1 Assemblercode

1.1 Speicherbereiche / -sektionen

1.1.1 Nicht initialisierter Speicherbereich

- wird fuer Programm allokiert, aber nicht initialisiert
- wird erst durch Programm mit Werten belegt
- unter NASM: "section .bss" (alle folgenden Zeilen gehoeren dem Segment an)

1.1.2 Initialisierter Speicherbereich

- Speicherbereich wird beim Laden von Programm mit diesen Werten geladen
- z.B. globale Variable aus einer Hochsprache
- unter NASM: "section .data" (alle folgenden Zeilen gehoeren dem Segment an)

1.2 Reservieren von nicht initialisiertem Speicher

- Reservierung durch "RES(B/W/D/Q) n" (n := Anzahl der Elemente)
- Reservierung von 1-8 Byte
- alle reservierten Bloecke werden direkt aneinandergereiht

1.3 Reservieren und initialisieren von Speicher

- Speicher kann mit festgelegten Werten initialisiert werden
- Reservierung durch "D(B/W/D/Q) n" (n:= Wert passend zur Datengroesse)
- Trennung durch Komma um mehrere Werte der selben Groesse anzulegen
  → DB 72, 101, 108, 108, 111 = "Hello"
- alle reservierten Bloecke werden direkt aneinandergereiht

1.4 Marken (Labels)

- Markos zum merken bestimmter Werte / Adressen
- z.B. "parameter1: DD 123" → "MOV eax, [parameter1]"
- Laberling von reserviertem Speicher → Zugriff im Code moeglich
- Platzhalter fuer bestimmte Werte mit "EQU n" (n := Wert)

1.5 Adressierung

1.5.1 Direkte Adressierung

- direkte Adressangabe (z.B. als Konstante)
- Speicherzugriff entweder als Quelle oder Ziel nicht beidem!
- z.B. "MOV [42349], eax"
- beide Operanden muessen in der Datengroesse uebereinstimmen
  → "MOV eax, bx" geht nicht! (16Bit zu 32Bit)
- Ein-Operand Befehle benoetigen Angabe von Datengroesse (WORD/DWORD/...)

1.5.2 Indirekte Adressierung

- Register: [Register] → "MOV dx, [ebx]"
- Based mode: [Register + Displacement] → "MOV dx, [ebx + 12]"
- Based indexed mode: [Register + Register] → "MOV dx, [ebx + ecx]"

Kombinationen moeglich:

- Based indexed mode with displacement:
  [Register + Register + displacement]
  → "MOV dx, [eax + ebx + 12]"

- Based indexed mode with displacement and scaling:
  [Register + Register x Scaling + Displacement]
  → "MOV dx, [eax + ebx x 2 + 12]"

- Skalierungsfaktor muss 2er Potenz sein (2 - 8 moeglich)
- Displacement muss eine 8Bit oder 32Bit Zahl sein

→ moegliche Gefahr: Fehlerhafte Speicherzellenzugriffe

1.6 Stack

- Stack waechst von oben nach unten
- ESP zeigt auf zuletzt abgelegtes Element
- Verwendung zum Zwischenspeichern / Unterprogrammaufruf /
  Allokieren fuer kleine Speicherbereiche

→ siehe Vorlesungsskripte fuer mehr

2 Speicherzugriff

- Tradeoff zwischen Rechenleistung und Speicherzugriff
- teure Speicherzugriffe → Loesung: Caching (Zwischenspeicher)

2.1 Caching

- Idee: Schneller Speicher zum Zwischenspeichern von Werten die zeitnah wiederverwendet werden sollen
- als Cache zwischen Hauptspeicher und Rechen- / Steuerwerk bekannt

2.2 Speicherpyramide

Register → Cache → Hauptspeicher → Hintergrundspeicher → Archivspeicher

→ Anstieg der Speicherkapazitaet
→ Abfall an Performanz

3 Datenuebergabe

3.1 Registeruebergabe

+ schnellste Moeglichkeit, kein Umweg ueber Speicher

− Anzahl der Parameter auf 4 begrenzt
− Parameter sind Caller saved → sie muessen von dem Callee Programm gesichert werden → Stack Operationen werden benoetigt zum sichern

3.2 Uebergabe mit festen Speicherpositionen

+ keine Stacknutzung
+ einfache Adressierung
+ einfaches Debugging, Parameterposition bleibt konstant
+ Parameter ohne Rekursion immer auslesbar

− keine Rekursion moeglich (Parameter wuerden sich ueberschreiben)
− keine variable Anzahl von Parametern moeglich → statische Anzahl
− Speicher wird immer allokiert auch falls dieser nie genutzt wird

3.3 Stackuebergabe

+ beliebige, auch variable Anzahl an Parametern
+ beliebige Verschachtelung und Rekursion moeglich

− umstaendliche Adressierung → Stacknutzung in der Funktion benoetigt ein Art Ankerregister um die originiale Parameterposition zu speichern

→ erster Parameter wird als letztes auf den Stack gepushed analog dazu die allgemeine Parameterreihenfolge