

Gesamtübersicht

- Informationsstrukturen
- Maschinenbefehle
- Adreß- und Registermodelle
- Adreßrechnung
- Speicherverwaltung
- Prozessoren
- Caches und TLBs

Informationsstrukturen

Die Bedeutung der Informationsstrukturen

... wird lediglich durch Anwendung der entsprechenden Maschinenbefehle bestimmt. Für sich gesehen sind alle Informationsstrukturen nur Bitketten.

Alternativen:

- Anfügen von Bedeutungskennzeichen (TAGs). Aus der Mode.
- Beschreibung der Bedeutung in Descriptoren (objektorientierte Zugriffsweise). Hat nur dann Bedeutung für die Architektur, wenn von Grund auf in Hardware unterstützt. Aus der Mode. (Auch JVM arbeitet nicht so.) Ansonsten hat Objektorientierung vor allem den Charakter einer Programmier- und Ordnungshilfe - trotz allen Rummels...

Elementare Datenstrukturen

- vorzeichenlose (natürliche) Binärzahlen (unsigned),
- ganze Binärzahlen (signed, Integers).

Zahlendarstellung:

- Wert und Vorzeichen,
- Einerkomplement,
- Zweierkomplement (Stand der Technik). Weshalb? - Wegen der Einfachheit:
 - Vorzeichen gilt als Binärstelle und muß nicht extra behandelt werden,
 - es ist nicht erforderlich, aufgrund der Vorzeichen und einer Größer-Kleiner-Betrachtung zu entscheiden, ob addiert oder subtrahiert werden soll,
 - Addition und Subtraktion im Zweierkomplement liefern bei gleichen Operandenbitmustern gleiche Resultate, unabhängig davon, ob es sich um vorzeichenlose oder vorzeichenbehaftete Zahlen handelt (nur Frage der Interpretation bzw. der Auswertung der Flagbits).

Für Vergleichen und Überlauferkennung von Bedeutung:

- bei vorzeichenlosen Zahlen: Ausgangsübertrag (CF) und Ergebnis = 0 (ZF),
- bei ganzen Zahlen: Vorzeichen (SF), Überlauf (OF) und Ergebnis = 0 (ZF).

Achtung: nachsehen, was CF bedeutet: Zweierkomplement-Ausgangsübertrag oder eine allgemeine Überlaufanzeige. *Frage:* wie wird sie gesetzt?

Wertebereich ganzer Zahlen (n Bits):

- größte (positive Zahl): $2^{n-1}-1 = 011...1B$,
- größte negative Zahl: $-1 = 111...1B$
- kleinste negative Zahl: $-2^{n-1} = 100...0B$

Rechnen mit längeren Zahlen (ADD/SUB):

Nach der ersten Verknüpfung Befehle mit Einspeisung des Ausgangsübertrages verwenden (ADC, SBC).

Rechnen mit kürzeren Zahlen

Auf Vorzeichenerweiterung achten. Vorzeichenstelle wird in alle höherwertigen Stellen eingetragen.

Multiplikation:

Aus Multiplikand und Multiplikator ergibt sich ein Produkt doppelter Länge (genauer: Produktlänge = Multiplikandenlänge + Multiplikatorlänge).

Verkürzung der Multiplikation (Software oder Mikroprogramm oder sequentiell gesteuerte Schleife):

- den kleineren Operanden als Multiplikator nehmen,
- die höchstwertige Eins im Multiplikanden auffinden und das Erreichen dieser Bitposition als Abbruchbedingung nutzen.

Die niedere Hälfte des Produkts hat das gleiche Bitmuster, gleichgültig ob vozeichenlos oder ganzzahlig multipliziert wird (nur Interpretationsfrage). Anwendung: zur Adreßrechnung kommt man notfalls mit ganzzahliger Multiplikation aus.

Division:

Dividend hat doppelte Länge des Divisors, Quotient hat Länge des Divisors. Befehl liefert typischerweise entweder Quotient oder Rest (Befehle DIV und REM).

Gleitkommazahlen

Näherungsweise Darstellung reeller Zahlen. Bestehen aus Vorzeichen, Exponent und Signifikand (Mantisse). Halblogarithmische Darstellung. Standard: IEEE 754-85. Binärzahlen. Darstellung: Vorzeichen + Wert. Längen: 32, 64, 80, 128 Bits. Grundoperationen (ADD, SUB, MUL, DIV usw.) liefern aus Operanden einer bestimmten Länge ein Ergebnis gleicher Länge. Rundung gehört zur Operation.

Festkomma-Arithmetik:

Gibt es in der Hardware nicht. Festkommarechnung muß stets auf Grundlage der Integer-Befehle ausprogrammiert werden. Formatbeispiele:

- der Wertebereich der Zahlendarstellung gibt das Einheitsintervall wieder ($-1...+1$),
- im 32-Bit-Wort wird das Komma in die Mitte gelegt, die höheren 16 Bits repräsentieren eine ganze Zahl, die niederen einen Binärbruch.

Dezimalzahlen

Heutzutage nur als BCD-Zahlen (Binary Coded Decimals), die wie Zeichenketten behandelt werden (variable Länge). Gepackte und ungepackte Darstellung. In der Hardware nur nachlässig unterstützt.

Alternative:

Rechnen mit rationalen (gebrochenen) Zahlen (auf Grundlage der Integer-Befehle auszuprogrammieren). Viel schneller als BCD, aber inkompatibel zu vorhandenen Datenbeständen.

Maschinenbefehle

- Befehlswirkungen
- Befehlsformate
- Adreßmodelle
- Registermodelle

Befehlswirkungen

Grundsätzlich zu unterscheiden:

- implizite Befehlswirkungen (= durch Schaltmittel in der Hardware (oder Abläufe des Interpreters) veranlaßt. Sind aus Befehlsformaten und Codierungen nicht erkennbar, sondern nur aus Beschreibung der Wirkungsweise der Hardware (Theory of Operation).
- codierte Befehlswirkungen (= durch Angaben im Befehl veranlaßt).

Die innerste Steuerschleife muß stets implizit sein (= unmittelbar in Hardware oder im Mikroprogramm oder im Interpreter (Simulator/Emulator) vergegenständlicht).

Typische Befehlswirkungen:

- Operationen
- Transporte
- Ein- und Ausgabe
- Entscheidungen über den weiteren Programmablauf (Verzweigungen)

- Bedingte Verzweigungen sind das eigentliche Kennzeichen des Universalrechners. -

- Die Leistung an sich bringen nur die Operationen. Alles andere hat nur Zubringer- und Komfortfunktionen. -

Weshalb wurden die Operationen ausgewählt, die in den heutigen Befehlslisten stehen?

Weil der Computer als Rechenmaschine erfunden wurde - demzufolge lag es nahe, die elementaren Operationen des numerischen Rechnens (= die Grundrechenarten) als Maschinenoperationen vorzusehen.

Alternativen:

- noch elementarer,
- noch komplexer (so daß sich die elementarerer Operationen als Spezialfälle ergeben).

a) noch elementarer:

- bedingte Transporte (wenn, dann...; Turing-Maschine)
- Aussagenkalkül über einen Adreßraum (Logische Maschine von Zuse). Es genügt eine einzige Verknüpfung, z. B. NAND.
- Informationswandlung über Tabellen (Table Lookup). Z. B. IBM 1620 (CADET; can't add, don't even try).
- Pauschaloperationen über Boolesche Vektoren (z. B. Carry-Save-Arithmetik). Elementare Rechenoperationen werden in ihre Bestandteile aufgelöst; diese werden dem Programmierer zur Verfügung gestellt. Kompliziertere Rechenabläufe werden auf solche Pauschaloperationen zurückgeführt (rekursive Maschinen).

b) noch komplexer:

- Spezialmaschinen,
- Unterstützung von Konstrukten höherer Programmiersprachen (IF...THEN, FOR..., WHILE... usw. als Maschinenoperationen, objektorientierte Zugriffweisen usw.),
- Unterstützung höher aggregierter regulärer Datenstrukturen (Vektoren, Matrizen usw.). Beispiel: Elementaroperationen eines universellen Matrizenkalküls. Vektoren sind Matrizen mit einer Zeile, Skalare sind Matrizen mit einer Zeile und einer Spalte. Feldrechenmaschine (Zuse), Datenstrukturmaschinen, Programmiersprache APL.

Alles, was irgendwie komplizierter ist (sei es hardwareseitig, sei es konzeptionell), ist heutzutage aus der Mode. Der aktuelle Stand der Technik:

- herkömmliche Befehlswirkungen (Grundrechenarten über elementare binäre Zahlenangaben (ganze und Gleitkommazahlen), elementare Transporte, Verzweigungen usw.)
- vergleichsweise einfache Befehlsformate (RISC)
- Pipelining ist (wegen der GHz (Marketing)) wichtiger als funktionelle Höherentwicklung
- Architektur nach wie vor auf dem Niveau der 70er/80er Jahre. Die eigentlichen Wundertaten werden vom Silizium und von den Compilern verrichtet...

Man bemüht sich, möglichst viele der herkömmlichen Operationen auszuführen (Pipelining, Superskalarmaschinen, SIMD, VLIW).

Bei der Ausarbeitung einer Befehlsliste ist es wichtiger, daß die Befehlsabläufe in ein bestimmtes Pipelining-Schema passen, als daß sie möglichst viel nutzbringende Arbeit verrichten.

Pipelining und konkurrierende Ausführung unabhängiger Abläufe nützt nur dann etwas, wenn viele gleichartige Operationen über Datenströme auszuführen sind.

Das Problem eines jeden Erfinders von Alternativen: er kann den praktischen Beweis (der Überlegenheit) nicht führen - denn dafür müßte er neben der Hardware auch vergleichbare System- und Anwendungssoftware schaffen.

Befehlsformate

Es geht um die Codierung der Angaben, die zwecks Funktionsauslösung in die Hardware einzubringen sind.

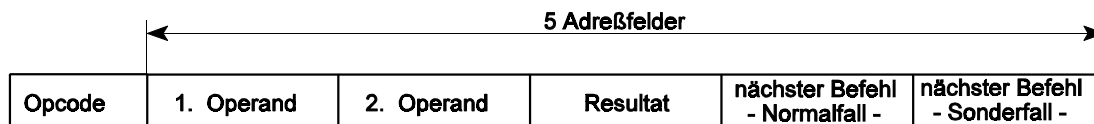
Grundsätzliche Alternativen der Funktionsauslösung:

- codegesteuert
- adreßgesteuert

Codegesteuerte Funktionsauslösung

Die Befehlswirkungen sind in Operationscodefeldern und Steuerbits des Befehls codiert. Im einfachsten Fall ist der Operationscode eine binär codierte Ordinalzahl (die einen Befehl aus der Menge aller Befehle auswählt).

Das Befehlsformat wird vor allem vom Adreß- und Registermodell bestimmt.



<Resultat> := <1. Operand> OP <2. Operand>

Registermodell:

nicht erforderlich...

Im Opcode ist auch codiert, welche Bedingungen als Normalfall und welche als Sonderfall gelten.

Operationsbefehle

Opcode	Adresse 1. Operand	Adresse 2. Operand	Adresse Resultat
--------	--------------------	--------------------	------------------

Transportbefehle

Opcode	Quelladresse	Zieladresse
--------	--------------	-------------

Verzweigungsbefehle

Opcode	Verzweigungsadresse
--------	---------------------

Unterprogrammruf:

1. Adreßrettung in fest adressierten Speicherzellen,
2. Zweiadreßformat (wie Transportbefehle) mit Verzweigungs- und Rettungsadresse

Registermodell:

Befehlszähler

Flags

Adreßgesteuerte Funktionsauslösung

Einfacher geht's nicht - es genügt ein einziger Befehlstyp. Funktionsauslösung = Transport auf bestimmte feste Adressen.

Funktionsadresse	Argumentangabe
-------------------------	-----------------------

Argumentangabe:

- Operandenadresse,
- Resultatadresse,
- Direktwert,
- Verzweigungsadresse

Adreß- und Registermodelle

Brauchen wir überhaupt ein Registermodell?

Im Grunde nein - es geht ganz ohne. Der Verzicht auf Register:

- hält die Architektur offen (den häufigen Entwurfsfehler "zuwenig Register" kann es gar nicht geben)
- erschwert Verständnis der Wirkungsweise (Stichwort: operationale Semantikdefinition)

Beispiel einer registerlosen Architekturspezifikation: die Java Virtual Machine (JVM).

Zwei Gründe, programmseitig zugängliche Register einzuführen:

- als Ordnungsmittel. Einschränkung des Programm-Kontextes bzw. Prozessorzustandes (wichtig bei Interrupts und Taskumschaltung). Reguläre, überschaubare Übergabestellen für Parameter. Verkürzung der Befehle (Registeradressen sind viel kürzer als Speicheradressen).
- als Beschleunigungsmittel (Register als Schnellspeicher). Kommt nur dann zur Wirkung, wenn Registersatz wirklich als Hardware^{*)} aufgebaut ist. Ziel: wenigstens 3 Zugriffe in jedem internen Prozessortakt (2 * Lesen, 1* Schreiben) mit der jeweils vollen Verarbeitungsbreite.

*) zum Gegenteil vgl. S/360.

Programmseitig zugängliche Register geben sowohl dem Hardware-Entwickler als auch dem Compiler-Autor klare Vorgaben in Hinsicht auf Optimierungsziele (Registersätze und ihre Zugriffswege, Placierung der Variablen).

Halten der Variablen in Registersätzen ermöglicht Ausnutzung des innewohnenden Parallelismus (rein speicherorientierte Architekturen sind demgegenüber inhärent sequentiell, teils wegen ihrer Wirkungsweise (Stackmaschine, v. Neumann-Einadreßmaschine), teils wegen des Aufwandes zum Erkennen und Ausnutzen des innewohnenden Parallelismus (Mehradreßmaschinen).

Fünfadreßmaschine:

<Resultat> := <Operand 1> **OP** <Operand 2>; nächster Befehl im Normalfall, nächster Befehl im Sonderfall

Dreiadreßmaschine:

<Resultat> := <Operand 1> **OP** <Operand 2>

Zweiadreßmaschine:

<Operand 1> := <Operand 1> **OP** <Operand 2>. Benötigt Transportbefehle.

Akkumulatur-Einadreßmaschine (v. Neumann):

<Akkumulator> := <Akkumulator> **OP** <Operand>. Benötigt implizites Register (Akkumulator) und Transportbefehle.

Zweiregister-Einadreßmaschine (Zuse Z3):

<R1> := <R1> **OP** <R2>; <R2> := 0. Implizite Registerzugriffe und Transportbefehle. 1. Laden nach R1, alle weiteren nach R2. Speichern/Ausgabe löscht R1. Danach bezieht sich der erste Ladebefehl wieder auf R1 usw. Speicheradresse 0 entspricht Register R1.

Mehrregister-Einadreßmaschine (1½Adreßmaschine):

<Register> := <Register> **OP** <Operand>. Oft verwendet (IBM 360, Intel usw.).

Klassenunterschiede in der Größe und Universalität der Registersätze.

- Nur ein Speicheroperand im Befehl erleichtert Implementierung eines virtuellen Speichers.

-

Mehrregister-Dreiadreßmaschine (Load-Store):

<Register 3> := <Register 1> **OP** <Register 2>. Universelle Lösung (RISC).

Mehrregister-Zweiadreßmaschine (Load-Store):

<Register 1> := <Register 1> **OP** <Register 2>. Gibt gelegentlich kompakteren Code (z. B. ARM THUMB). Auch P/F ist so ausgelegt.

Register: Arbeitsregister oder Schnellspeicher für Variable?

Wenn die Register als Schnellspeicher verwendet werden, kann man es sich nur selten leisten, die Variablen beim Rechnen zu überschreiben - da sie typischerweise noch gebraucht werden. Arbeitsregister müssen extra beiseite gesetzt werden. Deshalb Dreiadreßprinzip sinnvoll.

Stackmaschine:

$\langle \text{TOS} \rangle := \langle \text{TOS} \rangle \text{ OP } \langle \text{TOS}+1 \rangle$ (Aus den obersten Einträgen im Stack wird das Resultat gebildet. Die Operanden-Einträge werden aus dem Stack entfernt. Das Resultat ist der neue oberste Stack-Eintrag (Top of Stack TOS)). Beispiel: JVM.

Übungsaufgabe:

Schreiben Sie ein Kleinstprogramm für eine Stackmaschine, das folgenden Ausdruck berechnet:

$$X := A + (B * (C - (D/E)))$$

Die Befehlsliste: PUSH, POP, ADD, SUB, MUL, DIV.

Eine wichtige Stackoperation: SWAP (tauscht die beiden obersten Einträge des Stacks gegeneinander aus). Anwendung: bedarfsweise vor Operationen, die nicht kommutativ sind.

Adreßrettung beim Unterprogrammruf:

- festes Register (Linkregister)
- feste Speicherposition
- auswählbares (Universal-) Register
- auswählbare (adressierte) Speicherposition
- Hardwarestack
- Stack im Arbeitsspeicher

Vorteile des Rettens in einem Register:

- einfach
- kein Speicherzugriff
- kein Overhead, wenn nur eine Aufrufebene (keine Schachtelung)

Vorteil des Linkregisters:

Keine Rettungsadrese im Befehl nötig, da implizit genutzt. Bits stehen für längere Aufrufadresse zur Verfügung.

Verzweigungen**Sprungdistanz**

Oft unzureichend. Kompromiß zwischen Bedingungsauswahl und Sprungdistanz. Viele Befehle werden zum Schließen von Schleifen genutzt, wozu meist eine geringe Sprungdistanz (mit Bezug auf den Befehlszähler) ausreicht.

Bezugsadresse:

- absolut (lineare Adresse 0),
- Basisregister (oder Segment),
- Befehlszähler

Springen und Überspringen (Skip)

Bei zu geringer Sprungdistanz in bedingten Verzweigungsbefehlen eine passende unbedingte Verzweigung überspringen (mit der komplementären Sprungbedingung). Auf diese Weise lassen sich auch Unterprogramme bedingt aufrufen, wenn es keine bedingten CALLs gibt

Weiter weg springen durch Überspringen der komplementären Sprungbedingung:

IF carry THEN GOTO weit_weg

Ideal wäre: BRANCH_ON_CARRY, weit_weg

Ausweichlösungen (je nach Befehlsvorrat):

SKIP_ON_NO_CARRY JUMP weit_weg	BRANCH_ON_NO_CARRY, weiter JUMP weit_weg weiter: ...
-----------------------------------	--

Aufgabe:

Es soll ein Unterprogramm OUT_OF_RANGE gerufen werden, falls das CARRY-Flag gesetzt ist.

Springen mit berechneten Adressen (Funktionsverzweigung)

Indirekte Verzweigung mit Adresse aus Register

Gewährleistung der Verschieblichkeit ohne Relativadressierung (im einzelnen Programm) oder ohne ladbare Bezugsadresse (z. B. nur PC-relative Verzweigungen):

Mit Header und Ladeprogramm. Befehlsmodifikation beim Laden. Header enthält Zeiger auf die zu modifizierenden Befehle (vgl. .EXE-Dateien, DLLs).

Eine typische Verzweigung:

IF A then B else C

Test A (Liefert eine Sprungentscheidung 1/0 = JA/NEIN)

BANCH IF 1, TO_C -- gehe zu else-Zweig

B ausführen

JUMP WEITER -- else-Zweig übergehen

TO_C: C ausführen

WEITER: ...

Denksportaufgabe:

Welchen Zweig setzt der clevere Programmierer nach hinten? Weshalb?

IF A then B else if X then C else D

```

    Test A
    BRANCH IF 0, TO_NEXT
    B ausführen
    JUMP WEITER
TO_NEXT: Test X
    BRANCH IF 0, TO_D
    C ausführen
    JUMP WEITER
    TO_D: D ausführen
WEITER: ...

```

Weitere Schwerpunkte*Adreßrechnung*

- Zugriff auf globale und lokale Variable
- Zugriff auf Elemente heterogener und homogener Datenstrukturen

Stack Frames

- lokale und globale Variable

Speicherverwaltung

- Adreßverlängerung
- Schutzvorkehrungen

Virtueller Speicher

- Segmentierung
- Objektorientierung
- Seitenverwaltung (Paging)
- Strukturen der Seitenverwaltung
- Abbildungsvermögen

Problem sehr langer virtueller Adressen:

Direktumsetzung über Tabellen zu aufwendig. Adressierungsvermögen viel größer als technisch realisierbares Abbildungsvermögen.

Caches und TLBs

Abbildungsprinzipien, Abbildungsvermögen (Cacheability), Optimierungsfragen. Einzelheiten der Cache-Auslegung (z. B. bei bestimmten Prozessoren) kommen nicht dran.

Interruptsystem und Kaltstart: kommen nicht dran.