**Тестирование вычислительной производительности  
платы БАГЕТ-ПЛК1-01**

Мощевикин А.П., ПетрГУ

Одноплатный микрокомпьютер БАГЕТ-ПЛК1-01 построен на микроконтроллере Комдив-МК К5500ВК018 с архитектурой КОМДИВ64 (совместима с MIPS).

Для оценки производительности процессорного ядра были проведены эксперименты с данными из встроенной оперативной памяти (512 Кб).

Для этого в памяти были организованы два статические volatile-массива arr\_d (целочисленные значения) и arr\_f (вещественные значения) по 10000 элементов. См. выдержки из исходного кода в приложении А.

Перед и после каждой тестовой операции проводилась засечка текущего времени с помощью макроса \_\_micros(). Значения меток времени запоминались во временные переменные, и распечатывались на экран только после проведения все тестовых операций.

Проведенные тестовые операции и результаты измерений времени приведены в таблице.

Второй столбик содержит общее время, затраченное на операцию в микросекундах.

В третьем столбике указано примерное количество тактов процессора, затрачиваемое на одну операцию. Тактовая частота 300 МГц, длительности такта – 3.33 нс.

|  |  |  |
| --- | --- | --- |
| **Тестовая операция** | **Время,**  **[мкс]** | **Количество тактов CPU на 1 значение / время операции** |
| 10000 раз выполненный инкремент volatile переменной | 134 | 4 тактов / 13 нс |
| Заполнение генератором псевдослучайных чисел массива из 10000 значений типа int32 | 16960 | 509 тактов / 1.7 мкс |
| Суммирование 10000 элементов массива значений типа int32 | 260 | 8 тактов / 26 нс |
| Формирование 10000 элементов массива значений типа float из соответствующих элементов массива значений типа int32 (через присвоение значения с преобразованием типа) | 434 | 13 тактов / 43 нс |
| Суммирование 10000 элементов массива значений типа float | 326 | 10 тактов / 33 нс |
| Суммирование 10000 элементов массива значений типа float, каждое текущее значение предварительно делится на 10.0 | 1193 | 36 тактов / 119 нс |
| Суммирование 10000 элементов массива значений типа float, каждое текущее значение предварительно умножается на 10.0 | 660 | 20 тактов / 66 нс |

В приложении Б приведена выдержка из дизассемблированного файла прошивки (на всякий случай, как протокол экспериментов). Необходимо отметить, что детальное изучение этого кода не проводилось.

Приложение А – Исходный код на Си

volatile int32\_t arr\_d[10000];

volatile float arr\_f[10000];

t1 = \_\_micros();

for(i=0;i<10000;i++) {

arr\_d[i] = genrand32(1000) - 500;

}

t2 = \_\_micros();

for(i=0;i<10000;i++) {

sum1 += arr\_d[i];

}

t3 = \_\_micros();

for(i=0;i<10000;i++) {

sum2 ++;

}

t4 = \_\_micros();

for(i=0;i<10000;i++) {

arr\_f[i] = (float)arr\_d[i];

}

t5 = \_\_micros();

for(i=0;i<10000;i++) {

sum\_f1 += arr\_f[i];

}

t6 = \_\_micros();

for(i=0;i<10000;i++) {

sum\_f2 += arr\_f[i]/10.0;

}

t7 = \_\_micros();

for(i=0;i<10000;i++) {

sum\_f3 += arr\_f[i]\*10.0;

}

t8 = \_\_micros();

\_\_print\_console("\r\nRND 10000 values [UL], T =");

\_\_print\_console(itoa(t2-t1, s, 10));

\_\_print\_console(" Sum (10000 rnd) = ");

\_\_print\_console(itoa(sum1, s, 10));

\_\_print\_console("\r\nSumming 10000 array elements [UL], T = ");

\_\_print\_console(itoa(t3-t2, s, 10));

\_\_print\_console("\r\nIncrementing (++) 10000 times, T = ");

\_\_print\_console(itoa(t4-t3, s, 10));

\_\_print\_console("\r\nConversion array\_Int[10000] -> array\_float[10000], T = ");

\_\_print\_console(itoa(t5-t4, s, 10));

\_\_print\_console("\r\nSumming 10000 array elements [float], T = ");

\_\_print\_console(itoa(t6-t5, s, 10));

\_\_print\_console(" Sum(10000 float) = ");

\_\_print\_console(itoa((uint32\_t)(sum\_f1), s, 10));

\_\_print\_console("\r\nSumming 10000 float values, divided by 10.0, T = ");

\_\_print\_console(itoa(t7-t6, s, 10));

\_\_print\_console(" Sum = ");

\_\_print\_console(itoa((uint32\_t)(sum\_f2), s, 10));

\_\_print\_console("\r\nSumming 10000 float values, multiplied by 10.0, T = ");

\_\_print\_console(itoa(t8-t7, s, 10));

\_\_print\_console(" Sum = ");

\_\_print\_console(itoa((uint32\_t)(sum\_f3), s, 10));

\_\_print\_console("\r\n");

Приложение Б – Дизассемблированный код (выдержка)

80000d30: 0c000106 jal 80000418 <kmk\_micros>

80000d34: 00000000 nop

80000d38: ae023914 sw v0,14612(s0)

80000d3c: 00008825 move s1,zero

80000d40: 24122710 li s2,10000

80000d44: 0c0004aa jal 800012a8 <genrand32>

80000d48: 240403e8 li a0,1000

80000d4c: 00111880 sll v1,s1,0x2

80000d50: 2442fe0c addiu v0,v0,-500

80000d54: 02031821 addu v1,s0,v1

80000d58: 26310001 addiu s1,s1,1

80000d5c: ac629c40 sw v0,-25536(v1)

80000d60: 1632fff8 bne s1,s2,80000d44 <main+0xf0>

80000d64: 00000000 nop

80000d68: 0c000106 jal 80000418 <kmk\_micros>

80000d6c: 00000000 nop

80000d70: ae023910 sw v0,14608(s0)

80000d74: 27a30028 addiu v1,sp,40

80000d78: 3c100001 lui s0,0x1

80000d7c: 00001025 move v0,zero

80000d80: 00708021 addu s0,v1,s0

80000d84: 24042710 li a0,10000

80000d88: 00021880 sll v1,v0,0x2

80000d8c: 02031821 addu v1,s0,v1

80000d90: 8c659c40 lw a1,-25536(v1)

80000d94: 8e0338f4 lw v1,14580(s0)

80000d98: 24420001 addiu v0,v0,1

80000d9c: 00651821 addu v1,v1,a1

80000da0: ae0338f4 sw v1,14580(s0)

80000da4: 1444fff9 bne v0,a0,80000d8c <main+0x138>

80000da8: 00021880 sll v1,v0,0x2

80000dac: 0c000106 jal 80000418 <kmk\_micros>

80000db0: 00000000 nop

80000db4: ae02390c sw v0,14604(s0)

80000db8: 27a30028 addiu v1,sp,40

80000dbc: 3c100001 lui s0,0x1

80000dc0: 24022710 li v0,10000

80000dc4: 00708021 addu s0,v1,s0

80000dc8: 8e0338f0 lw v1,14576(s0)

80000dcc: 2442ffff addiu v0,v0,-1

80000dd0: 24630001 addiu v1,v1,1

80000dd4: ae0338f0 sw v1,14576(s0)

80000dd8: 1440fffb bnez v0,80000dc8 <main+0x174>

80000ddc: 00000000 nop

80000de0: 0c000106 jal 80000418 <kmk\_micros>

80000de4: 00000000 nop

80000de8: ae023908 sw v0,14600(s0)

80000dec: 27a30028 addiu v1,sp,40

80000df0: 3c100001 lui s0,0x1

80000df4: 00001025 move v0,zero

80000df8: 00708021 addu s0,v1,s0

80000dfc: 24052710 li a1,10000

80000e00: 00021880 sll v1,v0,0x2

80000e04: 02032021 addu a0,s0,v1

80000e08: 8c849c40 lw a0,-25536(a0)

80000e0c: 24420001 addiu v0,v0,1

80000e10: 44840000 mtc1 a0,$f0

80000e14: 27a40028 addiu a0,sp,40

80000e18: 46800020 cvt.s.w $f0,$f0

80000e1c: 00831821 addu v1,a0,v1

80000e20: e4600000 swc1 $f0,0(v1)

80000e24: 1445fff7 bne v0,a1,80000e04 <main+0x1b0>

80000e28: 00021880 sll v1,v0,0x2

80000e2c: 0c000106 jal 80000418 <kmk\_micros>

80000e30: 00000000 nop

80000e34: ae023904 sw v0,14596(s0)

80000e38: 27a30028 addiu v1,sp,40

80000e3c: 3c100001 lui s0,0x1

80000e40: 00001025 move v0,zero

80000e44: 00708021 addu s0,v1,s0

80000e48: 24042710 li a0,10000

80000e4c: 00021880 sll v1,v0,0x2

80000e50: 27a50028 addiu a1,sp,40

80000e54: 00a31821 addu v1,a1,v1

80000e58: c4620000 lwc1 $f2,0(v1)

80000e5c: c60038ec lwc1 $f0,14572(s0)

80000e60: 24420001 addiu v0,v0,1

80000e64: 46020000 add.s $f0,$f0,$f2

80000e68: e60038ec swc1 $f0,14572(s0)

80000e6c: 1444fff8 bne v0,a0,80000e50 <main+0x1fc>

80000e70: 00021880 sll v1,v0,0x2

80000e74: 0c000106 jal 80000418 <kmk\_micros>

80000e78: 3c128000 lui s2,0x8000

80000e7c: d6442778 ldc1 $f4,10104(s2)

80000e80: ae023900 sw v0,14592(s0)

80000e84: 27a30028 addiu v1,sp,40

80000e88: 3c100001 lui s0,0x1

80000e8c: 00001025 move v0,zero

80000e90: 00708021 addu s0,v1,s0

80000e94: 24042710 li a0,10000

80000e98: 00021880 sll v1,v0,0x2

80000e9c: 27a50028 addiu a1,sp,40

80000ea0: 00a31821 addu v1,a1,v1

80000ea4: c4600000 lwc1 $f0,0(v1)

80000ea8: c60238e8 lwc1 $f2,14568(s0)

80000eac: 24420001 addiu v0,v0,1

80000eb0: 46000021 cvt.d.s $f0,$f0

80000eb4: 46240003 div.d $f0,$f0,$f4

80000eb8: 460010a1 cvt.d.s $f2,$f2

80000ebc: 46220000 add.d $f0,$f0,$f2

80000ec0: 46200020 cvt.s.d $f0,$f0

80000ec4: e60038e8 swc1 $f0,14568(s0)

80000ec8: 1444fff4 bne v0,a0,80000e9c <main+0x248>

80000ecc: 00021880 sll v1,v0,0x2

80000ed0: 0c000106 jal 80000418 <kmk\_micros>

80000ed4: 3c110001 lui s1,0x1

80000ed8: d6442778 ldc1 $f4,10104(s2)

80000edc: 27a30028 addiu v1,sp,40

80000ee0: ae0238fc sw v0,14588(s0)

80000ee4: 24042710 li a0,10000

80000ee8: 00001025 move v0,zero

80000eec: 00718021 addu s0,v1,s1

80000ef0: 00021880 sll v1,v0,0x2

80000ef4: 27a50028 addiu a1,sp,40

80000ef8: 00a31821 addu v1,a1,v1

80000efc: c4600000 lwc1 $f0,0(v1)

80000f00: c60238e4 lwc1 $f2,14564(s0)

80000f04: 24420001 addiu v0,v0,1

80000f08: 46000021 cvt.d.s $f0,$f0

80000f0c: 46240002 mul.d $f0,$f0,$f4

80000f10: 460010a1 cvt.d.s $f2,$f2

80000f14: 46220000 add.d $f0,$f0,$f2

80000f18: 46200020 cvt.s.d $f0,$f0

80000f1c: e60038e4 swc1 $f0,14564(s0)

80000f20: 1444fff4 bne v0,a0,80000ef4 <main+0x2a0>

80000f24: 00021880 sll v1,v0,0x2

80000f28: 0c000106 jal 80000418 <kmk\_micros>