Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Ульяновский государственный технический университет»
Кафедра «Вычислительная техника»

Организация вычислительных машин и систем

Лабораторная работа №3
«Исследование функциональных возможностей системы команд микропроцессора»

Выполнил
Студент группы ИВТбд-21
Ведин В. А.
Проверил(а):
ст. преподаватель кафедры «ВТ»
Лылова А.В.

Ульяновск
2024

ОГЛАВЛЕНИЕ

ЗАДАНИЕ 3

ТЕКСТОВОЕ ОПИСАНИЕ АЛГОРИТМА 4

АЛГОРИТМ ПРОГРАММЫ 5

Код программы на «С»: 5

АНАЛИЗ КОДА НА ЯЗЫКЕ ASSEMBLER 6

СОБСТВЕННЫЕ ПРЕДЛОЖЕНИЕ ПО ОПТИМИЗАЦИИ 16

ИТОГИ 17

ЗАДАНИЕ

Требуется запрограммировать на языке C алгоритм, который выводит частное и остаток при делении двух целых положительных чисел в системе счисления с основанием p. Далее требуется откомпилировать полученную программу в любом доступном компиляторе, позволяющему получить исходный код на языке Assembler, с оптимизацией кода по скорости, а далее то же самое, только с оптимизацией кода по размеру. Затем требуется провести анализ исходного кода на языке Assembler, результатом которого будет ответ на вопрос, за счет чего компилятор добивается нужного метода оптимизации. На основе данного анализа предложить свои методы оптимизации.

ТЕКСТОВОЕ ОПИСАНИЕ АЛГОРИТМА

Алгоритм принимает на вход 3 числа: делимое, делитель и основание системы счисления. Далее высчитывается частное и остаток, после чего они переводятся в систему счисления с заданным основанием. Далее формируются полноценные строки с уже переведёнными в заданную систему счислениями числами, и программа их выводит.

АЛГОРИТМ ПРОГРАММЫ

Код программы на «С»:

```c
#include <stdio.h>

int main() {
    int num1, num2, base;
    scanf("%d %d %d", &num1, &num2, &base);
    int quotient = num1 / num2;
    int remainder = num1 % num2;
    char quotient_str[50], remainder_str[50];
    char *quotient_ptr = quotient_str, *remainder_ptr = remainder_str;
    do {
        int q_remainder = quotient % base;
        if (q_remainder < 10)
            *quotient_ptr++ = q_remainder + '0';
        else
            *quotient_ptr++ = q_remainder - 10 + 'A';
        quotient /= base;
    }while (quotient != 0);
    *quotient_ptr = '\0';
    do {
        int r_remainder = remainder % base;
        if (r_remainder < 10)
            *remainder_ptr++ = r_remainder + '0';
        else
            *remainder_ptr++ = r_remainder - 10 + 'A';
        remainder /= base;
    }while (remainder != 0);
    *remainder_ptr = '\0';
    for (int i = 0, j = quotient_ptr - quotient_str - 1; i < j; i++, j--) {
        char temp = quotient_str[i];
        quotient_str[i] = quotient_str[j];
        quotient_str[j] = temp;
    }
    for (int i = 0, j = remainder_ptr - remainder_str - 1; i < j; i++, j--) {
        char temp = remainder_str[i];
        remainder_str[i] = remainder_str[j];
        remainder_str[j] = temp;
    }
    printf("%s\n%s\n", quotient_str, remainder_str);
}
```

АНАЛИЗ КОДА НА ЯЗЫКЕ ASSEMBLER

В ходе выполнения данной лабораторной работы было получено два исходных кода на языке Assembler, первый путем оптимизации кода по скорости (Флаг –O3) и второй – по размеру (Флаг –Os).

Дизассемблирование происходило таким образом:

`gcc -m32 -O3 -masm=intel main.c -g -o o3.o` # Компиляция с флагом оптимизации по производительности

`gcc -m32 -Os -masm=intel main.c -g -o os.o` # Компиляция с флагом оптимизации по размеру

`objdump -d -M intel -S o3.o > 03.asm` # декомпиляция и вывод кода ассемблера в файл o3.asm

`objdump -d -M intel -S os.o > 0s.asm` # декомпиляция и вывод кода ассемблера в файл os.asm

Исходный код программы (оптимизация по размеру)

```
os.o:     формат файла elf32-i386


Дизассемблирование раздела .init:

00001000 <_init>:
1000:   53                      push   ebx
1001:   83 ec 08                sub    esp,0x8
1004:   e8 d7 01 00 00          call   11e0
<__x86.get_pc_thunk.bx>
1009:   81 c3 eb 2f 00 00       add
ebx,0x2feb
100f:   8b 83 f4 ff ff ff       mov
eax,DWORD PTR [ebx-0xc]
1015:   85 c0                   test   eax,eax
1017:   74 02                   je     101b
<_init+0x1b>
1019:   ff d0                   call   eax
101b:   83 c4 08                add    esp,0x8
101e:   5b                      pop    ebx
101f:   c3                      ret

Дизассемблирование раздела .plt:

00001020 <__libc_start_main@plt-0x10>:
1020:   ff b3 04 00 00 00       push   DWORD
PTR [ebx+0x4]
1026:   ff a3 08 00 00 00       jmp    DWORD
PTR [ebx+0x8]
102c:   00 00                   add    BYTE PTR
[eax],al
...

00001030 <__libc_start_main@plt>:
```

Исходный код программы (оптимизация по скорости)

```
o3.o:     формат файла elf32-i386


Дизассемблирование раздела .init:

00001000 <_init>:
    1000:       53
        push   ebx
    1001:       83 ec 08                sub
esp,0x8
    1004:       e8 27 02 00 00
        call   1230
<__x86.get_pc_thunk.bx>
    1009:       81 c3 eb 2f 00 00       add
ebx,0x2feb
    100f:       8b 83 f4 ff ff ff       mov
eax,DWORD PTR [ebx-0xc]
    1015:       85 c0
        test   eax,eax
    1017:       74 02                   je
101b <_init+0x1b>
    1019:       ff d0
        call   eax
    101b:       83 c4 08                add
esp,0x8
    101e:       5b                      pop
ebx
    101f:       c3                      ret

Дизассемблирование раздела .plt:

00001020 <__libc_start_main@plt-0x10>:
    1020:       ff b3 04 00 00 00
        push   DWORD PTR [ebx+0x4]
```

Комментарии

Оптимизация по скорости была произведена за счёт более эффективного распределения данных по регистрам процессора и стеку, из-за чего была снижена активность запросов к ОЗУ.

Команда LEA означает «Эффективный адрес загрузки» с ее помощью можно загрузить указатель на объект, который необходимо адресовать, в то время как команда MOV означает «Значение загрузки» и уже с ее помощью можно получить фактическое значение по этому адресу.

То есть, при использовании первой команды (LEA) процессор делает всё то же, что делает при использовании второй (MOV), но пропускает последний шаг – извлечение значений по адресу. Вместо этого он

```
1030:   ff a3 0c 00 00 00      jmp    DWORD
PTR [ebx+0xc]
1036:   68 00 00 00 00         push   0x0
103b:   e9 e0 ff ff ff         jmp    1020
<_init+0x20>

00001040 <printf@plt>:
1040:   ff a3 10 00 00 00      jmp    DWORD
PTR [ebx+0x10]
1046:   68 08 00 00 00         push   0x8
104b:   e9 d0 ff ff ff         jmp    1020
<_init+0x20>

00001050 <__isoc99_scanf@plt>:
1050:   ff a3 14 00 00 00      jmp    DWORD
PTR [ebx+0x14]
1056:   68 10 00 00 00         push   0x10
105b:   e9 c0 ff ff ff         jmp    1020
<_init+0x20>

Дизассемблирование раздела .text:

00001060 <main>:
#include <stdio.h>

int main() {
1060:   8d 4c 24 04            lea
ecx,[esp+0x4]
1064:   83 e4 f0               and
esp,0xfffffff0
1067:   ff 71 fc               push   DWORD
PTR [ecx-0x4]
106a:   55                     push   ebp
106b:   89 e5                  mov    ebp,esp
106d:   57                     push   edi
106e:   e8 6a 02 00 00         call   12dd
<__x86.get_pc_thunk.di>
1073:   81 c7 81 2f 00 00      add
edi,0x2f81
1079:   56                     push   esi
107a:   53                     push   ebx
107b:   51                     push   ecx
107c:   81 ec 98 00 00 00      sub    esp,0x98
1082:   65 a1 14 00 00 00      mov
eax,gs:0x14
1088:   89 45 e4               mov    DWORD
PTR [ebp-0x1c],eax
108b:   31 c0                  xor    eax,eax
int num1, num2, base;
scanf("%d %d %d", &num1, &num2, &base);
108d:   8d 85 7c ff ff ff      lea
eax,[ebp-0x84]
1093:   89 fb                  mov    ebx,edi
1095:   50                     push   eax
1096:   8d 85 78 ff ff ff      lea
eax,[ebp-0x88]
109c:   50                     push   eax
109d:   8d 85 74 ff ff ff      lea
eax,[ebp-0x8c]
10a3:   50                     push   eax
10a4:   8d 87 14 e0 ff ff      lea
eax,[edi-0x1fec]
10aa:   50                     push   eax
10ab:   e8 a0 ff ff ff         call   1050
<__isoc99_scanf@plt>

int quotient = num1 / num2;
10b0:   8b 85 74 ff ff ff      mov
eax,DWORD PTR [ebp-0x8c]
int remainder = num1 % num2;

char quotient_str[50], remainder_str[50];

char *quotient_ptr = quotient_str,
*remainder_ptr = remainder_str;
10b6:   8d 5d 80               lea
ebx,[ebp-0x80]

do {
```

```
    1026:       ff a3 08 00 00 00      jmp
DWORD PTR [ebx+0x8]
    102c:       00 00                  add
BYTE PTR [eax],al
        ...

00001030 <__libc_start_main@plt>:
    1030:       ff a3 0c 00 00 00      jmp
DWORD PTR [ebx+0xc]
    1036:       68 00 00 00 00
        push   0x0
    103b:       e9 e0 ff ff ff         jmp
1020 <_init+0x20>

00001040 <printf@plt>:
    1040:       ff a3 10 00 00 00      jmp
DWORD PTR [ebx+0x10]
    1046:       68 08 00 00 00
        push   0x8
    104b:       e9 d0 ff ff ff         jmp
1020 <_init+0x20>

00001050 <__isoc99_scanf@plt>:
    1050:       ff a3 14 00 00 00      jmp
DWORD PTR [ebx+0x14]
    1056:       68 10 00 00 00
        push   0x10
    105b:       e9 c0 ff ff ff         jmp
1020 <_init+0x20>

Дизассемблирование раздела .text:

00001060 <main>:
#include <stdio.h>

int main() {
    1060:       8d 4c 24 04            lea
ecx,[esp+0x4]
    1064:       83 e4 f0               and
esp,0xfffffff0
    1067:       ff 71 fc
        push   DWORD PTR [ecx-0x4]
    106a:       55
        push   ebp
    106b:       89 e5                  mov
ebp,esp
    106d:       57
        push   edi
    106e:       56
        push   esi
    106f:       e8 b9 02 00 00
        call   132d
<__x86.get_pc_thunk.si>
    1074:       81 c6 80 2f 00 00      add
esi,0x2f80
    107a:       53
        push   ebx
    107b:       51
        push   ecx
    107c:       81 ec 98 00 00 00      sub
esp,0x98
    1082:       89 b5 60 ff ff ff      mov
DWORD PTR [ebp-0xa0],esi
    int num1, num2, base;
    1088:       89 f3                  mov
ebx,esi
    108a:       65 a1 14 00 00 00      mov
eax,gs:0x14
    1090:       89 45 e4               mov
DWORD PTR [ebp-0x1c],eax
    1093:       31 c0                  xor
eax,eax
    scanf("%d %d %d", &num1, &num2,
&base);
    1095:       8d 85 7c ff ff ff      lea
eax,[ebp-0x84]
    109b:       50
        push   eax
```

складывает в регистр сам адрес.

К тому же, иногда, компилятор с флагом оптимизации по размеру (-Os) использует LEA вместо MOV, т.к. LEA – весит 3 байта, а MOV – 4.

Можно заметить, для оптимизации скорости компилятор в некоторых местах использует инструкцию test вместо cmp (выделено зеленым цветом), заметно это в циклах for. Это связано с тем, что test – это инструкция, выполняющая побитовую операцию И (and), что работает быстрее, чем сравнение регистров.

Также можно заметить, что для оптимизации скорости используется (выделено голубым цветом) sub/add вместо dec/inc. Т.к. sub и add работают быстрее, чем dec и inc

Были также замеены изменения в условихя перехода, JMP – безусловный переход, JNE – переход в случае неравенства.

Также использованы прыжки в виде jle, jge, jg, je, jl. Т.е. прыжки, применяющиеся после команды test, cmp, xor и так далее.

Замечена разница в printf (выделено фиолетовым цветом). При оптимизации по скорости данные выводятся из регистров, а при оптимизации по размеру данные выводятся из стека.

```
int q_remainder = quotient % base;
10b9:   83 c4 10                add    esp,0x10
10bc:   89 9d 60 ff ff ff       mov    DWORD
PTR [ebp-0xa0],ebx
int remainder = num1 % num2;
10c2:   99                      cdq
10c3:   f7 bd 78 ff ff ff       idiv   DWORD
PTR [ebp-0x88]
10c9:   89 d6                   mov    esi,edx
int q_remainder = quotient % base;
10cb:   8b 95 7c ff ff ff       mov
edx,DWORD PTR [ebp-0x84]
10d1:   89 95 64 ff ff ff       mov    DWORD
PTR [ebp-0x9c],edx
10d7:   99                      cdq
if (q_remainder < 10)
*quotient_ptr++ = q_remainder + '0';
10d8:   43                      inc    ebx
10d9:   f7 bd 64 ff ff ff       idiv   DWORD
PTR [ebp-0x9c]
else
*quotient_ptr++ = q_remainder - 10 + 'A';
10df:   8d 4a 37                lea
ecx,[edx+0x37]
if (q_remainder < 10)
10e2:   83 fa 09                cmp    edx,0x9
10e5:   7f 03                   jg     10ea
<main+0x8a>
*quotient_ptr++ = q_remainder + '0';
10e7:   8d 4a 30                lea
ecx,[edx+0x30]
10ea:   88 4b ff                mov    BYTE PTR
[ebx-0x1],cl
quotient /= base;
}while (quotient != 0);
10ed:   85 c0                   test   eax,eax
10ef:   75 e6                   jne    10d7
<main+0x77>
char *quotient_ptr = quotient_str,
*remainder_ptr = remainder_str;
10f1:   8d 4d b2                lea
ecx,[ebp-0x4e]
*quotient_ptr = '\0';
10f4:   c6 03 00                mov    BYTE PTR
[ebx],0x0
10f7:   89 8d 5c ff ff ff       mov    DWORD
PTR [ebp-0xa4],ecx

do {
int r_remainder = remainder % base;
10fd:   89 f0                   mov    eax,esi
if (r_remainder < 10)
*remainder_ptr++ = r_remainder + '0';
10ff:   41                      inc    ecx
1100:   99                      cdq
1101:   f7 bd 64 ff ff ff       idiv   DWORD
PTR [ebp-0x9c]
1107:   89 c6                   mov    esi,eax
else
*remainder_ptr++ = r_remainder - 10 + 'A';
1109:   8d 42 37                lea
eax,[edx+0x37]
if (r_remainder < 10)
110c:   83 fa 09                cmp    edx,0x9
110f:   7f 03                   jg     1114
<main+0xb4>
*remainder_ptr++ = r_remainder + '0';
1111:   8d 42 30                lea
eax,[edx+0x30]
1114:   88 41 ff                mov    BYTE PTR
[ecx-0x1],al
remainder /= base;
}while (remainder != 0);
1117:   85 f6                   test   esi,esi
1119:   75 e2                   jne    10fd
<main+0x9d>
*remainder_ptr = '\0';
```

```
    109c:       8d 85 78 ff ff ff       lea
eax,[ebp-0x88]
    10a2:       50
        push   eax
    10a3:       8d 85 74 ff ff ff       lea
eax,[ebp-0x8c]
    10a9:       50
        push   eax
    10aa:       8d 86 14 e0 ff ff       lea
eax,[esi-0x1fec]
    10b0:       50
        push   eax
    10b1:       e8 9a ff ff ff
        call   1050 <__isoc99_scanf@plt>

    int quotient = num1 / num2;
    10b6:       8b 85 74 ff ff ff       mov
eax,DWORD PTR [ebp-0x8c]
    char quotient_str[50],
remainder_str[50];

    char *quotient_ptr = quotient_str,
*remainder_ptr = remainder_str;

    do {
        int q_remainder = quotient %
base;
    10bc:       8b b5 7c ff ff ff       mov
esi,DWORD PTR [ebp-0x84]
    10c2:       83 c4 10                add
esp,0x10
    int remainder = num1 % num2;
    10c5:       99                      cdq
        int q_remainder = quotient %
base;
    10c6:       89 b5 64 ff ff ff       mov
DWORD PTR [ebp-0x9c],esi
    char *quotient_ptr = quotient_str,
*remainder_ptr = remainder_str;
    10cc:       8d 75 80                lea
esi,[ebp-0x80]
    int remainder = num1 % num2;
    10cf:       f7 bd 78 ff ff ff
        idiv   DWORD PTR [ebp-0x88]
    char *quotient_ptr = quotient_str,
*remainder_ptr = remainder_str;
    10d5:       89 f3                   mov
ebx,esi
    10d7:       89 d1                   mov
ecx,edx
    10d9:       8d b4 26 00 00 00 00    lea
esi,[esi+eiz*1+0x0]
        int q_remainder = quotient %
base;
    10e0:       99                      cdq
        if (q_remainder < 10)
            *quotient_ptr++ =
q_remainder + '0';
    10e1:       83 c3 01                add
ebx,0x1
    10e4:       f7 bd 64 ff ff ff
        idiv   DWORD PTR [ebp-0x9c]
        if (q_remainder < 10)
    10ea:       83 fa 09                cmp
edx,0x9
    10ed:       0f 8f ed 00 00 00       jg
11e0 <main+0x180>
            *quotient_ptr++ =
q_remainder + '0';
    10f3:       83 c2 30                add
edx,0x30
    10f6:       88 53 ff                mov
BYTE PTR [ebx-0x1],dl
        else
            *quotient_ptr++ =
q_remainder - 10 + 'A';
        quotient /= base;
    }while (quotient != 0);
```

```
for (int i = 0, j = quotient_ptr -
quotient_str - 1; i < j; i++, j--) {
111b:   8b 85 60 ff ff ff      mov
eax,DWORD PTR [ebp-0xa0]
*remainder_ptr = '\0';
1121:   c6 01 00               mov    BYTE PTR
[ecx],0x0
for (int i = 0, j = quotient_ptr -
quotient_str - 1; i < j; i++, j--) {
1124:   29 c3                  sub    ebx,eax
1126:   31 c0                  xor    eax,eax
1128:   4b                     dec    ebx
1129:   39 d8                  cmp    eax,ebx
112b:   7d 20                  jge    114d
<main+0xed>
char temp = quotient_str[i];
112d:   8a 54 05 80            mov    dl,BYTE
PTR [ebp+eax*1-0x80]
1131:   88 95 64 ff ff ff      mov    BYTE PTR
[ebp-0x9c],dl
quotient_str[i] = quotient_str[j];
1137:   8a 54 1d 80            mov    dl,BYTE
PTR [ebp+ebx*1-0x80]
113b:   88 54 05 80            mov    BYTE PTR
[ebp+eax*1-0x80],dl
quotient_str[j] = temp;
113f:   8a 95 64 ff ff ff      mov    dl,BYTE
PTR [ebp-0x9c]
for (int i = 0, j = quotient_ptr -
quotient_str - 1; i < j; i++, j--) {
1145:   40                     inc    eax
quotient_str[j] = temp;
1146:   88 54 1d 80            mov    BYTE PTR
[ebp+ebx*1-0x80],dl
for (int i = 0, j = quotient_ptr -
quotient_str - 1; i < j; i++, j--) {
114a:   4b                     dec    ebx
114b:   eb dc                  jmp    1129
<main+0xc9>
}

for (int i = 0, j = remainder_ptr -
remainder_str - 1; i < j; i++, j--) {
114d:   8d 45 b2               lea
eax,[ebp-0x4e]
1150:   29 c1                  sub    ecx,eax
1152:   49                     dec    ecx
1153:   39 ce                  cmp    esi,ecx
1155:   7d 13                  jge    116a
<main+0x10a>
char temp = remainder_str[i];
1157:   8a 44 35 b2            mov    al,BYTE
PTR [ebp+esi*1-0x4e]
remainder_str[i] = remainder_str[j];
115b:   8a 54 0d b2            mov    dl,BYTE
PTR [ebp+ecx*1-0x4e]
115f:   88 54 35 b2            mov    BYTE PTR
[ebp+esi*1-0x4e],dl
for (int i = 0, j = remainder_ptr -
remainder_str - 1; i < j; i++, j--) {
1163:   46                     inc    esi
remainder_str[j] = temp;
1164:   88 44 0d b2            mov    BYTE PTR
[ebp+ecx*1-0x4e],al
for (int i = 0, j = remainder_ptr -
remainder_str - 1; i < j; i++, j--) {
1168:   eb e8                  jmp    1152
<main+0xf2>
}

printf("%s\n%s\n", quotient_str,
remainder_str);
116a:   50                     push   eax
116b:   8d 87 1d e0 ff ff      lea
eax,[edi-0x1fe3]
1171:   89 fb                  mov    ebx,edi
1173:   ff b5 5c ff ff ff      push   DWORD
PTR [ebp-0xa4]
    10f9:       85 c0
        test   eax,eax
    10fb:       75 e3                  jne
10e0 <main+0x80>
    *quotient_ptr = '\0';
    10fd:       c6 03 00               mov
BYTE PTR [ebx],0x0
    1100:       89 ca                  mov
edx,ecx
    char *quotient_ptr = quotient_str,
*remainder_ptr = remainder_str;
    1102:       8d 7d b2               lea
edi,[ebp-0x4e]
    1105:       89 f9                  mov
ecx,edi
    1107:       89 d0                  mov
eax,edx
    1109:       8d b4 26 00 00 00 00   lea
esi,[esi+eiz*1+0x0]

    do {
        int r_remainder = remainder %
base;
    1110:       99                     cdq
        if (r_remainder < 10)
            *remainder_ptr++ =
r_remainder + '0';
    1111:       83 c1 01               add
ecx,0x1
    1114:       f7 bd 64 ff ff ff
        idiv   DWORD PTR [ebp-0x9c]
        if (r_remainder < 10)
    111a:       83 fa 09               cmp
edx,0x9
    111d:       0f 8f a5 00 00 00      jg
11c8 <main+0x168>
            *remainder_ptr++ =
r_remainder + '0';
    1123:       83 c2 30               add
edx,0x30
    1126:       88 51 ff               mov
BYTE PTR [ecx-0x1],dl
        else
            *remainder_ptr++ =
r_remainder - 10 + 'A';
        remainder /= base;
    }while (remainder != 0);
    1129:       85 c0
        test   eax,eax
    112b:       75 e3                  jne
1110 <main+0xb0>
    *remainder_ptr = '\0';

    for (int i = 0, j = quotient_ptr -
quotient_str - 1; i < j; i++, j--) {
    112d:       29 f3                  sub
ebx,esi
    *remainder_ptr = '\0';
    112f:       c6 01 00               mov
BYTE PTR [ecx],0x0
    for (int i = 0, j = quotient_ptr -
quotient_str - 1; i < j; i++, j--) {
    1132:       31 c0                  xor
eax,eax
    1134:       83 eb 01               sub
ebx,0x1
    1137:       85 db
        test   ebx,ebx
    1139:       7e 2b                  jle
1166 <main+0x106>
    113b:       89 8d 64 ff ff ff      mov
DWORD PTR [ebp-0x9c],ecx
    1141:       8d b4 26 00 00 00 00   lea
esi,[esi+eiz*1+0x0]
        char temp = quotient_str[i];
    1148:       0f b6 14 06
        movzx  edx,BYTE PTR [esi+eax*1]
        quotient_str[i] =
quotient_str[j];
```

```
1179:   ff b5 60 ff ff ff       push   DWORD
PTR [ebp-0xa0]
117f:   50                      push   eax
1180:   e8 bb fe ff ff          call   1040
<printf@plt>

return 0;
1185:   8b 45 e4                mov
eax,DWORD PTR [ebp-0x1c]
1188:   65 2b 05 14 00 00 00    sub
eax,DWORD PTR gs:0x14
118f:   74 05                   je     1196
<main+0x136>
1191:   e8 5a 01 00 00          call   12f0
<__stack_chk_fail_local>
1196:   8d 65 f0                lea
esp,[ebp-0x10]
1199:   31 c0                   xor    eax,eax
119b:   59                      pop    ecx
119c:   5b                      pop    ebx
119d:   5e                      pop    esi
119e:   5f                      pop    edi
119f:   5d                      pop    ebp
11a0:   8d 61 fc                lea
esp,[ecx-0x4]
11a3:   c3                      ret
11a4:   66 90                   xchg   ax,ax
11a6:   66 90                   xchg   ax,ax
11a8:   66 90                   xchg   ax,ax
11aa:   66 90                   xchg   ax,ax
11ac:   66 90                   xchg   ax,ax
11ae:   66 90                   xchg   ax,ax

000011b0 <_start>:
11b0:   31 ed                   xor    ebp,ebp
11b2:   5e                      pop    esi
11b3:   89 e1                   mov    ecx,esp
11b5:   83 e4 f0                and
esp,0xfffffff0
11b8:   50                      push   eax
11b9:   54                      push   esp
11ba:   52                      push   edx
11bb:   e8 18 00 00 00          call   11d8
<_start+0x28>
11c0:   81 c3 34 2e 00 00       add
ebx,0x2e34
11c6:   6a 00                   push   0x0
11c8:   6a 00                   push   0x0
11ca:   51                      push   ecx
11cb:   56                      push   esi
11cc:   ff b3 f8 ff ff ff       push   DWORD
PTR [ebx-0x8]
11d2:   e8 59 fe ff ff          call   1030
<__libc_start_main@plt>
11d7:   f4                      hlt
11d8:   8b 1c 24                mov
ebx,DWORD PTR [esp]
11db:   c3                      ret
11dc:   66 90                   xchg   ax,ax
11de:   66 90                   xchg   ax,ax

000011e0 <__x86.get_pc_thunk.bx>:
11e0:   8b 1c 24                mov
ebx,DWORD PTR [esp]
11e3:   c3                      ret
11e4:   66 90                   xchg   ax,ax
11e6:   66 90                   xchg   ax,ax
11e8:   66 90                   xchg   ax,ax
11ea:   66 90                   xchg   ax,ax
11ec:   66 90                   xchg   ax,ax
11ee:   66 90                   xchg   ax,ax
11f0:   e8 e4 00 00 00          call   12d9
<__x86.get_pc_thunk.dx>
11f5:   81 c2 ff 2d 00 00       add
edx,0x2dff
11fb:   8d 8a 20 00 00 00       lea
ecx,[edx+0x20]
1201:   8d 82 20 00 00 00       lea
eax,[edx+0x20]
```

```
    114c:       0f b6 0c 1e
        movzx  ecx,BYTE PTR [esi+ebx*1]
    1150:       88 0c 06                mov
BYTE PTR [esi+eax*1],cl
    for (int i = 0, j = quotient_ptr -
quotient_str - 1; i < j; i++, j--) {
    1153:       83 c0 01                add
eax,0x1
        quotient_str[j] = temp;
    1156:       88 14 1e                mov
BYTE PTR [esi+ebx*1],dl
    for (int i = 0, j = quotient_ptr -
quotient_str - 1; i < j; i++, j--) {
    1159:       83 eb 01                sub
ebx,0x1
    115c:       39 d8                   cmp
eax,ebx
    115e:       7c e8                   jl
1148 <main+0xe8>
    1160:       8b 8d 64 ff ff ff       mov
ecx,DWORD PTR [ebp-0x9c]
    }

    for (int i = 0, j = remainder_ptr -
remainder_str - 1; i < j; i++, j--) {
    1166:       29 f9                   sub
ecx,edi
    1168:       31 c0                   xor
eax,eax
    116a:       83 e9 01                sub
ecx,0x1
    116d:       85 c9
        test   ecx,ecx
    116f:       7e 1f                   jle
1190 <main+0x130>
    1171:       8d b4 26 00 00 00 00    lea
esi,[esi+eiz*1+0x0]
        char temp = remainder_str[i];
    1178:       0f b6 1c 07
        movzx  ebx,BYTE PTR [edi+eax*1]
        remainder_str[i] =
remainder_str[j];
    117c:       0f b6 14 0f
        movzx  edx,BYTE PTR [edi+ecx*1]
    1180:       88 14 07                mov
BYTE PTR [edi+eax*1],dl
    for (int i = 0, j = remainder_ptr -
remainder_str - 1; i < j; i++, j--) {
    1183:       83 c0 01                add
eax,0x1
        remainder_str[j] = temp;
    1186:       88 1c 0f                mov
BYTE PTR [edi+ecx*1],bl
    for (int i = 0, j = remainder_ptr -
remainder_str - 1; i < j; i++, j--) {
    1189:       83 e9 01                sub
ecx,0x1
    118c:       39 c8                   cmp
eax,ecx
    118e:       7c e8                   jl
1178 <main+0x118>
    }

    printf("%s\n%s\n", quotient_str,
remainder_str);
    1190:       8b 9d 60 ff ff ff       mov
ebx,DWORD PTR [ebp-0xa0]
    1196:       83 ec 04                sub
esp,0x4
    1199:       57
        push   edi
    119a:       8d 83 1d e0 ff ff       lea
eax,[ebx-0x1fe3]
    11a0:       56
        push   esi
    11a1:       50
        push   eax
    11a2:       e8 99 fe ff ff
        call   1040 <printf@plt>
```

```
1207:   39 c8                   cmp    eax,ecx
1209:   74 1d                   je     1228
<__x86.get_pc_thunk.bx+0x48>
120b:   8b 82 e8 ff ff ff      mov
eax,DWORD PTR [edx-0x18]
1211:   85 c0                   test   eax,eax
1213:   74 13                   je     1228
<__x86.get_pc_thunk.bx+0x48>
1215:   55                      push   ebp
1216:   89 e5                   mov    ebp,esp
1218:   83 ec 14                sub    esp,0x14
121b:   51                      push   ecx
121c:   ff d0                   call   eax
121e:   83 c4 10                add    esp,0x10
1221:   c9                      leave
1222:   c3                      ret
1223:   2e 8d 74 26 00         lea
esi,cs:[esi+eiz*1+0x0]
1228:   c3                      ret
1229:   8d b4 26 00 00 00 00   lea
esi,[esi+eiz*1+0x0]
1230:   e8 a4 00 00 00         call   12d9
<__x86.get_pc_thunk.dx>
1235:   81 c2 bf 2d 00 00      add
edx,0x2dbf
123b:   55                      push   ebp
123c:   89 e5                   mov    ebp,esp
123e:   53                      push   ebx
123f:   8d 8a 20 00 00 00      lea
ecx,[edx+0x20]
1245:   8d 82 20 00 00 00      lea
eax,[edx+0x20]
124b:   83 ec 04                sub    esp,0x4
124e:   29 c8                   sub    eax,ecx
1250:   89 c3                   mov    ebx,eax
1252:   c1 e8 1f                shr    eax,0x1f
1255:   c1 fb 02                sar    ebx,0x2
1258:   01 d8                   add    eax,ebx
125a:   d1 f8                   sar    eax,1
125c:   74 14                   je     1272
<__x86.get_pc_thunk.bx+0x92>
125e:   8b 92 fc ff ff ff      mov
edx,DWORD PTR [edx-0x4]
1264:   85 d2                   test   edx,edx
1266:   74 0a                   je     1272
<__x86.get_pc_thunk.bx+0x92>
1268:   83 ec 08                sub    esp,0x8
126b:   50                      push   eax
126c:   51                      push   ecx
126d:   ff d2                   call   edx
126f:   83 c4 10                add    esp,0x10
1272:   8b 5d fc                mov
ebx,DWORD PTR [ebp-0x4]
1275:   c9                      leave
1276:   c3                      ret
1277:   2e 8d b4 26 00 00 00   lea
esi,cs:[esi+eiz*1+0x0]
127e:   00
127f:   90                      nop
1280:   f3 0f 1e fb             endbr32
1284:   55                      push   ebp
1285:   89 e5                   mov    ebp,esp
1287:   53                      push   ebx
1288:   e8 53 ff ff ff         call   11e0
<__x86.get_pc_thunk.bx>
128d:   81 c3 67 2d 00 00      add
ebx,0x2d67
1293:   83 ec 04                sub    esp,0x4
1296:   80 bb 20 00 00 00 00   cmp    BYTE PTR
[ebx+0x20],0x0
129d:   75 28                   jne    12c7
<__x86.get_pc_thunk.bx+0xe7>
129f:   8b 83 f0 ff ff ff      mov
eax,DWORD PTR [ebx-0x10]
12a5:   85 c0                   test   eax,eax
12a7:   74 12                   je     12bb
<__x86.get_pc_thunk.bx+0xdb>
12a9:   83 ec 0c                sub    esp,0xc
    return 0;
    11a7:       83 c4 10                add
esp,0x10
    11aa:       8b 45 e4                mov
eax,DWORD PTR [ebp-0x1c]
    11ad:       65 2b 05 14 00 00 00    sub
eax,DWORD PTR gs:0x14
    11b4:       75 3d                   jne
11f3 <main+0x193>
    11b6:       8d 65 f0                lea
esp,[ebp-0x10]
    11b9:       31 c0                   xor
eax,eax
    11bb:       59                      pop
ecx
    11bc:       5b                      pop
ebx
    11bd:       5e                      pop
esi
    11be:       5f                      pop
edi
    11bf:       5d                      pop
ebp
    11c0:       8d 61 fc                lea
esp,[ecx-0x4]
    11c3:       c3                      ret
    11c4:       8d 74 26 00             lea
esi,[esi+eiz*1+0x0]
                    *remainder_ptr++ =
r_remainder - 10 + 'A';
    11c8:       83 c2 37                add
edx,0x37
    11cb:       88 51 ff                mov
BYTE PTR [ecx-0x1],dl
    }while (remainder != 0);
    11ce:       85 c0
        test   eax,eax
    11d0:       0f 85 3a ff ff ff       jne
1110 <main+0xb0>
    11d6:       e9 52 ff ff ff          jmp
112d <main+0xcd>
    11db:       2e 8d 74 26 00          lea
esi,cs:[esi+eiz*1+0x0]
                    *quotient_ptr++ =
q_remainder - 10 + 'A';
    11e0:       83 c2 37                add
edx,0x37
    11e3:       88 53 ff                mov
BYTE PTR [ebx-0x1],dl
    }while (quotient != 0);
    11e6:       85 c0
        test   eax,eax
    11e8:       0f 85 f2 fe ff ff       jne
10e0 <main+0x80>
    11ee:       e9 0a ff ff ff          jmp
10fd <main+0x9d>
    11f3:       e8 48 01 00 00
        call   1340
<__stack_chk_fail_local>
    11f8:       66 90
        xchg   ax,ax
    11fa:       66 90
        xchg   ax,ax
    11fc:       66 90
        xchg   ax,ax
    11fe:       66 90
        xchg   ax,ax

00001200 <_start>:
    1200:       31 ed                   xor
ebp,ebp
    1202:       5e                      pop
esi
    1203:       89 e1                   mov
ecx,esp
    1205:       83 e4 f0                and
esp,0xfffffff0
```

```
12ac:   ff b3 1c 00 00 00      push   DWORD
PTR [ebx+0x1c]
12b2:   ff 93 f0 ff ff ff      call   DWORD
PTR [ebx-0x10]
12b8:   83 c4 10               add    esp,0x10
12bb:   e8 30 ff ff ff         call   11f0
<__x86.get_pc_thunk.bx+0x10>
12c0:   c6 83 20 00 00 00 01   mov    BYTE PTR
[ebx+0x20],0x1
12c7:   8b 5d fc               mov
ebx,DWORD PTR [ebp-0x4]
12ca:   c9                     leave
12cb:   c3                     ret
12cc:   8d 74 26 00            lea
esi,[esi+eiz*1+0x0]
12d0:   f3 0f 1e fb            endbr32
12d4:   e9 57 ff ff ff         jmp    1230
<__x86.get_pc_thunk.bx+0x50>

000012d9 <__x86.get_pc_thunk.dx>:
12d9:   8b 14 24               mov
edx,DWORD PTR [esp]
12dc:   c3                     ret

000012dd <__x86.get_pc_thunk.di>:
12dd:   8b 3c 24               mov
edi,DWORD PTR [esp]
12e0:   c3                     ret
12e1:   66 90                  xchg   ax,ax
12e3:   66 90                  xchg   ax,ax
12e5:   66 90                  xchg   ax,ax
12e7:   66 90                  xchg   ax,ax
12e9:   66 90                  xchg   ax,ax
12eb:   66 90                  xchg   ax,ax
12ed:   66 90                  xchg   ax,ax
12ef:   90                     nop

000012f0 <__stack_chk_fail_local>:
12f0:   f3 0f 1e fb            endbr32
12f4:   56                     push   esi
12f5:   5e                     pop    esi
12f6:   e8 0e 00 00 00         call   1309
<__x86.get_pc_thunk.ax>
12fb:   05 f9 2c 00 00         add
eax,0x2cf9
1300:   83 ec 0c               sub    esp,0xc
1303:   ff 90 ec ff ff ff      call   DWORD
PTR [eax-0x14]

00001309 <__x86.get_pc_thunk.ax>:
1309:   8b 04 24               mov
eax,DWORD PTR [esp]
130c:   c3                     ret

Дизассемблирование раздела .fini:

00001310 <_fini>:
1310:   53                     push   ebx
1311:   83 ec 08               sub    esp,0x8
1314:   e8 c7 fe ff ff         call   11e0
<__x86.get_pc_thunk.bx>
1319:   81 c3 db 2c 00 00      add
ebx,0x2cdb
131f:   83 c4 08               add    esp,0x8
1322:   5b                     pop    ebx
1323:   c3                     ret
```

```
1208:       50
               push   eax
1209:       54
               push   esp
120a:       52
               push   edx
120b:       e8 18 00 00 00
               call   1228 <_start+0x28>
1210:       81 c3 e4 2d 00 00      add
ebx,0x2de4
1216:       6a 00
               push   0x0
1218:       6a 00
               push   0x0
121a:       51
               push   ecx
121b:       56
               push   esi
121c:       ff b3 f8 ff ff ff
               push   DWORD PTR [ebx-0x8]
1222:       e8 09 fe ff ff
               call   1030
<__libc_start_main@plt>
1227:       f4                     hlt
1228:       8b 1c 24               mov
ebx,DWORD PTR [esp]
122b:       c3                     ret
122c:       66 90
               xchg   ax,ax
122e:       66 90
               xchg   ax,ax

00001230 <__x86.get_pc_thunk.bx>:
1230:       8b 1c 24               mov
ebx,DWORD PTR [esp]
1233:       c3                     ret
1234:       66 90
               xchg   ax,ax
1236:       66 90
               xchg   ax,ax
1238:       66 90
               xchg   ax,ax
123a:       66 90
               xchg   ax,ax
123c:       66 90
               xchg   ax,ax
123e:       66 90
               xchg   ax,ax
1240:       e8 e4 00 00 00
               call   1329
<__x86.get_pc_thunk.dx>
1245:       81 c2 af 2d 00 00      add
edx,0x2daf
124b:       8d 8a 20 00 00 00      lea
ecx,[edx+0x20]
1251:       8d 82 20 00 00 00      lea
eax,[edx+0x20]
1257:       39 c8                  cmp
eax,ecx
1259:       74 1d                  je
1278 <__x86.get_pc_thunk.bx+0x48>
125b:       8b 82 e8 ff ff ff      mov
eax,DWORD PTR [edx-0x18]
1261:       85 c0
               test   eax,eax
1263:       74 13                  je
1278 <__x86.get_pc_thunk.bx+0x48>
1265:       55
               push   ebp
1266:       89 e5                  mov
ebp,esp
1268:       83 ec 14               sub
esp,0x14
126b:       51
               push   ecx
126c:       ff d0
               call   eax
126e:       83 c4 10               add
esp,0x10
```

```
    1271:      c9
       leave
    1272:      c3                        ret
    1273:      2e 8d 74 26 00            lea
esi,cs:[esi+eiz*1+0x0]
    1278:      c3                        ret
    1279:      8d b4 26 00 00 00 00      lea
esi,[esi+eiz*1+0x0]
    1280:      e8 a4 00 00 00
       call   1329
<__x86.get_pc_thunk.dx>
    1285:      81 c2 6f 2d 00 00         add
edx,0x2d6f
    128b:      55
       push   ebp
    128c:      89 e5                     mov
ebp,esp
    128e:      53
       push   ebx
    128f:      8d 8a 20 00 00 00         lea
ecx,[edx+0x20]
    1295:      8d 82 20 00 00 00         lea
eax,[edx+0x20]
    129b:      83 ec 04                  sub
esp,0x4
    129e:      29 c8                     sub
eax,ecx
    12a0:      89 c3                     mov
ebx,eax
    12a2:      c1 e8 1f                  shr
eax,0x1f
    12a5:      c1 fb 02                  sar
ebx,0x2
    12a8:      01 d8                     add
eax,ebx
    12aa:      d1 f8                     sar
eax,1
    12ac:      74 14                     je
12c2 <__x86.get_pc_thunk.bx+0x92>
    12ae:      8b 92 fc ff ff ff         mov
edx,DWORD PTR [edx-0x4]
    12b4:      85 d2
       test   edx,edx
    12b6:      74 0a                     je
12c2 <__x86.get_pc_thunk.bx+0x92>
    12b8:      83 ec 08                  sub
esp,0x8
    12bb:      50
       push   eax
    12bc:      51
       push   ecx
    12bd:      ff d2
       call   edx
    12bf:      83 c4 10                  add
esp,0x10
    12c2:      8b 5d fc                  mov
ebx,DWORD PTR [ebp-0x4]
    12c5:      c9
       leave
    12c6:      c3                        ret
    12c7:      2e 8d b4 26 00 00 00      lea
esi,cs:[esi+eiz*1+0x0]
    12ce:      00
    12cf:      90                        nop
    12d0:      f3 0f 1e fb
       endbr32
    12d4:      55
       push   ebp
    12d5:      89 e5                     mov
ebp,esp
    12d7:      53
       push   ebx
    12d8:      e8 53 ff ff ff
       call   1230
<__x86.get_pc_thunk.bx>
    12dd:      81 c3 17 2d 00 00         add
ebx,0x2d17
    12e3:      83 ec 04                  sub
esp,0x4
```

```
    12e6:      80 bb 20 00 00 00 00   cmp
BYTE PTR [ebx+0x20],0x0
    12ed:      75 28                  jne
1317 <__x86.get_pc_thunk.bx+0xe7>
    12ef:      8b 83 f0 ff ff ff      mov
eax,DWORD PTR [ebx-0x10]
    12f5:      85 c0
       test   eax,eax
    12f7:      74 12                  je
130b <__x86.get_pc_thunk.bx+0xdb>
    12f9:      83 ec 0c               sub
esp,0xc
    12fc:      ff b3 1c 00 00 00
       push   DWORD PTR [ebx+0x1c]
    1302:      ff 93 f0 ff ff ff
       call   DWORD PTR [ebx-0x10]
    1308:      83 c4 10               add
esp,0x10
    130b:      e8 30 ff ff ff
       call   1240
<__x86.get_pc_thunk.bx+0x10>
    1310:      c6 83 20 00 00 00 01   mov
BYTE PTR [ebx+0x20],0x1
    1317:      8b 5d fc               mov
ebx,DWORD PTR [ebp-0x4]
    131a:      c9
       leave
    131b:      c3                     ret
    131c:      8d 74 26 00            lea
esi,[esi+eiz*1+0x0]
    1320:      f3 0f 1e fb
       endbr32
    1324:      e9 57 ff ff ff         jmp
1280 <__x86.get_pc_thunk.bx+0x50>

00001329 <__x86.get_pc_thunk.dx>:
    1329:      8b 14 24               mov
edx,DWORD PTR [esp]
    132c:      c3                     ret

0000132d <__x86.get_pc_thunk.si>:
    132d:      8b 34 24               mov
esi,DWORD PTR [esp]
    1330:      c3                     ret
    1331:      66 90
       xchg   ax,ax
    1333:      66 90
       xchg   ax,ax
    1335:      66 90
       xchg   ax,ax
    1337:      66 90
       xchg   ax,ax
    1339:      66 90
       xchg   ax,ax
    133b:      66 90
       xchg   ax,ax
    133d:      66 90
       xchg   ax,ax
    133f:      90                     nop

00001340 <__stack_chk_fail_local>:
    1340:      f3 0f 1e fb
       endbr32
    1344:      56
       push   esi
    1345:      5e                     pop
esi
    1346:      e8 0e 00 00 00
       call   1359
<__x86.get_pc_thunk.ax>
    134b:      05 a9 2c 00 00         add
eax,0x2ca9
    1350:      83 ec 0c               sub
esp,0xc
    1353:      ff 90 ec ff ff ff
       call   DWORD PTR [eax-0x14]

00001359 <__x86.get_pc_thunk.ax>:
```

```
    1359:      8b 04 24              mov
eax,DWORD PTR [esp]
    135c:      c3                    ret

Дизассемблирование раздела .fini:

00001360 <_fini>:
    1360:      53
       push   ebx
    1361:      83 ec 08              sub
esp,0x8
    1364:      e8 c7 fe ff ff
       call   1230
<__x86.get_pc_thunk.bx>
    1369:      81 c3 8b 2c 00 00     add
ebx,0x2c8b
    136f:      83 c4 08              add
esp,0x8
    1372:      5b                    pop
ebx
    1373:      c3                    ret
```

СОБСТВЕННЫЕ ПРЕДЛОЖЕНИЕ ПО ОПТИМИЗАЦИИ

Одним из наиболее примитивных, но в тоже время достаточно эффективных способов оптимизации кода является ручная оптимизация, которая реализуется непосредственно самим программистом. Выше была рассмотрена автоматическая реализация, осуществляемая посредством компилятора GCC, но здесь, разумеется, нужно учитывать сложность алгоритма, и бывают случаи, когда программист может справится с задачей оптимизации кода лучше машины, так как точно знает какой результат он хочет получить. В целом, всё зависит от конкретно поставленной задачи. Например, стоит обращать внимание на циклы, что внутри них происходит, что внутри них вызывается, так же стоит обратить внимание на обращение к памяти, избавляться от лишних и ненужных обращений. Также можно избавиться от лишних библиотек, функционал которых переписать или использовать внутренний функционал языка, если это возможно. Например, вместо printf использовать raw вывод. Также можно использовать другие флаги оптимизации.

ИТОГИ

По итогам данной работы был написан алгоритм на языке «C», который был дизассемблирован под разными флагами оптимизации (скорость и размер). В ходе разбора дизассемблерного кода был проведен анализ, в ходе которого стало примерно ясно как компилятор оптимизирует программы с разными флагами и какими путями он это делает. Также были изучены системы команд микропроцессоров семейства i80x86.