Operatoren

Montag, 8. April 2019 07:44

Priorität	Assoziativität	Operatoren
1 (höchste)	Links-nach-rechts	Funktionsaufruf (), Indexzugriff [], Elementzugriffe -> .
2	Rechts-nach-Links	Vorzeichen + -, logisches/bitweises NOT ! ~, prä-/post-Inkrement/Dekrement ++ --, Adresse &, Zeigerdereferenzierung *, Tyumwandlung (typ), sizeof
3	Links-nach-rechts	Mutliplikation, Division, Modulo * / %
4	Links-nach-rechts	Addition, Subtraktion + -
5	Links-nach-rechts	Links-/Rechtsshift << >>
6	Links-nach-rechts	kleiner/größer (gleich) < <= > >=
7	Links-nach-rechts	Gleich, ungleich == !=
8	Links-nach-rechts	Bitweises AND &
9	Links-nach-rechts	Bitweises XOR ^
10	Links-nach-rechts	Bitweises OR |
11	Links-nach-rechts	Logisches AND &&
12	Links-nach-rechts	Logisches OR ||
13	Rechts-nach-Links	Bedingung ?
14	Rechts-nach-Links	(zusammengesetzte) Zuweisung = *= /= %= += -= &= ^= |= <<= >>=
15 (niedrigste)	Links-nach-rechts	Kommaoperator ,

Operatoren: Arithmetisch
Bitweise
Relationale

Arithmetisch: +, − , *, /

Betrag / 2-er Kompl. gleich verschieden

+: ⟶ ADD

```
int i;
short s;
```

```
i = i + s;  ⟶  mov EAX, DWORD PTR [i]
               [mov BX, WORD PTR [s]
               [movsx EBX, BX
                ADD EAX, EBX
                mov DWORD PTR [i], EAX
```

```
MOV DWORD PTR [i], EAX
```

```
movsx EBX, WORD PTR [s]
ADD   EBX, DWORD PTR [i]
MOV   DWORD PTR [i], EBX
```

(i = i + s ≙ i += s)

```
movsx EBX, WORD PTR [s]
ADD   DWORD PTR [i], EBX
```

Weiterer Sonderfall:

i = i + 5;	i = i + 1;
≙ i += 5;	≙ i += 1;
	≙ i++;
ADD DWORD PTR [i], 5	INC DWORD PTR [i]

Betrag / 2-er Komplement

Bsp.:

2-er Kompl.		Betrag
−1	1111 1111	255
+ +1	0000 0001	1
0	(1) (1)111 111	256
	0000 0000	

Carry-Flag – Vorzeichenlos
= 1 ≙ Resultat ungültig

XOR = 0

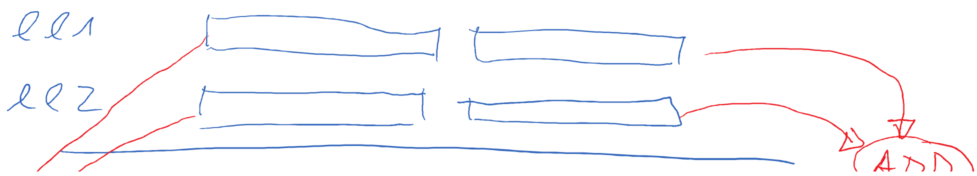
Overflow-Flag – Vorzeichenbehaftet

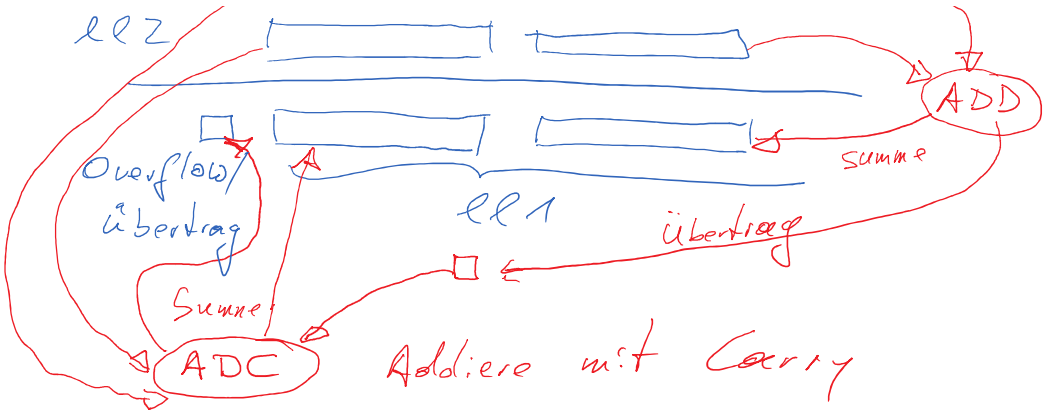
Hardwaregrenzen:

long long ll1, ll2;

ll1 = ll1 + ll2;

≙ ll1 += ll2;






```
MOV  EAX, DWORD PTR [ll2]
ADD  DWORD PTR [ll1], EAX
MOV  EAX, DWORD PTR [ll2+4]
ADC  DWORD PTR [ll1+4], EAX
```

übertrag wird erzeugt und hier verwendet

MOV ändert keine Flags!

Floating Point:

D8 /0	FADD *m32fp*
DC /0	FADD *m64fp*

FADD

Addiere floating point

DA /0	FIADD *m32int*
DE /0	FIADD *m16int*

FiADD

2-er Komplement

Kodierten Wert

HW-Beschränkung: - Bitbreite
- Kodierung (float, signed)

Bsp.:

```
float f;
int i;
unsigned int u;
```

f = f + i; → FLD DWORD PTR [f]

FIADD DWORD PTR [i]
FSTP DWORD PTR [f]

Frage (bleibt hier offen): ?

f = f + u;

Multiplikation / Division:

unsigned Multiplikation: MUL

- Operandenfestlegung (AL, AX, EAX
- Produkt ist doppelt breit als Quelloperanden

F6 /4	MUL *r/m8*	Unsigned multiply (AX ← AL * *r/m8*)
F7 /4	MUL *r/m16*	Unsigned multiply (DX:AX ← AX * *r/m16*)
F7 /4	MUL *r/m32*	Unsigned multiply (EDX:EAX ← EAX * *r/m32*)

Bsp.: 8-Bit 255 * 2

AL BL

1111 1111 * 0000 0010

AX 0000 0001 | 1111 1110

16 Bit 255 * 256 (≙ 2^8)

AX 0000 0000 | 1111 1111 * BX 0000 0001 | 0000 0000

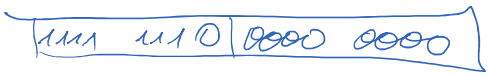
DX 0000 0000 | 0000 0000 * AX 1111 1111 | 0000 0000

16 Bit 255 * 512 (≙ 2^9)

DX AX

DX

AX



2-er Komplement Multiplikation: iMUL

S. Handbuch

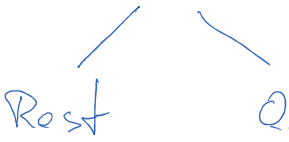
+ Erweiterte Operanden Kombinationen

d.h. bis zu 3 Operanden

Division:

Betrag

Div



Rest Quotient

2-er Komplement

iDiv

Relationale Operationen:

cmp / Vergleich

Jcc / Jump if Condition Code is met

Jmp