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 Abschittes)

- 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 Register-speicher
  - 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 Rückwärts-sprünge
- 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 Anwesiung

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) undtrap-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)

I (>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)

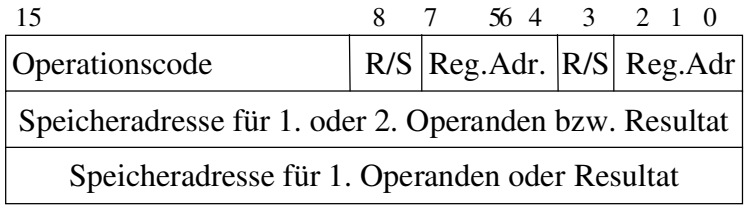
- Stack-Maschine: alle arithmetischen Operationen werden auf einem Keller ohne explizite Angabe der Operanden durchgeführt, dabei werden jeweils die beiden obersten Stackelemente verknüpft
- PUSH, POP mit Adressen
- einfache Übersetzung arithm. Ausdrücke, weitere Optimierungen nicht möglich
- Beispiel: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