1 Speicher

- in 386er Architektur 4 Byte → ausschliessliche Adressierung im 4 Byte Format
- Prozessor bietet Byte-weise Adressierung im Maschinencode an
- Probelmatik: Speicherzugriffe ueber mehrere Speicherzellen fuehren zur Ueberschreibung anderer Werte
- Loesung: Aufteilen von Ladebefehl → BE0-3 → Verhindert Ueberschreibung

1.1 Little vs. Big Endian

- kleinste adressierbare Speicherzelle ist ein Byte (8 Bits)
- unterschiedliche Moeglichkeiten zusammenhaengende Daten abzuspeichern

→ x86: Little Endian: kleinstes Byte zuerst
→ somit kann man auch 8Bit Wert einlesen falls dieser in 8Bit passt

1.2 386DX Register

1.2.1 General Purpose Register

- EAX / EBX / ECX / EDX
- ESI
- EDI
- EBP (Stack Frame Pointer)
- ESP (Stack Pointer)

- Abwaertskompatibilitaet → Adressierung der unteren 16Bit moeglich
- (untere 16Bit Register) AX / BX / CX / DX
  → Unterteilung in higher / lower 8 Bit (z.B. AH und AL)
- direkte Angabe von 16Bit aber auch 8Bit Register in Befehlen moeglich

1.2.2 Segment Registers

- CS / SS / DS / ES / FS / GS

1.2.3 Instruction Pointers and Flags

- EIP
- EFLAGS (werden zu bestimmten Zustaenden gesetzt)

2 Assemblercode

2.1 Assemblerrealisierungen

2.1.1 Intel-Syntax

- Zielregister als erstes Register
- SimpleASM (SASM) → graphische Oberflaeche zu NASM
- Compiler: Netwide Assembler (NASM), GNU Assembler

2.1.2 AT&T-Syntax

- Zielregister als letztes Register
- Register Praefix: "%"
- Adressen und Daten Praefix: "$"
- Compiler: GNU Assembler

2.2 wichtige Grundrechenarten

2.2.1 Multiplikation

- IMUL (Ziel := Ziel x Quelle) → vorzeichenlos
- MUL (EDX:EAX := EAX x Quelle) → niedere 32Bits in EAX / vorzeichenlos
- IMUL (nur mit Quelle) → wie MUL / vorzeichenbehaftet

2.2.2 Division

- DIV (EAX := (EDX:EAX) / Quelle) → Rest in EDX / vorzeichenlos
- IDIV (nur mit Quelle) → wie DIV / vorzeichenbehaftet