```
// 256 Bit Vector, der später zu den 4 extrahierten Bytes
// ein Offset hinzufügt durch Addition
__m256i_u passes = _mm256_setr_epi16(
    256 * 3, 256 * 2, 256 * 1, 256 * 0,
    256 * 3, 256 * 2, 256 * 1, 256 * 0,
    256 * 3, 256 * 2, 256 * 1, 256 * 0,
    256 * 3, 256 * 2, 256 * 1, 256 * 0
);

// 4 Zahlen, von denen die Radixe bestimmt werden sollen,
// werden in einem 256 Bit Vector gespeichert
__m256i_u values = _mm256_setr_epi32(
    0U, vector.at(i), 0U, vector.at(i + 1),
    0U, vector.at(i + 2), 0U, vector.at(i + 3)
);

// 256 Bit Vector, der angibt, wie der Vektor mit den 4 Zahlen verändert werden soll
__m256i_u shufflevector = _mm256_setr_epi8(
    0b00000111, 0b10000000, 0b00000110, 0b10000000,
    0b00000101, 0b10000000, 0b00000100, 0b10000000,
    0b00001111, 0b10000000, 0b00001110, 0b10000000,
    0b00001101, 0b10000000, 0b00001100, 0b10000000,
    0b00000111, 0b10000000, 0b00000110, 0b10000000,
    0b00000101, 0b10000000, 0b00000100, 0b10000000,
    0b00001111, 0b10000000, 0b00001110, 0b10000000,
    0b00001101, 0b10000000, 0b00001100, 0b10000000
);

// Ein neuer 256 Bit Vector mit der Veränderung wird erstellt
__m256i_u shuffled =  _mm256_shuffle_epi8(values, shufflevector);
// Dem neuen 256 Bit Vector werden die Offsets hinzuaddiert
__m256i_u result = _mm256_adds_epu16(shuffled, passes);
```