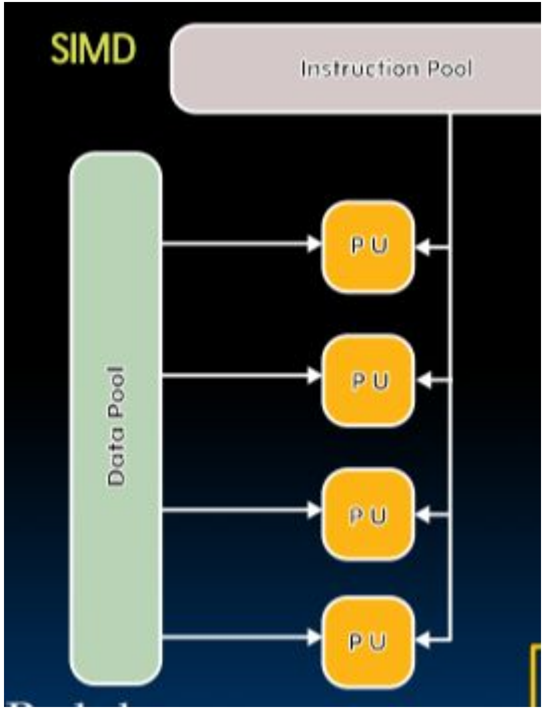
Lab 7

61C Summer 2023

SIMD

- We will be focusing on SIMD today (single instruction multiple streams of data)

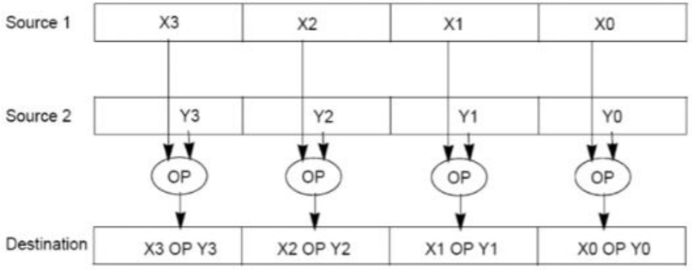


SIMD Intrinsics

- An intrinsic function is a function whose implementation is handled by the compiler
- We will be using SIMD intrinsic functions to speed up our programs

SIMD Intrinsics

- Our SISD (single instruction single data stream) instructions operate on 32 bits
  - Ex. Element wise adding two arrays of size 4 takes 4 instructions {1,2,3,4} + {1,2,3,4} = {2,4,6,8}
- SIMD functions use large (128 bits) registers to store and operate on more value at the same time
  - Now we can stuff 4 ints into 1 register and it will only take us 1 instruction to add 4 ints to 4 ints



Intel Intrinsic Functions

- Guide: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html
- __m128i _mm_setzero_si128() - returns a 128-bit zero vector
- __m128i _mm_loadu_si128(__m128i *p) - returns 128-bit vector stored at pointer p
- __m128i _mm_add_epi32(__m128i a, __m128i b) - returns vector (a_0 + b_0, a_1 + b_1, a_2 + b_2, a_3 + b_3)

Intel Intrinsics

- void _mm_storeu_si128(__m128i *p, __m128i a) - stores 128-bit vector a into pointer p
- __m128i _mm_cmpgt_epi32(__m128i a, __m128i b) - returns the vector (a_i > b_i ? 0xffffffff : 0x0 for i from 0 to 3) (useful as mask for and)
- __m128i _mm_and_si128(__m128i a, __m128i b) - returns vector (a_0 & b_0, a_1 & b_1, a_2 & b_2, a_3 & b_3), where & represents the bitwise and operator

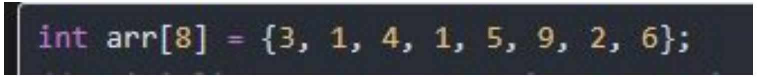
Intel Intrinsics Examples

- We have two 4 int wide arrays, arr1, arr2, add arr1 and arr2 element wise and store it in arr1
  - __m128i vec1 = _mm_loadu_si128((__m128i*) arr1)
  - __m128i vec2 = _mm_loadu_si128((__m128i*) arr2)
  - __m128i result = _mm_add_epi32(vec1, vec2)
  - _mm_storeu_si128((__m128i*) arr1, result)

We have to load arrays from memory into vector registers

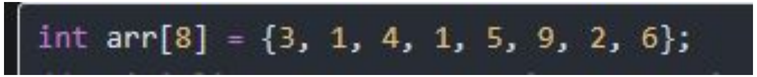
We update arr1

Another Intrinsics Example

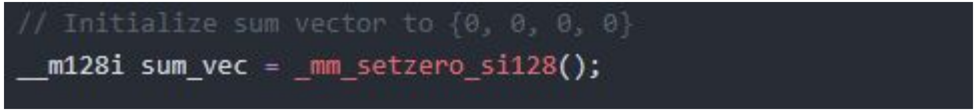


- Given `int arr[8] = {3, 1, 4, 1, 5, 9, 2, 6};` we want to write SIMD code that adds all the elements up together

Another Intrinsics Example



- Given `int arr[8] = {3, 1, 4, 1, 5, 9, 2, 6};` we want to write SIMD code that adds all the elements up together

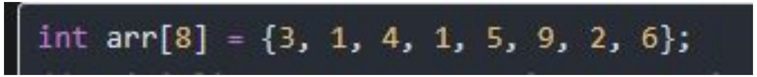


```
// Initialize sum vector to {0, 0, 0, 0}
__m128i sum_vec = _mm_setzero_si128();
```

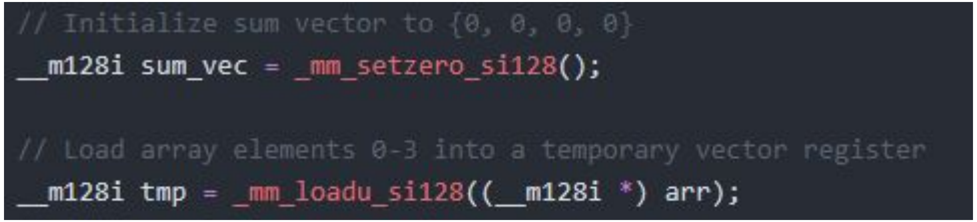
sum_vec:	0	0	0	0

We first create sum_vec(4 ints wide set to all 0s)
to store our sum

Another Intrinsics Example



- Given `int arr[8] = {3, 1, 4, 1, 5, 9, 2, 6};` we want to write SIMD code that adds all the elements up together

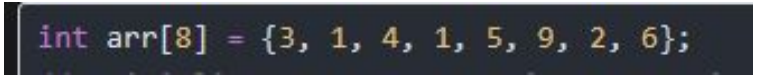


```
// Initialize sum vector to {0, 0, 0, 0}
__m128i sum_vec = _mm_setzero_si128();

// Load array elements 0-3 into a temporary vector register
__m128i tmp = _mm_loadu_si128((__m128i *) arr);
```

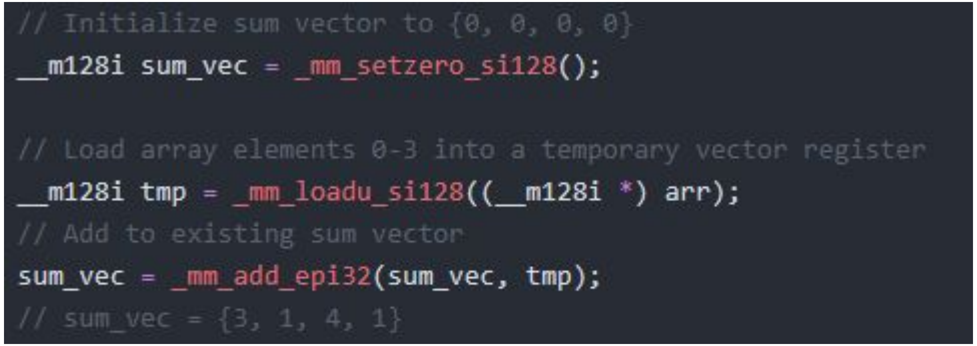
sum_vec:	0	0	0	0
tmp:	3	1	4	1

Next we load in 4 ints (elems 0-3) to a temp vector

Another Intrinsics Example



- Given `int arr[8] = {3, 1, 4, 1, 5, 9, 2, 6};` we want to write SIMD code that adds all the elements up together

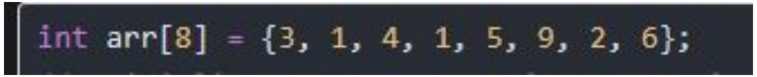


```
// Initialize sum vector to {0, 0, 0, 0}
__m128i sum_vec = _mm_setzero_si128();

// Load array elements 0-3 into a temporary vector register
__m128i tmp = _mm_loadu_si128((__m128i *) arr);
// Add to existing sum vector
sum_vec = _mm_add_epi32(sum_vec, tmp);
// sum_vec = {3, 1, 4, 1}
```

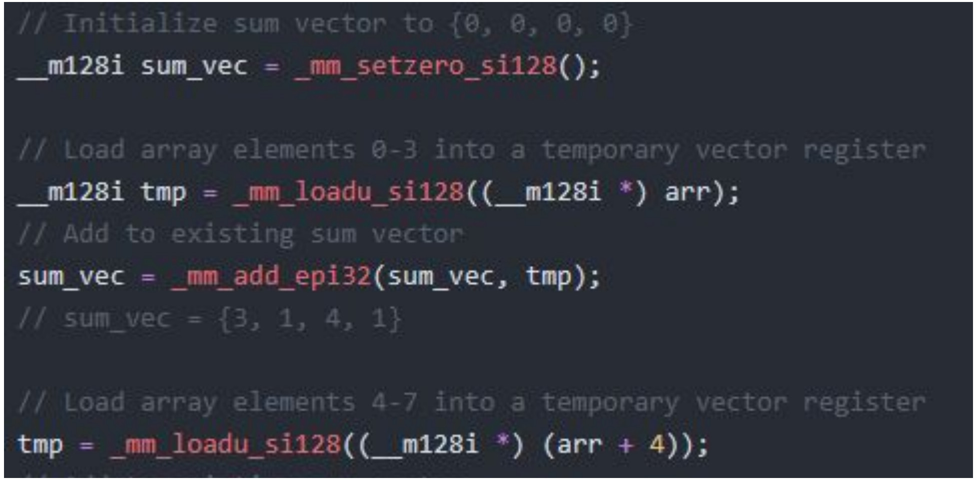
sum_vec:	3	1	4	1
tmp:	3	1	4	1

We add sum_vec and tmp together

Another Intrinsics Example



- Given `int arr[8] = {3, 1, 4, 1, 5, 9, 2, 6};` we want to write SIMD code that adds all the elements up together

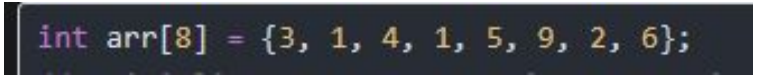


```c
// Initialize sum vector to {0, 0, 0, 0}
__m128i sum_vec = _mm_setzero_si128();

// Load array elements 0-3 into a temporary vector register
__m128i tmp = _mm_loadu_si128((__m128i *) arr);
// Add to existing sum vector
sum_vec = _mm_add_epi32(sum_vec, tmp);
// sum_vec = {3, 1, 4, 1}

// Load array elements 4-7 into a temporary vector register
tmp = _mm_loadu_si128((__m128i *) (arr + 4));
```

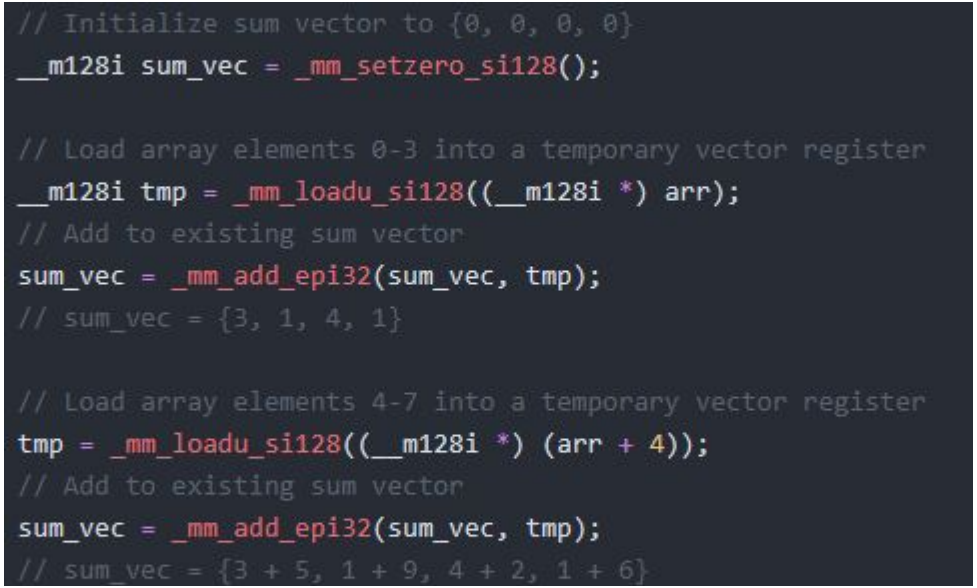
sum_vec:	3	1	4	1
tmp:	5	9	2	6

We load in the next 4 elems of arr into tmp

Another Intrinsics Example



- Given `int arr[8] = {3, 1, 4, 1, 5, 9, 2, 6};` we want to write SIMD code that adds all the elements up together

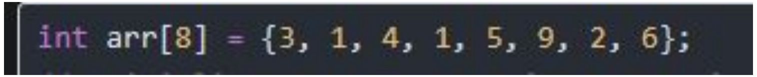


```
// Initialize sum vector to {0, 0, 0, 0}
__m128i sum_vec = _mm_setzero_si128();

// Load array elements 0-3 into a temporary vector register
__m128i tmp = _mm_loadu_si128((__m128i *) arr);
// Add to existing sum vector
sum_vec = _mm_add_epi32(sum_vec, tmp);
// sum_vec = {3, 1, 4, 1}

// Load array elements 4-7 into a temporary vector register
tmp = _mm_loadu_si128((__m128i *) (arr + 4));
// Add to existing sum vector
sum_vec = _mm_add_epi32(sum_vec, tmp);
// sum_vec = {3 + 5, 1 + 9, 4 + 2, 1 + 6}
```

sum_vec:	8	10	6	7
tmp:	5	9	2	6

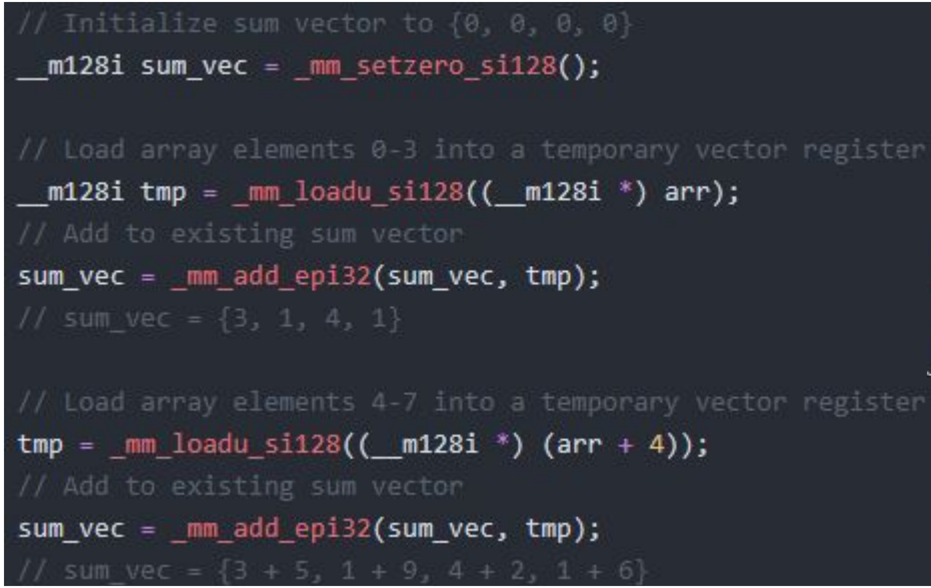
Once again we add sum_vec and tmp

Another Intrinsics Example

- Given 

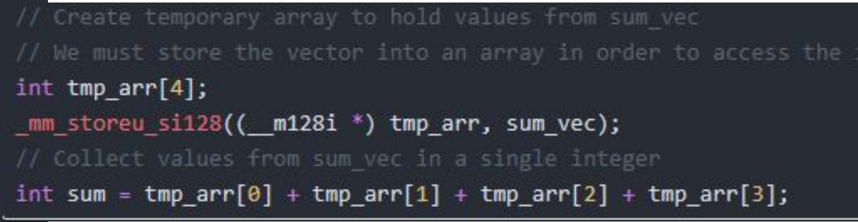
```c
int arr[8] = {3, 1, 4, 1, 5, 9, 2, 6};
```

we want to write SIMD code that adds all the elements up together



```c
// Initialize sum vector to {0, 0, 0, 0}
__m128i sum_vec = _mm_setzero_si128();

// Load array elements 0-3 into a temporary vector register
__m128i tmp = _mm_loadu_si128((__m128i *) arr);
// Add to existing sum vector
sum_vec = _mm_add_epi32(sum_vec, tmp);
// sum_vec = {3, 1, 4, 1}

// Load array elements 4-7 into a temporary vector register
tmp = _mm_loadu_si128((__m128i *) (arr + 4));
// Add to existing sum vector
sum_vec = _mm_add_epi32(sum_vec, tmp);
// sum_vec = {3 + 5, 1 + 9, 4 + 2, 1 + 6}
```

sum_vec:	8	10	6	7
tmp:	5	9	2	6



```c
// Create temporary array to hold values from sum_vec
// We must store the vector into an array in order to access the
int tmp_arr[4];
_mm_storeu_si128((__m128i *) tmp_arr, sum_vec);
// Collect values from sum_vec in a single integer
int sum = tmp_arr[0] + tmp_arr[1] + tmp_arr[2] + tmp_arr[3];
```

Finally, we store sum_vec into a temporary array and then add up all 4 elements of that array

Loop unrolling

- Unrolling a loop (more operations per iteration of loop) will result in slightly improved performance
- Unrolling a loop with SIMD functions will result in even better performance

Loop Unrolling Example

```
int N = 100;
int arr[N];

for (int i = 0; i < N; i += 1) {

        arr[i] = i;

}
```

```
int N = 100;
int arr[N];

for (int i = 0; i < N; i += 4) {

        arr[i] = i;
        arr[i + 1] = i + 1;
        arr[i + 2] = i + 2;
        arr[i + 3] = i + 3;

}
```

Loop Unrolling Example with tail case

```
int N = 103;
int arr[N];

for (int i = 0; i < N; i += 1) {

        arr[i] = i;

}
```

```
int N = 103;
int arr[N];

for (int i = 0; i < N / 4 * 4; i += 4) {
        arr[i] = i;
        arr[i + 1] = i + 1;
        arr[i + 2] = i + 2;
        arr[i + 3] = i + 3;
}

for (int i = N / 4 * 4; i < N; i += 1) {
        arr[i] = i;
}
```