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

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 = 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) 0000 0000	256

Carry-Flag — vorzeichenlos = 1 ≙ Resultat ungültig

Overflow-Flag – Vorzeichenbehaftet

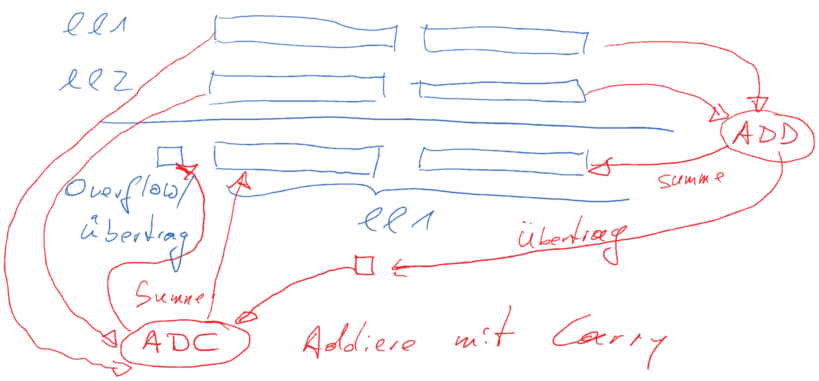
XOR = 0

Hardwaregrenzen:

long long ll1, ll2;

ll1 = ll1 + ll2;

≙ ll1 += ll2;



Addiere mit Carry

```
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 → ADD

und hier verwendet → ADC

MOV ändert keine Flags!

Floating Point:

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

FADD

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

FiADD

DC /0	FADD *m64fp*	DE /0	FIADD *m16int*

FADD — Addiere — floating point

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
- Operandenfestgelegt (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: 1111 1111 × BL: 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 | 0 . . . | . . . 01 | AX | 111 110 | 0000 0000 |

2-er Komplement Multiplikation: iMUL

s. Handbuch, Erweiterung auf 3 explizite Operanden

+ Erweiterte Operanden Kombinationen

d.h. bis zu 3 Operanden, beschränkt auf Instruktions codierungs-randbedingungen

Division:

Betrag

Div

Rest Quotient

2-er Komplement

iDiv

Relationale Operationen:

cmp / Vergleich jcc / Jump if Condition Code is met jmp

Bsp.: int i,j,k;

i = j==k; ⟷

```
mov EAX, DWORD PTR [j]
cmp EAX, DWORD PTR [k]
JZ EINS
MOV EAX, 0
JMP NULL
EINS:
MOV EAX, 1
NULL:
mov DWORD PTR [i], EAX
```

<, <=, >, >=, !=, ...

Typabhängig: unsigned, signed, float (später)

Carry Flag — Overflow Flag

i = j < k;
↳ C = 1
JB (below)

i = j > k
JA (above)

O = 1, bzw. O ≠ S
JL (less)

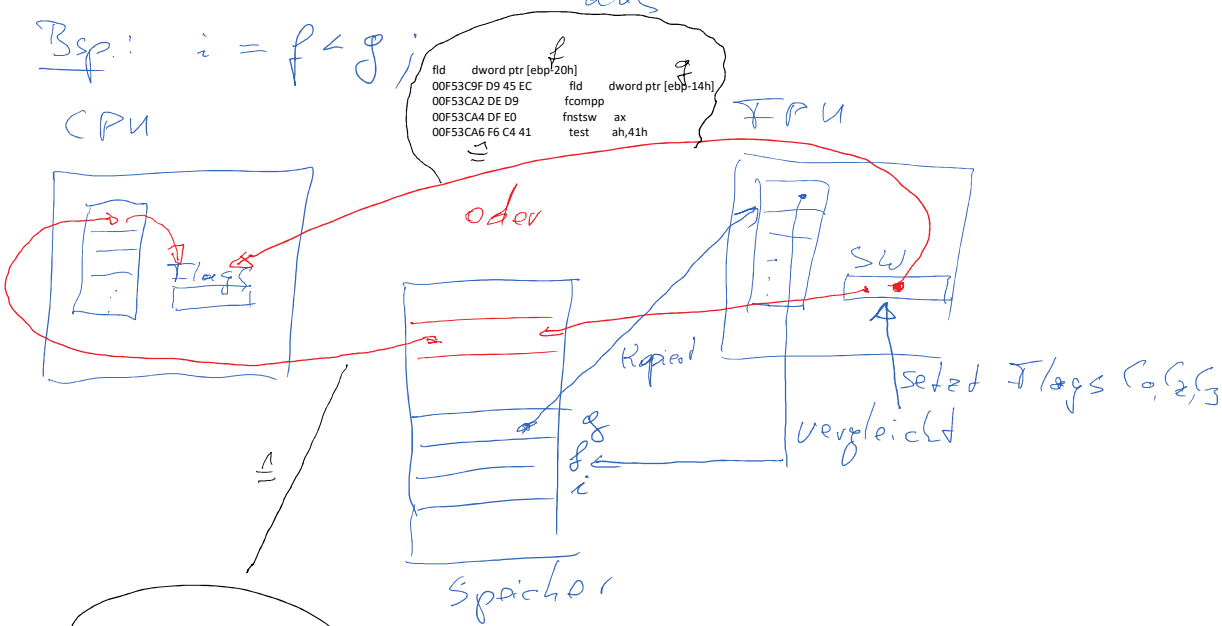
JG (greater)

Jetzt: Typ float

int i;
float f, g;

→ Ansatz:
fcomp
⇒ FPU braucht die Operanden
fcomp setzt 3 Bits im Statuswort
⇒ Diese werden in CPU Flag-Register kopiert
⇒ CPU führt Jcc Instruktion aus

Bsp: i = f < g;




```
{
        int i;
        float g, f;
        unsigned short statuswort;

        f = 5; g = 7;
        i = f < g;
        __asm{
                FLD DWORD PTR [f]
                FCOMP DWORD PTR [g]
                FSTSW WORD PTR [statuswort]
                MOV AX, WORD PTR [statuswort]
                SAHF

                JB eins
                MOV EAX, 0
                JMP null
eins:
                MOV EAX, 1
null:
                MOV DWORD PTR [i], EAX
        }
```

Flags:

Bit	15-12	11	10	9	8	7	6	5	4	3	2	1	0
Flag		OF	DF	IF	TF	SF	ZF		AF		PF		CF

```
            JMP null
eins:       MOV EAX, 1
null:       MOV DWORD PTR [i], EAX
        }
}
```

Bitoperatoren:

&, |, >>, << Java >>>

und	oder	rechts-Shift	links-Shift	rechtsshift
AND	OR	SHR	SHL	SAR

Bsp: Test, ob Wert negativ ist:

```
int i;
i = -1;
__asm{
  // AND DWORD PTR [i], 0x80000000
     // i=0 -> vorher nicht negativ...
  //SHR DWORD PTR [i], 31
     // wie zuvor

  SAR DWORD PTR [i], 32
     // i=0 -> vorher nicht negativ
     // i=-1 -> vorher negativ
}
```