03FFH

Bereitstellung der Vektornummer

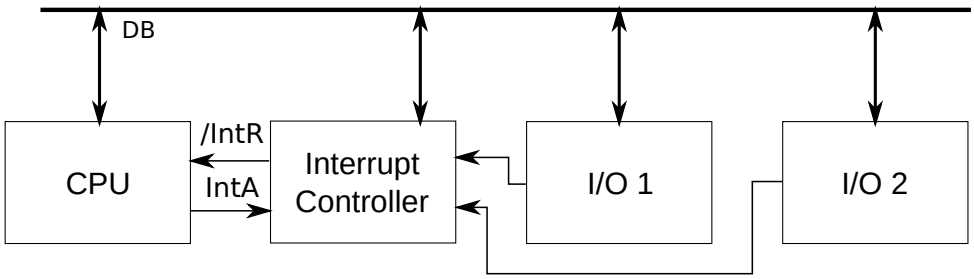
dezentral – durch Interrupt anmeldendes Gerät



- Interruptforderungen aller Geräte sind ODER-verknüpft
- jede Quelle besitzt eine *Quittungslogik*
  - erzeugt *Interrupt Request* **`/IntR`**
  - erkennt *Interrupt Acknowledge* **`IntA`**
  - liefert Vektornummer
- Hardwarepriorisierung bei gleichzeitiger Anmeldung – *Daisy Chain*

Bereitstellung der Vektornummer

zentral – durch Interrupt-Controller



- Geräte melden Interrupt-Bedarf an *Interrupt Controller*
- Interrupt Controller
  - entscheidet über Weiterleitung
  - meldet **`/IntR`** bei CPU an
  - erkennt **`IntA`**
  - liefert Verktornummer
- Priorisierung durch Int. Controller $\Rightarrow$ konfigurierbar

Interrupt-Freigabe

- Die Annahme externer Interrupts ist sperrbar (`I`-Flag).
  - `CLI` — Sperrung
  - `STI` — Freigabe
- Gesperrte Interrupt werden nicht ignoriert;
  Bearbeitung wird bis zur Freigabe verzögert.
- Nach einem *Reset* sind Interrupts stets gesperrt!

Anwendung:

- Schutz kritischer Programmabschnitte vor Unterbrechung
  - Realisierung zeitkritischer Abläufe
  - Initialisierung
  - Wahrung von Datenkonsistenz