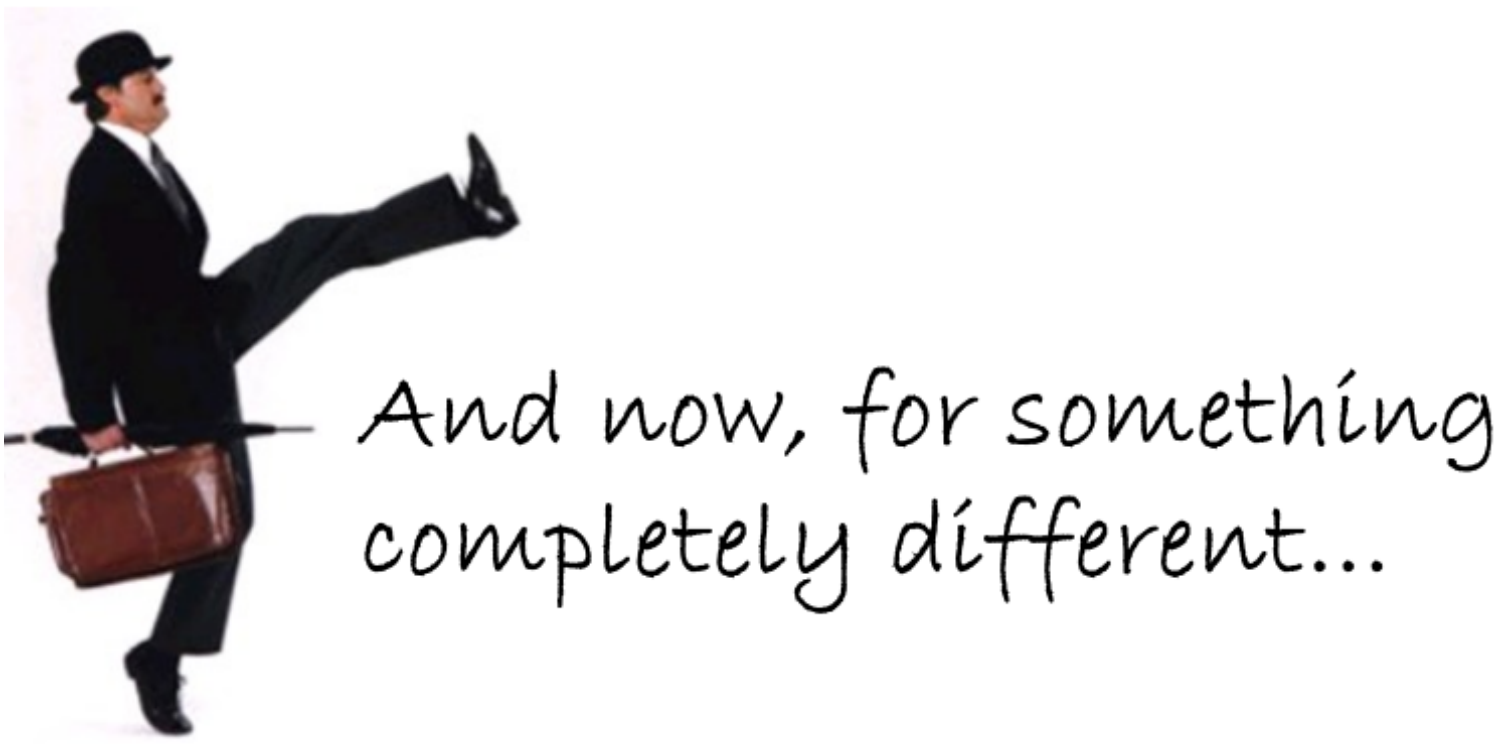
Hardwarenahe Programmierung I

U. Kaiser, R. Kaiser, M. Stöttinger, S. Reith

(HTTP: http://www.cs.hs-rm.de/~kaiser
EMail: robert.kaiser@hs-rm.de)

Wintersemester 2021/2022



https://slideplayer.com/slide/5134164/

Technische Umsetzung

- In diesem Kapitel wird dargestellt, welche technischen Mechanismen von Computern zur Verfügung gestellt werden, wie dadurch eine programmierbare Maschine bereitgestellt wird, und wie damit die vorgestellten Konzepte realisiert werden können.

Fragen...

- Wie werden Datenstrukturen technisch (d.h. in einem Computer) realisiert?
- Wie werden Programme technisch (d.h. von einem Computer) ausgeführt?

Diese Themen werden ausführlich in Lehrveranstaltungen wie Grundlagen der Informatik, Rechnerarchitektur und Mikroprozessortechnik behandelt. Hier nur das Nötigste.

Speicher

Hochschule RheinMain

Wir haben gesehen (s.o.): Sowohl Programme als auch Daten werden *gespeichert*.

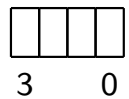
- Speicher sind das „Gedächtnis“ des Computers
- Erlauben das Halten von Daten bzw. Zuständen
- Technische Grundlage sind **Speicherzellen**: Bistabile Elemente die eindeutig einen von zwei möglichen Zuständen annehmen und halten können $\rightarrow$ 1 „Bit“.
- Beispiele:
  - Schalter, Flip-Flop, Reflexionseigenschaften (CD/DVD), Magnetisierungsrichtung (Festplatte), Ladungen (DRAM oder Flash-Speicher) ...

Zusammenfassungen zu *Worten*

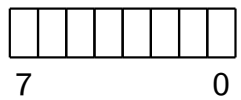
Hochschule RheinMain

- Durch Zusammenfassen von Bits zu einer Gruppe werden **Worte** gebildet.
- Ein aus Halbleitern (Flip-Flops) bestehendes Wort wird auch als **Register** bezeichnet.
- Mit einem Wort von n Bit lassen sich 2^n verschiedene Bitmuster (d.h. Zustände) repräsentieren:

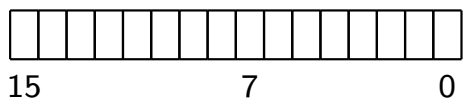
4 Bit: *Nibble* (*Tetrade*)



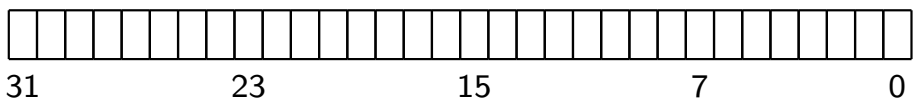
8 Bit: *Byte*



16 Bit: *Halbwort*[1]



32 Bit: *(Maschinen-)Wort*[1]



64 Bit: *Doppelwort*[1] .....

128 Bit: *Quadwort*[1] .....

- Schreib- und lesbare Worte bzw. Register können (u.A.) Variablen realisieren.

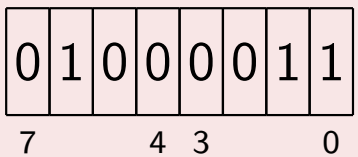
[1] Gilt für eine 32-bit Maschine (d.h. 32 bit Wortgröße)

Interpretation von Bitmustern

Hochschule RheinMain

- Worte speichern Bitmuster
- Ihre Bedeutung ergibt sich aus der Art ihrer Verarbeitung

Beispiel



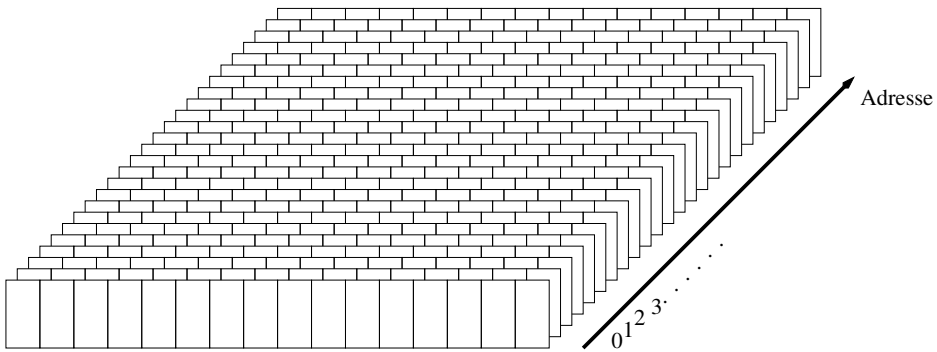
Könnte z.B. bedeuten:

Bedeutung	**Verarbeitung als**
Zeichen „C”	ASCII Zeichen
8-Bit-Zahl 67	Arithmetik-Operand
Befehl „INC BX”	IA-32 Befehl

Speicher

Hochschule RheinMain

- Der Hauptspeicher eines Rechners ist eine Folge (ein „Array“) von (Speicher-)Worten
- Die Auswahl eines Wortes erfolgt durch Angabe einer **Adresse** ($\rightarrow$ einer Zahl)



- Adressen können ebenfalls als Bitmuster in Registern oder Speicherworten gehalten werden.

Größere Informationsmengen

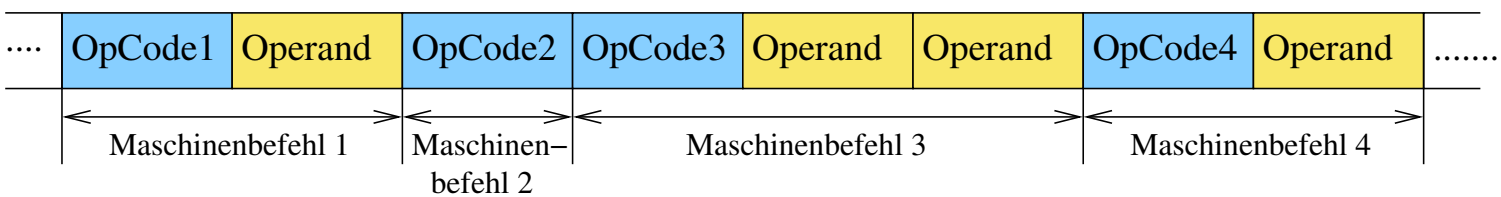
- Meistgebrauchte Einheit z.B. zur Angabe von Dateigrößen, Speichergrößen, etc. ist das Byte
  (... obwohl gerade Speicher i.d.R. in *Worten* organisiert ist ...)
- Größere Datenmengen werden als Vielfache von Bytes angegeben:

2^{10} Byte	= ein Kilobyte	= 1 KByte	= 1024 Byte	= 1024 Byte
2^{20} Byte	= ein Megabyte	= 1 MByte	= 1024 KByte	= 1.048.576 Byte
2^{30} Byte	= ein Gigabyte	= 1 GByte	= 1024 MByte	= 1.073.741.824 Byte
2^{40} Byte	= ein Terabyte	= 1 TByte	= 1024 GByte	= 1.099.511.627.776 Byte
2^{50} Byte	= ein Petabyte	= 1 PByte	= 1024 TByte	= 1.125.899.906.842.624 Byte
2^{60} Byte	= ein Exabyte	= 1 EByte	= 1024 PByte	= 1.152.921.504.606.846.976 Byte

Prozessor: Funktionsprinzip

Hochschule RheinMain

- Prozessoren führen *Maschinenbefehle* aus, z.B.:
  - Daten aus dem Speicher holen (*load*) oder in den Speicher schreiben (*store*)
  - Arithmetische Operationen auf einem oder mehreren Operanden
- Maschinenbefehle werden durch Bitmuster dargestellt (sogenannte *Operationscodes* oder kurz *Opcodes*)
- Opcodes werden vom Prozessor gelesen und interpretiert
- Viele Maschinenbefehle benötigen zum Opcode noch Operanden (Beispiel: Speicheradresse, auf die zugegriffen werden soll)
- *Maschinencode* ist demnach eine Sequenz zusammenhängender Maschinenbefehle, die ihrseits aus Opcodes und ggf. Operanden bestehen

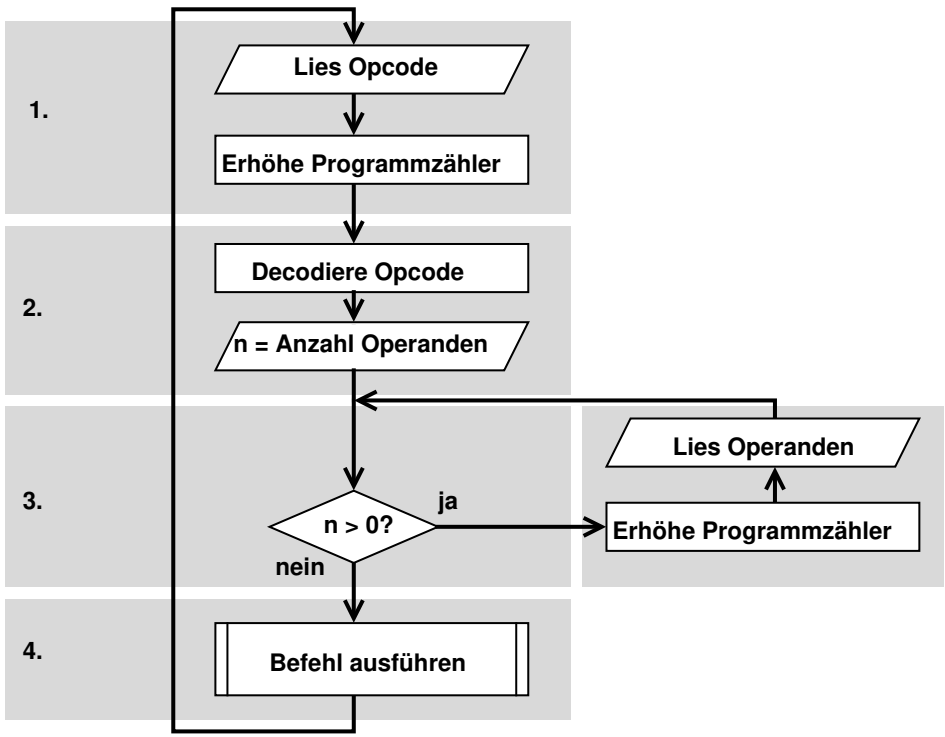


Programmausführung

Hochschule RheinMain

- Der *Programmzähler* (engl. *program counter* (PC)), ein spezielles Register des Prozessor, enthält die Adresse des nächsten Befehls
- Befehlszyklus:

1. **Opcode fetch**: Programmzähler enthält die Adresse des auszuführenden Befehls. Dieser wird aus dem Speicher gelesen, der wird Programmzähler erhöht
2. **Decodieren**: Bit-weiser Vergleich des Opcode mit bekannten Mustern, um seine Bedeutung zu entscheiden
3. **Operand fetch**: Die Anzahl der Operanden (n) ergibt sich aus dem Opcode. Diese werden nun ebenfalls geholt (und der Programmzähler dabei weiter erhöht)
4. **Execute**: Der Befehl wird ausgeführt



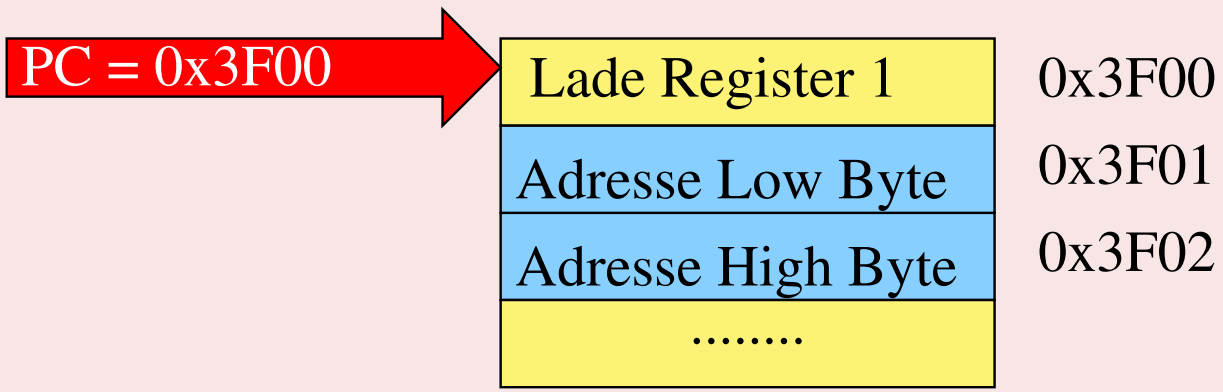
Speicherzugriffe

Hochschule RheinMain

- Je nach Operandenzahl kann ein Maschinenbefehl einen oder mehrere Speicherzugriffe beinhalten:

Beispiel (8-Bit Prozessor):

Laden einer 16-bit Adresse in Register 1:



- Einige Befehle (z.B. Inkrementieren eines Registerinhaltes) benötigen keine Operanden
- Es gibt Rechnerarchitekturen, bei denen alle Operanden im Opcode enthalten sind ($\rightarrow$ Feste Opcode-Größe, Voraussetzung für *Skalarität*)

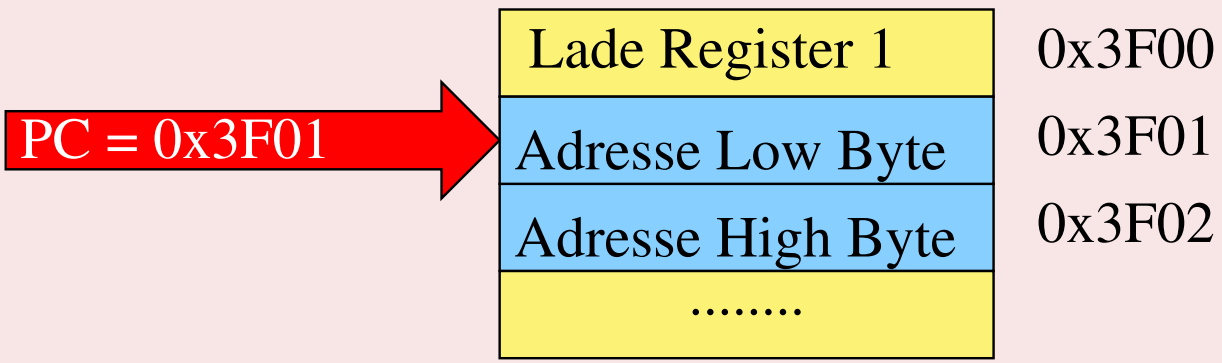
Speicherzugriffe

Hochschule RheinMain

- Je nach Operandenzahl kann ein Maschinenbefehl einen oder mehrere Speicherzugriffe beinhalten:

Beispiel (8-Bit Prozessor):

Laden einer 16-bit Adresse in Register 1:



- Einige Befehle (z.B. Inkrementieren eines Registerinhaltes) benötigen keine Operanden
- Es gibt Rechnerarchitekturen, bei denen alle Operanden im Opcode enthalten sind ($\rightarrow$ Feste Opcode-Größe, Voraussetzung für *Skalarität*)

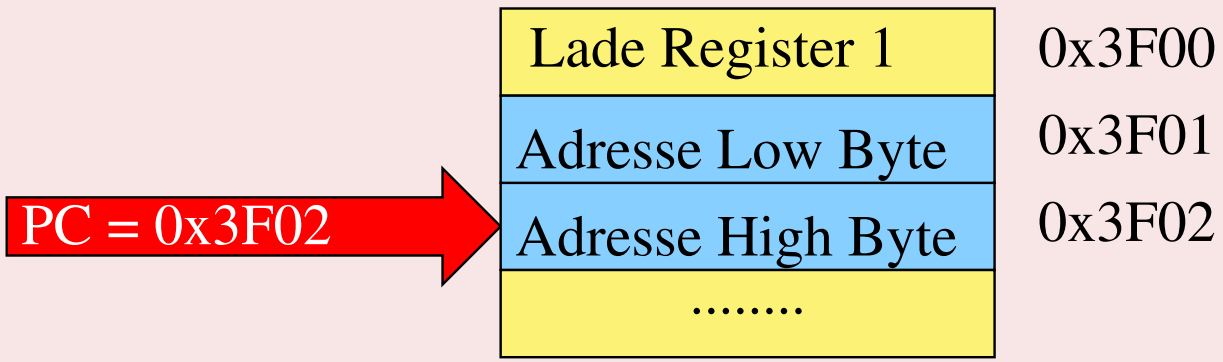
Speicherzugriffe

Hochschule RheinMain

- Je nach Operandenzahl kann ein Maschinenbefehl einen oder mehrere Speicherzugriffe beinhalten:

Beispiel (8-Bit Prozessor):

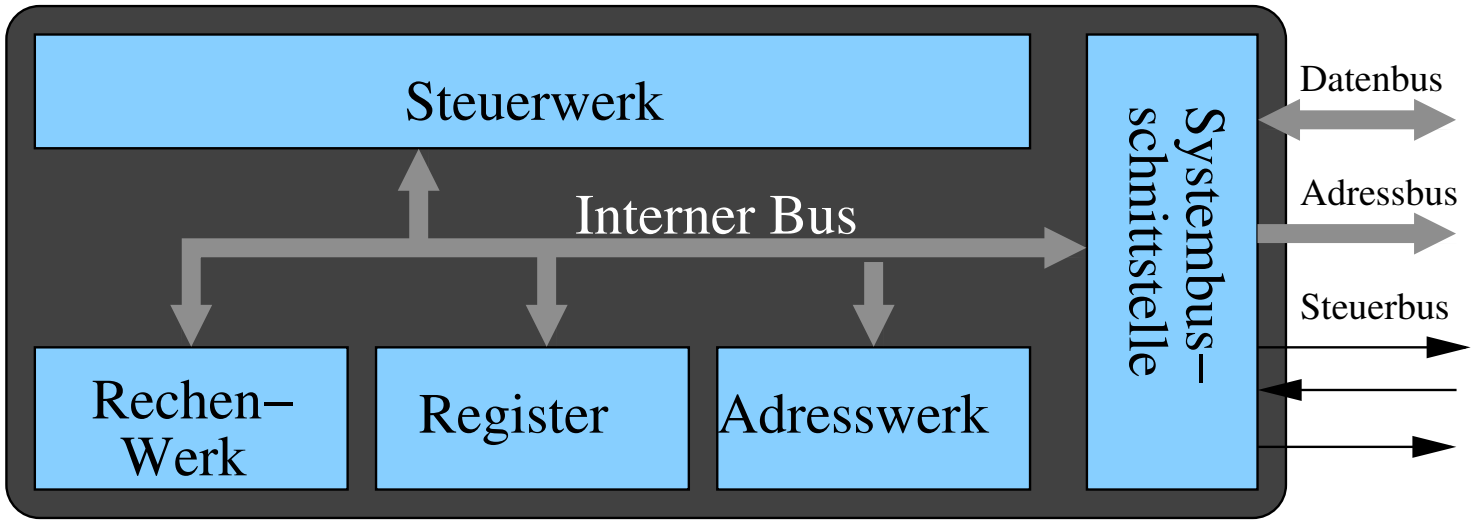
Laden einer 16-bit Adresse in Register 1:



- Einige Befehle (z.B. Inkrementieren eines Registerinhaltes) benötigen keine Operanden
- Es gibt Rechnerarchitekturen, bei denen alle Operanden im Opcode enthalten sind ($\rightarrow$ Feste Opcode-Größe, Voraussetzung für *Skalarität*)

Interner Aufbau eines Mikroprozessors

- Intern besteht ein Mikroprozessor aus mehreren Baugruppen:



- Register: interne Datenspeicher
- Rechenwerk: Arithmetisch-logische Verknüpfungen
- Steuerwerk: Ablaufsteuerung
- Adresswerk: Adresserzeugung
- Systembusschnittstelle: Treiber, etc.

Endlicher Automat

Abstrakt betrachtet ist ein Computer ein *endlicher Automat*:

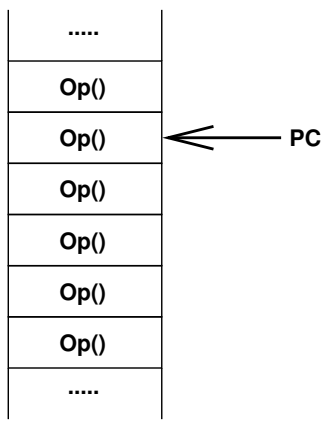
- $Neuerzustand = Operation(Alterzustand)$
- **Digital**rechner: Zustände werden durch Bitmuster dargestellt
- Anzahl der Zustände: $2^{AnzahlBits}$
- Beispiel: 1GB ($= 2^{30}$ Byte) Speicher $\Rightarrow 2^{(8 \cdot 2^{30})}$ Zustände
  $\Rightarrow$ Ziemlich viele (aber doch endlich) Zustände,
- *Operation*() ist eine Boolesche Funktion (vgl. Grundlagen der Informatik: „Schaltnetz“)
- Auswahl der *Operation*() erfolgt durch den *Programmzähler*

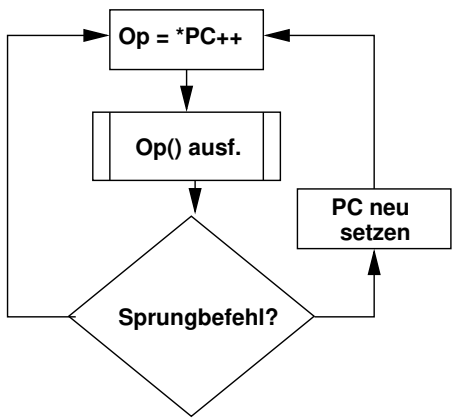
Programmzähler

Hochschule RheinMain

Ein Teil des Zustandes wird durch **Register** repräsentiert

- Der **Programmzähler** (*PC*, *IP*, *EIP*, ...) ist das erste und wichtigste davon
- Zeigt auf die nächste auszuführende Operation im Speicher





- Verzweigungsoperationen setzen $PC = Sprungziel$

Registersatz

- Allgemein: Register = Gruppe von Flipflops (N Stück) mit gemeinsamer Steuerung $\rightarrow$ N Bit breiter Datenspeicher
- I.d.R. hat ein Prozessor mehrere 8-/16-/32- oder 64-Bit Register
- Moderne RISC[2]-Prozessoren haben ≥ 32 Register
- Register sind das (einzige!) „Gedächtnis“ eines Prozessors

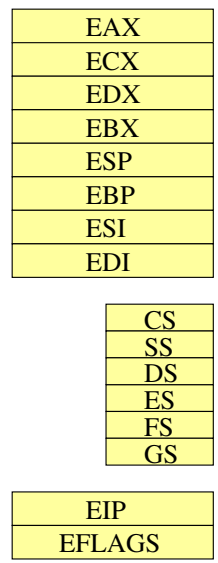
$\rightarrow$ Der Zustand eines Prozessors ist durch seine Registerinhalte vollständig beschrieben(!)

- Die meisten Register können durch Maschinenbefehle direkt bearbeitet werden („Variablen des Maschinenprogrammierers“)
- Zugriffe auf Register sind i.d.R. erheblich schneller als Speicher
- Daneben gibt es mitunter auch „unsichtbare“ Register, die nur intern verwendet werden

[2] Reduced instruction set computer, wird noch behandelt

Spezial- und Universal-Register

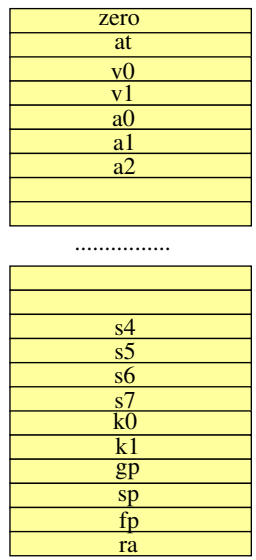
- Universalregister (engl. *General Purpose Registers*) können von vielen Maschinenbefehlen für unterschiedlichste Inhalte verwendet werden (Daten, Adressen)
- Spezialregister haben eine bestimmte, „hartverdrahtete“ Funktion, z.B. Programmzähler, Stackpointer[3], Akkumulator (früher), Nullregister, Rückkehradresse, etc..
- Beispiel: IA-32 Registersatz
  - 7 Universalregister (32 Bit)
  - 6 „Segment”-Register (16 Bit)
  - ESP: Stackpointer
  - EIP: Programmzähler
  - „historisch gewachsen” ...



[3] Dem Adresswerk zuzurechnen

Spezial- und Universal-Register

- Universalregister (engl. *General Purpose Registers*) können von vielen Maschinenbefehlen für unterschiedlichste Inhalte verwendet werden (Daten, Adressen)
- Spezialregister haben eine bestimmte, „hartverdrahtete“ Funktion, z.B. Programmzähler, Stackpointer[3], Akkumulator (früher), Nullregister, Rückkehradresse, etc..
- Beispiel: MIPS R3000
  - 30 Universalregister (32 Bit, Verwendung durch Konvention (*ABI*) festgelgt)
  - r0 (zero) → Konstante „0”
  - r31 (ra) → Rücksprungadresse
  - lo/hi → Ergebnisregister für 32x32 bit Multiplikation



[3] Dem Adresswerk zuzurechnen

Spezial- und Universal-Register

- Universalregister (engl. *General Purpose Registers*) können von vielen Maschinenbefehlen für unterschiedlichste Inhalte verwendet werden (Daten, Adressen)
- Spezialregister haben eine bestimmte, „hartverdrahtete“ Funktion, z.B. Programmzähler, Stackpointer[3], Akkumulator (früher), Nullregister, Rückkehradresse, etc..
- Beispiel: Atmel AVR
  - 32 Register (8 Bit)
  - r26 bis r31: paarweise als 16-bit Adressregister nutzbar
  - Register erscheinen ab Adresse 0 im Adressraum

Register	Speicheradresse
R0	0x0000
R1	0x0001
R2	0x0002
R3	0x0003
R4	0x0004
R5	0x0005
R6	0x0006
R7	0x0007
R8	0x0008
.....	
R20	0x0014
R21	0x0015
R22	0x0016
R23	0x0017
R24	0x0018
R25	0x0019
R26 X–reg low	0x001A
R27 X–reg hi	0x001B
R28 Y–reg low	0x001C
R29 Y–reg hi	0x001D
R30 Z–reg low	0x001E
R31 Z–reg hi	0x001F

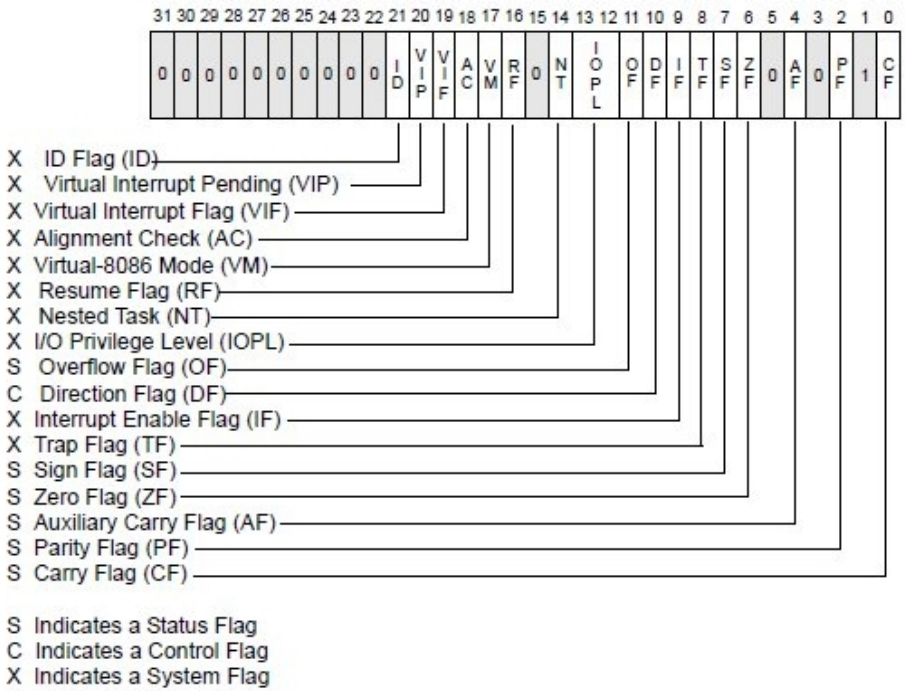
[3] Dem Adresswerk zuzurechnen

Spezielle Register zur Prozessorsteuerung

Hochschule RheinMain

- *Statusregister*, *Machine Status Word*, *Flag-Register* (Name je nach Hersteller / Architektur unterschiedlich, Funktion vergleichbar)
  - Zulassen / Sperren von Unterbrechungen
  - Privilegierter / nicht-privilegierter Modus
  - Ergebnisse Arithmetischer Operationen (Null, Überlauf, etc)

- Dem Rechenwerk zugeordnet
- Beispiel: IA-32
  EFLAGS Register:



Operationen

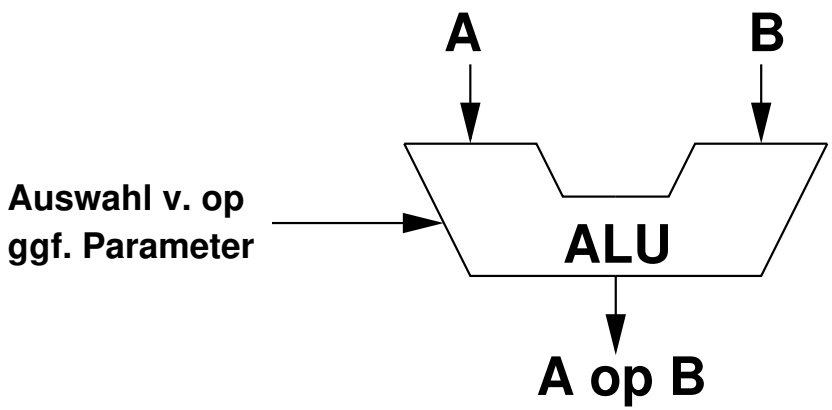
Die verfügbaren Operationen können klassifiziert werden

- Arithmetisch-Logische Operationen
  - $(+, -, \cdot, :, \mathit{AND}, \mathit{OR}, \mathit{XOR}, \ldots)$
- Datentransport
  - Zugriff auf E/A oder Speicher
- Kontrollfluss
  - (bedingte) Sprünge, Unterprogrammaufruf und -Rückkehr
- Steuerung und Konfiguration der Maschine
  - Interrupt sperren, Ausnahmebehandlung, etc.

Arithmetisch-Logische Operationen

Implementiert als **Schaltnetz** (ALU: Arithmetic Logic Unit)

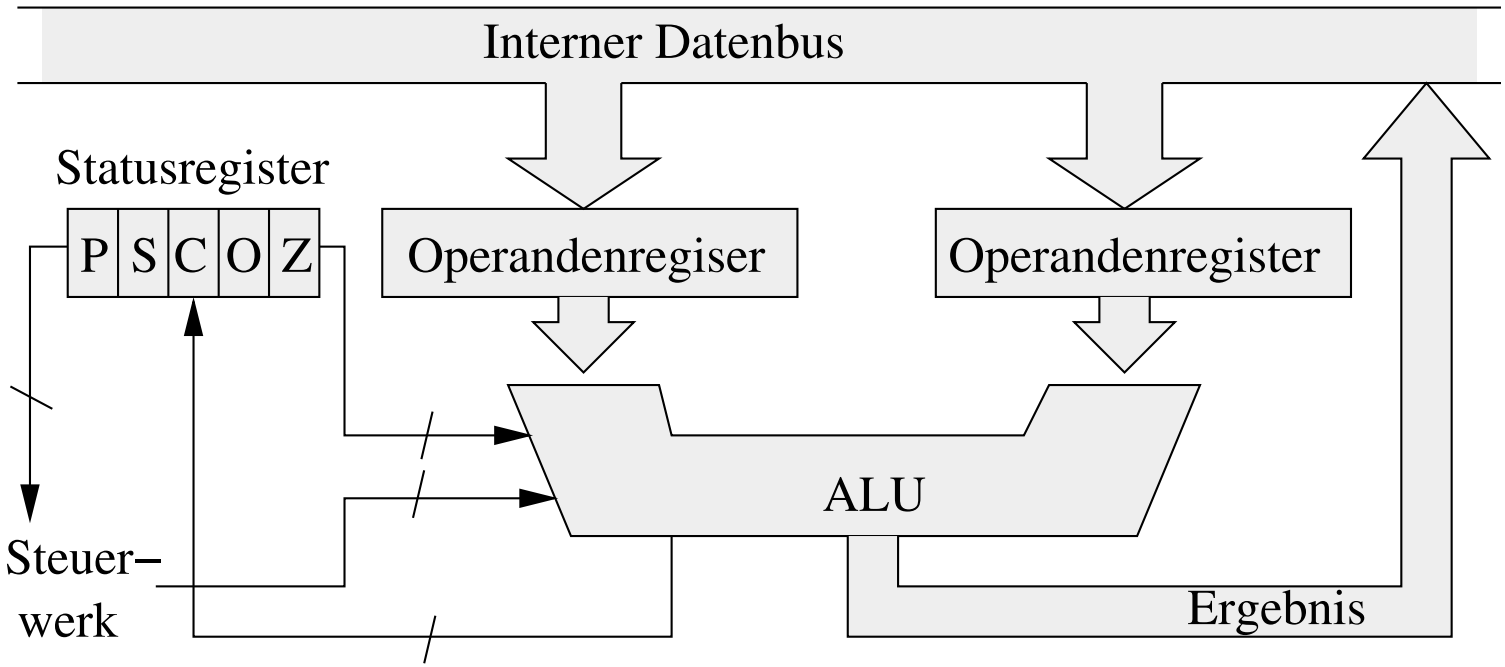
- In der Regel[4] zwei Eingänge für Operanden
- Ein Ausgang für Ergebnis
- Steuereingang wählt Operation aus und enthält ggf weitere Parameter (z.B. *shift amount*)



- Evtl. nur Strichrechnung und Schiebeoperationen
  → Multiplikation und Division algorithmisch (Mikroprogramme)

[4] aber nicht zwingend, vgl. Signalprozessoren

Rechenwerk



- Daten der beiden Operandenregister werden miteinander verknüpft
- Vielzahl möglicher Verknüpfungen wählbar:
  - UND, ODER, Äquivalenz, Antivalenz, rechts-schieben, links-schieben, ..
  - Addition, Subtraktion, Inkrement, Dekrement, ..
- Ergebnisse: Verknüpfungs-Bitmuster + „Flags“ → werden ins Flag-Register übernommen

Flags

Hochschule RheinMain

- Informationen über den Verlauf der letzten ALU-Operation werden ins „Flag-“Register übernommen
- Typische Flag-Bits:
  - **Zero (Z)**: Gesetzt (d.h. „1“), wenn das letzte Ergebnis Null war, sonst gelöscht („0“)
  - **Carry (C)**: Übertragsbit bei Addition, „Borgen“ bei Subtraktion (für vorzeichenlose Zahlen)
  - **Overflow (O)**: Überlauf bei Addition oder Subtraktion (für vorzeichenbehaftete Zahlen)
  - **Sign (S)**: Kopie des MSB (d.h. bei vorzeichenbehafteten Zahlen: des Vorzeichenbits) des Ergebnisses
  - **Parity (P)**: Zeigt an, ob die Anzahl der Einsen im Ergebnis gerade oder ungerade ist
- Bedingte Sprungbefehle werden durch Flags gesteuert (z.B. „Branch if (not) Zero“, „Branch if (not) Carry“ etc.)

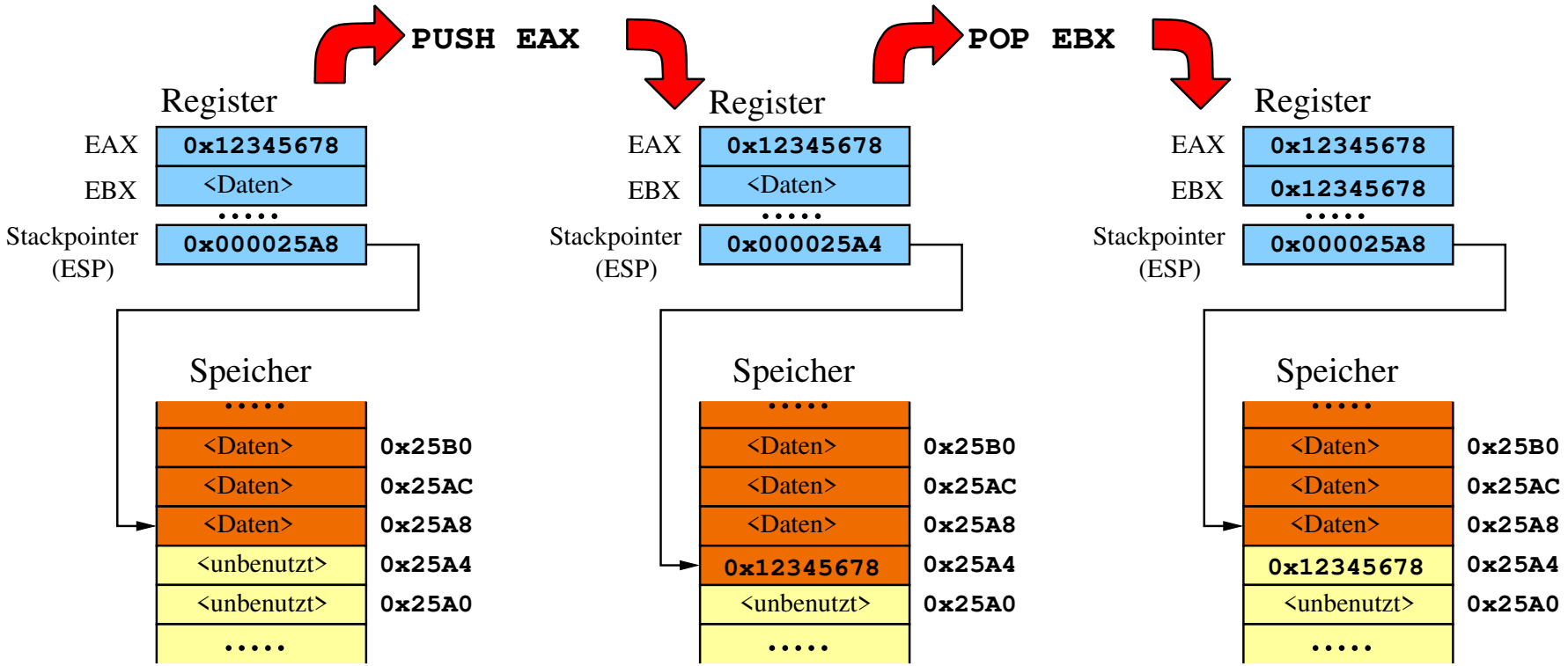
Stack

- Spezielles Register: der *Stackpointer* (SP)
- Autoinkrementierender/-dekrementierender Zeiger
- Dient zur Verwaltung einer LIFO[5]-Datenstruktur (Stapel, engl. *Stack*)
- Wird z.B. von C benutzt, um lokale Variablen anzulegen, Aufrufparameter zu übergeben und Rückkehradressen zwischenzuspeichern
- Bei den meisten Architekturen „wächst“ der Stack nach „unten“, d.h. zu kleiner werdenden Adressen hin
- Spezielle Maschinenbefehle wie „PUSH“ und „POP“
- Wird später noch genauer diskutiert ...

Beispiel: PUSH und POP

Hochschule RheinMain

- Hier: *Prädekrement* und *Postinkrement*



- Stackpointer zeigt auf „Top of Stack“

→ zuletzt gespeichertes Datum

Maschinenprogramme (1)

- In einem Programm werden komplexe Aufgaben in Einzelschritte zerlegt
- Diese Einzelschritte entsprechen Maschinenbefehlen (Opcodes + Operanden)
- Beispiel: Berechne `A = 9 * B - 1`, Schritte:
  1. Lade Inhalt von Variable `B` in Register 1
  2. Kopiere Register 1 $\rightarrow$ Register 2
  3. Schiebe Register 1 3x nach links (entspricht Multiplikation mit 8)
  4. Addiere Register 2 zu Register 1 $\rightarrow$ jetzt Multiplikation mit 9
  5. Dekrementiere Register 1 $\rightarrow$ enthält jetzt `9 * B - 1`
  6. Speichere Register 1 in Variable `A`

$\rightarrow$ Insgesamt 6 Maschinenbefehle

Maschinenprogramme (2)

Hochschule RheinMain

Entsprechender Maschinencode (z.B.)

10100001 0xA1	00000000 0x00	00101010 0x2A	10001011 0x8B	11011000 0xD8	11000001 0xC1	11100000 0xE0
00000010 0x02	00000011 0x03	11000011 0xC3	01000000 0x40	10100011 0xA3	00000000 0x00	00101000 0x28

→ D.h. ausführbares Programm (z.B. EXE-Datei) enthält hier die Bytefolge A1 00 2A 8B D8 C1 E0 02 03 C3 40 A3 00 28

- Prinzipiell könnte auf diese Ebene programmiert werden, aber ..
  - unhandlich, unflexibel, kaum änderbar (z.B. Einfügen auch nur eines Bytes → alle nachfolgenden Sprungziele verschieben sich..)
  - Programmcode ist faktisch nicht lesbar: Keine Variablennamen, keine Labels, etc.
  - Keine Kommentare möglich

Assemblersprache (1)

- Maschinenbefehle werden durch ca. 3 Zeichen lange Kürzel (*Mnemonics*) bezeichnet
  - `MV`, `MOV`, `MOVE` – Move: Daten bewegen
  - `SHL`, `SHR`, `ASL`, `LSL` – (Arithmetic/logic) shift left/right: Bitweises Schieben
  - `ADD`, `SUB`, `MUL`, `DIV` – Addieren/Subtrahieren Dividieren, Multiplizieren
  - `AND`, `OR`, `XOR` – Bitweises UND/ODER, exclusiv-ODER
  - `INC`, `DEC` – Inkrementieren, Dekrementieren
  - ....
- Register werden mit Namen referenziert: `EAX`, `R0`, `SP`, ...
- Speicheradressen (d.h. Variablen im Speicher, aber auch Sprungziele) werden mit Namen (*Labels*) bezeichnet
- Optional kann jeder Maschinenbefehl kommentiert werden

→ Programmme werden lesbar und verständlich

Assemblersprache (2)

- Assemblerprogramme stellen nach wie vor eine exakte Beschreibung des Programmcodes dar
- D.h. es existiert eine eindeutige Abbildung zwischen Assemblerprogramm und Maschinencode

Beispiel: `A=9*B-1`

```
mov ax,B    ;Variable B in Register AX
mov bx,ax   ;Kopieren in Register BX
shl ax,3    ;AX = AX * 8 -> AX = 8 * B
add ax,bx   ;AX = AX + BX -> AX = 9 * B
dec ax      ;AX dekrementieren -> AX = 9 * B - 1
mov A,ax    ;Speichere AX in Variable A
```

Assembler

- Ein *Assembler* („Montierer") übersetzt Assemblerprogramme in Maschinencode

Beispiel: `A=9*B-1`

Assemblerprogramm	Maschinencode
`mov ax,B`	A1 002A
`mov bx,ax`	8B D8
`shl ax,3`	C1 E0 02
`add ax,bx`	03 C3
`dec ax`	40
`mov A,ax`	A3 0028

- Assemblerprogramme sind spezifisch für die jeweilige Rechnerarchitektur

Ausblick: Compiler

- Hochsprachen (z.B. C) sind maschinenunabhängig

→ Ein Hochspracheprogramm muss für eine bestimmte Maschine in Maschinensprache umgewandelt werden, damit es von dieser ausgeführt werden kann

- Hochsprachenübersetzer *Compiler* übersetzen Hochsprachenprogramme in Assembler oder direkt in Maschinensprache

Zusammenfassung

- Speicher in Computern ist in *Worten* organisiert
- Die Auswahl von Speicherworten geschieht durch *Adressen*
- Speicherworte enthalten Bitmuster, deren Bedeutung sich aus der Art ihrer Verarbeitung ergibt
- Prozessoren verarbeiten Bitmuster als *Maschinenbefehle*
- Prozessoren haben *Register*, die als Quelle/Ziel von arithmetischen Operationen, aber auch als Adressen dienen können.
- Der *Programmzähler* und der *Stackpointer* sind spezielle Adressregister
- In Maschinenprogrammen werden komplexere Operationen aus einzelnen Maschinenbefehlen zusammengesetzt
- Assemblersprache ist eine leichter lesbare Form der Maschinensprache