ISA-Ebene mit Beispielen aus IA-32 und SPARC V9

Die Folgende Zusammenfassung ist ein ergänzendes Handout zu meiner Präsentation „**Instruction Set Architecture mit Beispielen von IA-32 (Pentium 4) und SPARC V9 (UltraSPARC III)**“ vom 08.02.2010.

ISA allgemein:

- Schnittstelle zwischen Compiler und Hardware. Durch sie wird von der Software auf die Hardware zugegriffen.
- Definiert die Maschinensprache des Prozessortypen.
- Kompromiss zwischen tatsächlicher Maschine (Mikroprozessorebene) und zum Verständnis nötiger Visualisierung.
- Die ISA ist für die Programmierung kompakt gehalten, das bedeutet, dass für den Programmierer unwichtige Aspekte und Vorgänge innerhalb der CPU bei der ISA-Definition stark vereinfacht oder sogar weggelassen werden. Solche sind z.B. einige Aktionen oder Werte (Pipelining, Zykluszeiten…), welche der Prozessor durchführt, die der Softwareentwickler aber sowieso nicht manipulieren kann. Auch passt eine ISA-Spezifikation meist auf viele Prozessoren, wodurch der Programmierer nicht jeden Prozessor und seine Mikroarchitektur bis ins Kleinste kennen muss. Es reicht das Wissen über die ISA-Ebene.
- Konzeptionelle Struktur der Funktionsweise von:
  Organisation des programmierbaren Speichers:
  Codierungen und Darstellungen von Datentypen und Datenstrukturen;
  Den Befehlssatz:
  Der sich idealerweise effizient in vorhandene und künftige Technologien implementieren lässt.
  Modelle für Befehls- und Datenzugriffe:
  Ausnahmebedingungen, Rechte (Modi);

Ablauf:

- Der Compiler übersetzt den Code in ISA-Programm, dieses wird dann wiederum vom Mikroprogramm interpretiert.

Warum (und für wen) ist die ISA von Bedeutung?

- Compilerbauer und Betriebssystemprogrammierer benötigen Wissen über Speichermodelle, Register, Datentypen, Instruktionen usw. können aber nicht Mikroprozessorarchitektur jedes Modells einer Serie berücksichtigen.
- Gutes Design entscheidet über die Leistungsfähigkeit eines Rechners (Einfluss ca. 25%).

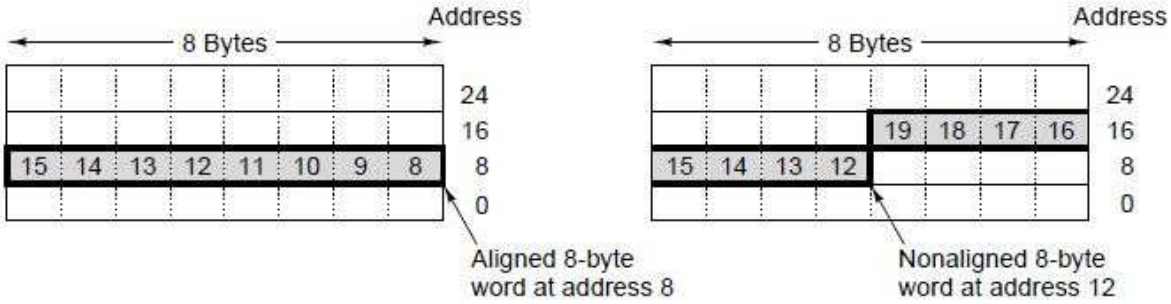
Wer legt die Spezifikationen fest?

- Spezifiziert werden die Architekturen von Industriekonsortien oder durch offizielle Definitionen (meist durch die Hersteller).

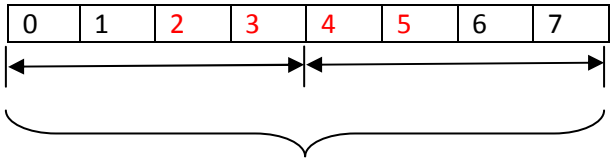
Speichermodelle:

Aufteilung in Zellen mit aufeinanderfolgenden Adressen. Zellen sind 8Bit-Folgen. Die 8 Bit einer Zelle(ein Byte) ergeben sich daraus, dass der ASCII-Zeichensatz(7Bit) + 1 Paritätsbit benutzt wird. (Falls sich Unicode durchsetzten würde, haben zukünftige Rechner 16Bit große Zellen, das würde dann einiges an Umstrukturierung verlangen).

Wörter ausrichten:



Viele Architekturen verlangen, dass Wörter an ihren natürlichen Grenzen ausgerichtet werden, da es Befehle gibt, welche ganze Worte manipulieren. Diese Befehle orientieren sich direkt an den Wortgrenzen (was äußerst performant ist). IA-32 macht dies, aus Gründen der Abwärtskompatibilität zum 8088 Prozessor(welcher die Zellen einzeln eingelesen hat (da 8-Bit-Prozessor)), nicht. Folglich muss zusätzliche Logik aufgewandt werden, um die gewünschten Daten zu erlangen:



Bei Wortgrenzenordnung müsste lediglich 0-3 oder 4-7 gelesen werden.

Bei Nichteinhalten der Ordnung, muss die doppelte Anzahl an Daten eingelesen werden, um die gewünschten Informationen „herausfiltern“ zu können.

Register:

- steuern die Programmausführung
- speichern Zwischenergebnisse (Speichern in Register ist wesentlich schneller als das Ablegen von Daten im Arbeitsspeicher)
- sind entweder sichtbar oder unsichtbar. Damit ist die Sichtbarkeit auf verschiedenen Ebenen gemeint; z.B. ist das Flag-Register (Program Status Word) bei den meisten Architekturen nur im Kernelmodus sichtbar.

Man unterscheidet zwischen…

…Spezialregistern: Spezialregistern sind spezifische Funktionen zugeordnet. Z.B. PC (**P**rogram **C**ounter: zeigt auf den nächsten auszuführenden Befehl).

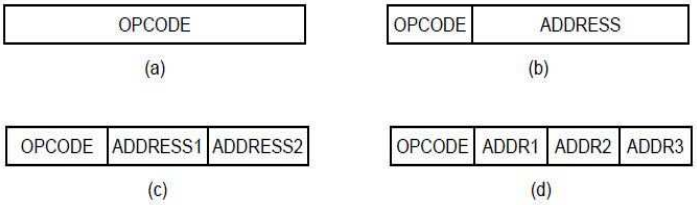
…Allgemeinen Registern: Allgemeine Register dienen zur Aufnahme von Zwischenergebnissen und wichtigen lokalen Variablen (sie werden mit Rn, also z.B. R5 beschrieben). Sie sind symmetrisch, was bedeutet, dass es egal ist ob eine Prozedur R5 oder R23 beschreibt.

…Hybriden: In den meisten Architekturen besitzen vereinzelte allgemeine Register spezifizierte Aufgaben. R5 kann also in der einen Architektur ein Spezialregister sein, ist aber in anderen Architekturen wiederrum frei verfügbar.

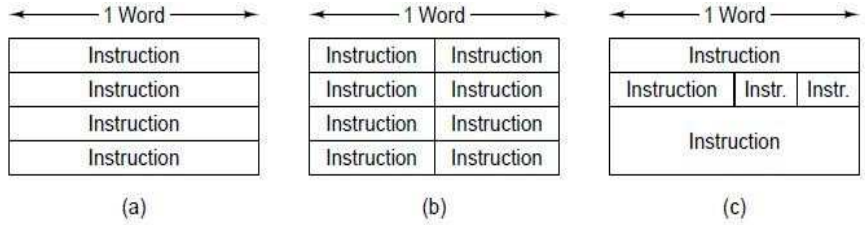
Befehlsformate:

Ein Befehl besteht aus einem Opcode und normalerweise zusätzlichen Informationen, die z.B. angeben woher Operanden kommen oder wo das Ergebnis abzulegen ist.

Wir sehen im Folgenden mehrere mögliche Formate für Befehle. Der 1. Teil eines Befehls ist immer der Opcode, der die Aufgabe des Befehls beschreibt. Ihm folgen keine (a) bis drei (d) Adressen.



Befehle entsprechen entweder Wortlängen (siehe unten (a)) oder sind kürzer/länger (siehe unten (b), (c)), je nach Architektur. Sind alle Befehle gleich lang hat das einen gewissen Vorteil, sie sind wesentlich leichter zu handhaben (da die Länge immer von vornherein bekannt ist). Allerdings sind sie somit auch richtige Platzverschwender.



Designkriterien für Befehlsformate:

Befehlslänge: Hier liegt das Hauptaugenmerk auf dem Befehls-Cache. Wenn dieser eine Bandbreite von n Bit/s besitzt, ein Befehl eine durchschnittliche Länge von m Bit hat, kann der Cache folglich höchstens n/m Bit/s liefern.

Ausreichen Platz: Im Befehlsformat muss ausreichend Platz vorhanden sein, um alle gewünschten Operationen ausdrücken zu können. -> Ausreichend Platz im Opcode um Befehl zu bezeichnen.

Anzahl der Bits: Die Anzahl der Bits im Adressfeld ist gleichzeitig die Genauigkeit, mit der der Speicher adressiert ist. Eine kurze Adresse bedeutet einen kurzen Befehl (was an sich vorteilhaft ist) aber es bedeutet auch gleichzeitig, dass z.B. ganze Worte adressiert werden und somit wieder zusätzliche Logik für das Herausfiltern von einzelnen Informationen erforderlich wird. Logik benötigt zusätzlichen Zeitaufwand.

IA-32 (Intel Architecture 32-Bit):

Spezifizierung für alle x86-Architekturen. Wobei der Befehlssatz selbst auch von AMD64 genutzt wird.

Kompatibilität:

- Ab dem 8086 (16Bit) aufwärts ist die IA-32 kompatibel (zumindest über bestimmte Modi s.u.). Enthält aber auch Überreste des 8008 (8-Bit) welcher wiederum auf dem 4004 (4-Bit) basiert.
- **Voll** kompatibel ist die Architektur ab dem 80386 (32-Bit), wobei später hinzugekommene Befehlssätze wie MMX oder SSE (1-5) ebenfalls inkludiert sind.

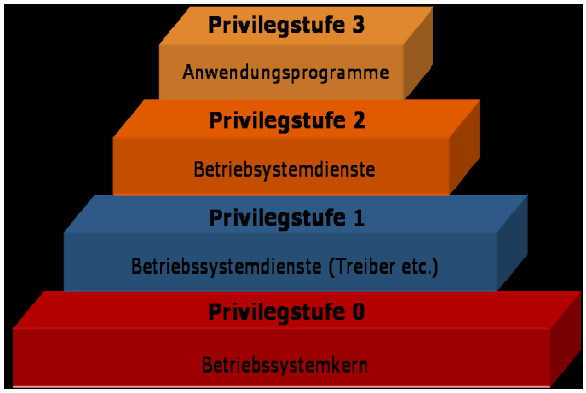
4004, 4040, 8008, 8080, 8085, 8086, 8088, 80186, 80188, 80286, i386, i486, 80386…

Der Pentium 4 kann in 3 Betriebsmodi arbeiten:

Real Mode: Er schaltet ab 8088 gemachte Neuerungen (Änderungen) aus, so dass ältere Betriebssysteme wie z.B. Dos gestartet werden können. Nachteil: Ein fehlerhaftes Programm kann den ganzen Rechner lahm legen.

Virtueller 8086 Modus: Ermöglicht das Ausführen alter 8086 – Programme (oder auch 8088) in geschützter Umgebung. Das Betriebssystem behält hierbei die Kontrolle über den ganzen Computer.
->Bei einem Fehler wird lediglich das Betriebssystem benachrichtigt und nicht gleich der Rechner zum Absturz gebracht.

Geschützter Modus: System läuft unter einer von 4 verschiedenen Privilegebenen (auch Ringe genannt), die von den PSW Bits gesteuert werden:



Ebene 0: Dieser Ring ist der Kernel Modus (in diesem arbeitet das Betriebssystem). Abstrakt gesagt entspricht er einem Vollzugriff auf die Hardware.

Ebene 1 und *Ebene 2* werden selten genutzt. Sie sind zuständig für verschiedene Betriebssystemdienste die unterschiedlichen Einschränkungen unterliegen.

Ebene 3: Ist auf Anwendungsprogramme spezialisiert. Zugriff auf bestimmte kritische Befehle und Steuerregister von Anwendungsseite sind gesperrt.

ISA-SPARC V9 (Scalable Processor ARChitecture):

SPARC war eine der ersten kommerziellen RISC-Architekturen und ist die Architektur von Sun Microsystems.
SPARC V9 (Version 9) ist die am häufigsten auftretende Sun Architektur: UltraSPARC I bis T2 und SPARC64 V bis VII. Im Speziellen bezieht sich alles Folgende auf den UltraSPARC III, weil sich auch die meisten meiner Quellen auf diesen Prozessor konzentriert haben, im Grunde sind aber alle UltraSPARC-Prozessoren auf der ISA-Ebene gleich. Die SPARC V9 ist zu allen eben genannten Prozessoren voll kompatibel, darum gibt es bei der Kompatibilität keine Abstufungen zu erwähnen.

Adressraum P4:

- Der Pentium 4 hat einen sehr großen Adressraum: 16384 Segmente mit je 2^{32} Byte. Dies liese sich auf keiner heutzutage verfügbaren Maschine implementieren. Die meisten Betriebssysteme nutzen allerdings sowieso nur 1 Segment (siehe Windows 32 Bit: maximal adressierbarer Speicher = 4GB).
- Jedes Byte besitzt seine eigene Adresse.
- Worte sind 32 Bit lang und nach dem Little-Endian-Format Adressiert. (Little-Endian-Format bedeutet: Niederwertigstes Byte hat niedrigste Adresse)

Adressraum UltraSPARC III:

- Besitzt einen linearen Adressraum von 2^{64} Byte. Das entspricht grob 18 ExaByte -> altbekanntes Problem: keine derzeit verfügbare Maschine kann so einen großen Adressraum implementieren.
- Der UltraSPARC III ist daher aktuell auf 2^{44} Byte beschränkt.

Datentypen im Vergleich:

Zeichen (char), boolesche Werte (boolean) und Zeiger existieren auf beiden der hier besprochenen Befehlssetarchitekturen.
Im Folgenden ist die jeweilige Unterstützung numerischer Datentypen gegenübergestellt:

Pentium 4

Type	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Signed integer	X	X	X	X	
Unsigned integer	X	X	X	X	
Binary coded decimal integer					
Floating point			X	X	X

Mit BCD

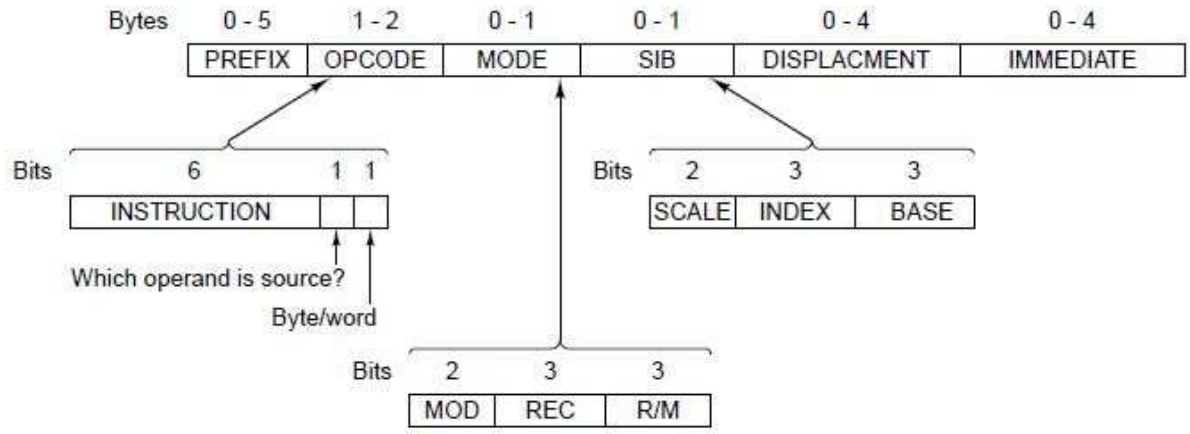
UltraSPARC III

Type	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Signed integer	X	X	X		
Unsigned integer	X	X	X		
Binary coded decimal integer	X				
Floating point			X	X	

Ohne BCD

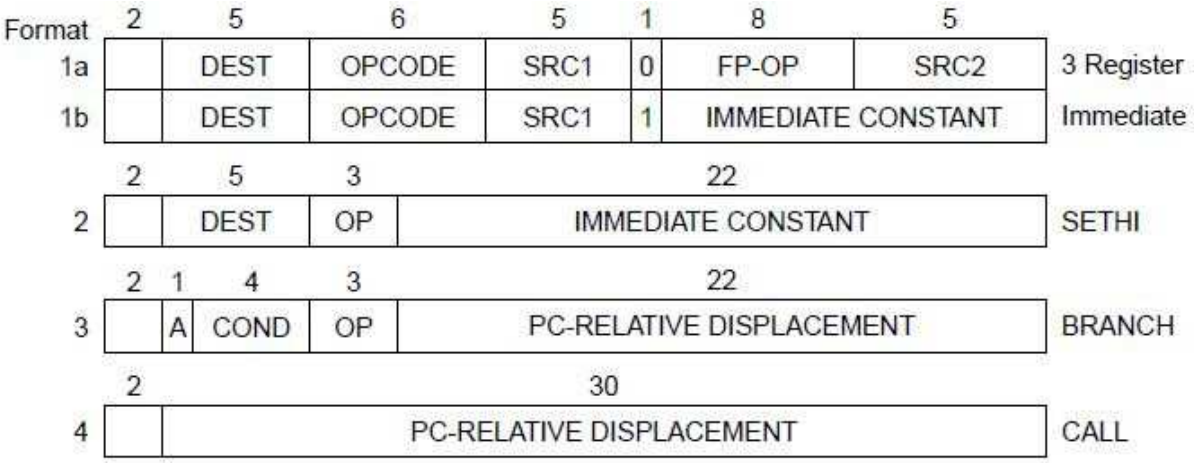
BCD (**B**inary **C**oded **D**ecimal):
Dezimalziffern werden bei diesem Datentyp einzeln dargestellt (meist 4 Bits pro Ziffer):
Das Byte 01010111 entspricht bei BCD nicht 87 sondern 57. 4 Bits pro Dezimalziffer (siehe Programmiersprache COBOL). Übrige Werte 10-15 sind nicht zugelassen.

Befehlsformate P4:



- Sehr komplex und unregelmäßig mit bis zu 6 Feldern variabler Länge wobei alle außer Opcode optional sind. Dieser Zustand ist durch die Tatsache zu erklären, dass die Architektur sich über viele Generationen hinweg entwickelt hat und somit immer wieder Verbesserungen bzw. zum Teil Verschlechterungen notwendig waren. Ginge es nach den Entwicklern würden die alten Problemverursacher von den Neuimplementierungen ausgeschlossen, doch leider finden auch nach Jahren Steinzeitrechner ihren Platz in der Welt des Kunden.
- Das Präfixbyte dient der Modifizierung von bestimmten Befehlen -> entspricht einem zusätzlichen Opcode, welcher die Wirkung des Befehls verändert. Diese Modifier sind mnemonische Extensions , z.B. für den Befehl **ADD**: ADD.A (welcher eine Umselektierung der übergebenen Operanden vornimmt).
- Die Bits des Opcode entsprechen der Beschreibung des Befehls, der Angabe über die Interpretation der Speicheradresse (entspricht in **REG** angegebenes Register Quelle oder Ziel?) und dem Bezug (entweder auf ein Byte oder auf ein Wort).
- Das MODE-Byte taucht bei Befehlen auf die einen Operanden mitliefern und beinhaltet Informationen über diesen. Es setzt sich zusammen aus MOD (das z.B. zusammen mit R/M zur Adressierungsart-Festlegung dient, oder mit Prefix zusammen zur Modifizierung von Befehlen dient), REG (das eine Registernummer enthält welche von Opcode genauer spezifiziert wird) und R/M (spezifiziert ein Register als Operand oder agiert mit MOD wie zuvor beschrieben).
- SIB: Wird bei bestimmten Modi verlangt. Es besteht aus drei Bereichen:
  SCALE (Skalierungsfaktor der mit INDEX multipliziert wird), INDEX (Registernummer des Index-Registers, welches einen Abstand (z.B. bei einem Befehl der zu einer Adresse springt, welche von der Ausgangsadresse 6 Bytes entfernt liegt) angibt) und BASE (Registernummer des Base-Registers, welches einen Konstanten Abstand enthält; also nicht mit SCALE multipliziert wird).
- DISCLAMER: Spezifiziert eine Speicheradresse (z.B. für eine Verschiebung). Hierauf kann sich SIB beziehen.
- IMMEDIATE: Enthält einen unmittelbaren Operanden. Dies ist eine literale Konstante z.B. 4#.

Befehlsformate UltraSPARC III:



32-Bit Befehle welche im Speicher ausgerichtet sind. Der Einfachheit halber sind hier nur die ursprünglichen Befehlsformate aufgeführt, mittlerweile gibt es allerdings schon 31 verschiedene.

- Die beiden ersten Bits bestimmen das Format.
- 1a) Einige Befehle spezifizieren 2 Quellregister (1. Operand (SRC1), 2. Operand(SRC2)). Beispiel: AND R1,R2,R3 -> Verknüpfe R3 mit R2 und schreibe Ergebnis in R1.
- 1b) Hier wird statt SRC2 und dem Floating Point Operator (FP-OP) eine literale Konstante übergeben.
- Bit 19 sagt bei 1 ob a oder b interpretiert werden soll.
- SETHI, BRANCH und CALL sind jeweils Befehle die ihre eigenen Formate besitzen, welche auf ihre speziellen Funktionen zugeschnitten sind. CALL ist beispielsweise ein Prozeduraufruf welcher nur eine einzige Adresse als Argument übergeben muss. Auch die hier fehlenden 26 Befehlsformate wurden spezifiziert und implementiert, weil sich einzelne Befehle in keines der ursprünglichen Befehlsformate drängen liesen.

Quellen (die 2 Hauptquellen sind fett-gedruckt)**:**

http://tams-www.informatik.uni-hamburg.de/lehre/2009ws/vorlesung/ram/doc/ram100108.pdf
„Computerarchitektur Strukturen-Konzepte-Grundlagen“ Andrew S. Tanenbaum
http://ca.itec.kit.edu/teaching/emp/ws09/emp09-01.pdf
http://www2.informatik.hu-berlin.de/rok/ca/SS05/data/slides/ca05_Befehlsformate.pdf
http://www.risc.uni-linz.ac.at/education/oldmoodle/mod/resource/view.php%3Fid=330.html
http://www.ra.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_RA/tgi2/01b-Einleitung.pdf
http://www.lrr.in.tum.de/~karlw/Karlsruhe/Mikroprozessoren/SS03/mp03-03.pdf

Beispielspezifikation: Intel IA-32 Spezifikation (z.B. Basic Architecture):
http://www.intel.com/products/processor/manuals