Labor 2 - Theoretische Unterstützung

Arithmetische Anweisungen

ADD

***Syntax**:*

```
add <regd>,<regs>; <regd> ← <regd> + <regs>
add <reg>,<mem>; <reg> ← <reg> + <mem>
add <mem>,<reg>; <mem> ← <mem> + <reg>
add <reg>,<con>; <reg> ← <reg> + <con>
add <mem>,<con>; <mem> ← <mem> + <con>
```

Semantik und Einschränkungen:

- Die beiden Operanden des Zusatzes müssen vom gleichen Typ sein (beide Bytes, beide Wörter, beide Doppelwörter);
- Während beide Operanden Register sein können, kann höchstens ein Operand ein Speicherplatz sein.

Beispiel:

```
add EDX,EBX; EDX ← EDX + EBX
add AX,[var]; AX ← AX + [var]
add [var],AX; [var] ← [var] + AX
add EAX,123456h; EAX ← EAX + 123456h
add BYTE [var],10; BYTE [var] ← BYTE [var] + 10
```

SUB

Syntax:

```
sub <regd>,<regs>; <regd> ← <regd> - <regs>
sub <reg>,<mem>; <reg> ← <reg> - <mem>
sub <mem>,<reg>; <mem> ← <mem> - <reg>
sub <reg>,<con>; <reg> ← <reg> - <con>
sub <mem>,<con>; <mem> ← <mem> - <con>
```

Semantik und Einschränkungen:

- Die beiden Operanden der Subtraktion müssen den gleichen Typ haben (beide Bytes, beide Wörter, beide Duplikate);

- Während beide Operanden Register sein können, kann höchstens ein Operand ein Speicherplatz sein.

Beispiel:

```
sub EDX,EBX; EDX ← EDX - EBX
sub AX,[var]; AX ← AX - [var]
sub [var],AX; [var] ← [var] - AX
sub EAX,123456h; EAX ← EAX - 123456h
sub byte [var],10; BYTE [var] ← BYTE [var] - 10
```

MUL

Syntax:

```
mul <op8>; AX ← AL * <op8>
mul <op16>; DX:AX ← AX * <op16>
mul <op32>; EDX:EAX ← EAX * <op32>
```

Semantik und Einschränkungen:

- Das Ergebnis der Multiplikationsoperation wird verglichen mit der Länge der Operanden auf einer doppelten Länge gehalten;
- Die MUL Anweisung führt die Multiplikationsoperation für Ganzzahlen ohne Vorzeichen aus.
- Es ist erforderlich, dass der erste Operand und das Ergebnis in den Registern gespeichert werden.Obwohl die Operation binär ist, wird nur ein Operand angegeben, da der andere und die Position des Ergebnisses immer festgelegt sind.
- Der explizite Operand kann ein Register oder eine Variable sein, aber kein unmittelbarer (konstanter) Wert.

Beispiel:

```
mul DH; AX ← AL * DH
mul DX; DX:AX ← AX * DX
mul EBX; EDX:EAX ← EAX * EBX
mul BYTE [mem8]; AX ← AL * BYTE [mem8]
mul WORD [mem16]; DX:AX ← AX * WORD [mem8]
```

DIV

Syntax:

```
div <reg8>; AL ← AX / <reg8>, AH ← AX % <reg8>
div <reg16>; AX ← DX:AX / <reg16>, DX ← DX:AX % <reg16>
div <reg32>; EAX ← EDX:EAX / <reg32>, EDX ← EDX:EAX % <reg32>
div <mem8>; AL ← AX / <mem8>, AH ← AX % <mem8>
div <mem16>; AX ← DX:AX / <mem16>, DX ← DX:AX % <mem16>
div <mem32>; EAX ← EDX:EAX / <mem32>, EDX ← EDX:EAX % <mem32>
```

Semantik und Einschränkungen:

- Die DIV-Anweisung führt die Divisionsoperation für Ganzzahlen ohne Vorzeichen aus.
- Es ist erforderlich, dass der erste Operand und das Ergebnis in den Registern gespeichert werden.
- Der erste Operand ist nicht angegeben und doppelt so lang wie der zweite Operand.
- Der explizite Operand kann ein Register oder eine Variable sein, aber kein unmittelbarer (konstanter) Wert.
- Durch Teilen einer großen Zahl in eine kleine Zahl besteht die Möglichkeit, dass das Ergebnis die Darstellungskapazität überschreitet. In diesem Fall wird der gleiche Fehler ausgelöst wie beim Teilen durch 0.

Beispiel:

```
div CL; AL ← AX / CL, AH ← AX % CL
div SI; AX ← DX:AX / SI, DX ← DX:AX % SI
div EBX; EAX ← EDX:EAX / EBX, EDX ← EDX:EAX % EBX
```

INC

Syntax:

```
inc <reg>; <reg> ← <reg> + 1
inc <mem>; <mem> ← <reg> + 1
```

Semantik und Einschränkungen:

- Erhöht den Inhalt des Operanden um 1.

Beispiel:

```
inc DWORD [var]; DWORD [var] ← DWORD [var] + 1
inc EBX; EBX ← EBX + 1
inc DL; DL ← DL + 1
```

DEC

Syntax:

```
dec <reg>; <reg> ← <reg> - 1
dec <mem>; <mem> ← <reg> - 1
```

Semantik und Einschränkungen:

- Dekrementieren Sie den Inhalt des Operanden mit 1.

Beispiel:

```
dec EAX; EAX ← EAX - 1
dec BYTE [mem]; [value] ← [value] - 1
```

NEG

Syntax:

```
neg <reg>; <reg> ← 0 - <reg>
neg <mem>; <mem> ← 0 - <mem>
```

Semantik und Einschränkungen:

- Die arithmetische Negation des Operanden findet statt.

Beispiel:

```
neg EAX; EAX ← 0 – EAX
```

Variable Declaration / Konstanten

Variable Declaration mit Anfangswert

```
a DB 0A2h ;Die Variable vom Typ BYTE wird deklariert und mit dem Wert 0A2h initialisiert
b DW 'ab' ;deklarieren Sie die Variable vom Typ WORD und initialisieren Sie mit dem Wert 'ab'
c DD 12345678h ;Deklarieren Sie die Variable vom Typ DOUBLE WORD und initialisieren Sie mit dem Wert 12345678h
d DQ 1122334455667788h ;Deklarieren Sie die Variable vom Typ QUAD WORD und initialisieren Sie mit dem Wert 1122334455667788h
```

Variable Declaration ohne Anfangswert

```
a RESB 1 ;1 Byte ist reserviert
b RESB 64 ;64 Bytes sind reserviert
c RESW 1 ;1 Wort ist reserviert
```

Definition von Konstanten

```
zece EQU 10 ;die Konstante 'zece' wird mit dem Wert 10 definiert
```

Zeichenerklärung

```
<op8>    - 8-Bit-Operand
<op16>   - 16-Bit-Operand
<op32>   - 32-Bit-Operand

<reg8>   - 8-bit Register
<reg16>  - 16-bit Register
<reg32>  - 32-bit Register
<reg>    - Register
<regd>   - Zielregister
<regs>   - Quellregister

<mem8>   - 8-Bit-Speichervariable
<mem16>  - 16-Bit-Speichervariable
<mem32>  - 32-Bit-Speichervariable
<mem>    - Speichervariable

<con8>   - Konstante (Sofortwert) auf 8 Bits
<con16>  - Konstante (Sofortwert) auf 16 Bits
<con32>  - Konstante (Sofortwert) auf 32 Bits
<con>    - Konstante (Sofortwert)
```