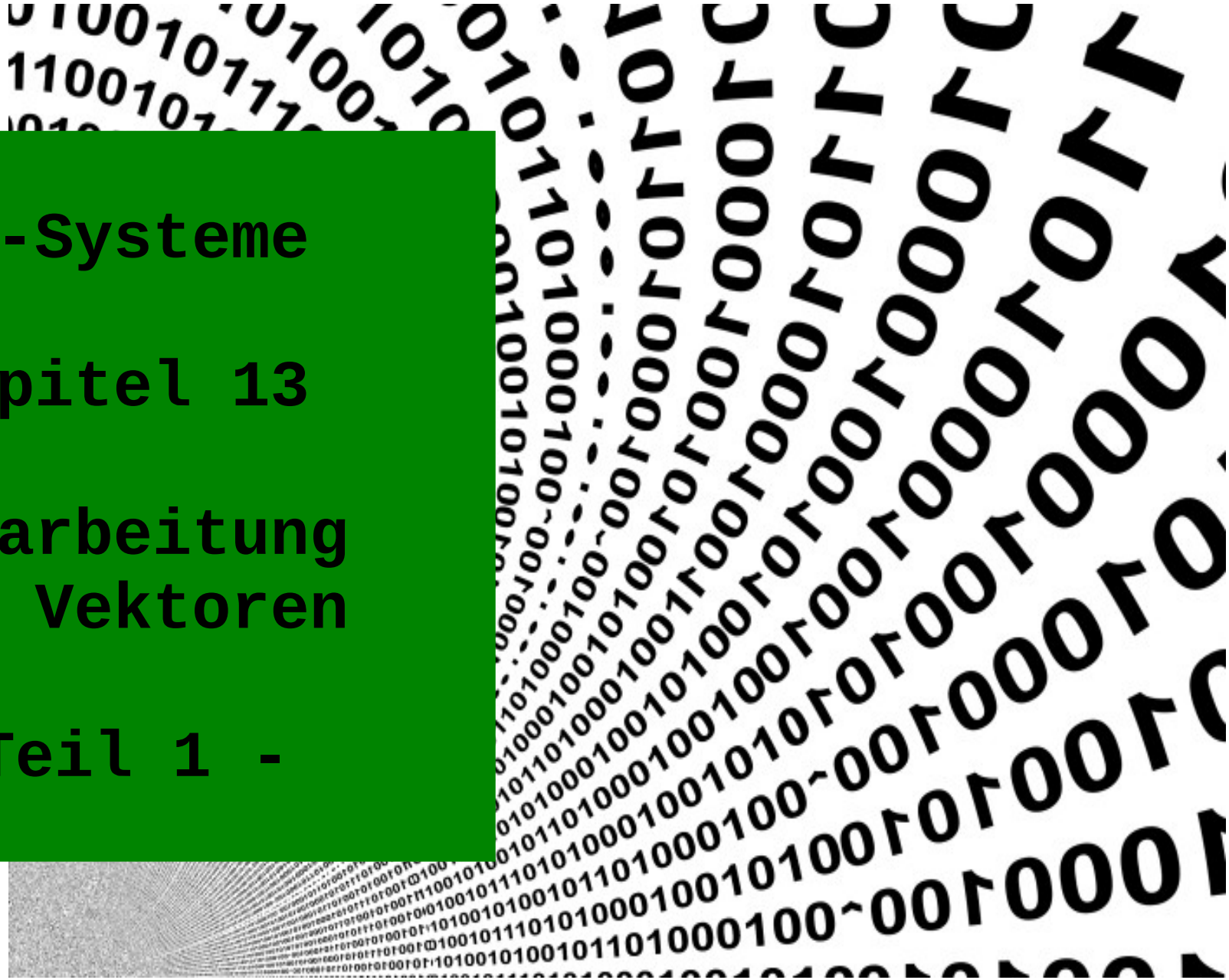


DS-Systeme

Kapitel 13

Verarbeitung von Vektoren

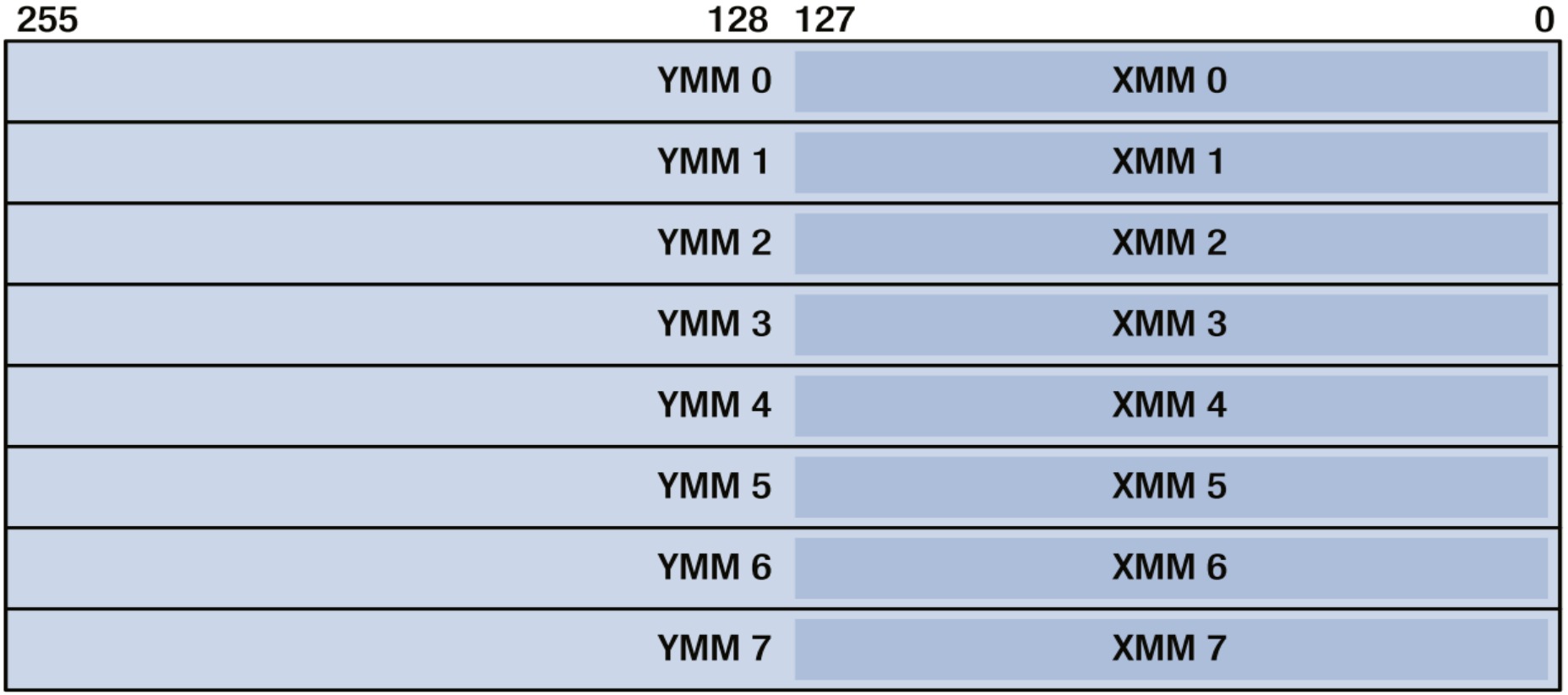
- Teil 1 -

Inhaltsverzeichnis

Vector Processing – Verarbeiten von Vektoren

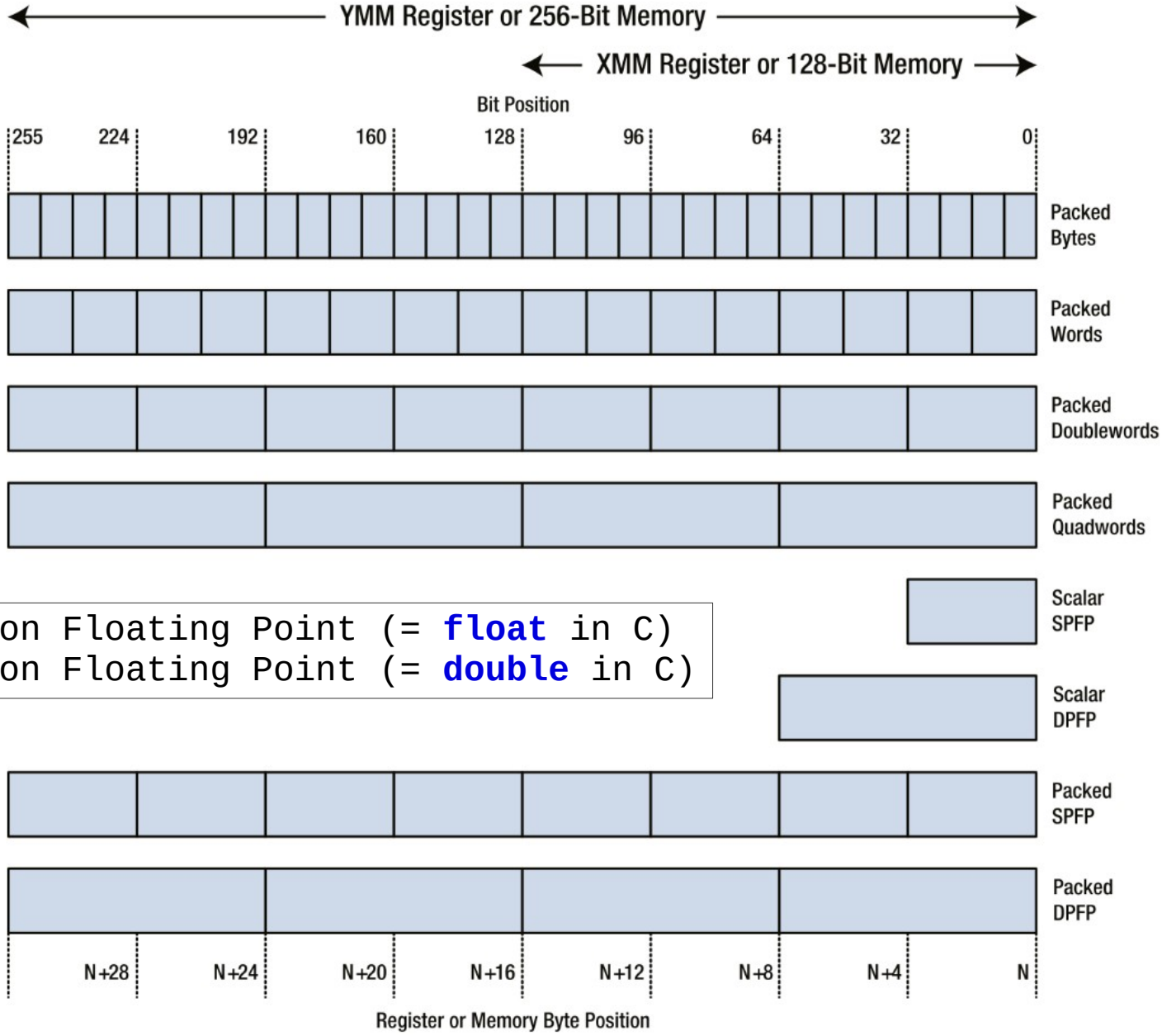
Unter Vektor Prozessing versteht man die **parallele Verarbeitung** von Gleitkomma- bzw. Ganzzahlwerten.

Je nach Standard (SSE, AVX, AVX2, AVX-512) wird mit den Registern XMM (128 Bit), YMM (256 Bit) und ZMM (512 Bit) gearbeitet.



Vector Processing – Verarbeiten von Vektoren

Dies ermöglicht z.B. eine **parallele** Verarbeitung von 4 oder 8 **float-** bzw. 2 oder 4 **double**-Werten.



SPFP = Single Precision Floating Point (= **float** in C)
DPFP = Double Precision Floating Point (= **double** in C)

Anwendungen des Vector Processings

- Beschleunigung von Algorithmen (z.B. Matrizenmultiplikation)
- Bildverarbeitung (Filter, Matrizenrechnung)
- Computergrafik (Blender, Render Engine)
- Cryptographie (Verschlüsselungsverfahren) (OpenSSL, Bitcoin)
- Videos kodieren / dekodieren (z.B. x264- bzw. x265-Format)
- Wissenschaftliche Berechnungen (FFTW, PRIME95 / MPrime)

Legende:

Blender	-	3D-Grafiksuite, mit der sich Körper modellieren, texturieren und animieren lassen
Render Engine	-	Software zur Erstellung einer Grafik aus Rohdaten
FFTW	-	**F**astest **F**ourier **T**ransform in the **W**est ist eine Softwarebibliothek zur Berechnung diskreter Fourier-Transformationen (DFTs).
PRIME95 / Mprime	-	Programme, die für die Primzahlfaktorisierung verwendet werden.

mov mit Floating Point

Instruction	Source 2	Source 1	Destination	Description //<availability>
vmovaps		X/M	X/M	Move aligned packed single-precision floating-point //AVX (SSE)
vmovapd		X/M	X/M	Move aligned packed double-precision floating-point values //AVX (SSE2)
vmovups		X/M	X/M	Move unaligned packed single-precision floating-point //AVX (SSE)
vmovupd		X/M	X/M	Move unaligned packed double-precision floating-point //AVX (SSE2)

Befehlsbedeutungen

a = aligned, **u** = unaligned, **p** = packed

Suffix:
s = single precision, **d** = double precision

Operanden:
X = xmm- oder ymm-Register
M = 128-Bit- oder 256-Bit-Speicher-Operand
maximal ein Operand kann Memory sein

Achtung:

für Vectoroperationen gilt:
<instr> <src2> <src1> <dest>
vorausges. **<src2>** ist vorhanden

für Skalare galt:
<instr> <src1> <src2> <dest>

Die **unaligned**-Befehle haben grundsätzlich eine schlechtere Performance als die **aligned**-Befehle.

mov mit Floating Point

Beispiel:

Assembler-Code:

```
.section .bss
.align 16
.lcomm tdataarr, 16

.section .data
.align 16
farr1:
  .float 1.2, 2.3, -3.4, 5.6
.align 16
farr2:
  .float 2.2, 3.3, 4.4, -6.6

.section .text
func:
  vmovaps farr1, %xmm1 # 1
  vmovaps farr2, %xmm2 # 2
  # …
  vmovaps %xmm0, tdataarr # 3
  ret
```

1. Alle float-Werte des Arrays **farr1** werden parallel in das Register **xmm1** kopiert.

2. Alle float-Werte des Arrays **farr2** werden parallel in das Register **xmm2** kopiert.

3. Je nach Funktion kann das gewünschte Endergebnis in **xmm0** stehen oder bei **call by reference** (weil dann eine Adresse vorhanden ist) an einen Speicherplatz kopiert werden.

   Hier: **xmm0** wird an die Adresse **tdataarr** kopiert.

mov Floating Point Parts

Instruction	Source 2	Source 1	Destination	Description //<availability>
vmovlps	M	X/M	X	Move two packed single-precision floating-point (lower part) to destination //AVX (SSE)
vmovlpd	M	X/M	X	Move double-precision floating-point value (lower part) to destination //AVX (SSE2)
vmovhps	M	X/M	X	Move two packed single-precision floating-point (higher part) to destination //AVX (SSE)
vmovhpd	M	X/M	X	Move double-precision floating-point value (higher part) to destination //AVX (SSE2)

Bitte in Liste korrigieren

Beschreibung korrigiert:

Befehlsbedeutungen

l = lower part of destination, **h** = higher part of destination, **p** = packed

Suffix:
s = single precision, **d** = double precision

Operanden:
X = xmm- oder ymm-Register
M = 128-Bit- oder 256-Bit-Speicher-Operand
maximal ein Operand kann Memory sein

Vorgehensweise bzgl. **vmovlps**, **vmovhps**, **vmovlpd** und **vmovhpd**:

src1 wird komplett nach **dest** kopiert, anschließend wird bei **l** der **lower part** von **dest** und bei **h** der **higher part** von **dest** mit dem **lower part** von **src2** überschrieben.

mov Floating Point Parts

Beispiel:

```
.section .data
.align 16
farr0:
  .float 0.0, 0.0, 0.0, 0.0
.align 16
farr1:
  .float -11.1, 12.4, 13.5, 14.9
.align 16
farr2:
  .float 21.1, -22.4, 23.5, 24.9
.align 16
farr3:
  .double 31.1, 32.9
.align 16
farr4:
  .double 41.1, 42.9

.section .text
.globl _start
.type _start, @function
```

Beschreibung:

farr0:

Inhalt wird mehrfach zur Initialisierung des Registers **xmm0** genutzt.

farr1, farr2:

Inhalte werden in Register **xmm1** bzw. **xmm2** kopiert.

farr3, farr4:

Inhalte werden in Register **xmm3** bzw. **xmm4** kopiert.

Fortsetzung -->

mov Floating Point Parts

Beispiel:

Fortsetzung

```
_start:
  vmovaps farr0, %xmm0  # xmm0 = 0
  vmovaps farr1, %xmm1
  vmovaps farr2, %xmm2
  # low part of farr1 to low part of xmm0, rest of xmm2
  vmovlps farr1, %xmm2, %xmm0
  vmovaps farr0, %xmm0  # xmm0 = 0
  # low part of farr1 to high part of xmm0, rest of xmm2
  vmovhps farr1, %xmm2, %xmm0

  vmovapd farr3, %xmm3
  vmovapd farr4, %xmm4
  vmovaps farr0, %xmm0  # xmm0 = 0
  # low part of farr3 to low part of xmm0, rest of xmm4
  vmovlpd farr3, %xmm4, %xmm0
  vmovaps farr0, %xmm0  # xmm0 = 0
  # low part of farr3 to high part of xmm0, rest of xmm4
  vmovhpd farr3, %xmm4, %xmm0

_end:
  movq $60, %rax
  xorq %rdi, %rdi
  syscall
```

Floating Point Shuffle

Instruction	Src3	Src2	Src1	Dest	Description //<availability>
vshufps	I8	X/M	X	X	Select from quadruplet of single-precision floating-point values in src1 and src2 using I8, interleaved result pairs are stored in dest //AVX (SSE)
vshufpd	I8	X/M	X	X	Shuffle two pairs of double-precision floating-point values from src1 and src2 using I8 to select from each pair, interleaved result is stored in dest //AVX (SSE2)

Befehlsbedeutungen:

Suffix:
s = single precision, **d** = double precision

Operanden:
X = xmm- oder ymm-Register
M = 128-Bit- oder 256-Bit-Speicher-Operand,
maximal ein Operand kann Memory sein

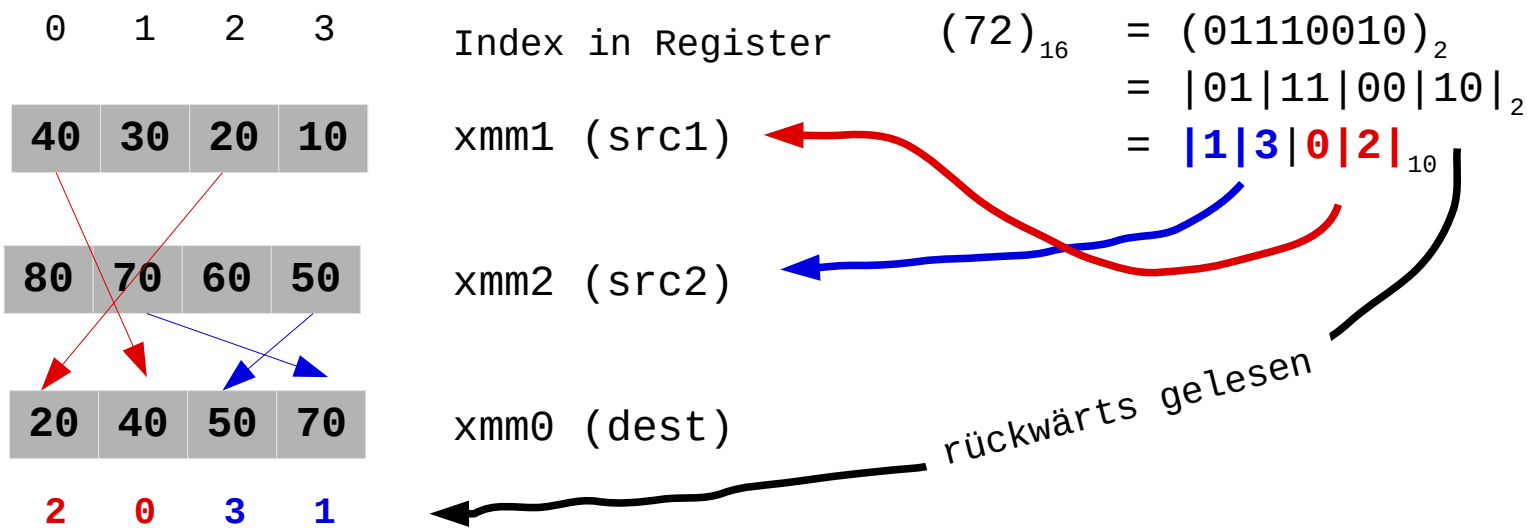
I8:
I8 (Immediate - 8 Bit) besteht aus mehreren 2-Bit-Gruppen, die Indizees darstellen (Leserichtung: rechts → links). Erste Hälfte (rechts) zeigt auf **src1**, zweite (links) auf **src2**.

Floating Point Shuffle

Shuffle-Befehle (engl. to shuffle / dt.: mischen) werden verwendet, um Datenelemente umzusortieren.

Beispiel mit 128 Bit:

```
                    src3,   src2,   src1,   dest
vshufps $0x72, %xmm2, %xmm1, %xmm0
```



Hinweis:

I8 (Immediate 8 Byte, hier 0x72) besteht aus mehreren 2-Bit-Gruppen, die Indizees darstellen. Leserichtung: rechts → links. Erste Hälfte (rechts) zeigt auf **src1**, zweite (links) auf **src2**.

Floating Point Shuffle

Beispiel:

Mit den Zahlen der vorherigen Seite.

shuffle.s

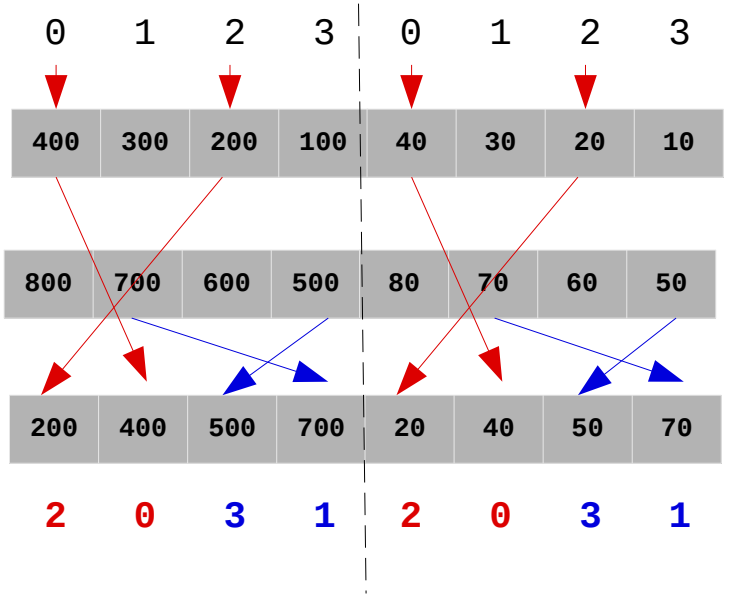
```
.section .data
.align 16
farr0:
  .float 0.0, 0.0, 0.0, 0.0
.align 16
farr1:
  .float 40.0, 30.0, 20.0, 10.0
.align 16
farr2:
  .float 80.0, 70.0, 60.0, 50.0

.section .text
.globl _start
.type _start, @function
_start:
  pushq %rbp
  movq %rsp, %rbp
  vmovaps farr0, %xmm0  # xmm0 = 0
  vmovaps farr1, %xmm1
  vmovaps farr2, %xmm2
  # shuffle
  vshufps $0x72, %xmm2, %xmm1, %xmm0

  # end
  movq $60, %rax
  xorq %rdi, %rdi
  popq %rbp
  syscall
```

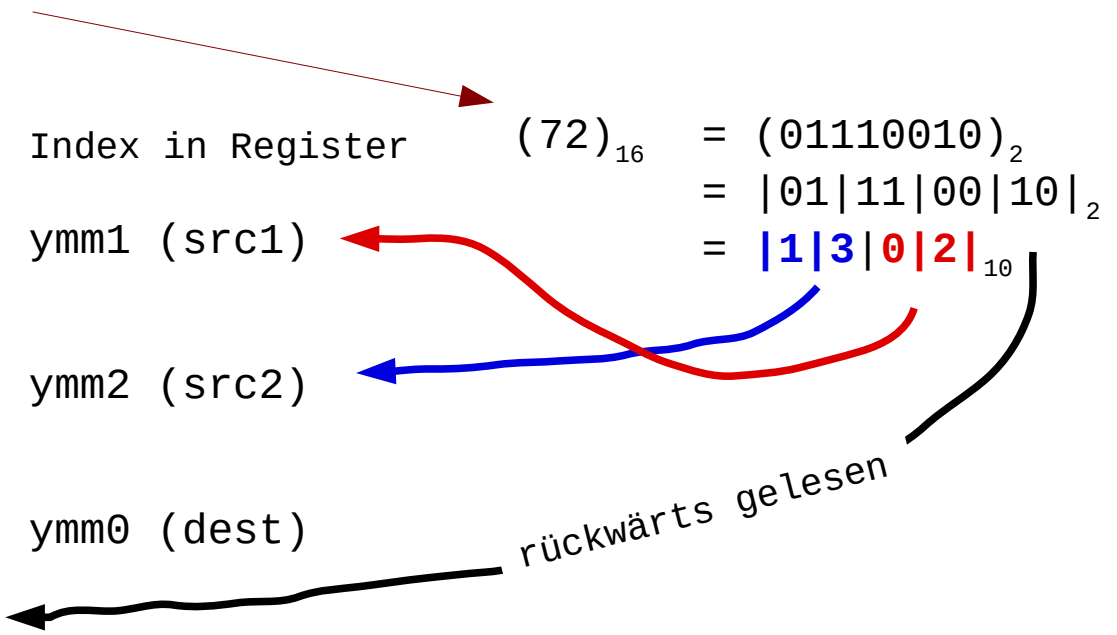
Floating Point Shuffle – mit ymm-Registern

Beispiel mit 256 Bit:

```
                  src3,    src2,    src1,    dest
vshufps $0x72, %ymm2, %ymm1, %ymm0
```

Indizees 0 bis 3 jetzt doppelt vorhanden.





Hinweis:

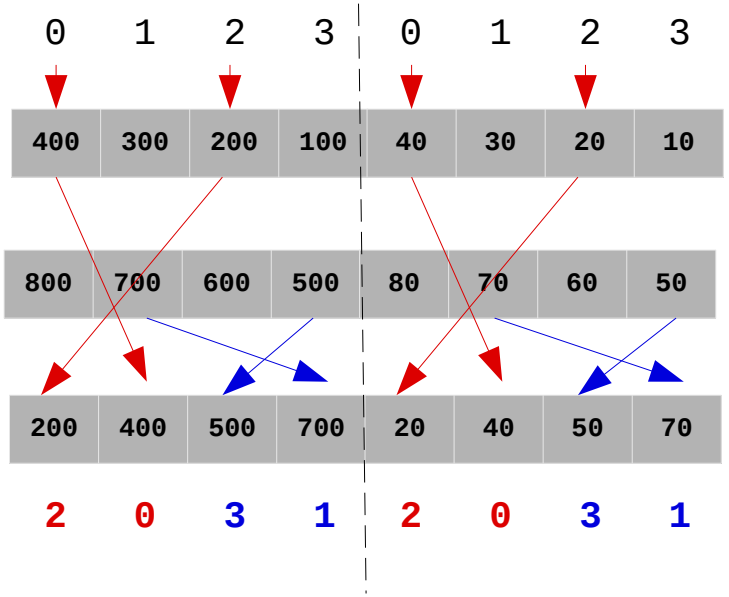
256 Bit werden als 2 x 128 Bit angesehen. Eine Durchmischung zwischen oberen und unteren 128 Bit ist nicht möglich.

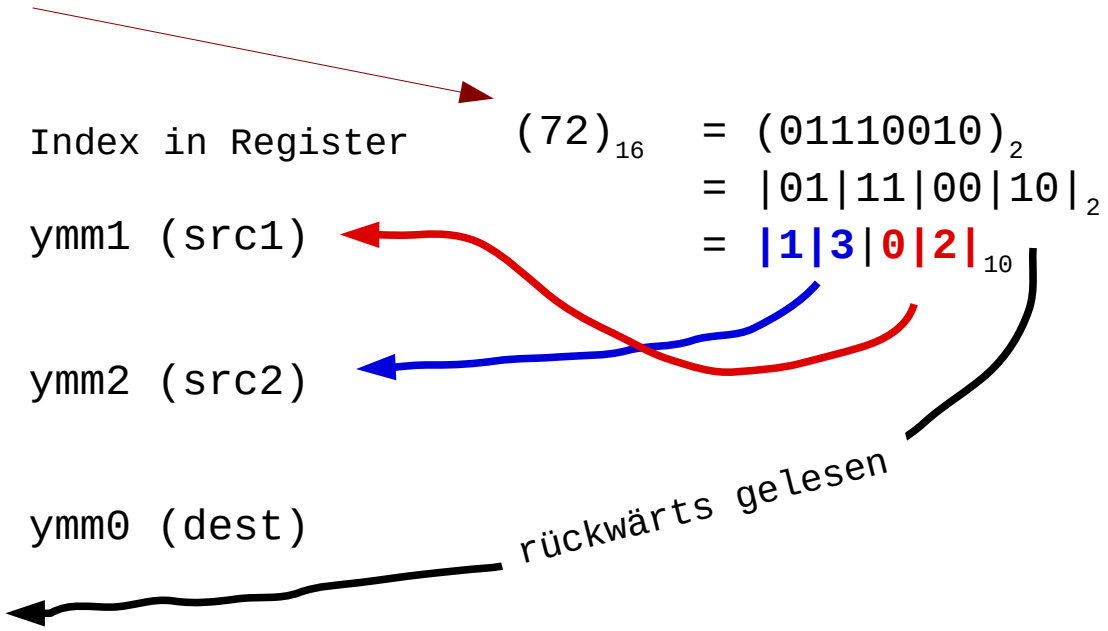
Floating Point Shuffle – mit ymm-Registern

Beispiel mit 256 Bit:

```
                  src3,    src2,    src1,    dest
vshufps $0x72, %ymm2, %ymm1, %ymm0
```

Indizees 0 bis 3 jetzt doppelt vorhanden.





Aufgabe:

Was müsste im Programm **shuffle.s** geändert werden, um die hier abgebildete Situation zu erhalten?

Hinweis:

256 Bit müssen als 2 x 128 Bit angesehen werden. Eine Durchmischung zwischen oberen und untern 128 Bit ist nicht möglich.