Lernzettel

Assemblersprache, Steuerkonstrukte und Adressierungsarten

Universität: Technische Universität Berlin
Kurs/Modul: Rechnerorganisation
Erstellungsdatum: September 20, 2025

Zielorientierte Lerninhalte, kostenlos!
Entdecke zugeschnittene Materialien für deine Kurse:

https://study.AllWeCanLearn.com

Rechnerorganisation

Lernzettel: Assemblersprache, Steuerkonstrukte und Adressierungsarten

(1) Assemblersprache – Grundlagen.
Assembler bilden die Brücke zwischen Maschinensprache und menschlich lesbarem Code. Mnemonics wie MOV, ADD, CMP entsprechen Maschinenbefehlen und werden durch den Assembler in Maschinencode übersetzt.
Typische Bausteine:

- Labels (Sprungziele) und direktionale Anweisungen (ORG, DB, DW, EQU, DEFINE),
- Makros und Makrodefinitionen zur Wiederverwendung von Code,
- Symbolische Adressen statt harter Zahlen.

Zweck der Assemblersprache: feine Kontrolle über Register und Speicher, klare Abbildung auf die Architektur, schnelle Optimierung an der Mikroarchitektur.

(2) Steuerkonstrukte – Bedingungen und Sprünge.
Steuerkonstrukte regeln den Ablauf eines Programms auf der Maschinensprachebene. Wichtige Konzepte:

- Bedingte Sprünge (Verzweigungen) basieren auf Statusbits/Flags, z. B. Null-Flag, Überlauf-Flag, Trennflag,
- Vergleichsoperationen (CMP, TEST) setzen Flags je nach Ergebnis,
- Sprungarten: unbedingte Sprünge (JMP), bedingte Sprünge (JE/JZ, JNE/JNZ, JG/JNGE, JL/JLZ, JC/JNC),
- Schleifen implementieren: Zählschleifen, While-ähnliche Strukturen,
- Sprungtabellen ermöglichen schnelle Dispatch bei switch-case-ähnlichen Strukturen.

Beispiel (schematisch):

```
CMP R1, R2
BEQ DONE
BNE CONTINUE
...
JMP END
CONTINUE: ...
DONE: STOP
```

(3) Adressierungsarten – Zugriff auf Speicher und Register.
Eine klare Kenntnis der Adressierungsmodi ist zentral, um korrekt mit Daten zu arbeiten. Wichtige Modi:

- Immediate addressing: Operand direkt im Befehl, z. B. `MOV R0, #5` (das # deutet auf einen unmittelbaren Wert hin),
- Direct addressing (direkter Speicherzugriff): Operand befindet sich an einer festen Adresse, z. B. `MOV R1, [0x1000]`,

- Indirect addressing (Indirekter Zugriff): Adresse wird in einem Register gehalten, z. B. `MOV R2, [R3]` oder `LOAD R2, [R3]`,
- Register addressing (Registerzugriff): Operand liegt in einem Register, z. B. `MOV R4, R5`,
- Basis- bzw. Indexadressierung (Basis + Index): Zugriff über eine Basisadresse plus Index, z. B. `MOV R6, [R1 + R2]`,
- Relative addressing (PC-relativ, insbesondere für Sprünge): Sprünge relativ zur aktuellen PC-Position, z. B. `JMP label` mit PC-Offset.

Beispiele (schematisch):

```
MOV R0, #10   # unmittelbarer Wert
MOV R1, [0x2000]   Direct
MOV R2, [R3]   Indirect
MOV R4, R5   Register
MOV R6, [R1 + 8]   Indexed/disp
JMP label   PC-relative Sprung
```

(4) Mini-Beispielprogramm – Ablaufsteuerung und Adressierung.
Ziel: Zähle von 0 bis 9 und springe zurück zu Beginn, bis der Zähler 9 erreicht ist.

```
ORG 0x1000
START: MOV R0, #0
 CMP R0, #9
 JG END
 ; Aktion mit R0 (z.B. Ausgabe)
 ADD R0, #1
 JMP START
END: HALT
```

(5) Notation und Mnemonics – Übersicht.
Typische Mnemonics (Beispiele):

- MOV – kopiert Werte zwischen Registern/Speicherzellen
- ADD, SUB – arithmetische Operationen
- CMP – Vergleich (setzt Flags)
- JMP – unbedingter Sprung
- JE/JZ, JNE/JNZ – bedingte Sprünge (basierend auf Flags)

Hinweis: Adressierungsarten und Befehle variieren je nach Architektur (z. B. ARM, x86, MIPS). Verlässliche Referenz: Architektur-Spezifikation des Zielsystems.

Hinweis zur Sicherheit und Verständlichkeit: Bei komplexeren Sprungstrukturen darauf achten, dass Pfade eindeutig sind und der Compiler/Assembler keine unklaren Optimierungen vornimmt. Halte Mengen an Code überschaubar und kommentiere kritisch, welche Adressierungsart wann sinnvoll ist.