Die Intel-Pentium CPU

Pentium

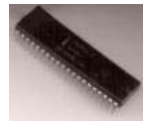
- Äußerst komplexes Ding
  - IA-32 Intel ® Architecture Software Developer's Manual
    - Vol 1: Basic Architecture: 426 Seiten
    - Vol 2: Instruction Set Reference: 976 Seiten
    - Vol 3: System programming guide: 780 Seiten
- vereinfachte Darstellung nicht sinnvoll

AK Pentium

- Ausgewählte Themen
- Pentium wichtig, da sehr verbreitet (auch kompatible Hersteller wie AMD)
- Pentium eigentlich Nachfolger des Intel 8080 aus 1975 und letztes Mitglied der 80x86-Familie

Packaging

- Intel 8080: 40-pin dual-in-line-package



- Pentium: viel größer, an die 300 bis 500 pins, 12 Watt

RISC oder CISC?

- Na ja, bei 900+ Seiten Instruction Set Dokumentation….
- Aber Einflüße der CISC-Technologie:
  - Einfachere Befehle durch Hardwired-Logic gesteuert
  - Microcode-Engine für komplexere Befehle

Pentium Busse

- Adressbus extern: 32 Bits
- Datenbus 64 Bits
- Interne CPU-Busse mit 128 oder 256 Bits
- Breitere Busse: langsamerer Speicher kann trotzdem mehr gleichzeitig anliefern

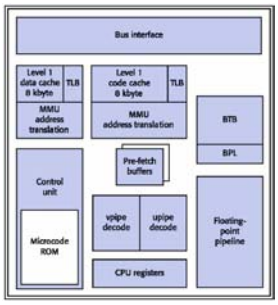
Memory

- Billiges, langsames DRAM unterstützt
- Schneller on-chip-cache: Level 1 cache:
  8KB für Daten, 8KB für Instruktionen (unterschiedliche Größen!)
- Burst access – vier Buszyklen (32 Byte, *cache line*)

Cache

- Trennung Daten/Code-Cache vorteilhaft für Parallelverarbeitung
- Zugriffe auf Cache von zwei Stufen gleichzeitig – bei zwei Caches weniger Kollisionen und Verzögerungen
- Meist 8KB Cache zu wenig – weiterer SRAM-Cache im System (L2, Level 2, zwischen L1 und Hauptspeicher)

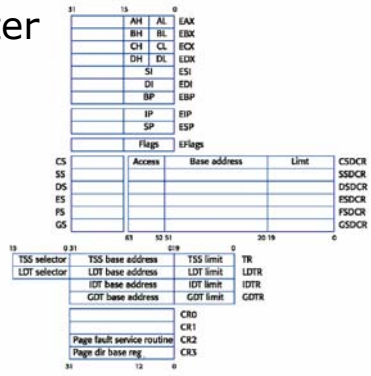
Blockdiagramm



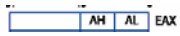
CPU-Register

- Grundlegender Registersatz unverändert seit 80386
- Viele Register mit besonderen Aufgaben

Register



EAX



- A-Register
- Auch Akkumulator
- Allzweckregister für Arithmetisch-logische Operationen
- Bei Multiplikation/Division implizit verwendet (nicht erwähnt)
- Auch in allen Datentransfers mit I/O-Ports verwickelt

EAX(2)

- Angesprochen als AL,AH (8 bit), AX (16 Bit, AL+AH)
- Beispielcode

```
MOV EAX,1234H       ; direkt laden
INC EAX             ; inkrement
CMP AL,'Q'          ; vergleich
MOV maxval, EAX     ; abspeichern
DIV DX              ; Division
```


EBX

- B-Register
- Basisregister
- Enthält Adressen, die die Startadresse einer Datenstruktur darstellen (wie zum Beispiel Arrays)

EBX

- Beispielcode

```
LEA EBX,marks      ;Adresse laden
MOV AL,[EBX]       ;EBX als Adresse
ADD EAX,EBX        ;Addition mit A
MOV EAX,table[BX] ;Index in
                   ;Tabelle
```


ECX

- C Register
- Count Register
- Als Zähler in Schleifen oder Bit-shifting-Vorgängen

ECX

Beispielcode

```
        MOV ECX,100
        …
For1:

        LOOP For1
; LOOP: DEC DCX, test for zero,
; JMP back to For1 if non-zero
```


EDX

- D Register
- Data Register
- Kann bei I/O-Vorgängen involviert sein
- Verwendet bei Integer-Multiplikation und Division
- Sonst zum Zwischenspeichern von Variablen

EDX

- Beispielcode

```
IN AL,DX    ;I/O-Adresse in DX
MUL DX      ;Multiplizere A mit DX
```


ESI

- Source Index Register
- Zeiger bei String- oder Array-Operationen im Datensegment
- Beispielcode

```
LEA ESI,dtable      ;Adresse laden
MOV AX,[EBX+ESI]    ;Basis +
                    ;Indexregister
                    ;=Adresse
```


EDI

- Destination Index Register
- Zeiger bei String- oder Array-Operationen im Datensegment
- Beispielcode

```
MOV [EDI],[ESI]
```


EBP

- Stack Base Pointer
- Stack frame pointer für Hochsprachen-Operationen
- Offset im Stack-Segment
- Beispielcode

```
ENTER 16     ; sichert EBP am Stack
             ; Kopiert ESP in EBP
             ; subtrahiert 16 vom EBP
```


EIP

- Instruction Pointer
- Program Counter
- Adresse der nächsten Instruktion im Code-Segment
- Beispielcode

```
JMP errors      ; neue Adresse in
                ; EIP
```


ESP

- Stack Pointer
- Adresse des nächsten verfügbaren Elements am Stack
- Beispielcode

```
Call sub        ; return-adresse
                ; am Stack
PUSH EAX        ; EAX auf Stack
```

EFLAG

- CPU-status-flags
- In allen bedingten Befehlen verwendet
- Beispielcode

```
JGE back   ; sprung bei ≥
LOOP back  ;
```


CS, DS, SS, ES, FS, GS

- Segment-Selektoren
- Code-Segment, Data-Segment, Stack-Segment, drei weitere Datensegmente
- Ursprünglich zur Erweiterung des Adressraumes von 16 Bit
- Um 4 Bit linksverschoben: 20 Bit (Adresse immer Segment + logische Adresse)

Und noch viele weitere…

- CSDCR-GSDCR
- TR
- IDTR
- GDTR
- LDTR
- CR3
- CR2
- MM0-MM7
- FP0-FP7

Segmentregister

- Protected mode: Teilt Hauptspeicher in verschiedene Abschnitte (Segmente)
- Werden nicht mehr direkt zur Adressbestimmung herangezogen
- Zeigen auf entsprechende Einträge in einer der Segment-Deskriptor-Tabellen (GDT, LDT)
- Eintrag enthält Startadresse eines Segmentes

Instruction set

- Pentium besitzt über 200 Instruktionen
- Verschiedene Adressierungsmodi
- Binärdarstellung zwischen einem und 15 Bytes
- „populäre" Instruktionen haben kürzere Darstellungen

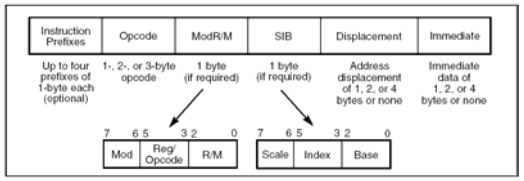
Befehlskategorien

1. Data transfer
2. Data Input/Output
3. Data manipulation
4. Transfer of control
5. Machine supervision

Struktur der Instruktionen

- Instruktionsdekodierung etwas komplexer als in unseren einfachen Beispielen
- Die meisten Instruktionen enthalten
  - Aktion
  - Operanden
  - Wohin mit dem Ergebnis

Struktur der Instruktionen



Prefixes

- Modifiziert Verhalten der Instruktion - meist Modifikation der Defaultwerte
- Bis zu vier Prefixes pro Instruktion
- Lock Prefix:
  führt Instruktion „atomar" – also ununterbrechbar – aus (kein anderer Prozessor kann auf Speicher zugreifen)

Prefixes

- Repeat-Prefix:
  Wiederholt Instruktion für jedes Element eines Strings
- Segment-override-Prefix:
  anderes Default-Segment
- Operand/Address-size Prefix:
  Andere Operanden/Adress-länge (16/32 Bit)
- Branch-Hint Prefix

Op-Code

- 1-3 Bytes
- Eventuell 3 weitere Bits im nächsten Feld (ModR/M)

ModR/M

- *Mod* + *r/m*: 32 mögliche Werte: 8 Register und 24 Adressierungs-arten
- *Reg/opcode*: Registernummer oder 3 Bits opcode (je nach primary Opcode)
- *r/m*: Register als Operand oder Zusatzbits für Mod

SIB

- Zusatzinfo zu Adressierungsart (je nach ModR/M)
- *Scale factor:* Skalierungsfaktor
- *Index field:* Register-Nummer des Indexregisters
- *Base field:* Register-Nummer des Basisregisters

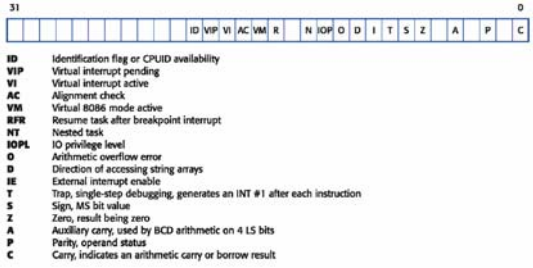
Displacement/Operand

- Bei manchen Adressierungsmodi nötig
- Folgen direkt dem ModR/M oder SIB-Byte

CPU-Statusflags

- CPU-Register
- Veränderung der Flags bei Arithmetisch/Logischen Operationen
- Keine Veränderung bei MOVs
- Statusflags wichtig für bedingte Operationen

Status-Flags



Status-Flags

- CF – Carry Flag
  bei arithmetischen Operationen: carry/borrow bei Addit./Subtr. Overflow-Bedingung für *unsigned-integer*
  - Kann durch Befehle (STC,CLC,CMC) verändert werden

Status-flags

- PF – Parity Flag
  wenn das niederwertigste Byte eine gerade Anzahl an 1-Bits hat
- ZF – Zero Flag
  bei arithmetischen Operationen: Ergebnis ist Null

Status-Flags

- SF – Sign Flag
  entspricht dem höchstwertigen Bit des Ergebnisses (Vorzeichenbit eines *signed integer*)
- OF – Overflow Flag
  bei Integer-Operationen: Ergebnis ist zu groß für eine positive / zu klein für eine negative Zahl
  Overflow-Bedingung für *signed integer*

Andere Flags

- DF – Direction Flag
  - Gibt Arbeitsrichtung von String-Instruktionen vor
- System Flags
  - IF – Interrupt enable flag
  - TF – Trap flag – fürs debuggen
  - ID – ID-Flag: CPU versteht CPUID-Instruktion

Adressierungsmodi

- Die meisten CPUs unterstützen verschiedene Adressierungsmodi
- Macht Instruktionssatz flexibler
- Unterstützt auch Hochsprachen
- Pentium: wenn zwei Operanden, auch zwei Adressierungsmodi möglich

Adressierungsmodi

- Data Register Direct
  `MOV EAX, EBX`
  - Drei Bit zur Identifikation
- Immediate Operand
  `MOV EAX, 1234`
  - Konstante wird direkt nach der Instruktion gespeichert
  - IP zeigt auf Operanden
  - Daher auch *IP indirect*

Adressierungsmodi

- Memory direct
  `MOV EAX, [variable]`
  - Adresse direkt nach der Instruktion gespeichert
  - IP zeigt auf Adresse
  - Adresse wird in temporäres Register gelesen

Adressierungsmodi

- Address register direct
  `LEA EAX, variable`
  - Lädt Memory-Adresse in Register (analog immediate)
  - Zum Initialisieren von Zeigern

Adressierungsmodi

- Register indirect

  `MOV EAX, [EBX]`

  - Register (EBX) enthält Adresse der Variablen (*zeigt auf Daten*)
  - Wert der Variablen wird geladen

Adressierungsmodi

- Indexed Register indirect with displacement

  `MOV EAX, [table+EBP+ESI]`

  - Basis-Adresse (eines Arrays), zu der Index-Registerinhalte addiert werden
  - Auch `MOV EAX, table[ESI]`

Adressierungsmodi

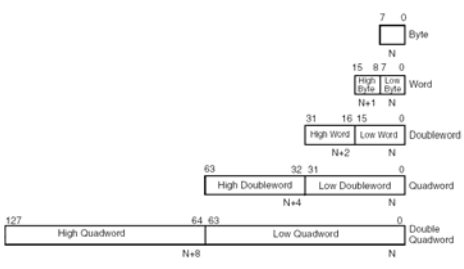
$$\underbrace{\begin{bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{bmatrix}}_{Base} + \left[\underbrace{\begin{pmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{pmatrix}}_{index} * \underbrace{\begin{pmatrix} 1 \\ 2 \\ 4 \\ 8 \end{pmatrix}}_{scale} \right] + \underbrace{\begin{bmatrix} none \\ 8-bit \\ 16-bit \\ 32-bit \end{bmatrix}}_{displacement}$$

Offset = Base + index*scale + displacement

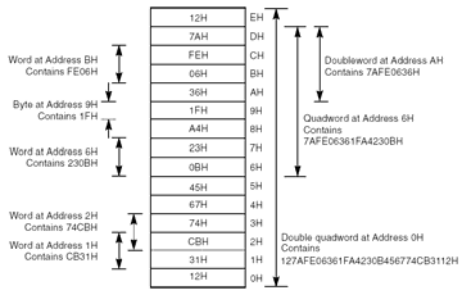
Offset

- Alle Kombinationen möglich
  - Displacement
  - Base
  - Base+Displacement
  - (Index*Scale)+Displacement
  - Base+Index+Displacement
  - Base+(Index*Scale)+Displacement

Fundamentale Datentypen



Fundamentale Datentypen



Alignment

- Words/… Double Quadwords benötigen mehrere Bytes
- Natural Boundaries:
  - Words: gerade Adressen
  - Double words: Adresse durch 4 teilbar
  - Quadwords: durch 8 teilbar

Alignment

- Alignment nicht immer erforderlich - steigert aber Geschwindigkeit
  - Zwei Speicherzugriffe für „unaligned memory access", einen für „aligned"
- Manche Instruktionen auf double quadwords erfordern allignment (0 mod 16), andere nicht.
  - General protection exception

Numerische Datentypen

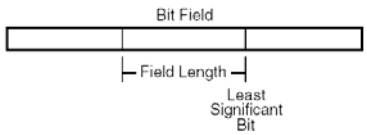
- Zusätzliche Interpretation der fundamentalen Typen
  - Byte/Word/Doubleword/Quadword unsigned integer
  - Byte/Word/Doubleword/Quadword signed integer
  - Single Precision Floating Point
  - Double Precision Floating Point
  - Double Extended Precision Floating Point

Pointer

- Near pointer
  - 32 Bit offset (*effective address*) in einem Segment
  - Für alle Speicherzugriffe in einem flachen Speichermodell
  - Referenz in segmentiertem Modell, wo Segment implizit
- Far pointer (48 Bit)
  - 16 bit Segment Selector, 32 bit offset

Andere Datentypen

- Bit-Feld
  - Beginn: beliebige Bit-Position in einem Byte
  - Max 32 Bit



Andere Datentypen

- String
  - Kontinuierliche Folge von Bytes, Words or Doublewords
  - Bit string
    - Beginn: beliebige Position in einem Byte
    - Max 2^{32}-1 Bits
  - Byte string
    - Von 0 bis 2^{32}-1 Bytes

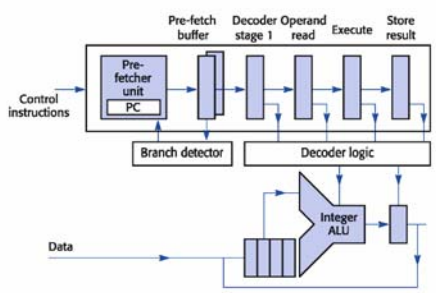
Pipelines

- Parallelverarbeitung steigert Geschwindigkeit
- *pre-fetching*: Instruktionen holen bevor sie gebraucht werden
- Partielles dekodieren und exekutieren – *pipelining*
- Bringt große Geschwindigkeitssteigerungen!

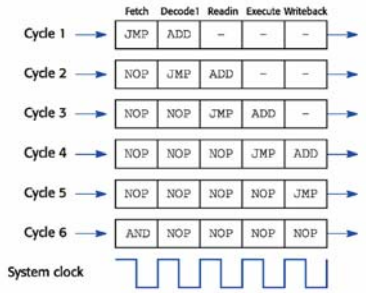
Pipelining

- Etwa 5 Zyklen pro Instruktion: 100 Mhz sind effektiv nur 20.
- Pipelining: eine Stufe pro Zyklus: bis zu fünf-fache Geschwindigkeit!
- Pipelining ist Multiprocessing!
- CPU-Hardware wird komplexer!

Pipelined decoding



Pipelining - Jumps



Superscalar processing

- Bisher: *Single instruction stream*
- Pentium kann zwei Instruktionen parallel ausführen
- Zwei Pipelines: U und V
- Nicht für floating-point Operationen – eigene Pipeline

Pipelining