Лабораторная работа № 7
Использование векторных расширений системы команд процессора на примере решения задач обработки одномерных массивов

Цель работы: познакомиться с базовыми принципами векторизации вычислений в соответствии с SIMD-принципом обработки информации.

Векторные расширения системы команд процессора работают в соответствии с SIMD-принципом обработки информации, в основе которого лежит выполнение однотипной операции над векторами операндов, размер которых определяется типом расширения (MMX, SSE, AVX) и используемым типом данных (целые числа различного размера (8-, 16-, 32-, 64-разрядные) – PADDB, PMULLW, PANDW и т.д., вещественные числа с плавающей точкой заданного типа (половинной, одинарной или двойной точности) – ADDPS, MULPD и т.д.). Иногда действия, выполняемые одной командой, могут быть различными (гетерогенными), что бывает полезно, например, при реализации операций с комплексными числами (ADDSUBPD и т.п.). Также в состав векторных расширений входят команды для перераспределения данных в составе обрабатываемых векторов (MOVHLxx, MOVLHxx, SHUFxx, PACKxx, UNPCKxx, MOVxDUP и др.).

Арифметические и логические команды векторных расширений подразделяются на 3 основных типа (см. рис. 1):

- скалярные – операция производится только для младших компонентов векторов (например, ADDSS);
- векторные – операция производится для всех пар операндов в составе пары заданных векторов (например, MULPD);
- горизонтальные – операция производится над компонентами одного вектора (например, HADDPS).

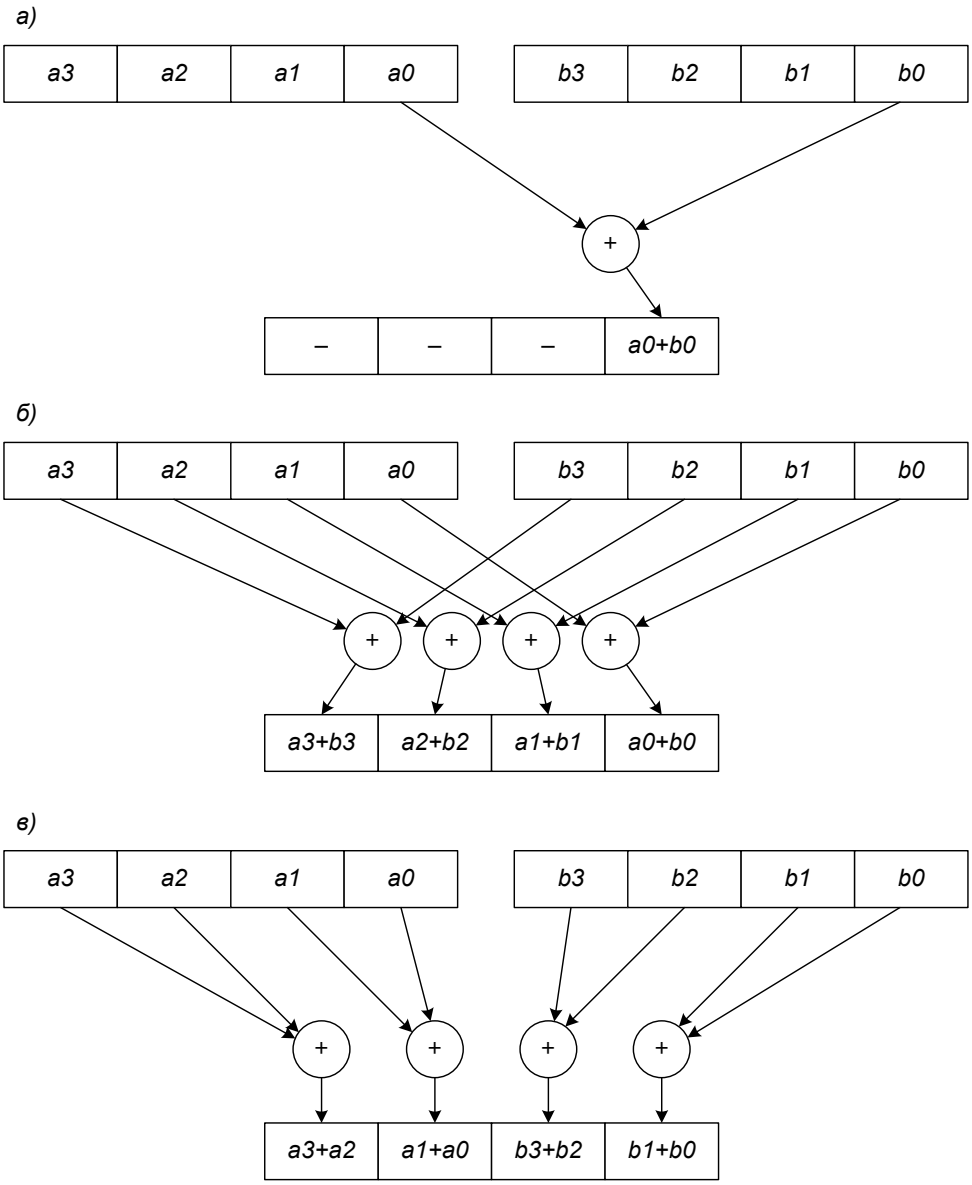

Рис. 1. Примеры скалярной (а), векторной (б) и горизонтальной (в) команд

Программирование с использованием векторных расширений возможно путем написания кода программы на языке ассемблера, другим способом является использованием высокоуровневых аналогов ассемблерных команд – т.н. интринсиков (англ. intrinsics). Например, для векторного сложения пары векторов C[3:0] := A[3:0] + B[3:0] вещественных чисел одинарной точности может быть использован код на языке ассемблера

```
MOVAPS      XMM0, [A]
ADDPS       XMM0, [B]
MOVAPS      [C], XMM0
```

либо интринсик

```
__m128 C = _mm_add_ps(__m128 A, __m128 B);
```

на языке высокого уровня. Для работы с интринсиками потребуется подключение ряда заголовочных файлов, например, immintrin.h.

При программировании SIMD-расширений на языке ассемблера возможно использование следующих регистров (табл. 1).

Таблица 1. Регистры векторных расширений системы команд процессора

Тип расширения	**Названия регистров**	**Размер регистра**
MMX	MM0, MM1, …, MM7	64 бита (8 байт)
SSE/SSE2/SSE3/SSSE3/SSE4	XMM0, XMM1, … XMM7 (32-битный режим) XMM0, XMM1, … XMM15 (64-битный режим)	128 бита (16 байт)
AVX/AVX2	YMM0, YMM1, …, YMM15	256 бита (32 байта)
AVX-512	ZMM0, ZMM1, …, ZMM31 (регистры общего назначения) k0, k1, …, k7 (регистры масок)	512 бит (64 байта) 64 бита (8 байт)

При использовании интринсиков распределением переменных по регистрам занимается компилятор прозрачно для программиста. При использовании регистров MM0–MM7 необходимо помнить, что они физически хранятся поверх регистров кольцевого стека сопроцессора ST(0)–ST(7), ввиду чего одновременное использование команд MMX и FPU недопустимо, а после последней команды расширения MMX перед использованием FPU команд необходимо использовать команду EMMS. В настоящее время команды расширений AVX/AVX2/AVX-512 имеют поддержку как целочисленных, так и вещественных операций, использование команды EMMS совместно с ними не требуется.

Регистры XMM являются младшими частями регистров YMM, а последние в свою очередь – младшими частями регистров ZMM (см. рис. 2). При написании кода использование команд различных расширений (например, AVX и SSE) не приветствуется, вместо этого в случае крайней необходимости рекомендуется использовать AVX-аналоги SSE-команд, обнуляющих старшие половины регистров (например, VADDPS с XMM-регистрами вместо ADDPS).

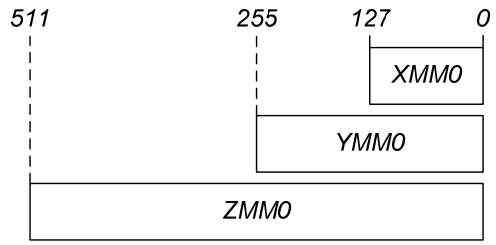

Рис. 2. XMM, YMM и ZMM регистры

Для очистки всех регистров (XMM0–XMM15, YMM0–YMM15, ZMM0–ZMM15 в 64-битном режиме) применяется команда VZEROALL, старшие регистры ZMM16–ZMM31 при этом не изменяются. В 32-битном режиме очистке подвергаются только младшие 8 регистров. Для очистки старших бит регистров YMM и ZMM (с номерами старше 128) применяется команда VZEROUPPER, что может быть полезно при переключении между AVX- и SSE-кодом.

Для объявления переменных при использовании интринсиков применяются следующие основные типы данных (табл. 2).

Таблица 2. Типы данных, используемые при программировании векторных расширений с использованием интринсиков

Обозначение	Тип расширения	Размер	Трактовка
__m64	MMX	64 бита (8 байт)	Вектор целочисленных значений
__m128i	SSE2	128 бита (16 байт)	
__m256i	AVX2	256 бита (32 байт)	
__m512i	AVX-512	512 бита (64 байт)	
__m128	SSE/SSE2/SSE3	128 бита (16 байт)	Вектор вещественных значений
__m256	AVX/AVX2	256 бита (32 байт)	
__m512	AVX-512	512 бита (64 байт)	

Интерпретация логического содержимого вектора определяется используемой командой (интринсиком). Например, PADDB – вектора байт, PADDW – вектора слов (каждый элемент – 2 байта), PADDD – вектора двойных слов (4 байта); _mm_add_ps() – вектора вещественных чисел одинарной точности, _mm_add_pd() – вектора вещественных чисел двойной точности.

При работе с векторами важным является выравнивание данных в памяти. При соблюдении правильного выравнивания, когда адрес очередной порции данных, загружаемых/выгружаемых из/в память, кратен размеру вектора (64, 128 или 256 бит в зависимости от расширения) операции обращения к подсистеме памяти будут выполняться максимально эффективно и потребуют передачи минимального количества 64-байтных порций данных между памятью и кэш-памятью различного уровня. В случае обращения к невыровненным данным может потребоваться большее количество обращений, либо будет сгенерировано исключение. В составе некоторых расширений присутствуют отдельные команды для обращения к выровненным или невыровненным данным (например, MOVAPS – MOV **Aligned** Parallel Single и MOVUPS – MOV **Unaligned** Parallel Single). Команды выровненного обращения к памяти обычно характеризуются меньшей латентностью.

Перечень основных команд SIMD-расширений приведен ниже (табл. 3–11).

Таблица 3. Основные команды расширения MMX (целочисленные операции)

Операция	Команда на языке ассемблера	Интринсик
Сложение	PADDB PADDW PADDD PADDQ	_mm_add_pi8() _mm_add_pi16() _mm_add_pi32() _mm_add_si64()
Вычитание	PSUBB PSUBW PSUBD PSUBQ	_mm_sub_pi8() _mm_sub_pi16() _mm_sub_pi32() _mm_sub_si64()
Умножение	PMULLW PMULHW	_mm_mullo_pi16() _mm_mulhi_pi16()
Частичное скалярное произведение	PMADDWD	_mm_madd_pi16()

Сравнение	PCMPEQ[B/W/D] PCMPGT[B/W/D]	_mm_cmpeq_pi[8/16/32]() _mm_cmpgt_pi[8/16/32]()
Минимум, максимум	PMINUB PMAXUB	_mm_min_pu8() _mm_max_pu8()
Пересылка, упаковка/распаковка, перемешивание	MOVD MOVQ PACKSSWB PACKSSDW PUNPCKHBW PUNPCKHWD PUNPCKHDQ UNPCKLBW UNPCKLWD UNPCKLDQ PSHUF[B/W]	_mm_cvtsi32_si64() _mm_unpackhi_pi8() _mm_unpackhi_pi16() _mm_unpackhi_pi32() _mm_unpacklo_pi8() _mm_unpacklo_pi16() _mm_unpacklo_pi32() _mm_shuffle_pi[8/16]()

Таблица 4. Основные команды расширения SSE (вещественные операции одинарной точности)

Операция	**Команда на языке ассемблера**	**Интринсик**
Скалярное сложение	ADDSS	_mm_add_ss()
Скалярное умножение	MULSS	_mm_mul_ss()
Скалярная обратная величина $\left(1/x\right)$	RCPSS	_mm_rcp_ss()
Скалярное деление	DIVSS	_mm_div_ss()
Скалярный квадратный корень	SQRTSS	_mm_sqrt_ss()
Скалярное сравнение	CMPSS	_mm_cmp_ss()
Скалярный минимум, максимум	MINSS MAXSS	_mm_min_ss() _mm_max_ss()
Векторное сложение	ADDPS	_mm_add_ps()
Векторное умножение	MULPS	_mm_mul_ps()
Векторная обратная величина $\left(1/x\right)$	RCPPS	_mm_rcp_ps()
Векторное деление	DIVPS	_mm_div_ps()
Векторный квадратный корень	SQRTPS	_mm_sqrt_ps()
Векторное сравнение	CMPPS	_mm_cmp_ps()
Векторный минимум, максимум	MINPS MAXPS	_mm_min_ps() _mm_max_ps()
Пересылка, упаковка/распаковка, перемешивание	MOVSS MOVAPS	_mm_move_ss() _mm_load_ss() _mm_store_ss() _mm_load_ps()

	MOVUPS SHUFPS, MOVHLPS, MOVLHPS, …	_mm_store_ps() _mm_loadu_ps() _mm_storeu_ps() _mm_shuffle_ps() …

Таблица 5. Основные команды расширения SSE2 (вещественные операции двойной точности)

Операция	**Команда на языке ассемблера**	**Интринсик**
Скалярное сложение	ADDSD	_mm_add_sd()
Скалярное умножение	MULSD	_mm_mul_sd()
Скалярное деление	DIVSD	_mm_div_sd()
Скалярный квадратный корень	SQRTSD	_mm_sqrt_sd()
Скалярное сравнение	CMPSD	_mm_cmp_sd()
Скалярный минимум, максимум	MINSD MAXSD	_mm_min_sd() _mm_max_sd()
Векторное сложение	ADDPD	_mm_add_pd()
Векторное умножение	MULPD	_mm_mul_pd()
Векторное деление	DIVPD	_mm_div_pd()
Векторный квадратный корень	SQRTPD	_mm_sqrt_pd()
Векторное сравнение	CMPPD	_mm_cmp_pd()
Векторный минимум, максимум	MINPD MAXPD	_mm_min_pd() _mm_max_pd()
Пересылка, упаковка/распаковка, перемешивание	MOVSD MOVAPD MOVUPD MOVHPD MOVLPD MOVMSKPD SHUFPD UNPCKHPD UNPCKLPD	_mm_load_sd() _mm_store_sd() _mm_move_sd() _mm_load_pd() _mm_store_pd() _mm_loadh_pd() _mm_loadl_pd() _mm_storeh_pd() _mm_storel_pd() …

Таблица 6. Основные команды расширения SSE2 (целочисленные операции)

Операция	**Команда на языке ассемблера**	**Интринсик**
Сложение	PADDB PADDW PADDD	_mm_add_epi8() _mm_add_epi16() _mm_add_epi32()

	PADDQ	_mm_add_epi64()
Вычитание	PSUBB PSUBW PSUBD PSUBQ	_mm_sub_epi8() _mm_sub_epi16() _mm_sub_epi32() _mm_sub_esi64()
Умножение	PMULLW PMULHW PMULLD	_mm_mullo_epi16() _mm_mulhi_epi16() _mm_mullo_epi32()
Частичное скалярное произведение	PMADDWD	_mm_madd_epi16()
Сравнение	PCMPEQ[B/W/D/Q] PCMPGT[B/W/D/Q]	_mm_cmpeq_epi[8/16/32/64]() _mm_cmpgt_epi[8/16/32/64]()
Минимум, максимум	PMINUB PMAXUB	_mm_min_epu8() _mm_max_epu8()
Пересылка, упаковка/распаковка, перемешивание	MOVSD MOVAPD MOVUPD MOVDAQ MOVDQU PACKSSWB PACKSSDW PUNPCKHBW PUNPCKHWD PUNPCKHDQ UNPCKLBW UNPCKLWD UNPCKLDQ PSHUF[B/W]	_mm_load_[sd/si256]() _mm_store_[sd/si256] () _mm_move_[sd/si256] () _mm_unpackhi_epi8() _mm_unpackhi_epi16() _mm_unpackhi_epi32() _mm_unpacklo_epi8() _mm_unpacklo_epi16() _mm_unpacklo_epi32() _mm_shuffle_epi[8/16]()

Таблица 7. Основные команды расширения SSE3/SSSE3

Операция	**Команда на языке ассемблера**	**Интринсик**
Горизонтальное сложение	PHADD[W/SW/D] HADDPS HADDPD	_mm_hadd_epi[16/32] _mm_hadds_epi[16/32] _mm_hadd_ps() _mm_hadd_pd()
Горизонтальное вычитание	PHSUB[W/SW/D] HSUBPS HSUBPD	_mm_hsub_epi[16/32] _mm_hsubs_epi[16/32] _mm_hsub_ps() _mm_hsub_pd()

Таблица 8. Основные команды расширения AVX (вещественные операции одинарной и двойной точности)

Операция	**Команда на языке ассемблера**	**Интринсик**

Скалярное сложение	VADDSS VADDSD	_mm_add_ss() _mm_add_sd()
Скалярное умножение	VMULSS VMULSD	_mm_mul_ss() _mm_mul_sd()
Скалярный квадратный корень	VSQRTSS VSQRTSD	_mm_sqrt_ss() _mm_sqrt_sd()
Скалярное деление	VDIVSS VDIVSD	_mm_div_ss() _mm_div_sd()
Векторное сложение	VADDPS VADDPD	_mm256_add_ps() _mm256_add_pd()
Векторное умножение	VMULPS VMULPD	_mm256_mul_ps() _mm256_mul_pd()
Векторный квадратный корень	VSQRTPS VSQRTPD	_mm256_sqrt_ps() _mm256_sqrt_pd()
Векторное деление	VDIVPS VDIVPD	_mm256_div_ps() _mm256_div_pd()
Векторное сравнение	VCMPPS VCMPPD	_mm256_cmp_ps() _mm256_cmp_pd()
Векторный минимум и максимум	VMINPS VMINPD VMAXPS VMAXPD	_mm256_min_ps() _mm256_min_pd() _mm256_max_ps() _mm256_max_pd()
Совмещенное сложение-умножение	VFMADD[123]PS VFMADD[123]PD	_mm256_fmadd_ps() _mm256_fmadd_pd()
Пересылка, перемешивание	VMOVAPD VMOVUPD VSHUFPS VSHUFPD …	_mm256_load_sd() _mm256_store_sd() _mm256_loadu_sd() _mm256_storeu_sd() _mm256_shuffle_ps() _mm256_shuffle_pd() …
Горизонтальное сложение	VHADDPS VHADDPD	_mm256_hadd_ps() _mm256_hadd_pd()

Таблица 9. Основные команды расширения AVX2 (целочисленные операции)

Операция	**Команда на языке ассемблера**	**Интринсик**
Сложение	VPADD[B/W/D/Q]	_mm256_add_epi[8/16/32/64]()
Вычитание	VPSUB[B/W/D/Q]	_mm256_add_epi[8/16/32/64]()
Скалярное умножение	VPMULLW VPMULHW VPMULLD	_mm256_mullo_epi16() _mm256_mulhi_epi16() _mm256_mullo_epi32()
Частичное скалярное произведение	VPMADDWD	_mm256_madd_epi16()
Сравнение	VPCMPEQ[B/W/D/Q] VPCMPGT[B/W/D/Q]	_mm256_cmpeq_epi[8/16/32/64]() _mm256_cmpgt_epi[8/16/32/64]()
Минимум, максимум	VPMINUB	_mm256_min_epu8()

	VPMINUW VPMINUB VPMINUW	_mm256_min_epu16() _mm256_max_epu8() _mm256_max_epu16()
Пересылка, упаковка/распаковка, перемешивание	VMOVDQA VMOVDQU VPACKSSWB VPACKSSDW VPUNPCKHBW VPUNPCKHWD VPUNPCKHDQ VUNPCKLBW VUNPCKLWD VUNPCKLDQ VPSHUF[B/W]	_mm256_load_si256() _mm256_loadu_si256() _mm256_store_si256() _mm256_storeu_si256() _mm256_packs_epi16() _mm256_packs_epi32() _mm256_unpackhi_epi8() _mm256_unpackhi_epi16() _mm256_unpackhi_epi32() _mm256_unpacklo_epi8() _mm256_unpacklo_epi16() _mm256_unpacklo_epi32() _mm256_shuffle_epi[8/16]()

Таблица 10. Основные команды расширения AVX-512 (целочисленные операции)

Операция	**Команда на языке ассемблера**	**Интринсик**
Сложение	VPADD[B/W/D/Q]	_mm512_add_epi[8/16/32/64]()
Вычитание	VPSUB[B/W/D/Q]	_mm512_add_epi[8/16/32/64]()
Скалярное умножение	VPMULLW VPMULHW VPMULLD	_mm512_mullo_epi16() _mm512_mulhi_epi16() _mm512_mullo_epi32()
Частичное скалярное произведение	VPMADDWD	_mm512_madd_epi16()
Сравнение	VPCMPEQ[B/W/D/Q] VPCMPGT[B/W/D/Q]	_mm512_cmpeq_epi[8/16/32/64]() _mm512_cmpgt_epi[8/16/32/64]()
Минимум, максимум	VPMINUB VPMINUW VPMINUB VPMINUW	_mm512_min_epu8() _mm512_min_epu16() _mm512_max_epu8() _mm512_max_epu16()
Пересылка, упаковка/распаковка, перемешивание	VMOVDQA VMOVDQU VPACKSSWB VPACKSSDW VPUNPCKHBW VPUNPCKHWD VPUNPCKHDQ VUNPCKLBW VUNPCKLWD VUNPCKLDQ	_mm512_load_si256() _mm512_loadu_si256() _mm512_store_si256() _mm512_storeu_si256() _mm512_packs_epi16() _mm512_packs_epi32() _mm512_unpackhi_epi8() _mm512_unpackhi_epi16() _mm512_unpackhi_epi32() _mm512_unpacklo_epi8() _mm512_unpacklo_epi16() _mm512_unpacklo_epi32()

	VPSHUF[B/W]	_mm512_shuffle_epi[8/16]()

Таблица 11. Основные команды расширения AVX-512 (вещественные операции одинарной и двойной точности)

Операция	**Команда на языке ассемблера**	**Интринсик**
Скалярное сложение	VADDSS VADDSD	_mm_add_ss() _mm_add_sd()
Скалярное умножение	VMULSS VMULSD	_mm_mul_ss() _mm_mul_sd()
Скалярный квадратный корень	VSQRTSS VSQRTSD	_mm_sqrt_ss() _mm_sqrt_sd()
Скалярное деление	VDIVSS VDIVSD	_mm_div_ss() _mm_div_sd()
Векторное сложение	VADDPS VADDPD	_mm512_add_ps() _mm512_add_pd()
Векторное умножение	VMULPS VMULPD	_mm512_mul_ps() _mm512_mul_pd()
Векторный квадратный корень	VSQRTPS VSQRTPD	_mm512_sqrt_ps() _mm512_sqrt_pd()
Векторное деление	VDIVPS VDIVPD	_mm512_div_ps() _mm512_div_pd()
Векторное сравнение	VCMPPS VCMPPD	_mm512_cmp_ps() _mm512_cmp_pd()
Векторный минимум и максимум	VMINPS VMINPD VMAXPS VMAXPD	_mm512_min_ps() _mm512_min_pd() _mm512_max_ps() _mm512_max_pd()
Совмещенное сложение-умножение	VFMADD[123]PS VFMADD[123]PD	_mm512_fmadd_ps() _mm512_fmadd_pd()
Пересылка, перемешивание	VMOVAPD VMOVUPD VSHUFPS VSHUFPD …	_mm512_load_sd() _mm512_store_sd() _mm512_loadu_sd() _mm512_storeu_sd() _mm512_shuffle_ps() _mm512_shuffle_pd() …

В настоящее время при разработке приложений с поддержкой SIMD-расширений необходимо ориентироваться на использование расширений AVX/AVX2, поддерживаемых абсолютным большинством современных процессоров, и на использование расширения AVX-512 в ближайшей временной перспективе. Расширения MMX и SSE в настоящее время считаются устаревшими и их использование на практике не рекомендуется.

Индивидуальные варианты заданий

Действия над векторами:

1. Поэлементное сложение: $c_i = a_i + b_i$.
2. Горизонтальное сложение: $s = \sum_{i=1}^{n} a_i = a_1 + a_2 + ... + a_n$.
3. Поэлементное умножение: $c_i = a_i \cdot b_i$.
4. Скалярное произведение: $d = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + ... + a_n b_n$.
5. Длина вектора (евклидова норма): $l = \sqrt{\sum_{i=1}^{n} a_i^2} = \sqrt{a_1^2 + a_2^2 + ... + a_n^2}$.
6. Нормировка значений: $a_i' = \frac{a_i - \min\limits_{j=1,n} a_j}{\max\limits_{j=1,n} a_j - \min\limits_{j=1,n} a_j} \cdot V$, V – максимальное значение диапазона $[0; V]$.
7. Масштабирование значений: $a_i' = a_i \cdot k$, $k \in \mathbb{R}$.
8. Смешивание значений: $c_i = a_i \cdot \alpha + b_i \cdot (1 - \alpha)$, $\alpha \in \mathbb{R}$, $0 \le \alpha \le 1$.
9. Сравнение значений: $c_i = \begin{cases} 0, & a_i < b_i, \\ 1, & a_i \ge b_i. \end{cases}$.
10. Среднее арифметическое: $s = \frac{1}{n} \sum_{i=1}^{n} a_i = \frac{a_1 + a_2 + ... + a_n}{n}$.
11. Среднее геометрическое: $s = \sqrt[n]{\prod_{i=1}^{n} a_i} = (a_1 \cdot a_2 \cdot ... \cdot a_n)^{1/n}$.
12. Среднее гармоническое: $s = \frac{n}{\sum_{i=1}^{n} \frac{1}{a_i}} = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + ... + \frac{1}{a_n}}$.
13. Манхеттенская норма: $s = \sum_{i=1}^{n} |x_i| = |x_1| + |x_2| + ... + |x_n|$.
14. p-норма: $s = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p} = \left(|x_1|^p + |x_2|^p + ... + |x_n|^p \right)^{1/p}$.
15. m-норма: $s = \max\limits_{i=1,n} |x_i| = \max\left(|x_1|, |x_2|, ..., |x_n| \right)$.

Размер массивов для упрощения допускается выбирать кратным размеру вектора в используемом векторном расширении системы команд процессора.

Таблица 12. Индивидуальные варианты заданий

Номер варианта	**Тип данных (тип результата)**	**Операция**	**Расширение**
1	int 8	1	MMX
2			SSE2
3			AVX2

4		2	MMX
5			SSE2
6			AVX2
7		3	MMX
8			SSE2
9			AVX2
10		4	MMX
11			SSE2
12			AVX2
13		5	MMX→SSE
14			SSE2→SSE
15			AVX2→AVX
16		6	MMX→SSE→MMX
17			SSE2→SSE→SSE2
18			AVX2→AVX→AVX2
19		7	MMX→SSE→MMX
20			SSE2→SSE→SSE2
21			AVX2→AVX→AVX2
22		8	MMX→SSE→MMX
23			SSE2→SSE→SSE2
24			AVX2→AVX→AVX2
25		9	MMX
26			SSE2
27			AVX
		10	MMX
			SSE2
			AVX2
		11	SSE
			AVX
		12	MMX→SSE
			AVX2→AVX
		13	MMX
			AVX2
		14	MMX→SSE
			AVX2→AVX
		15	MMX
			AVX2
28	int 16	1	MMX
29			SSE2
30			AVX2
31		2	MMX
32			SSE2
33			AVX2
34		3	MMX
35			SSE2
36			AVX2
37		4	MMX
38			SSE2

39			AVX2
40		5	MMX→SSE
41			SSE2→SSE
42			AVX2→AVX
43		6	MMX→SSE→MMX
44			SSE2→SSE→SSE2
45			AVX2→AVX→AVX2
46		7	MMX→SSE→MMX
47			SSE2→SSE→SSE2
48			AVX2→AVX→AVX2
49		8	MMX→SSE→MMX
50			SSE2→SSE→SSE2
51			AVX2→AVX→AVX2
52		9	MMX
53			SSE2
54			AVX
55		10	MMX
56		11	SSE2
57			AVX2
58			SSE
59		12	AVX
60			MMX→SSE
61			AVX2→AVX
62		13	MMX
63			AVX2
64			MMX→SSE
65		14	MMX
66			AVX2
67			MMX→SSE
68		15	AVX2→AVX
69			MMX
70			AVX2
71	int 32	1	MMX→SSE
72			SSE2→SSE
73			AVX2→AVX
74		2	MMX→SSE
75			SSE2→SSE
76			AVX2→AVX
77		3	MMX→SSE
78			SSE2→SSE
79			AVX2→AVX
80		4	MMX→SSE
81			SSE2→SSE
82			AVX2→AVX
83		5	MMX→SSE
84			SSE2→SSE

85			AVX2→AVX
86		6	MMX→SSE→MMX
87			SSE2→SSE→SSE2
88			AVX2→AVX→AVX2
89		7	MMX→SSE→MMX
90			SSE2→SSE→SSE2
91			AVX2→AVX→AVX2
92		8	MMX→SSE→MMX
93			SSE2→SSE→SSE2
94			AVX2→AVX→AVX2
95		9	MMX
96			SSE2
97			AVX
98	float	1	SSE/SSE3
99			AVX/FMA3
100		2	SSE/SSE3
101			AVX/FMA3
102		3	SSE
103			AVX
104		4	SSE/SSE3
105			AVX/FMA3
106		5	SSE/SSE3
107			AVX/FMA3
108		6	SSE
109			AVX
110		7	SSE
111			AVX
112		8	SSE
113			AVX
114		9	SSE
115			AVX
116		10	SSE2
117			AVX2
118		11	SSE
119			AVX
120		12	SSE
121			AVX
122		13	SSE
123			AVX
124		14	SSE
125			AVX
126		15	SSE

127	double		AVX
128		1	SSE2
129			AVX
130		2	SSE2
131			AVX
132		3	SSE2
133			AVX
134		4	SSE2
135			AVX
136		5	SSE2
137			AVX
138		6	SSE2
139			AVX
140		7	SSE2
141			AVX
142		8	SSE2
143			AVX
144		9	SSE2
145			AVX
146		10	SSE2
147			AVX
148		11	SSE2
149			AVX
150		12	SSE2
151			AVX
152		13	SSE2
153			AVX
154		14	SSE2
155			AVX
156		15	SSE2
157			AVX

Замечание. При выполнении целочисленного умножения учесть, что результат умножения в битах в два раза больше размера операндов (например, при умножении 8-битных чисел результат будет 16-битным).

Примеры программ

Пример 1. Сумма двух векторов вещественных чисел одинарной точности на ассемблере с использованием расширения SSE.

```
const int N = 1024;

__declspec(align(16)) float a[N], b[N], c[N];

// Инициализация значений массивов
for (size_t i=0; i<N; i++)
{
    a[i] = i;
    b[i] = i*i;
}

// Обработка с использованием команд расширения SSE
```

```
__asm{
    LEA     RAX, [A]
    LEA     RBX, [B]
    LEA     RDX, [C]
    XOR     RCX, RCX            // ECX = i

@@Loop:
    MOVAPS  XMM0, [RAX+RCX]     // XMM0 = (a[i+3], a[i+2], a[i+1], a[i])
    ADDPS   XMM0, [RBX+RCX]     // XMM0 = (a[i+3]+b[i+3], a[i+3]+b[i+2], a[i+1]+b[i+1], a[i]+b[i])
    MOVAPS  [RDX+RCX], XMM0

    ADD     RCX, 16
    CMP     RCX, N
    JNZ     @@Loop
}
```

Замечание. Код может быть реализован без завершающей команды CMP если присвоить регистру RCX начальное значение –N с последующим его инкрементом на 16 до достижения нулевого значения, что можно проверить командой JZ по значениям флагов, установленных предыдущей командой ADD.

Пример 2. Поиск минимального значения вектора слов с использованием интринсиков и команд расширения AVX2.

```
const int N = 1024;

__declspec(align(16)) __int16 a[N];

// Инициализация значений массива
for (size_t i=0; i<N; i++)
    a[i] = i;

// Обработка с использованием команд расширения AVX2

// Поиск минимального значения
__m256i min_vector = _mm256_setzero_si256();

for (size_t i=0; i<N; i+=32)
{
    __m256i curr_vector = _mm256_load_si256(a[i]);

    min_vector = _mm256_min_epu16(min_vector, curr_vector);
}

// Редукция вектора до одного значения

__m128i hi = _mm256_extractf128_ps(min_vector, 1);
__m128i lo = _mm256_castps256_ps128(min_vector);

lo = _mm256_min_epu16(lo, hi);
__m128i shuf = _mm_movehl_ps(lo, lo);

lo = _mm_min_epu16(lo, shuf);
shuf = _mm_shuffle_ps(lo, lo, 1);

lo = _mm_min_epu16(lo, shuf);

__int16 min_value = _mm_cvtsi128_si32(lo) & 0x000000FF;
```

Задание

В соответствии с индивидуальным вариантом реализовать заданные действия тремя (четырьмя в случае наличия поддержки расширения AVX-512) способами:

1. На языке высокого уровня стандартными средствами работы с массивами, код откомпилировать в конфигурациях Debug и Release.
2. С использованием указанного расширения системы команд процессора с использованием скалярных команд (если подобные команды поддерживаются).

3. С использованием указанного расширения системы команд процессора с использованием векторных и горизонтальных команд (горизонтальные команды использовать в случае их наличия в указанном расширении и при необходимости в соответствии с индивидуальным вариантом).
4. С использованием расширения AVX-512 (в случае его поддержки процессором).

Убедиться в том, что результирующие массивы совпадают. Измерить время выполнения операции для массивов большого размера (например, $n = 1\,000\,000$), реализовать раскрутку циклов на 2, 4 и 8 итераций. Сравнить времена между собой, сделать выводы о целесообразности использования векторизации.