```
	double * const dA = &dAC[0];
	double * const dC = &dAC[1];
	double * const dB = &dArr[2];
	double * const dResult = &dArr[4];
	double * const dX1 = &dResult[1];
	double * const dX2 = &dResult[0];

	__m128d r__zero_zero, r__c_a, r__uORb_b, r__2cORbOR2a_2a,
		r__zero_bb, r__sqrtDiscriminant_zero, r_result;

	r__zero_zero = _mm_set_pd(0., 0.); // init

	REPEATOR(REPEAT_COUNT,
		TWO_VALUES_SELECTOR(*dA, 4., A);
		TWO_VALUES_SELECTOR(*dB, 3., B);
		TWO_VALUES_SELECTOR(*dC, 1., C);
		r__c_a = _mm_load_pd(dAC);
		// r__uORb_b = _mm_load_pd1(dB);
		r__uORb_b = _mm_load1_pd(dB);
		// b b
		r__uORb_b = _mm_unpacklo_pd(r__uORb_b, r__uORb_b);
		// (etap 1)
		r__2cORbOR2a_2a = _mm_add_pd(r__c_a, r__c_a);
		// b 2c
		r_result = _mm_unpackhi_pd(r__2cORbOR2a_2a, r__uORb_b);
		// b 2a
		r__2cORbOR2a_2a = _mm_unpacklo_pd(r__2cORbOR2a_2a, r__uORb_b);
		// bb 4ac (etap 2)
		r_result = _mm_mul_pd(r_result, r__2cORbOR2a_2a);
		r__zero_bb = _mm_unpackhi_pd(r_result, r__zero_zero);
		// zero Discriminant (etap 3)
		r_result = _mm_sub_sd(r__zero_bb, r_result);
		// zero sqrtDiscriminant (etap 4)
		r_result = _mm_sqrt_sd(r_result, r_result);
		r__sqrtDiscriminant_zero = _mm_shuffle_pd(r_result, r_result, 1);
		// sqrtDiscriminant -sqrtDiscriminant (etap 5)
		r_result = _mm_sub_sd(r__sqrtDiscriminant_zero, r_result);
		// (etap 6)
		r_result = _mm_sub_pd(r_result, r__uORb_b);
		// 2a 2a
		r__2cORbOR2a_2a = _mm_unpacklo_pd(r__2cORbOR2a_2a, r__2cORbOR2a_2a);
		// (etap 7)
		r_result = _mm_div_pd(r_result, r__2cORbOR2a_2a);
		_mm_store_pd(dResult, r_result);
	)
}

void printResult(char * const title, double * const dArr, unsigned int runTime){
	double * const dAC = dArr;
	double * const dA = &dAC[0];
	double * const dC = &dAC[1];
	double * const dB = &dArr[2];
	double * const dResult = &dArr[4];
	double * const dX1 = &dResult[1];
	double * const dX2 = &dResult[0];

	printf("%s:\r\n", title);
	printf("%fx^2 + %fx + %f = 0;\r\n", *dA, *dB, *dC);
	printf("x1 = %1.0f; x2 = %1.0f;\r\n", *dX1, *dX2);
	printf("run time: %dns\r\n\r\n", runTime);
}

int main() {
	double * const dArr = (double *)_mm_malloc(6 * sizeof(double), 16);
```

```
        double * const dAC = dArr;
        double * const dA = &dAC[0];
        double * const dC = &dAC[1];
        double * const dB = &dArr[2];
        double * const dResult = &dArr[4];
        double * const dX1 = &dResult[1];
        double * const dX2 = &dResult[0];

        double startTime, endTime;

        // native (only x86, if auto vectorization by compiler is off)
        startTime = getCurrentTime();
        run_native(dArr);
        endTime = getCurrentTime();
        printResult("x86",
               dArr,
               (unsigned int)((endTime - startTime) * (1000000000 / REPEAT_COUNT)));

        // SSE2
        startTime = getCurrentTime();
        run_SSE2(dArr);
        endTime = getCurrentTime();
        printResult("SSE2",
               dArr,
               (unsigned int)((endTime - startTime) * (1000000000 / REPEAT_COUNT)));

        _mm_free(dArr);

        printf("Press any key to continue . . .");
        getchar();
        return 0;
}

#pragma GCC pop_options
```

4. Спрощене завдання.

Скласти програму (C/C++), яка дозволяє провести порівняння двох реалізацій розв'язання квадратного рівняння(звичайний послідовний код та код на основі інструкцій SSE2, що відображає потоковий граф алгоритму) за характеристикою часової складності для таких вхідних даних:

n	**Вхідні дані**		
	c	**b**	**a**
1	111	112	107
2	145	146	141
3	165	166	161
4	185	186	181
5	205	206	201
6	225	226	221
7	245	246	241
8	265	266	261
9	285	286	281
10	305	306	301
11	325	326	321
12	345	346	341
13	365	366	361
14	385	386	381
15	405	406	401
16	425	426	421

17	445	446	441
18	465	466	461
19	485	486	481
20	505	506	501
21	525	526	521
22	545	546	541
23	565	566	561
24	585	586	581
25	605	606	601
26	625	626	621
27	645	646	641
28	665	666	661
29	685	686	681
30	705	706,	701
n – порядковий номер у журналі			

**** Для отримання 50% балів за лабораторну роботу можна використати наявний програмний код з лістингу 3.1. Для отримання 100% балів за лабораторну роботу потрібно написати власний код.***

5. Завдання базового рівня.

Скласти програму (C/C++), яка дозволяє провести порівняння двох реалізацій обчислення виразу(звичайний послідовний код та код на основі інструкцій SSE2, що відображає потоковий граф алгоритму) за характеристикою часової складності для вхідних даних, що вводяться під час роботи програми.

Варіант	**Вираз**
1	y = (a + b) + (c + d) + (e + f) + (g + h)
2	y = (a + b) - (c + d) + (e + f) - (g + h)
3	y = (a + b) * (c + d) + (e + f) * (g + h)
4	y = (a + b) / (c + d) + (e + f) / (g + h)
5	y = (a + b) + (c + d) + (e + f) + (g + h)
6	y = (a + b) - (c + d) + (e + f) - (g + h)
7	y = (a + b) * (c + d) + (e + f) * (g + h)
8	y = (a + b) / (c + d) + (e + f) / (g + h)
9	y = (a + b) + (c + d) + (e - f) + (g - h)
10	y = (a + b) - (c + d) + (e - f) - (g - h)
11	y = (a + b) * (c + d) + (e - f) * (g - h)
12	y = (a + b) / (c + d) + (e - f) / (g - h)
13	y = (a + b) + (c + d) + (e - f) + (g - h)
14	y = (a + b) - (c + d) + (e - f) - (g - h)
15	y = (a + b) * (c + d) + (e - f) * (g - h)
16	y = (a + b) / (c + d) + (e - f) / (g - h)
17	y = (a - b) + (c - d) + (e + f) + (g + h)
18	y = (a - b) - (c - d) + (e + f) - (g + h)
19	y = (a - b) * (c - d) + (e + f) * (g + h)
20	y = (a - b) / (c - d) + (e + f) / (g + h)
21	y = (a - b) + (c - d) + (e + f) + (g + h)
22	y = (a - b) - (c - d) + (e + f) - (g + h)
23	y = (a - b) * (c - d) + (e + f) * (g + h)
24	y = (a - b) / (c - d) + (e + f) / (g + h)
25	y = (a - b) + (c - d) + (e - f) + (g - h)
26	y = (a - b) - (c - d) + (e - f) - (g - h)
27	y = (a - b) * (c - d) + (e - f) * (g - h)
28	y = (a - b) / (c - d) + (e - f) / (g - h)
29	y = (a - b) + (c - d) + (e - f) + (g - h)
30	y = (a - b) - (c - d) + (e - f) - (g + h)
n – порядковий номер у журналі	

6. Зміст звіту

- Титульний лист;
- Завдання;
- Алгоритм рішення завдання;
- Код програми;
- Екранна форма з результатами роботи програми;
- Висновки.

```
	double * const dA = &dAC[0];
	double * const dC = &dAC[1];
	double * const dB = &dArr[2];
	double * const dResult = &dArr[4];
	double * const dX1 = &dResult[1];
	double * const dX2 = &dResult[0];

	__m128d r__zero_zero, r__c_a, r__uORb_b, r__2cORbOR2a_2a,
		r__zero_bb, r__sqrtDiscriminant_zero, r_result;

	r__zero_zero = _mm_set_pd(0., 0.); // init

	REPEATOR(REPEAT_COUNT,
		TWO_VALUES_SELECTOR(*dA, 4., A);
		TWO_VALUES_SELECTOR(*dB, 3., B);
		TWO_VALUES_SELECTOR(*dC, 1., C);
		r__c_a = _mm_load_pd(dAC);
		// r__uORb_b = _mm_load_pd1(dB);
		r__uORb_b = _mm_load1_pd(dB);
		// b b
		r__uORb_b = _mm_unpacklo_pd(r__uORb_b, r__uORb_b);
		// (etap 1)
		r__2cORbOR2a_2a = _mm_add_pd(r__c_a, r__c_a);
		// b 2c
		r_result = _mm_unpackhi_pd(r__2cORbOR2a_2a, r__uORb_b);
		// b 2a
		r__2cORbOR2a_2a = _mm_unpacklo_pd(r__2cORbOR2a_2a, r__uORb_b);
		// bb 4ac (etap 2)
		r_result = _mm_mul_pd(r_result, r__2cORbOR2a_2a);
		r__zero_bb = _mm_unpackhi_pd(r_result, r__zero_zero);
		// zero Discriminant (etap 3)
		r_result = _mm_sub_sd(r__zero_bb, r_result);
		// zero sqrtDiscriminant (etap 4)
		r_result = _mm_sqrt_sd(r_result, r_result);
		r__sqrtDiscriminant_zero = _mm_shuffle_pd(r_result, r_result, 1);
		// sqrtDiscriminant -sqrtDiscriminant (etap 5)
		r_result = _mm_sub_sd(r__sqrtDiscriminant_zero, r_result);
		// (etap 6)
		r_result = _mm_sub_pd(r_result, r__uORb_b);
		// 2a 2a
		r__2cORbOR2a_2a = _mm_unpacklo_pd(r__2cORbOR2a_2a, r__2cORbOR2a_2a);
		// (etap 7)
		r_result = _mm_div_pd(r_result, r__2cORbOR2a_2a);
		_mm_store_pd(dResult, r_result);
	)
}

void printResult(char * const title, double * const dArr, unsigned int runTime){
	double * const dAC = dArr;
	double * const dA = &dAC[0];
	double * const dC = &dAC[1];
	double * const dB = &dArr[2];
	double * const dResult = &dArr[4];
	double * const dX1 = &dResult[1];
	double * const dX2 = &dResult[0];

	printf("%s:\r\n", title);
	printf("%fx^2 + %fx + %f = 0;\r\n", *dA, *dB, *dC);
	printf("x1 = %1.0f; x2 = %1.0f;\r\n", *dX1, *dX2);
	printf("run time: %dns\r\n\r\n", runTime);
}

int main() {
	double * const dArr = (double *)_mm_malloc(6 * sizeof(double), 16);
```

```
        double * const dAC = dArr;
        double * const dA = &dAC[0];
        double * const dC = &dAC[1];
        double * const dB = &dArr[2];
        double * const dResult = &dArr[4];
        double * const dX1 = &dResult[1];
        double * const dX2 = &dResult[0];

        double startTime, endTime;

        // native (only x86, if auto vectorization by compiler is off)
        startTime = getCurrentTime();
        run_native(dArr);
        endTime = getCurrentTime();
        printResult("x86",
               dArr,
               (unsigned int)((endTime - startTime) * (1000000000 / REPEAT_COUNT)));

        // SSE2
        startTime = getCurrentTime();
        run_SSE2(dArr);
        endTime = getCurrentTime();
        printResult("SSE2",
               dArr,
               (unsigned int)((endTime - startTime) * (1000000000 / REPEAT_COUNT)));

        _mm_free(dArr);

        printf("Press any key to continue . . .");
        getchar();
        return 0;
}

#pragma GCC pop_options
```

4. Спрощене завдання.

Скласти програму (C/C++), яка дозволяє провести порівняння двох реалізацій розв'язання квадратного рівняння(звичайний послідовний код та код на основі інструкцій SSE2, що відображає потоковий граф алгоритму) за характеристикою часової складності для таких вхідних даних:

n	**Вхідні дані**		
	c	**b**	**a**
1	111	112	107
2	145	146	141
3	165	166	161
4	185	186	181
5	205	206	201
6	225	226	221
7	245	246	241
8	265	266	261
9	285	286	281
10	305	306	301
11	325	326	321
12	345	346	341
13	365	366	361
14	385	386	381
15	405	406	401
16	425	426	421

17	445	446	441
18	465	466	461
19	485	486	481
20	505	506	501
21	525	526	521
22	545	546	541
23	565	566	561
24	585	586	581
25	605	606	601
26	625	626	621
27	645	646	641
28	665	666	661
29	685	686	681
30	705	706,	701
n – порядковий номер у журналі			

**** Для отримання 50% балів за лабораторну роботу можна використати наявний програмний код з лістингу 3.1. Для отримання 100% балів за лабораторну роботу потрібно написати власний код.***

5. Завдання базового рівня.

Скласти програму (C/C++), яка дозволяє провести порівняння двох реалізацій обчислення виразу(звичайний послідовний код та код на основі інструкцій SSE2, що відображає потоковий граф алгоритму) за характеристикою часової складності для вхідних даних, що вводяться під час роботи програми.

Варіант	**Вираз**
1	y = (a + b) + (c + d) + (e + f) + (g + h)
2	y = (a + b) - (c + d) + (e + f) - (g + h)
3	y = (a + b) * (c + d) + (e + f) * (g + h)
4	y = (a + b) / (c + d) + (e + f) / (g + h)
5	y = (a + b) + (c + d) + (e + f) + (g + h)
6	y = (a + b) - (c + d) + (e + f) - (g + h)
7	y = (a + b) * (c + d) + (e + f) * (g + h)
8	y = (a + b) / (c + d) + (e + f) / (g + h)
9	y = (a + b) + (c + d) + (e - f) + (g - h)
10	y = (a + b) - (c + d) + (e - f) - (g - h)
11	y = (a + b) * (c + d) + (e - f) * (g - h)
12	y = (a + b) / (c + d) + (e - f) / (g - h)
13	y = (a + b) + (c + d) + (e - f) + (g - h)
14	y = (a + b) - (c + d) + (e - f) - (g - h)
15	y = (a + b) * (c + d) + (e - f) * (g - h)
16	y = (a + b) / (c + d) + (e - f) / (g - h)
17	y = (a - b) + (c - d) + (e + f) + (g + h)
18	y = (a - b) - (c - d) + (e + f) - (g + h)
19	y = (a - b) * (c - d) + (e + f) * (g + h)
20	y = (a - b) / (c - d) + (e + f) / (g + h)
21	y = (a - b) + (c - d) + (e + f) + (g + h)
22	y = (a - b) - (c - d) + (e + f) - (g + h)
23	y = (a - b) * (c - d) + (e + f) * (g + h)
24	y = (a - b) / (c - d) + (e + f) / (g + h)
25	y = (a - b) + (c - d) + (e - f) + (g - h)
26	y = (a - b) - (c - d) + (e - f) - (g - h)
27	y = (a - b) * (c - d) + (e - f) * (g - h)
28	y = (a - b) / (c - d) + (e - f) / (g - h)
29	y = (a - b) + (c - d) + (e - f) + (g - h)
30	y = (a - b) - (c - d) + (e - f) - (g + h)
n – порядковий номер у журналі	

6. Зміст звіту

- Титульний лист;
- Завдання;
- Алгоритм рішення завдання;
- Код програми;
- Екранна форма з результатами роботи програми;
- Висновки.

```
	double * const dA = &dAC[0];
	double * const dC = &dAC[1];
	double * const dB = &dArr[2];
	double * const dResult = &dArr[4];
	double * const dX1 = &dResult[1];
	double * const dX2 = &dResult[0];

	__m128d r__zero_zero, r__c_a, r__uORb_b, r__2cORbOR2a_2a,
		r__zero_bb, r__sqrtDiscriminant_zero, r_result;

	r__zero_zero = _mm_set_pd(0., 0.); // init

	REPEATOR(REPEAT_COUNT,
		TWO_VALUES_SELECTOR(*dA, 4., A);
		TWO_VALUES_SELECTOR(*dB, 3., B);
		TWO_VALUES_SELECTOR(*dC, 1., C);
		r__c_a = _mm_load_pd(dAC);
		// r__uORb_b = _mm_load_pd1(dB);
		r__uORb_b = _mm_load1_pd(dB);
		// b b
		r__uORb_b = _mm_unpacklo_pd(r__uORb_b, r__uORb_b);
		// (etap 1)
		r__2cORbOR2a_2a = _mm_add_pd(r__c_a, r__c_a);
		// b 2c
		r_result = _mm_unpackhi_pd(r__2cORbOR2a_2a, r__uORb_b);
		// b 2a
		r__2cORbOR2a_2a = _mm_unpacklo_pd(r__2cORbOR2a_2a, r__uORb_b);
		// bb 4ac (etap 2)
		r_result = _mm_mul_pd(r_result, r__2cORbOR2a_2a);
		r__zero_bb = _mm_unpackhi_pd(r_result, r__zero_zero);
		// zero Discriminant (etap 3)
		r_result = _mm_sub_sd(r__zero_bb, r_result);
		// zero sqrtDiscriminant (etap 4)
		r_result = _mm_sqrt_sd(r_result, r_result);
		r__sqrtDiscriminant_zero = _mm_shuffle_pd(r_result, r_result, 1);
		// sqrtDiscriminant -sqrtDiscriminant (etap 5)
		r_result = _mm_sub_sd(r__sqrtDiscriminant_zero, r_result);
		// (etap 6)
		r_result = _mm_sub_pd(r_result, r__uORb_b);
		// 2a 2a
		r__2cORbOR2a_2a = _mm_unpacklo_pd(r__2cORbOR2a_2a, r__2cORbOR2a_2a);
		// (etap 7)
		r_result = _mm_div_pd(r_result, r__2cORbOR2a_2a);
		_mm_store_pd(dResult, r_result);
	)
}

void printResult(char * const title, double * const dArr, unsigned int runTime){
	double * const dAC = dArr;
	double * const dA = &dAC[0];
	double * const dC = &dAC[1];
	double * const dB = &dArr[2];
	double * const dResult = &dArr[4];
	double * const dX1 = &dResult[1];
	double * const dX2 = &dResult[0];

	printf("%s:\r\n", title);
	printf("%fx^2 + %fx + %f = 0;\r\n", *dA, *dB, *dC);
	printf("x1 = %1.0f; x2 = %1.0f;\r\n", *dX1, *dX2);
	printf("run time: %dns\r\n\r\n", runTime);
}

int main() {
	double * const dArr = (double *)_mm_malloc(6 * sizeof(double), 16);
```

```
        double * const dAC = dArr;
        double * const dA = &dAC[0];
        double * const dC = &dAC[1];
        double * const dB = &dArr[2];
        double * const dResult = &dArr[4];
        double * const dX1 = &dResult[1];
        double * const dX2 = &dResult[0];

        double startTime, endTime;

        // native (only x86, if auto vectorization by compiler is off)
        startTime = getCurrentTime();
        run_native(dArr);
        endTime = getCurrentTime();
        printResult("x86",
               dArr,
               (unsigned int)((endTime - startTime) * (1000000000 / REPEAT_COUNT)));

        // SSE2
        startTime = getCurrentTime();
        run_SSE2(dArr);
        endTime = getCurrentTime();
        printResult("SSE2",
               dArr,
               (unsigned int)((endTime - startTime) * (1000000000 / REPEAT_COUNT)));

        _mm_free(dArr);

        printf("Press any key to continue . . .");
        getchar();
        return 0;
}

#pragma GCC pop_options
```

4. Спрощене завдання.

Скласти програму (C/C++), яка дозволяє провести порівняння двох реалізацій розв'язання квадратного рівняння(звичайний послідовний код та код на основі інструкцій SSE2, що відображає потоковий граф алгоритму) за характеристикою часової складності для таких вхідних даних:

n	**Вхідні дані**		
	c	**b**	**a**
1	111	112	107
2	145	146	141
3	165	166	161
4	185	186	181
5	205	206	201
6	225	226	221
7	245	246	241
8	265	266	261
9	285	286	281
10	305	306	301
11	325	326	321
12	345	346	341
13	365	366	361
14	385	386	381
15	405	406	401
16	425	426	421

17	445	446	441
18	465	466	461
19	485	486	481
20	505	506	501
21	525	526	521
22	545	546	541
23	565	566	561
24	585	586	581
25	605	606	601
26	625	626	621
27	645	646	641
28	665	666	661
29	685	686	681
30	705	706,	701
n – порядковий номер у журналі			

**** Для отримання 50% балів за лабораторну роботу можна використати наявний програмний код з лістингу 3.1. Для отримання 100% балів за лабораторну роботу потрібно написати власний код.***

5. Завдання базового рівня.

Скласти програму (C/C++), яка дозволяє провести порівняння двох реалізацій обчислення виразу(звичайний послідовний код та код на основі інструкцій SSE2, що відображає потоковий граф алгоритму) за характеристикою часової складності для вхідних даних, що вводяться під час роботи програми.

Варіант	**Вираз**
1	y = (a + b) + (c + d) + (e + f) + (g + h)
2	y = (a + b) - (c + d) + (e + f) - (g + h)
3	y = (a + b) * (c + d) + (e + f) * (g + h)
4	y = (a + b) / (c + d) + (e + f) / (g + h)
5	y = (a + b) + (c + d) + (e + f) + (g + h)
6	y = (a + b) - (c + d) + (e + f) - (g + h)
7	y = (a + b) * (c + d) + (e + f) * (g + h)
8	y = (a + b) / (c + d) + (e + f) / (g + h)
9	y = (a + b) + (c + d) + (e - f) + (g - h)
10	y = (a + b) - (c + d) + (e - f) - (g - h)
11	y = (a + b) * (c + d) + (e - f) * (g - h)
12	y = (a + b) / (c + d) + (e - f) / (g - h)
13	y = (a + b) + (c + d) + (e - f) + (g - h)
14	y = (a + b) - (c + d) + (e - f) - (g - h)
15	y = (a + b) * (c + d) + (e - f) * (g - h)
16	y = (a + b) / (c + d) + (e - f) / (g - h)
17	y = (a - b) + (c - d) + (e + f) + (g + h)
18	y = (a - b) - (c - d) + (e + f) - (g + h)
19	y = (a - b) * (c - d) + (e + f) * (g + h)
20	y = (a - b) / (c - d) + (e + f) / (g + h)
21	y = (a - b) + (c - d) + (e + f) + (g + h)
22	y = (a - b) - (c - d) + (e + f) - (g + h)
23	y = (a - b) * (c - d) + (e + f) * (g + h)
24	y = (a - b) / (c - d) + (e + f) / (g + h)
25	y = (a - b) + (c - d) + (e - f) + (g - h)
26	y = (a - b) - (c - d) + (e - f) - (g - h)
27	y = (a - b) * (c - d) + (e - f) * (g - h)
28	y = (a - b) / (c - d) + (e - f) / (g - h)
29	y = (a - b) + (c - d) + (e - f) + (g - h)
30	y = (a - b) - (c - d) + (e - f) - (g + h)
n – порядковий номер у журналі	

6. Зміст звіту

- Титульний лист;
- Завдання;
- Алгоритм рішення завдання;
- Код програми;
- Екранна форма з результатами роботи програми;
- Висновки.

```
	double * const dA = &dAC[0];
	double * const dC = &dAC[1];
	double * const dB = &dArr[2];
	double * const dResult = &dArr[4];
	double * const dX1 = &dResult[1];
	double * const dX2 = &dResult[0];

	__m128d r__zero_zero, r__c_a, r__uORb_b, r__2cORbOR2a_2a,
		r__zero_bb, r__sqrtDiscriminant_zero, r_result;

	r__zero_zero = _mm_set_pd(0., 0.); // init

	REPEATOR(REPEAT_COUNT,
		TWO_VALUES_SELECTOR(*dA, 4., A);
		TWO_VALUES_SELECTOR(*dB, 3., B);
		TWO_VALUES_SELECTOR(*dC, 1., C);
		r__c_a = _mm_load_pd(dAC);
		// r__uORb_b = _mm_load_pd1(dB);
		r__uORb_b = _mm_load1_pd(dB);
		// b b
		r__uORb_b = _mm_unpacklo_pd(r__uORb_b, r__uORb_b);
		// (etap 1)
		r__2cORbOR2a_2a = _mm_add_pd(r__c_a, r__c_a);
		// b 2c
		r_result = _mm_unpackhi_pd(r__2cORbOR2a_2a, r__uORb_b);
		// b 2a
		r__2cORbOR2a_2a = _mm_unpacklo_pd(r__2cORbOR2a_2a, r__uORb_b);
		// bb 4ac (etap 2)
		r_result = _mm_mul_pd(r_result, r__2cORbOR2a_2a);
		r__zero_bb = _mm_unpackhi_pd(r_result, r__zero_zero);
		// zero Discriminant (etap 3)
		r_result = _mm_sub_sd(r__zero_bb, r_result);
		// zero sqrtDiscriminant (etap 4)
		r_result = _mm_sqrt_sd(r_result, r_result);
		r__sqrtDiscriminant_zero = _mm_shuffle_pd(r_result, r_result, 1);
		// sqrtDiscriminant -sqrtDiscriminant (etap 5)
		r_result = _mm_sub_sd(r__sqrtDiscriminant_zero, r_result);
		// (etap 6)
		r_result = _mm_sub_pd(r_result, r__uORb_b);
		// 2a 2a
		r__2cORbOR2a_2a = _mm_unpacklo_pd(r__2cORbOR2a_2a, r__2cORbOR2a_2a);
		// (etap 7)
		r_result = _mm_div_pd(r_result, r__2cORbOR2a_2a);
		_mm_store_pd(dResult, r_result);
	)
}

void printResult(char * const title, double * const dArr, unsigned int runTime){
	double * const dAC = dArr;
	double * const dA = &dAC[0];
	double * const dC = &dAC[1];
	double * const dB = &dArr[2];
	double * const dResult = &dArr[4];
	double * const dX1 = &dResult[1];
	double * const dX2 = &dResult[0];

	printf("%s:\r\n", title);
	printf("%fx^2 + %fx + %f = 0;\r\n", *dA, *dB, *dC);
	printf("x1 = %1.0f; x2 = %1.0f;\r\n", *dX1, *dX2);
	printf("run time: %dns\r\n\r\n", runTime);
}

int main() {
	double * const dArr = (double *)_mm_malloc(6 * sizeof(double), 16);
```

```
        double * const dAC = dArr;
        double * const dA = &dAC[0];
        double * const dC = &dAC[1];
        double * const dB = &dArr[2];
        double * const dResult = &dArr[4];
        double * const dX1 = &dResult[1];
        double * const dX2 = &dResult[0];

        double startTime, endTime;

        // native (only x86, if auto vectorization by compiler is off)
        startTime = getCurrentTime();
        run_native(dArr);
        endTime = getCurrentTime();
        printResult("x86",
               dArr,
               (unsigned int)((endTime - startTime) * (1000000000 / REPEAT_COUNT)));

        // SSE2
        startTime = getCurrentTime();
        run_SSE2(dArr);
        endTime = getCurrentTime();
        printResult("SSE2",
               dArr,
               (unsigned int)((endTime - startTime) * (1000000000 / REPEAT_COUNT)));

        _mm_free(dArr);

        printf("Press any key to continue . . .");
        getchar();
        return 0;
}

#pragma GCC pop_options
```

4. Спрощене завдання.

Скласти програму (C/C++), яка дозволяє провести порівняння двох реалізацій розв'язання квадратного рівняння(звичайний послідовний код та код на основі інструкцій SSE2, що відображає потоковий граф алгоритму) за характеристикою часової складності для таких вхідних даних:

n	**Вхідні дані**		
	c	**b**	**a**
1	111	112	107
2	145	146	141
3	165	166	161
4	185	186	181
5	205	206	201
6	225	226	221
7	245	246	241
8	265	266	261
9	285	286	281
10	305	306	301
11	325	326	321
12	345	346	341
13	365	366	361
14	385	386	381
15	405	406	401
16	425	426	421

17	445	446	441
18	465	466	461
19	485	486	481
20	505	506	501
21	525	526	521
22	545	546	541
23	565	566	561
24	585	586	581
25	605	606	601
26	625	626	621
27	645	646	641
28	665	666	661
29	685	686	681
30	705	706,	701
n – порядковий номер у журналі			

**** Для отримання 50% балів за лабораторну роботу можна використати наявний програмний код з лістингу 3.1. Для отримання 100% балів за лабораторну роботу потрібно написати власний код.***

5. Завдання базового рівня.

Скласти програму (C/C++), яка дозволяє провести порівняння двох реалізацій обчислення виразу(звичайний послідовний код та код на основі інструкцій SSE2, що відображає потоковий граф алгоритму) за характеристикою часової складності для вхідних даних, що вводяться під час роботи програми.

Варіант	**Вираз**
1	y = (a + b) + (c + d) + (e + f) + (g + h)
2	y = (a + b) - (c + d) + (e + f) - (g + h)
3	y = (a + b) * (c + d) + (e + f) * (g + h)
4	y = (a + b) / (c + d) + (e + f) / (g + h)
5	y = (a + b) + (c + d) + (e + f) + (g + h)
6	y = (a + b) - (c + d) + (e + f) - (g + h)
7	y = (a + b) * (c + d) + (e + f) * (g + h)
8	y = (a + b) / (c + d) + (e + f) / (g + h)
9	y = (a + b) + (c + d) + (e - f) + (g - h)
10	y = (a + b) - (c + d) + (e - f) - (g - h)
11	y = (a + b) * (c + d) + (e - f) * (g - h)
12	y = (a + b) / (c + d) + (e - f) / (g - h)
13	y = (a + b) + (c + d) + (e - f) + (g - h)
14	y = (a + b) - (c + d) + (e - f) - (g - h)
15	y = (a + b) * (c + d) + (e - f) * (g - h)
16	y = (a + b) / (c + d) + (e - f) / (g - h)
17	y = (a - b) + (c - d) + (e + f) + (g + h)
18	y = (a - b) - (c - d) + (e + f) - (g + h)
19	y = (a - b) * (c - d) + (e + f) * (g + h)
20	y = (a - b) / (c - d) + (e + f) / (g + h)
21	y = (a - b) + (c - d) + (e + f) + (g + h)
22	y = (a - b) - (c - d) + (e + f) - (g + h)
23	y = (a - b) * (c - d) + (e + f) * (g + h)
24	y = (a - b) / (c - d) + (e + f) / (g + h)
25	y = (a - b) + (c - d) + (e - f) + (g - h)
26	y = (a - b) - (c - d) + (e - f) - (g - h)
27	y = (a - b) * (c - d) + (e - f) * (g - h)
28	y = (a - b) / (c - d) + (e - f) / (g - h)
29	y = (a - b) + (c - d) + (e - f) + (g - h)
30	y = (a - b) - (c - d) + (e - f) - (g + h)
n – порядковий номер у журналі	

6. Зміст звіту

- Титульний лист;
- Завдання;
- Алгоритм рішення завдання;
- Код програми;
- Екранна форма з результатами роботи програми;
- Висновки.