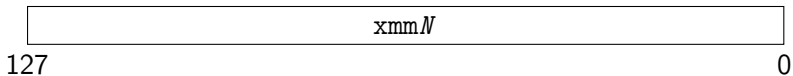
SSE-Register

- 16 weitere Register
  - xmm0 bis xmm15



  - Für skalare Berechnungen sind nur die unteren 32 bzw. 64 Bit von Relevanz

- xmm-Registern verwendbar für Floating-Point Berechnungen

Konstanten

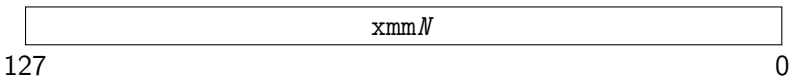
- XOR-Instruktionen für xmm-Register
  - `pxor dst, src`
- Floating-Point Konstanten können aus dem Speicher (z.B. `.rodata`) geladen werden
  - `movss dst, src`
  - Beispiel: `movss xmm0, [rip + .Lconstx]`
- Moves zwischen General-Purpose und xmm-Registern möglich
  - `movd` / `movq`
  - Keine Konvertierung!
  - Gut für Bitmanipulationen

Arithmetik – Teil 1

- Namenskonvention bei Instruktionen:
  - 'ss' $\rightarrow$ Scalar Single (Precision)
  - 'sd' $\rightarrow$ Scalar Double (Precision)

Arithmetik – Teil 2

- `addss dest, src` – Addition zweier Floating-Point Werte
  - `src`: Register oder Speicher



  - Untere 32 bit für Addition mit `addss`

- `subss dest, src` – Subtraktion zweier Floating-Point Werte
  - Analog zu `addss`

Arithmetik – Teil 3

- `divss dest, src` – Division mit zwei Floating-Point Werten
  - Unterschied zu `div`: Operanden werden nicht implizit gefolgert, explizite Angabe
- `mulss dest, src` – Multiplikation mit zwei Floating-Point Werten
  - Analog zu `divss`
- Konstanter Divisor: Multiplikation mit Kehrwert bevorzugen

Vergleiche

- `ucomiss op1, op2` - Skalarer Vergleich zweier Floating-Point Werte
- Flags gesetzt in Abhängigkeit des Ergebnisses
  - Ermöglicht Sprünge mittels jCC
  - Aber: Condition Codes für vorzeichenlose Vergleiche

```
1   cmpFloat:
2     ucomiss xmm1, xmm0
3     jp .Lunordered // xmm0 or xmm1 NaN
4     jb .Llesser    // xmm1 < xmm0
5     ja .Lgreater   // xmm1 > xmm0
6     je .Lequal     // xmm1 == xmm0
```

- `ucomiss` behandelt Vergleiche mit NaN gesondert
  - Überprüfbar mit `jp` bzw. `jnp`

Codebeispiel

```c
#include <stdio.h>

extern float func(float x);

int main(int argc, char** argv) {
    float res = func(2.0);
    printf("Result: %f\n", res);
}
```

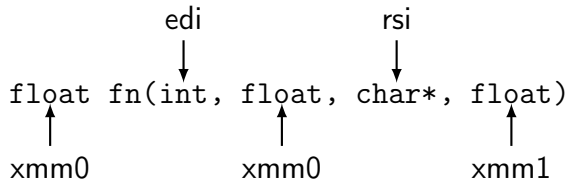
```
.intel_syntax noprefix
.global func
.text
func:
    mov r8, 1
    cvtsi2ss xmm1, r8
    divss xmm1, xmm0
    movss xmm0, xmm1
    ret
```

Erweiterte Calling Convention - Teil 1

- Registerknappheit motiviert Erweiterung der CC
- Floating-Point Rückgabewert: `xmm0`
- Floating-Point Argumente: `xmm0` bis `xmm7` (weitere auf Stack)
- Wichtig: Alle Register sind caller-saved/temporär

Erweiterte Calling Convention - Teil 2

- Kombinationen von FP und Int/Ptr Argumenten
  - Seperate Durchnummerierung der Register




```
1  ex:
2      mov rdi, 1
3      movss xmm0, [rip + .Lconstx]
4      mov rsi, 2
5      movss xmm1, [rip + .Lconsty]
6      call fn
7      <...>
```

Quiz: movss

Kreuzen Sie alle richtigen Antworten in Bezug auf movss an.

- ☐ Wenn der zweite Operand ein Speicherzugriff ist, werden die Bits `[32;127]` des xmm-Registers auf Null gesetzt
- ☐ Wenn der zweite Operand ein Register ist, dann werden nur die Bits `[0;31]` des Ziel-Registers geändert
- ☐ `movss` benötigt immer mindestens einen Speicheroperanden
- ☐ `movss` erlaubt es, Single-Precision Floating-Point Werte zwischen Registern bzw. Registern und dem Speicher zu bewegen

Quiz: cvtsi2ss

Kreuzen Sie alle richtigen Antworten in Bezug auf cvtsi2ss an.

- ☐ Für diese Instruktion gibt es kein Gegenstück für Double-Precision Floats
- ☐ Die Instruktion konvertiert eine Ganzzahl in einen Single-Precision Floating-Point Wert
- ☐ Die Instruktion konvertiert eine Ganzzahl in einen Double-Precision Floating-Point Wert
- ☐ Die Komponente ‘si’ steht für Single Integer

Quiz: cvtss2si

Kreuzen Sie alle richtigen Antworten in Bezug auf cvtss2si an.

- ☐ Der Floating-Point Wert kann aus dem Speicher bezogen werden
- ☐ Die Instruktion konvertiert einen Single-Precision Floating-Point Wert in eine Ganzzahl
- ☐ Es findet keine Rundung bei der Konvertierung statt
- ☐ Der erste Operand ist immer xmm1

Quiz: func

Was berechnet die Funktion ‘func’?

☐ $\frac{x}{8}$ ☐ $\frac{1}{x}$ ☐ x^2

```
func:
  mov r8, 1
  ctvsi2ss xmm1, r8
  divss xmm1, xmm0
  movss xmm0, xmm1
  ret
```

Quiz: Calling Convention

In welchem SSE-Register liegt der Rückgabewert?

`xmm3`

`xmm16`

`xmm0`

```
func:
  mov r8, 1
  ctvsi2ss xmm1, r8
  divss xmm1, xmm0
  movss xmm0, xmm1
  ret
```