Resim-Befehle

Transportbefehle

- MOV Akku {Konstante, Register}
  - Mov Akku 3 — 3 in den Akku
  - Mov Akku Reg 5 — Inhalt von Reg 5 in den Akku
- MOV {Register} Akkumulator
  - Mov Reg 5 Akku — Akku in das Reg 5

Arithmetikbefehle

- ADD Akku {Konstante, Register}
  - ADD Akku 44 — Addiert 44 zum Akku
  - ADD Akku Reg 3 — Addiert den Inhalt von Reg 3 zum Akku
- SUB Akku {Konstante, Register}
  - SUB Akku 44 — Subtrahiert 44 vom Akku
  - SUB Akku Reg 3 — Subtrahiert Reg 3 vom Akku
- MOD Akku {Konstante, Register}
  - MOD Akku 44 — Berechnet den Rest von Akku und 44
  - MOD Akku Reg 3 — Berechnet den Rest von Akku und Reg 3
- MULT Akku {Konstante, Register}
  - MULT Akku 44 — Multipliziert 44 mit dem Akku
  - MULT Akku Reg 3 — Multipliziert Reg 3 mit dem Akku
- DIV Akku {Konstante, Register}
  - DIV Akku 44 — Dividiert den Akku durch 44
  - DIV Akku Reg 3 — Dividiert den Akku durch Reg 3
- INC Akku — Erhöht den Akku um eins
- DEC Akku — Verringert den Akku um eins

Logik-Befehle

- AND Akku {Konstante, Register}
  - AND Akku 44 — Bitweises logisches Und Akku mit 44
  - AND Akku Reg 3 — Bitweises logisches Und Akku mit Reg 3
- OR Akku {Konstante, Register}
  - OR Akku 44 — Bitweises logisches Oder Akku mit 44
  - OR Akku Reg 3 — Bitweises logisches Oder Akku mit Reg 3
- XOR Akku {Konstante, Register}
  - XOR Akku 44 — Bitweises logisches XOR Akku mit 44
  - XOR Akku Reg 3 — Bitweises logisches XOR Akku mit Reg 3
- NOT Akku — Bitweises negieren des Akkus

Schiebe-Befehle

- SHL Akku {Konstante, Register}
  - SHL Akku 2 — Verschieben des Akkus um 2 Bit nach links
  - SHL Akku Reg 3 — Verschieben des Akkus um Reg 3 Bit nach links
- SHR Akku {Konstante, Register}
  - SHR Akku 2 — Verschieben des Akkus um 2 Bit nach rechts
  - SHR Akku Reg 3 — Verschieben des Akkus um Reg 3 Bit nach rechts
- ROL Akku Konstante
  - ROL Akku 1
    - Verschieben des Akkus um 1 Bit nach links, das MSB wandert in das carry-Flag. Das LSB erhält den vorherigen Wert des Carry-Flags
- ROR Akku Konstante
  - ROR Akku 1
    - Verschieben des Akkus um 1 Bit nach rechts, das MSB erhält den Wert des Carry-Flags, das LSF wandert in das carry-Flag.
  -

Beispiele:

- `00011001  SHL ergibt 00110010`
- `00011001  SHR ergibt 00001100`
- `00011001  ROL ergibt 00110010 wenn Carry nicht gesetzt`
- `00011001  ROL ergibt 00110011 wenn Carry gesetzt`
- `00011001  ROR ergibt 00001100 wenn Carry nicht gesetzt`
- `00011001  ROR ergibt 10001100 wenn Carry gesetzt`

Sprungbefehle

- CMP AKKU {Konstante, Register}
  - CMP AKKU 4
    - Vergleicht den Akku mit einer Konstanten. Die beiden Werte werden subtrahiert.
  - CMP Reg 2
    - Vergleicht den Akku mit dem Inhalt des Registers 2. Die beiden Werte werden subtrahiert.
- JUMP Adresse
- JZ: nach Adresse wenn Zero-Flag gesetzt
- JNZ: nach Adresse wenn Zero-Flag nicht gesetzt
- JS: nach Adresse wenn Sign-Flag gesetzt
- JNS: nach Adresse wenn Sign-Flag nicht gesetzt
- JG: nach Adresse wenn Akku>0
- JGE: nach Adresse wenn Akku >= 0
- JL: nach Adresse wenn Akku <0
- JLE: nach Adresse wenn Akku <=0
- JC: nach Adresse wenn Carry-Flag gesetzt
- JNC: nach Adresse wenn Carry-Flag nicht gesetzt
- JP: nach Adresse wenn Parity-Flag gesetzt
- JNP: nach Adresse wenn Parity-Flag nicht gesetzt

E/A-Befehle

- LiesZahl in E/A Einheit
    - LiesZahl "Eingabe"
- IN {Akku, Register}
    - IN Akku — Kopiert den Wert aus der E/A-Einheit in den Akku
    - IN Reg 3 — Kopiert den Wert aus der E/A-Einheit in Reg 3
- OUT {Akku, Register}
    - OUT Akku — Ausgabe des Akkus in den Editor
    - OUT Reg 3 — Ausgabe des Reg 3 in den Editor
- OUT NL — Neue Zeile
- OUT String "Ergebnis" — Ausgabe eines Textes

Eingabedialoge im Debugger:

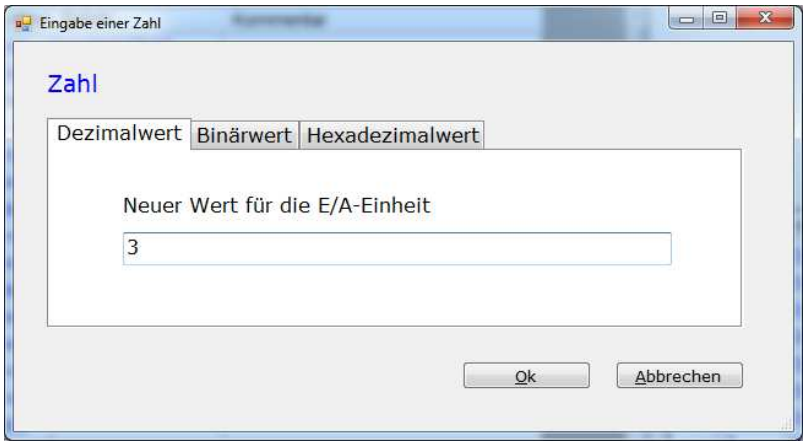

Abbildung 1 Eingabe einer dezimalen Zahl im Debugger

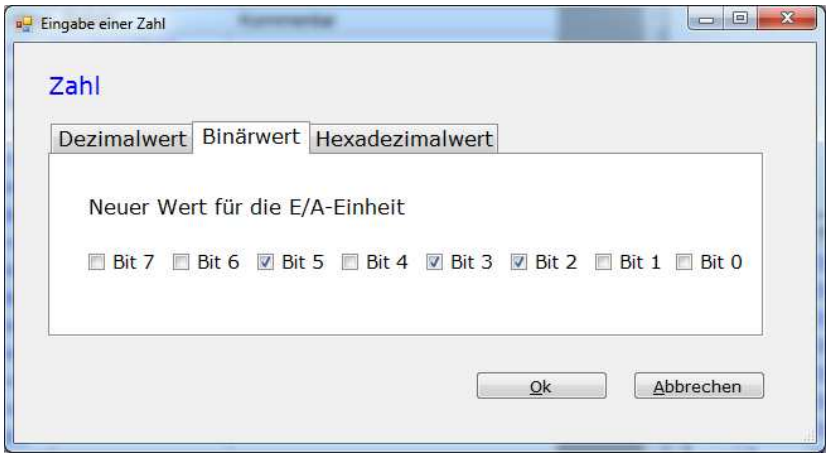

Abbildung 2 Eingabe einer binären Zahl

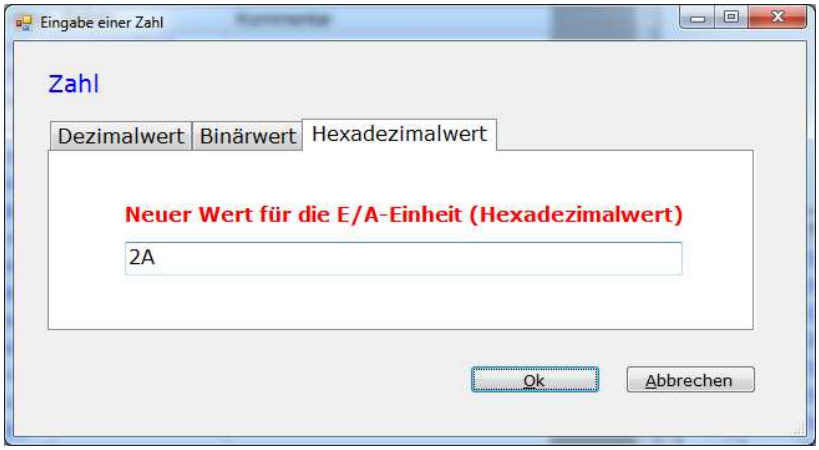

Abbildung 3 Eingabe einer hexadezimalen Zahl

Steuerbefehle

- Clear Carry
  - Löscht das Carry-Flag
- Goto Sub / IRET
- NOP (No Operation)

Befehlsgruppen

Dieses Kapitel zeigt, wie man die einzelnen Befehle in sinnvolle Gruppen resp. Aufgaben, einteilen kann. Es gibt immer wieder „Teilaufgaben“, die in den Programmen gelöst werden müssen. Dieses Kapitel zeigt für die wichtigsten Teilaufgaben die Lösungen.

Einlesen einer Zahl

- LiesZahl "x"
- In Reg 5

Eine Zahl wird in die Eingabeeinheit eingelesen. Danach wird es in das Register fünf eingetragen.

Register mit einem Wert belegen

Diese Aufgabe kann man nur über den Akkumulator realisieren:

- Mov Akku Konstante Mov Akku 42
- Mov Reg(i) Akku Mov Reg 5 Akku

Das Register und die Konstante muss angegeben werden.

Operation mit einem Register und die Speicherung in ein neues Register

Diese Aufgabe kann man nur über den Akkumulator realisieren:

- Mov Akku Reg(i) // Register in den Akkumulator
- Operation Akku Reg(j) // add, sub, div, mod, mult
- Mov Reg(k) Akku // Wert in das Zielregister schreiben

Beispiel:

- Mov Akku Reg 3 // Akku erhält den Wert des Registers 3
- Add Akku Reg 2 // Zum Akku wird der Inhalt des Reg 2 addiert
- Mov Reg 4 Akku // Ergebnis wird im Register 4 gespeichert

- Mov Akku Reg(i) // Register in den Akkumulator
- Operation Akku 55 // add, sub, div, mod, mult
- Mov Reg(k) Akku // Wert in das Zielregister schreiben

Beispiel:

- Mov Akku Reg 3 // Akku erhält den Wert des Registers 3
- Add Akku 42 // Zum Akku wird die Zahl 42 addiert
- Mov Reg 4 Akku // Ergebnis wird im Register 4 gespeichert

Die Register i,j,k müssen ausgewählt werden. Können aber auch teilweise identisch sein

Sprung, wenn der Akku größer als der Inhalt des Registers 2 ist

- Mov Akku 42
- CMP Akku Reg 2 // in Akku steht die Differenz „Akku-Reg2“
- JG Akku Marker1 // Jump greater, wenn Differenz > 0

Die Sprungmarke wird mit einem Namen ganz links mit einem Doppelpunkt eingetragen.
Beispiel:
Marker1: NOP

Sprung, wenn der Akku größer gleich Null ist

```
M1:
    // Anweisungen
    Mov  Akku  42          // Beispiel oder Reg 1
    CMP  Akku  Reg 3       // in Akku steht die Differenz  „42-Reg 3“
    JGE  M1                // Jump greater equal          if (42>=Reg3)
```

Sprung, wenn der Akku kleiner Null ist

```
M1:
    // Anweisungen
    Mov  Akku  Reg 3       // Akku=Reg 3
    CMP  Akku  42          // in Akku steht die Differenz  „Reg3-42“
    JL  M1                 // Jump less than          if (Reg3<42)
```

Sprung, wenn der Akku kleiner gleich Null ist

```
M1:
    // Anweisungen
    Mov  Akku  Reg 3       // Akku=Reg 3
    CMP  Akku  42          // in Akku steht die Differenz  „Reg3-42“
    JLE  M1                // Jump less equal          if (Reg3<=42)
```

Sprung, wenn der Akku ungleich Null ist

```
M1:
    // Anweisungen
    Mov  Akku  Reg 3       // Akku=Reg 3
    CMP  Akku  42          // in Akku steht die Differenz  „Reg3-42“
    JNZ  M1                // Jump Not Zero          if (Reg3<>42)
```

1 Aufgaben

1.1 Addieren zwei eingegebene Zahlen

Java:

```
int a;
int b;
int c;

a= readZahl();
b= readZahl();

c=a+b;

Syso("Ergebnis ist"+c);
```

Resim:

```
Register 0          a
Register 1          b
Register 2          c

    LiesZahl "Eingabe a"
    In Reg 0
    LiesZahl "Eingabe b"
    In Reg 1
    Mov Akku Reg 0              // akku=a
    Add akku reg 1              // akku=akku+b
    Mov reg 2 akku              // c=akku

    Out NL                  // NewLine
    Out String "Ergebnis"
    Out reg 2
```

1.2 Berechne die Summe von 1 bis n

$$s = \sum_{i=1}^{n} i$$

Java:

```
int s=0;
int i=1;
int n=10;

while (i<=n) {
    s+=I;
    i++;
}
System.out.Println("Ergebnis ist: "+s);
```

Resim:

Register 0	n	
Register 1	i	Laufindex von 1 bis n
Register 2	summe	

```
        LiesZahl "Eingabe n"
        In Reg 0
        Mov Akku 0              // s=0
        Mov Reg 2 Akku
        Mov Akku 1              // i=1
        Mov Reg 1 Akku
M1:     mov akku reg 2          // akku=s
        Add akku reg 1          // akku=akku+i
        Mov reg 2 akku          // s=akku

        Mov akku reg 1          // akku=i
        Inc akku                // akku++
        Mov reg 1 akku          // i=akku

        Mov akku reg 0          // akku = n
        Cmp akku reg 1          // akku = n – i
        Jge M1                  // Sprung, wenn n>=i

        Out NL                  // NewLine
        Out String "Ergebnis"
        Out reg 2
```