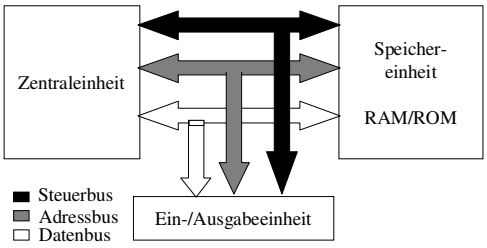
Kap.2
Befehlsschnittstelle

Prozessoren,
externe Sicht

RA Überblick

- Einleitung
- Hardwareentwurf
- Befehlsschnittstelle
- Mikroarchitektur
- Speicherarchitektur
- Ein-/Ausgabe
- Multiprozessorsysteme, ...

von Neumann-Architektur



- Kernstück ist **Zentraleinheit**
- **Speicher**, in denen Daten abgelegt werden
- **Ein- und Ausgabebausteine**, mit denen Kommunikationen mit Außenwelt hergestellt wird

- Komponenten durch **Busse** verbunden
- Vorteile:
  - Verringerung der Anzahl der Leitungen
  - Erweiterbarkeit/Skalierbarkeit
- Probleme bei bidirektionalen Bussen
  - zu jedem Zeitpunkt nur ein Zugriff

Zentraleinheit

- Auch Mikroprozessor oder CPU (=central processing unit)
- Aufgaben:
  - Steuerung
  - Holen und Interpretieren von Befehlen
  - Ausführen von Befehlen
  - Organisation des Datenaustauschs mit der Umwelt

Architektur einer Zentraleinheit

- Komponenten:
  - Steuerwerk
  - Operationswerk
- Steuerwerk für Befehlsverarbeitung
- Operationswerk für Datenverarbeitung

Schichtenstruktur

2 Befehlsschnittstelle

- **2.1** elementare Datentypen, Operationen
- **2.2** logische Speicherorganisation
- **2.3** Maschinenbefehlssatz
- **2.4** Klassifikation von Befehlssätzen
- **2.5** Unterbrechungen
- **2.6** Prozesse

2.1 Elementare Datentypen, Operationen

- Welche Datentypen kann der Prozessor verarbeiten?
  - Bitvektoren
  - ganze Zahlen
  - Gleitpunktzahlen
- Welche Operatoren können auf diesen Datentypen ausgeführt werden?

Kap.2 Befehlsschnittstelle

Prozessoren, externe Sicht

- **2.1** elementare Datentypen, Operationen
- **2.2** logische Speicherorganisation
- **2.3** Maschinenbefehlssatz
- **2.4** Klassifikation von Befehlssätzen
- **2.5** Unterbrechungen
- **2.6** Prozesse

- Speicher „spiegeln" den Zustand eines Rechensystems, bzw. eines Programms

  „Konfiguration"

- Wir unterscheiden Hauptspeicher und Registerspeicher.

Hauptspeicher

- „großes" adressierbares Array
- i.d.R. Byte-Adressierung, d.h. jedes Byte besitzt eine eigene, eindeutige Adresse
- größere Datentypen (z.B. int) sind mehreren Bytes zugeordnet
  (little endian, big endian: siehe unten)

- Aus Peformance Gründen ist Schreiben einzelner Bytes/Bits oft gar nicht möglich
- Ausrichtung an Wortgrenzen:

  Alignment

Alignment

- zusätzliches Problem: ausgerichteter (aligned) und nicht ausgerichteter (misaligned) Zugriff auf Speicher:

Objektadresse	Ausgerichtete Byteabstände	Nicht ausgerichtete Byteabstände
Byte	0,1,2,3,4,5,6,7	ausgeschlossen
Halbwort	0, 2, 4, 6	1, 3, 5, 7
Wort	0, 4	1,2,3, 5,6,7
Doppelwort	0	1,2,3,4,5,6,7

- "misaligned" Zugriffe (die in der Regel erlaubt sind) sparen Speicherplatz aber kosten Zeit bzw. erfordern zusätzlichen Hardwareaufwand

- maximale Größe:
  m Bits für Adresse
  - 2^m adressierbare Speicherzellen

  - 16-bit Adressen bei DEC PDP-11
  - 24-bit Adressen bei IBM Großrechnern
  - 64-bit bei DEC Alpha, SuperSPARC

Wie sind die Bytes numeriert?

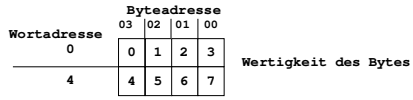
- little endian
  am wenigsten signifikanter Teil enthält die niedrigste Byte Adresse

- big endian
  der signifikanteste Teil enthält die niedrigste Byte Adresse

Little Endian - Big Endian

- Little Endian: z.B. 80x86, DEC PDP11/VAX, DEC ALPHA

- Big Endian: z.B. MIPS, MC680x0, IBM 360/370



- von Rechner zu Rechner verschieden
- manche unterstützen beide Formate
- i.d.R. merkt der Benutzer nichts davon, es sei denn man möchte auf einzelne Bytes explizit zugreifen
- bei Strings sind big endians natürlicher, bei Zahlen little endians

- Soviel zur logischen (Haupt-)Speichersicht

  strukturelle Eigenschaften später

- jetzt:

Registerspeicher

- Viel kleiner als Hauptspeicher, aber auch viel schneller
- kann genutzt werden, um lokale Informationen in Registern zu halten und so die Ausführung von Programmen zu beschleunigen

Designparameter

- Größe
  je nach Architektur unterschiedlich große Registerspeicher
  - kleine Registerbank: wenige Bit zur Adressierung, aber für schnelle Ausführung ist dann schnelle Speicherarchitektur notwendig
  - große Zahl: Befehlsgröße wächst, Ausführungszeit klein

- Funktionalität
  - homogen: alle Register haben gleiche Funktionalität
    - einfach zu nutzen, man muß eventuell angeben wie Inhalt zu interpretieren ist
  - inhomogen: „special purpose register"
    - erleichtert schnelle Ausführung von speziellen Operationen

Registerfunktionen

- **Adressregister**
- **Datenregister**
- **Spezialregister**

Spezialregister

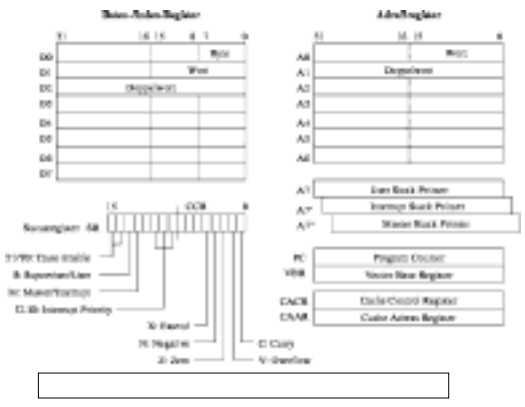
- **Akkumulator-Register** (ACCU)
  - speichert einen Operanden
  - unterstützt Ausführung von 'Schieben'
- **Programmzähler** (Befehlszähler)
  - erzeugt durch Inkrementieren aufeinanderfolgende Programmadressen
  - bei Sprungbefehlen neuen Wert laden

- **Instruktionsregister**
  - nimmt den aus dem Speicher gelesenen Befehl auf
- **Statusregister**
  - Informationen, die aus Operationen resultieren (Überlauf, Übertrag, Vorzeichen)

- **Stack-Register** (Stapel-Register)
  - aktueller Stand des Stackpointers
  - Verwaltung von Rücksprungadressen und lokalen Variablen bei Unterprogrammen

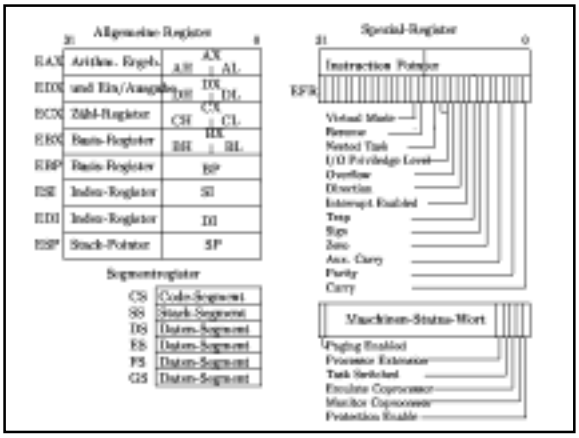
Beispiele

- **Motorola MC 68000 (1980--)**
  - 16-bit Prozessor, interne Struktur 32 bit
  - 16 weitgehend homogene 32-bit Register (8 Daten-, 8 Adress-)
  - 5 Spezialregister
  - optional 8 80-bit floating point Register



- **MIPS I/II**
  - 32-bit RISC Prozessor
  - 32 weitgehend homogene 32-bit Register (bis auf Register mit Nr. 0 und 1)
  - 32 32-bit floating-point-Register (einzeln für single precision, paarweise für double precision)

- **Intel 80386/486 (1985-90)**
  - 32-bit Prozessor
  - 16 Register, 10 32-bit, 6 16-bit jedes Register hat (mehr oder weniger) spezielle Aufgaben
  - 8 80-bit floating-point-Register



Kap.2 Befehlsschnittstelle

Prozessoren, externe Sicht

- **2.1** elementare Datentypen, Operationen
- **2.2** logische Speicherorganisation
- **2.3** Maschinenbefehlssatz
- **2.4** Klassifikation von Befehlssätzen
- **2.5** Unterbrechungen
- **2.6** Prozesse

- Wie nutze ich das Potential von Haupt- und Registerspeicher?
- Welche Befehle?
- Wie speichert man Daten ab?
- Wie berechnet man Adressen?

2.3.1 Befehlstypen

- MPs moderner Bauart haben ca. 30 bis 200 Befehle
- Überwiegende Anzahl der MPs haben kleinen Befehlssatz
  - muß kein Nachteil sein, da kleiner Befehlssatz übersichtlicher, einfacher anwendbar
  - bei speziellen Problemstellungen große Befehlssätze mit vielen Adressierungsarten von Vorteil

  (siehe auch Klassifikation am Ende des Abschnittes)

- Befehlssatz immer ein **Kompromiss**
  - Wünsche aus Anwendersicht
  - technisch (sinnvoll) machbar
- Zu beachten
  - Anzahl der Worte pro Befehl
  - Verarbeitungsbreite der Operationen
  - ...

- Untergliederung
  - Datentransportbefehle
  - arithmetische Befehle
  - logische Befehle
  - bitverarbeitende Befehle
  - Schiebe- und Rotationsbefehle
  - Stringbefehle
  - Sprungbefehle
  - Systembefehle

Datentransportbefehle (1)

- **Transport** eines Datums von Quelle zu Ziel
  - Quelle und Ziel im Haupt- oder Registerspeicher
  - auch Übertragung zwischen E/A-Toren und Registerspeicher
  - „Transport" eigentlich nicht richtig, da nichts von Quelle entfernt wird
    - besser: Kopie

Datentransportbefehle (2)

- Beispiele in Assembler-Schreibweise
  - Register->Register, Byte
    - MOVEB R0, R1
  - Register->Speicher, Wort
    - MOVEW R3,711
  - Speicher->Speicher, Doppelwort
    - MOVEL CELL1,CELL2

Datentransportbefehle (3)

- Spezialfälle
  - MOVEM (move multiple)
    - lädt oder speichert *n* aufeinander folgende Registerinhalte aus bzw. in einen Speicherbereich
  - PUSH, POP
    - legt Datum auf Stack bzw. entnimmt es
    - Korrektur des Stackpointers erfolgt automatisch
  - CLR (clear)
    - lädt Wert Null in Register oder Speicherzelle

Datentransportbefehle (4)

- EXC (exchange)
  - vertauscht zwei Operanden miteinander
  - idR muß einer davon im Register stehen
- SWAP
  - vertauscht zwei Registerhälften miteinander
  - dadurch ev. Zugriff auf höherwertige Bits eines Registerwortes

Arithmetische Befehle (1)

- Festkomma-Arithmetik für Bytes, Worte und Doppelworte
  - ab 16-Bit 'Standard'
- Beispiele
  - ADD (add), ADDC (add with carry)
  - SUB (subtract), SUBC (subtract with carry)
  - MULU (multiply), DIVU (divide)
    - Multiplikant und Multiplikator haben einfache Wortlänge, Produkt ist Doppelwort
    - Division analog

Arithmetische Befehle (2)

- ABCD, SBCD
  - Addition/Subtraktion von BCD-Zahlen
- ABCDC, SBCDC
  - analog inklusive Carry
- INC (increment), DEC (decrement)
  - anwendbar auf Register und Speicher
- NEG (negate)
  - bilde Zweierkomplement einer Zahl
  - entspricht Multiplikation mit -1

Arithmetische Befehle (3)

- CMP (compare)
  - Bedingungsbits (condition codes) werden manipuliert
  - können von Programmverzweigung abgefragt werden
- Alle Befehle werden durch ALU 'abgedeckt'
- Gleitkomma-Operationen?

Arithmetische Befehle (4)

- Gleitkomma-Operation werden oft von der ALU abgedeckt
  --> Zwei Möglichkeiten
  - (softwaremäßige) Emulation: Zerlegung der Operation in Elementarbefehle
    - langsam
  - Koprozessoren
    - extra Hardware
    - schnell

Logische Befehle

- ALU realisiert auch logische Funktionen
- Bedingungsbits (condition bits) werden beeinflußt
- Befehle, z.B.
  - OR (Disjunktion)
  - AND (Konjunktion)
  - XOR (Antivalenz)
  - NOT (Negation)

Bitverarbeitende Befehle

- Betroffene Operandenbits werden durch die in einer Maske auf Eins gesetzten Bitpositionen adressiert
  - BTST (test masked bits)
    - beeinflußt Zero-Flag der Bedingungsbits
  - BSET (test and set masked bits) und BCLR (test and clear masked bits)
    - beeinflußt Zero-Flag der Bedingungsbits
    - anschließend setzen der Bits auf Eins bzw. Null

Schiebe- und Rotationsbefehle

- Verschieben eines Operanden um *n* Bitstellen
- Man unterscheidet (gemäß Behandlung der Datenformatgrenzen)
  - logisches Schieben
  - arithmetisches Schieben
  - Rotieren

Stringbefehle

- Operationen auf Byte- und Wortketten
- Befehle
  - MOVES (move string)
    - kopiert zusammenhängende Kette in anderen (zusammenhängenden) Speicherbereich
    - Länge in allgemeinem Register angegeben
  - CMPS (compare string)
    - Vergleich einer Zeichenkette
    - Länge in allgemeinem Register angegeben

Sprungbefehle (1)

- Zur Ablaufsteuerung von Programmen
- **Klassifikation**
  - bedingte Sprungbefehle
  - unbedingte Sprungbefehle
  - Unterprogrammaufrufe
  - Rückkehr zum rufenden Programmabschnitt (Unterprogrammbeendigung)
  - Unterbrechung

Sprungbefehle (2)

- Gestatten Vorwärts- und Rückwärtssprünge
- Befehle
  - JMP (jump) oder BRA (branch)
    - bei JMP absoluter, unbedingter Sprung, Befehlszähler wird mit im Befehl angegebener Programmadresse geladen
    - bei BRA relativer, bedingter Sprung, Offset wird addiert wenn Bedingung erfüllt, sonst fortfahren mit der im Code folgenden, nächsten Anweisung

Sprungbefehle (3)

- BGT (branch greater, signed)
- BGE (branch greater or equal, signed)
- ....
- BEQ (branch equal)
- BNE (branch not equal)
- BVC (branch overflow clear)
- BVS (branch overflow set)
- BCC (branch carry clear)
- ...

Sprungbefehle (4)

- Unterprogrammaufrufe bzw. Unterbrechungsbehandlung
  - sichere Befehlszählerstand
  - Verzweigung zum Unterprogramm
  - Abarbeitung
  - nehme Rücksprungadresse von Stack
  - Rücksprung zum rufenden Programmabschnitt

Sprungbefehle (5)

- Unterschied bei Unterprogrammaufrufen und Unterbrechungsbehandlung
  - bei Unterprogrammaufruf Adresse des Unterprogramms in Befehl enthalten
  - Prozessorstatus wird nicht automatisch gerettet
  - bei Unterbrechungsbehandlung Sprungadresse fest vorgegeben oder aus Vektortabelle
  - Statusregister wird auf Stack gerettet

Sprungbefehle (6)

- Befehle
  - JSR (jump subroutine)
    - Rücksprungadresse auf Stack
    - Sprung zu im Befehl angegebener Unterprogrammadresse
  - RTS (return from subroutine)
    - Rücksprungadresse von Stack und Verzweigen
  - RTE (return from exception processing)
    - ähnlich zu RTS
    - Statusregister wird zurückgeladen

Systembefehle (1)

- Steuerung des Systemzustandes
  - Unterscheidung zwischen privilegierten Befehlen (nur im Systemmodus) und trap-Befehlen (software-mäßig erzwungener Übergang von Normalmodus in Systemmodus)
- Befehle
  - MOVSR (move status, privilegierter Befehl)
    - Zugriff auf Statusregister (schreibend, lesend)
    - Prozessorstatus kann verändert werden

Systembefehle (2)

- MOVCC (move condition code)
  - Zugriff auf Bedingungsbits
  - auch in Normalmodus zulässig
- MOVNSP (move normal stack pointer, privilegierter Befehl)
  - in Systemmode Zugriff auf Normalstackpointer
- NOP (no operation)
  - führt keine Operation aus

Systembefehle (3)

- STOP (stop processing, privilegierter Befehl)
  - stoppt Programmausführung
- RESET (reset external device, privilegierter Befehl)
  - Initialisierung von Systemkomponenten
- TRAP (trap unconditionally) und TRAPV (trap on overflow)
  - Programmunterbrechung, Adresse aus Vektortabelle

2.3.2 Adressierungsarten

„Anfangsjahre“ der Digitalrechner:

- alle Adressen der Operanden und Sprungziele sind absolute (physikalische) Adressen
- Programme und Daten sind vollständig lageabhängig,
  Programmieren einer Schleife erfordert Änderungen im Programmcode während des Programmlaufes

heute:

- Ziel ist Berechnung der effektiven Adresse dynamisch zur Laufzeit
- Benutzt werden dazu Konstanten, Register und andere Speicherplätze
- Unterscheidungskriterium ist die Zahl der Zugriffe auf den Speicher

Adressierungsarten

- **0-stufige Speicheradressierung**
  - implizite Register-Adressierung
  - explizite Register-Adressierung
  - immediate
- **1-stufige Speicheradressierung**
  - direkt
  - Register-indirekt
  - indiziert
  - Programmzähler-relativ

Adressierungsarten (2)

- **2-stufige Speicheradressierung**
  - indirekt absolut
  - indirekt Register-indirekt
  - indirekt indiziert
  - indiziert indirekt
  - indirekt Programmzähler-relativ
- **(>2)-stufige Speicheradressierung**

Erläuterungen und Beispiele (1)

- **0-stufig,** implizite Register-Adressierung
  - durch Befehl wird Register implizit spezifiziert, in dem sich Operand befindet
  - LSRA <=> „verschiebe den Inhalt des Akkus um eine Bitposition nach rechts"
  - CLC = „clear carry flag"
- **0-stufig,** explizite Register-Adressierung
  - Register im Op-code angegeben
  - DEC R0 <=> „Dekrementiere Inhalt von R[0]"

Erläuterungen und Beispiele (2)

- **0-stufig,** immediate
  - Operand ist im Befehl (als Konstante) enthalten
  - LD D3,#$A3 <=> D[3]:=$A3
- **1-stufig,** direkt
  - LD D3,$A374 <=> D[3]:=M[$A374]
- **1-stufig,** Register-indirekt
  - LD D3,(A4) <=> D[3]:=M[A4]

Erläuterungen und Beispiele (3)

- Varianten: Prä/Post- De/Increment z.B. für PUSH, POP

  Operand befindet sich im Speicher und Register beinhaltet effektive Adresse für Operanden, vor/nach jedem Speicherzugriff wird Inhalt des spezifizierten Registers um 1, 2 oder 4 Byte (je nach Inhalt) erhöht

- MOVE R1, (R0)+ oder MOVE R1, -(R0)

Erläuterungen und Beispiele (4)

- **1-stufig,** indiziert
  - effektive Adresse ergibt sich durch Addition eines „Index" zu einem Basiswert
  - Speicher-relativ
    Basis absolute Adresse, Index im Register
    ST R1,$A704(R0) <=> M[$A704+R[0]] := R[1]
  - Register-relativ
    Basis in Basisregister, Index absolut
    CLR $A7(B0) <=> M[B[0]+$A7]:=0
  - Register-relativ mit Index
    DEC $A7(B0)(I0)+ <=>
    ```
    M[B[0]+I[0]+$A7]:=
      M[B[0]+I[0]+$A7]-1;
    I[0]:=I[0]+1;
    ```


Erläuterungen und Beispiele (5)

- **1-stufig,** Programmzähler-relativ
  - zu Inhalt des Befehlszählers wird Konstante (Distanz) addiert um Adresse des Operanden zu generieren
  - LBRA $7FFF <=> „Verzweige ‚unbedingt' zu der Speicherzelle, deren Adressdistanz zum aktuellen PC 32767 (=$7FFF) ist"
- **2-stufig**
  Ergebnis der erste Berechnung liefert Adresse im Speicher, die wiederum Adresse bzw. Offset für folgende Adressrechnung enthält
  --> „Speicherindirekt"

Erläuterungen und Beispiele (7)

- **(>2)-stufig**

  nur in Ausnahmefällen realisiert,

  fortgesetzte Interpretation des gelesenen Speicherwortes als Adresse des nächsten Speicherwortes nennt man auch Dereferenzieren

  (siehe z.B. Realisierung von PROLOG mittels der *Warren Abstract Machine*)
  [Kogge: The architecture of symbolic computing, McGraw-Hill, 1991]

Adressierungsarten

Adressierungsart	Beispiel	Wirkung	Anwendung
Register	Add R4,R3	R4=R4+R3	Wert im Register
Immediate	Add R4,#3	R4=R4+3	Operand Konstante
Displacement/ Basisadress.	Add R4,100(R1)	R4=R4+M[100+R1]	lokale Variable
Reg.indirekt	Add R4,(R1)	R4=R4+M[R1]	Pointer (Zeiger)
Indiziert	Add R3,(R1+R2)	R3=R3+M[R1+R2]	Feld-Adressierung R1-Basis, R2-Index
Direkt/Absolut	Add R1,(1001)	R1=R1+M[1001]	statische Daten
Speicher-indirekt	Add R1,@(R3)	R1=R1+M[M[R3]]	Speicherplatz als Pointer (Wert=*p)
Autoinkrement	Add R1,(R2)+	R1=R1+M[R2] R2=R2+d	Für Zugriff auf Felder in Schleifen
Autodekrement	Add R1,-(R2)	R2=R2-d R1=R1+M[R2]	(wie Autoinkr.)
Skaliert/ Indiziert	Add R1, 100(R2)[R3]	R1=R1 +M[100+R2+R3*d]	Felder mit Daten der Länge d

Erläuterungen und Beispiele (8)

- Große Anzahl von Adressierungsarten kommt ursprünglich aus dem Wunsch, kompakten Programmcode zu erstellen
- mit Aufkommen von RISC: Zahl der Adressierungsarten hat sich reduziert mit dem Ziel der potentiell schnelleren Ausführung
- bei SOCs („system on chip"): benötigter Speicher wichtig, deshalb dort häufig *keine* RISC-Prozessoren

2.3.3 n-Adress-Maschinen

n-Adress-Maschinen, -befehl

- Bis jetzt wurden Alternativen bei Befehlsvorrat und Adressierungsarten diskutiert
- weiteres Klassifikationsmerkmal: Zahl und Art der Operanden bei dem betrachteten Befehlsformat?

3-Adressmaschinen, -befehl (1)

- *Dyadische* Operation
  - Verknüpfung von zwei Operanden A = B *op* C
- Befehle enthalten Angaben über
  - Art der Operation (Operationscode)
  - Adresse des ersten Operanden (erste Quelladresse)
  - Adresse des zweiten Operanden (zweite Quelladresse)
  - Adresse des Resultates (Zieladresse)

3-Adressmaschinen, -befehl (2)

- Werden alle vier Angaben in einem Befehl zusammengefaßt, so entsteht **Dreiadressbefehl**

Op-code	1. Quelle	2. Quelle	Ziel

- Beispiel
  - Operationscode mit 32 Bit
  - Prozessor arbeitet mit 32 Bit
  - Befehlslänge (32+3*32)=104 Bit

3-Adressmaschinen, -befehl (3)

- Befehlsformat
  - 'unhandlich'
  - lang
  - in Beispiel: vier 32-bit Worte verwendet
- **Ziel:** Verringerung der Anzahl der Adressen innerhalb eines Befehls
  - Reduzierung der Länge des Befehls

3-Adressmaschinen, -befehl (4)

- Möglichkeiten zur Reduktion
  - **verdeckte Adressierung**: eine Adresse ist Ziel und Quelle zugleich
  - **implizite Adressierung**: Befehl bezieht sich auf ein Register, das Quelle für den ersten Operanden ist (Registeradresse ist implizit im Operationscode enthalten)
  - **Kurzadressen**: z.B. Basisregister und Displacement

2-Adressmaschinen, -befehl (2)

- Typ SS, als Ziel wird eine Quelle benutzt
- Befehl besteht aus drei Worten

opcode	op1	op2
add	$4786,	$5567
M[$4786] = M[$4786]+M[$5567]		

11/2-Adressmaschinen, -befehl (1)

- Typ RS, eine Adresse ist Register, Registernummer kann in der Regel noch im ersten Befehlswort kodiert werden
- Befehl besteht aus 2 Worte

opcode + regnr	op2
add R1,	$4A67
R1 := R1 + M[$4A67]	

1-Adressmaschinen, -befehl (1)

- MP besitzt **ausgezeichnetes** Register
  - **Akkumulator**
  - Adresse nicht explizit, sondern implizit enthalten
  - bei dyadischen Befehlen: Quelle für ersten Operanden und Ziel für Resultat
  - nach Befehlsausführung wird alter Akkumulatorwert mit Verknüpfungsergebnis überschrieben

1-Adressmaschinen, -befehl (2)

- durch Doppeladressierung fallen zwei Adressen zusammen
  - verdeckte Adressierung
- im Befehl nur noch der Operationscode und der zweite Operand
- spezielle Lade- und Speicherbefehle erlauben Datentransfer zwischen Akkumulator und anderen Registern bzw. Speicher

1-Adressmaschinen, -befehl (3)

- Kompakte Darstellung
- Aber: bei dyadischem Befehl **drei** Befehle notwendig
  - lade Akkumulator mit Inhalt der Speicherzelle
  - verknüpfe Inhalt des Akkumulators mit zweitem Operanden
  - speichere Inhalt des Akkumulators

0-Adressmaschinen, -befehl (1)

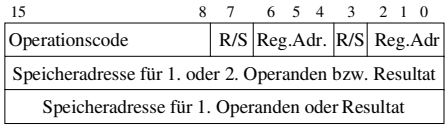
- Stack-Maschine: alle arithmetischen Operationen werden auf einem Keller ohne explizite Angabe der Operanden durchgeführt, dabei werden jeweils die beiden obersten Stackelemente verknüpft
- PUSH, POP mit Adressen
- einfache Übersetzung arithm. Ausdrücke, weitere Optimierungen nicht möglich
- Beispiel: Gleitkommaprozessor der Intel 80x86-Reihe

Zusammenfassung (1)

- Bei 8-bit MPs häufig 1-Adressbefehl
  - nur ein Akkumulator
- Bei 16-bit MPs mehrere Register mit gleicher Funktionalität
- **Identifikation** eines Registers erfolgt innerhalb des Befehls
  - 3-4 Bit (je nach Anzahl der Register)

Zusammenfassung (2)

- Beispiel:



  - 8 **allgemeine** Register
  - explizite Adressierung vorausgesetzt
  - verdeckte Adressierung verwendet

Zusammenfassung (3)

- Befehl kann bis zu **zwei** explizite Adressen beinhalten
  - lassen sich wahlweise auf allgemeine Register (prozessorintern) oder den Arbeitsspeicher bzw, Peripheriegeräte (prozessorextern) in allen möglichen Kombinationen beziehen
- in Befehl zwei Bits (R/S)
  - entscheiden, ob Zugriff auf Registersatz oder Speicher

Zusammenfassung (4)

- Vier verschiedene Befehlstypen
  - **Register-Register-Befehl**
    - beide Operanden und das Resultat prozessorintern
    - Befehl besteht nur aus einem Wort
  - **Register-Speicher-Befehl**
    - erster Operand Register, zweiter aus Speicher
    - Ergebnis in Speicher
    - Befehl besteht aus zwei Worten

Zusammenfassung (5)

- **Speicher-Register-Befehl**
  - erster Operand aus Speicher, zweiter aus Register
  - Ergebnis in Register
  - Befehl besteht aus zwei Worten
- **Speicher-Speicher-Befehl**
  - beide Operanden aus Speicher
  - Befehl besteht aus drei Worten

Zusammenfassung (6)

- Moderne Rechner erlauben i.d.R. 2-3 Operanden für typ. ALU-Befehl
- dabei kann max. Zahl von erlaubten Speicheradressen geringer sein

Max Speicheradressen	Max Operanden	Rechner
0	3	SPARC, MIPS
1	2	Intel 80x86, MR 68000
2	2	IBM 360
3	3	VAX

Kap.2 Befehlsschnittstelle

Prozessoren, externe Sicht

- **2.1** elementare Datentypen, Operationen
- **2.2** logische Speicherorganisation
- **2.3** Maschinenbefehlssatz
- **2.4** Klassifikation von Befehlssätzen
- **2.5** Unterbrechungen
- **2.6** Prozesse

Mikroprozessorklassen (1)

- MPs mit komplexem Instruktionssatz (**CISC**=complex instruction set computer)
  - verfügen über Satz von zum Teil komplexen Befehlen
  - Struktur des Befehlssatzes ist fest vorgegeben und nicht veränderbar (Makrobefehle)
- Mikroprogrammierbare MPs
  - Satz von Mikrobefehlen steht zur Verfügung
  - können zur Definition von Makrobefehlen genutzt werden

Mikroprozessorklassen (2)

- MPs mit reduziertem Befehlssatz (**RISC**=reduced instruction set computer)
  - einfache Befehle
  - in einem Verarbeitungsschritt ausführbar
- MPs mit niedrigem Befehlssatz (**low-RISC**)
  - bis zu 16 Befehle
  - für dezidierte Problemlösung besonders gut angepaßt
- weitere Klassen

Mikroprozessorklassen (3)

- Im folgenden zunächst:

  CISC, RISC, CISC versus RISC

CISC (1)

- Erster moderne Rechner (IBM 360, 1965) war CISC-Rechner
- Charakterisierung:
  - mächtige Befehlssätze (über 100 Befehle)
  - komplexe Maschinenbefehle
  - viele Adressierungsarten
  - mehrere Datentypen
  - mikrokodierter Befehlssatz (Steuerung durch Mikroprogramm)

CISC (2)

- Orthogonaler Befehlssatz, d.h. jeder Datentyp kann mit vielen Adressierungsarten angesprochen werden
- wenige Register
- Maschinenbefehle haben unterschiedliche Ausführungszeiten
- Mikroprogramm als Interpretationsebene (Mikroprogrammsteuerwerk kontrolliert Befehlsausführung)

CISC (3)

- Argumente für CISC:
  - komplexe Befehle schließen **semantische Lücke** zwischen Software und Hardware
  - Diskrepanz zw. interner Verarbeitungsgeschwindigkeit und Speichergeschwindigkeit reduziert
  - Komplexe Befehle reduzieren Speicherplatzbedarf (bezogen auf Hauptspeicher)
  - komplexe Befehle **vereinfachen Compiler**
  - **Zuverlässigkeit** (mehr Funktionalität unterhalb der Maschinensprache)
    - „Hardware ist sicher, Software enthält Fehler"

CISC (4)

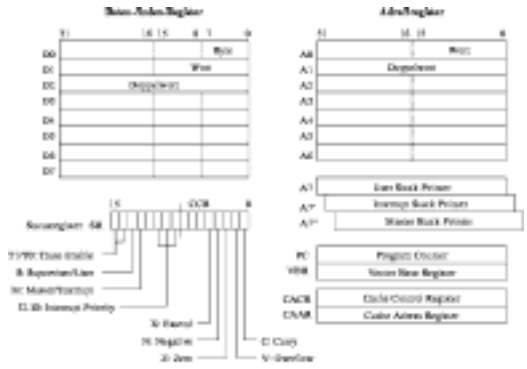
- Argumente gegen CISC:
  - Hardware ist **nicht** zuverlässiger als Software
    - aber schwieriger abzuändern bei Fehlern
  - Cache-Strategien für Programmspeicher sehr gut
    - fast alle Zugriffe auf Programmteile in einem Mikroprogrammzyklus möglich

Beispiele

- **Motorola MC68000 (1979)**
  - 16-Bit Prozessor, interne Struktur 32-Bit
  - Charakterisierung
    - allgemeiner Registersatz
    - 2 Stacks
    - 2 Betriebszustände
    - privilegierte Befehle
    - Adressraum von 16 MByte

Motorola (2)

- für Anwender benutzbare Register
  - 8 32-Bit Datenregister (D0 bis D7)
  - 7 32-Bit Adreßregister (A0 bis A6)
  - 2 32-Bit-Stackpointerregister (A7 und A7')
  - 1 16-Bit Statusregister
  - 1 32-Bit Befehlsregister
- Datenformate
  - Byte (8 Bit), Wort (16 Bit), Doppelwort (32 Bit)
  - Doppelworttransport erfordert zwei Zugriffe
    - da 16-Bit Datenbus
  - Wörter und Doppelwörter an geraden Adressen

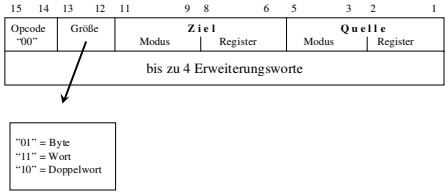


Motorola (3)

- Addressierungsarten
  - 14 werden unterstützt
  - Addressierungart nicht an Befehlstyp gebunden
- Befehlsformate und Befehlssatz
  - über 70 Befehle
  - bestehen aus 1 bis 5 16-Bit Wörtern

Motorola (4)

- Format der 68000-Maschinenbefehle am Beispiel des MOVE-Befehls



MOVE-Befehle des 68000

Modus	Register-Feld	Erweit.-worte	Assembler-Notation	Adressierungstyp	effektiver Operand ggf. Seiteneffekt
"000"	n	0	Dn	Register-Adressierung	D[n]
"001"	n	0	An	(Adreß-)Register-Adress.	A[n]
"010"	n	0	(An)	(Adreß-)Register indir.	Speicher[A[n]]
"011"	n	0	(An)+	(Adreß-)Register indir. mit Postincrement(-1,-2,4	Speicher[A[n]]; A[n]:=A[n]+i
"100"	n	0	-(An)	(Adreß-)Register indir. mit Prädecrement	A[n]:=A[n]-i; Speicher[A[n]]
"101"	n	1	d(An)	Relative A. mit 16-Bit Distanz	Speicher[d+A[n]]
"110"	n	1	d(An,Xm)	Register-Relative Adr. mit Index: Xm kann Daten- oder Adreßregister sein	Speicher[d+A[n]+X[m]]
"111"	"000"	1	d	direkte Adressierung	Speicher[d]
"111"	"001"	2	d	direkte Adr. (32 Bit)	Speicher[d]
"111"	"010"	1	d(PC),d(*)	Programmzähler-relativ	Speicher[PC+d+2]
"111"	"011"	1	d(*,Xn)	Programmzähler-relativ mit Index	Speicher[PC+d+2+X[n]]
"111"	"100"	1-2	#zahl	unmittelbare Adressierung	zahl

Intel 8086

- aufwärtskompartibel zu 8-Bit Prozessor 8080A/8085
- für 8- und 16-Bit Verarbeitung ausgelegt
- Charakterisierung
  - segmentierte Speicheradressierung
  - dynamisches Verschieben von Programm-, Daten- und Speicherbereichen
  - Interrupt-System
  - Unterstützung von Mehrprozessorbetrieb
  - Adreßraum von 1 MByte

Intel (2)

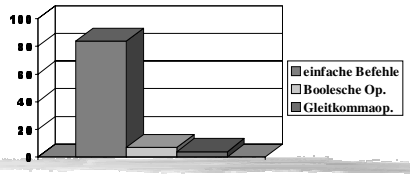
- für Anwender benutzbare Register
  - 4 allgemeine Register
  - 4 Pointer- und Indexregister
  - 4 Segmentregister
  - 1Statusregister
  - 1Befehlszähler
- im Gegensatz zu MC68000 Verwendungsmöglichkeit der Register eingeschränkt und genau festgelegt

Intel (3)

- Datenformate
  - arithmetische und logische Operationen können Byte- oder wortweise durchgeführt werden
  - weitere Datenformate
    - dezimale Operanden in BCD-Darstellung (gepackte Form) und ASCII-Darstellung (ungepackte Form)
    - Byte- und Wortketten
- Adressierungsarten
  - 24 werden unterstützt
- Befehlsformate und Befehlssatz
  - 77 Befehle und bestehen aus 1 bis 6 Byte

CISC-Probleme (1)

- komplexe Befehle werden (von Compiler) kaum genutzt
- Statistik auf DEC VAX 11/780



CISC-Probleme (2)

- Mehr als 90% der ausgeführten Befehle sind **einfach**
  - können schnell ausgeführt werden
- Unterstützung der restlichen 10% durch komplexe Maschinenbefehle **erhöht** idR Ausführungszeit der 90%
  - 10% der auszuführenden Befehle schneller machen zu Lasten der übrigen 90%?

Designprinzip schneller Rechner

- Was sind die **Schlüsseloperationen**?
- **Datenpfadentwurf** für diese Operationen
- Entwurf der Maschinenoperationen
  - jede Operation in einem Datenpfadzyklus ausfürbar -> kein Mikroprogramm vonnöten
- Erweiterung des Befehlssatzes nur dann, wenn Maschine dadurch nicht langsamer wird

RISC (1)

- Nur **wichtige** Befehle werden realisiert
- **RISC** kein neuer Prozessortyp
  - idR von-Neumann-Architektur
- **Kleine** Befehlsätze (max. 100 Befehle)
- Wenige (notwendige) Adressierungsarten
  - nur zwei Befehle für Speicherzugriff LOAD/STORE-Architektur
- Verzicht auf Mikrocode
  - festverdrahtetes Steuerwerk

RISC (2)

- Befehle in **einem** Zyklus verarbeiten
- Aufwandsverlagerung von Hardware in die Compiler
- **Synergie** zwischen Hardware und Compiler (anstatt interpretierende Mikroprogramme)
- Großer interner Registersatz
- Leistungsfähige Speicherhierarchie

RISC (3)

- Einsparung des Mikrobefehlssatzes
  - Performance-Gewinn durch Einsparung einer Transformationsebene
- Schnelles Dekodieren der Befehle
  - durch einfaches Befehlsformat
- Schnelle Ausführung
  - Register-orientiert durch LOAD/STORE
  - meisten Instruktionen in einem Zyklus ausführbar

RISC (4)

- Optimierende Compiler
  - wegen einfacher Hardware
- *Kürzere Entwurfszeit*
  - einfaches Design
- Große Speicher verfügbar
  - mehr speicherverbrauchende Ansätze von Architekturlösungen werden realisiert
  - Optimierung der Zeiteffizienz anstelle der Speichereffizienz

Beispiel:

- **MIPS RS2000/3000 (1982-89)**
  - 32-bit RISC Prozessor
  - 32 weitgehend homogene 32-bit Register (bis auf Register mit Nr. 0 und 1)
  - 32 32-bit floating-point-Register (einzeln für single precision, paarweise für double precision)

MIPS (2)

- LOAD/STORE Architektur
- alle Befehle haben 32-bit Format
- arithm. Befehle sind vom Typ RRR (3-Adress-Befehle auf Reg. Beschränkt)
- Assembler hat Pseudo-Befehle (werden durch Sequenzen von Maschinenbefehlen ersetzt)

MIPS(3)

Befehlsformate der Rechner MIPS R2000/3000

	Größe [bit]	6	5	5	5	5	6
Arithmetische Befehle	Format R	op	rs	rt	rd	shamt	funct
Immediate + LOAD/STORE	Format I	op	rs	rt	adr-teil		
Bedingte Sprungbefehle	Format J	op	Sprungadresse				

RISC versus CISC (1)

- Einfache Instruktionen (in einem Zyklus ausführbar)
- Befehle werden durch Hardware ausgeführt
- Einfache Instruktionsformate
- Zum Teil komplexe Instruktionen (benötigen z.T. mehrere Zyklen)
- Einsatz eines Mikroprogramms
- Variable Instruktionsformate

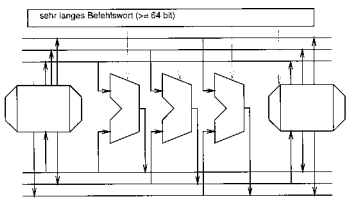
RISC versus CISC (2)

- LOAD/STORE-Architektur
- Nur wenige Instruktionen
- Komplexität in den Compilern
- Viele Befehle dürfen Hauptspeicher adressieren(z.T. sehr aufwendige Adressierungsmodi)
- Viele Instruktionen
- Komplexität in der Hardware

Weitere Mikroprozessorklassen Klassen von Befehlssätzen

- DSP-Befehlssätze
  - eingesetzt in eingebetteten Systemen
  - angepaßt an die spezifischen Aufgaben
- EPIC- und VLIW- Befehlssätze
  - größere Zahl von funktionellen Einheiten
  - Speicher mit mehreren Ports
  - „breite" Befehle steuern alle Einheiten gleichzeitig

VLIW-Architektur (Beispiel)



Weitere Mikroprozessorklassen Klassen von Befehlssätzen

- Multimedia-Befehle
  - Erweiterung bei RISC und CISC
- ASIPs
- Abstrakte Maschinen
  - Java Abstract Machine

Kap.2 Befehlsschnittstelle

Prozessoren, externe Sicht

- **2.1** elementare Datentypen, Operationen
- **2.2** logische Speicherorganisation
- **2.3** Maschinenbefehlssatz
- **2.4** Klassifikation von Befehlssätzen
- **2.5** Unterbrechungen
- **2.6** Prozesse

2.5 Unterbrechungen

- Situationen während der Bearbeitung von Programmen, die besondere Behandlung erfordern

Exceptions / Traps

Definition
prozessorinternes Ereignis, welches den normalen Programmablauf unterbricht, heißt **Exception**.

Beispiele
- arithmetischer Überlauf
- Division durch Null
- ungültiger Befehlscode
- ...

Wie werden Exceptions behandelt?

Szenario
Der gerade aktuelle Prozessorbefehl verursachte die Ausnahmesituation.

Behandlung
Je nach Art der Exception und des Prozessors gibt es zwei Möglichkeiten:
- Ein bestimmtes *Flag* in einem *Register* wird gesetzt.
  - Der Programmierer muss den Inhalt des Flags im folgenden Programm auswerten und entsprechend reagieren.
- Es wird eine definierte Ausnahmeroutine für eine bestimmte Exception gestartet (siehe Interrupt).

Interrupts

Definition
externes Ereignis, welches den Programmablauf verändert, heißt **Interrupt**

Interrupts werden durch asynchron zum Prozessor laufende externe Geräte erzeugt und dienen zur Kommunikation mit dem Prozessor.

Beispiele:
- Eingabe über Tastatur/Maus
- Grafikkarte benötigt neue Daten
- Datenübertragung (Festplatte,Netzwerk) beendet
- ...

Interrupts und Exceptions

Zur Namensgebung eine Bemerkung

Interrupts wurden ursprünglich für die Behandlung von Anfragen externer Geräte eingeführt. Derselbe Mechanismus kann auch zur Behandlung von intern generierten Exceptions genutzt werden.

Ereignis	ausgelöst	Bezeichnung
arithmetischer Überlauf	intern	Exception
undefinierter Opcode	intern	Exception
I/O-Request	extern	Interrupt
Hardware-Fehlfunktion	extern	Exception oder Interrupt

Interrupts und Exceptions

Wesentlicher Unterschied

- Exceptions sind **synchron** zum ausgeführten Programm,
- Interrupts hingegen **asynchron**

➢ Exceptions sind reproduzierbar, indem bei gleichem Systemstatus das Programm mit den gleichen Eingaben wiederholt gestartet wird.

Bearbeitung einer Exception

- wie die Interrupts **durch das Betriebssystem**
  (... siehe nächste Folie)
- **durch die Hardware**
  - ➢ **Vorteil gegenüber Softwarelösung**
    - Überprüfung auf Vorliegen einer Exception kann in der Regel im gleichen Zyklus erfolgen, in dem der entsprechende Maschinenbefehl abgearbeitet wird.
    - schnellere Abarbeitung der Befehle

Wie werden Interrupts behandelt?

- Ein Interruptwird asynchron zum gerade laufenden Programm ausgelöst und erfordert eine Behandlung durch das Betriebssystem.
- Es wird eine spezielle Ausnahmeroutine, die **Unterbrechungsbehandlungs-routine** oder auch **Interrupthandler** genannt, aufgerufen.

Beispiel

Der Nutzer drückt eine Taste der Tastatur

- CPU unterbricht aktuelles Programm,
- Tastaturcode wird überprüft,
- weitere Aktionen (z.B. nach Ctrl-Alt-Del) oder Speicherung im Tastaturpuffer,
- Tätigkeit wird an unterbrochener Stelle im alten Zustand fortgesetzt

Interrupthandler: Ursache der Ausnahme bestimmen

Das Betriebssystem benötigt Informationen über die Ursache der Ausnahme.

Zwei mögliche Varianten:

- **Causeregister**
  Bei Auftreten eines Interrupts wird ein bestimmtes Bit (*Flag*) im Causeregister gesetzt. Dieses kann vom Handler abgefragt werden.
- **Vectored Interrupt**
  Dem Betriebssystem wird ein Index aus der **Interrupttabelle** bereitgestellt, in der die Adressen für die zu speziellen Interrupts gehörenden Routinen (**ISR**: **I**nterrupt-**S**ervice-**R**outine) gespeichert sind. Im Ausnahmefall wird die entsprechende Routine aufgerufen und durch den Befehl **RTI** (**R**eturn-**f**rom-**I**nterrupt) verlassen.

MIPS: Exception-Codes im Cause-Register

- CPU, der sogenannte Koprozessor 0, verfügt über spezielle Register für die Ausnahmebehandlung

Register	Nr.	Erläuterung
Status	12	Interrupt Maske und Enable Bits
Cause	13	Exception Type
EPC	14	Adresse des Befehls, der die Exception auslöste

Im Falle einer Exception erfolgt ein Sprung an eine von der CPU-Hardware definierte Adresse (SPIM: 0x8000 0080). Der Exception Code kann nun dem Cause-Register des Koprozessors 0 entnommen werden (Bit 5-2)

MIPS Exception-Codes

Code	Name	Erläuterung
0	INT	**Int**errupt (externe Unterbrechung)
4	ADDRL	**Addr**ess error during **l**oading
5	ADDRS	**Addr**ess error during **s**toring
6	IBUS	Busfehler beim Laden des Befehls
7	DBUS	Busfehler beim Speichern von Daten
8	SYSCALL	Betriebssystemaufruf durch SYSCALL
9	BKPT	**B**rea**kp**oin**t** erreicht
10	RI	**R**eserved **I**nstruction: Verwendung eines priviligierten Befehls
12	OVF	**Ov**erflow: arithmetischer Überlauf

Interrupt-Tabelle (Vektortabelle)

- enthält die Startadressen der **ISR**,
  - jede Zeile muss jeweils eine 4-Byte Adresse aufnehmen können
- Die Indizes werden als **Interrupt-Vektornummern** (**IVN**) oder mit **INT** bezeichnet
- liegt oftmals auf den ersten Adressen des Hauptspeichers

INTEL CPU Exception Table

INT	Function
0	Divide by zero
1	Single step (→ debugging)
2	Non-maskable interrupt (NNI)
3	Breakpoint
4	Overflow
5	Bound range exceeded
6	Invalid opcode
7	Coprocesor not available
8	Double fault exception
9	Coprocessor segment overrun
A	Invalid task state segment (?)
B	Segment is not present
C	Stack exception (Überlauf)
D	General protection exception
E	Page fault
F	Reserved
10	Coprocessor error

IBM PC (Hardware) Interrupt-Tabelle

INT	Function
0	Timer (55 ms – Intervall)
1	Keyboard
3	COM2 and COM4 (serieller Port)
4	COM1 and COM3 (serieller Port)
5	LPT2 (paralleler Port)
6	Floppy Disk
7	LPT1 (paralleler Port)
8	Real Time Clock (CMOS Clock)
C	PS2 – Mausport
D	Numeric Coprocessor Error
E	IDE0 – Festplatte / CDROM / ...
F	IDE1 – Festplatte / CDROM / ...

Maskierte / Nichtmaskierte Interrupts

Prozessoren unterscheiden zwischen maskierten und nichtmaskierten Interrupts:

- **maskierter Interrupt:**
  - wird nur dann ausgeführt, wenn das **I**nterrupt **E**nable Bit (**IE**) in einem dafür vorgesehenen Register gesetzt ist. (MIPS: Bits 8-15 im Status-Register)
  - Der Nutzer kann den Registerinhalt ändern.
- **nichtmaskierter Interrupt (NMI):**
  - wird auf jeden Fall ausgeführt. Bei NMI handelt es sich üblicherweise um Ausnahmesituationen, die die Funktionalität des Systems gefährden.

Interrupt in Interrupt-Service-Routine

Szenario

Es besteht eine gewisse Wahrscheinlichkeit, daß innerhalb einer ISR (z.B. durch ein I/O-Device verursacht) ein weiteres I/O-Device seinen Interrupt auslöst!

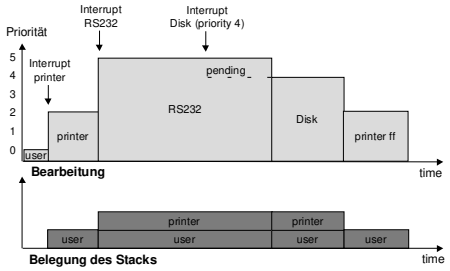
Mögliche Lösungen

- Erster Befehl in der ISR ist Ausmaskierung aller Interrupts
  - keine weiteren Interrupts möglich
- Vergabe von Interruptprioritäten derart, daß ein kritisches I/O-Device eine höhere Priorität, als ein weniger kritisches Device hat.

Interrupt-Priorität

- Falls ein Device mit Priorität *n* einen Interrupt auslöst, so startet die ISR mit Priorität *n* . Wird ein Interrupt mit einer Priorität *m>n* ausgelöst, so wird die aktuelle ISR unterbrochen und eine neue ISR der Priorität *m* sofort gestartet.
- Der Versuch eines Gerätes mit einer niedrigeren Priorität als *n* die ISR zu unterbrechen, wird ignoriert und der Interruptcode wird zur späteren Ausführung zwischengespeichert (**pending Interrupt**).
- Nach Abarbeitung der ISR geht das System auf die user-Priorität (**userpriority 0**) zurück.

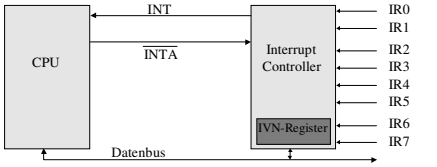
Interrupt-Prioritäten: Illustration



Interrupt-Prioritäten

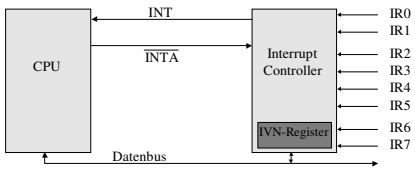
- Prozessoren mit nur einem **Interruptlevel** (x86) ist es selbst nicht möglich, verschiedenen Geräten verschiedene Prioritäten zuzuordnen.
- Abhilfe schafft in diesem Fall ein **Interrupt-Controller**, der extern mit dem Prozessor verbunden wird (z.B. 8259A).
- Dieser externe Controller verwaltet die Prioritäten. Für den Fall dass keine ISR aktiv ist oder ein Interrupt eine laufende ISR unterbrechen darf, signalisiert er dem Prozessor einen Interrupt, falls ein solcher bei ihm anliegt
- Da er vor dem Auslösen eines weiteren Interrupts mit niedriger Priorität wissen muss, ob die ISR beendet wurde, benötigt er eine Rückmeldung vom Prozessor.

Interrupt-Controller



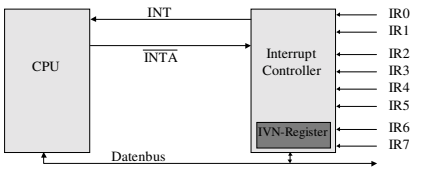
- I/O-Device *j* teilt dem Controller durch Setzen der Signalleitung IR*j* mit, daß eine Ausnahmesituation vorliegt.
- Die Kennung des Interrupts wird in das Register **IVN** eingetragen.
- Der Controller gibt die Meldung durch Setzen des Signals INT an die CPU weiter.
- Ist die CPU bereit, so teilt sie dies dem Controller durch Setzen des Signals INTA mit.
- Der Interrupt-Controller legt den Inhalt seines IVN-Registers auf den Datenbus.

Interrupt-Controller ff



- Der Interrupthandler der CPU nimmt den Inhalt des IVN-Registers vom Bus und rettet ihn in einem eigenen Register. Hiermit weiß der Prozessor, welches periphere Gerät den Interrupt ausgelöst hat.
- Vor der Abarbeitung des Interrupts rettet die CPU den Befehlszähler, sowie Steuer- und Statusregister auf dem Stack.
- Die CPU schlägt in der Interruptvektortabelle nach, um mit Hilfe der IVN die Startadresse der entsprechenden Interrupt-Serviceroutine (ISR) zu finden.

Interrupt-Controller ff



- Die CPU führt die ISR aus. Falls sie ISR Register verwendet, so muß sie deren Inhalt vorher auf dem Stack sichern.
- Vor Beendigung der ISR holt diese die geretteten Registerinhalte vom Stack.
- Die CPU stellt den ehemaligen Zustand (Befehlszähler, Steuer- und Statusregister) wieder her.
- Dem Interrupt-Controller wird die Abarbeitung des Interrupts mitgeteilt.

IBM PC (Hardware) Interrupt-Tabelle

geordnet nach Priorität

INT	function
0	Timer (55 ms Intervall)
1	Keyboard service
8	Real time clock
9	-- Einsteckplätze
10	-- Einsteckplätze
11	-- Einsteckplätze
12	PS2 – Mouseport
13	Numeric coprocessor error
14	IDE0 – Festplatte / CDROM / ...
15	IDE1 – Festplatte / CDROM / ...
3	COM2 oder COM4 (serieller Port)
4	COM1 oder COM3 (serieller Port)
5	LPT2 (paralleler Port)
6	Floppy Disk
7	LPT1 (paralleler Port)

Kap.2 Befehlsschnittstelle

Prozessoren, externe Sicht

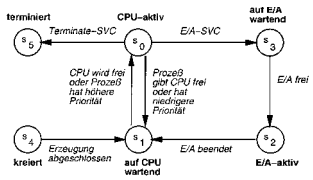
- **2.1** elementare Datentypen, Operationen
- **2.2** logische Speicherorganisation
- **2.3** Maschinenbefehlssatz
- **2.4** Klassifikation von Befehlssätzen
- **2.5** Unterbrechungen
- **2.6** Prozesse

2.6 Prozesse

- unabhängige oder lose gekoppelte Aufgaben werden als Prozesse modelliert
- Prozesse werden „parallel" bearbeitet
- Mehrprozessbetrieb durch Dispatcher
  - Zuteilung des Prozessors
  - Prozesswechsel

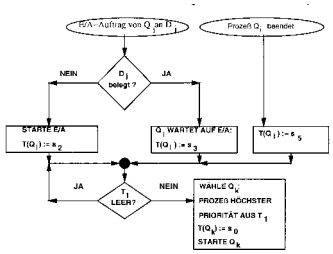
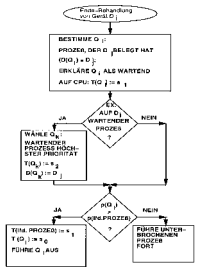
- Übergänge zwischen Prozesszuständen sind ereignisgesteuert, ausgelöst durch
  - E/A Aufträge
  - Timer-Interrupts, EXIT
  - Abschluss der Prozess-Initialisierung
  - Abschluss E/A Auftrag
  - Freigabe des Prozessors

Übergänge zwischen Prozesszuständen



- Arbeitsweise des Dispatchers
- Prozesskontrollblock (PCB)
  - Daten für Dispatcher zur Verwaltung der Prozesse
    - Prozesszustand, -kennung, -Priorität
    - Registerinhalte
    - Speicherverwaltungsdaten
    - E/A-Status

Behandlung von E/A-Aufträgen und Prozess-Terminierungen



Ende-Behandlung von E/A-Aufträgen