МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

«Ручная оптимизация ассемлерного кода»

студента 2 курса, группы 20205

Муратова Максима Александровича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Доцент
Власенко А. Ю.

Новосибирск 2021

СОДЕРЖАНИЕ

ЦЕЛИ

- Проанализировать ассемблерный листинг и понять, что он делает.
- Вручную провести оптимизацию кода.
- Выявить, какая оптимизация оказалась наиболее эффективной.

ЗАДАНИЕ

1. Не нарушив корректность программы, провести оптимизацию его ассемблерного кода.
2. Замерить время выполнения программы на каждом этапе оптимизации.
3. Сравнить скорость исходной программы и программы после всех оптимизаций.

ОПИСАНИЕ РАБОТЫ

Прежде чем начать оптимизацию, необходимо разобаться, что именно делает программа. Для этого нужно было тщательно его проанализировать и декомпилировать; это очень трудоёмкий процесс. Результат декомпиляции представлен в ПРИЛОЖЕНИИ 4.

Оптимизация ассемблерного кода была проведена в 5 этапов:

1. Перенос переменных со стека в регистры процессора. Это было проведено только с теми переменными, которые использовались в циклах, переносить на регистры вещественные переменные я не стал. Например, в обоих циклах счётчик был перенесён в регистр `%r12`.
2. Ускорение переходов данных путём замены связки переходов на одну инструкцию. Первая оптимизация позволила сократить операции присваивания. Например:
   ```
   movl %r12d, %eax
   cltq
   leaq 0(,%rax,8), %rdx
   ```
   Перешло в
   ```
   leaq 0(,%r12,8), %rdx
   ```
3. Упрощение самого вычислительного алгоритма. В `func2()` было замечено, что в цикле к переменной res прибавляется число, которое умножалось на константу. Гораздо проще сначала всё сложить, и только после цикла умножить на константу.
   До:
   ```
   call     sin
   movapd   %xmm0, %xmm1
   movsd    .LC4(%rip), %xmm0
   mulsd    %xmm0, %xmm1
   movsd    %xmm1, -24(%rbp)
   leaq     0(,%r12,8), %rdx
   movq     %r14, %rax
   addq     %rdx, %rax
   movq     (%rax), %rax
   movq     %rax, %xmm0
   call     cos
   #…
   .L6:
   movsd    -8(%rbp), %xmm0
   ```
   После:
   ```
   call     sin
   movsd    %xmm0, -24(%rbp)
   leaq     0(,%r12,8), %rdx
   movq     %r14, %rax
   addq     %rdx, %rax
   movq     (%rax), %rax
   movq     %rax, %xmm0
   ```

```
call     cos
#…
.L6:
movsd    -8(%rbp), %xmm0
movsd    .LC4(%rip), %xmm1
mulsd    %xmm1, %xmm0
```

4. Раскрутка цикла в соотношении 5:1. Первый цикл был удлинён в 5 раз, чтобы во столько же раз уменьшить количество вызовов конструкции `if`

Все оптимизации проводились с осторожностью, так как далеко не всегда использование оптимизаций приводило к увеличению производительности. Так, эти оптимизации не были применены:

- Упрощение преобразование числа типа int в double:
  До:
  `pxor %xmm0, %xmm0`
  `cvtsi2sd %eax, %xmm0`
  После:
  `cvtsi2sd %eax, %xmm0`
- Раскрутка второго цикла в соотношении 5:1. Возможно, падение производительности было вызвано тем, что инструции после второй раскрутки перестали помещаться в кэш `L1i`, что приводило бы к замедлению времени выполнения программы.

Все версии программы замерялись 6 раз при помощи утилиты time, для этого достаточно было вызвать скрипт `tester.sh` (ПРИЛОЖЕНИЕ 2) командой:

```
$ ./tester.sh
```

Вызывать отдельную команду для сборки ассемблерных листингов не нужно, для этого есть специальный файл сборки `Makefile` (ПРИЛОЖЕНИЕ 1), который срабатывает при вызове в tester.sh команды `$ make`

Протокол результата замеров времени работы всех программ записан в файл `report.txt`, с ним можно ознакомиться в ПРИЛОЖЕНИИ 3.

Из этого протокола мы для всех версий программы выпишем строчки с наименьшим реальным временем

Название	**REAL, с**	**USER, с**	**SYS, с**
`prog0`	5.83	5.57	0.25
`prog1`	5.52	5.26	0.25
`prog2`	5.36	5.12	0.22
`prog3`	5.36	5.12	0.23
`prog4`	5.32	5.06	0.26

Сравнение мы проведём так: на сколько процентов уменьшилось время (real) выполнения программы после очередной оптимизации:

1. `prog1`: -5,31%
2. `prog2`: -2,90%
3. `prog3`: -0,00%
4. `prog4`: -0,75%

Итоговое ускорение составляет 8,75%.

ЗАКЛЮЧЕНИЕ

1. Самой эффективной оптимизацией оказался перенос переменных в регистры.
2. Также действенным оказалось сокращение цепочек перемещения данных, но это стало доступно только после оптимизации выше.
3. Упрощение самого вычислительного алгоритма, вопреки ожиданиям, не смогло ускорить программу.
4. Ручная оптимизация ускорила программу на 8,75%.
5. Не всегда оптимизации могут повысить производительность, нужно учитывать особенности процессора.

Приложение 1. *Makefile*
файл сборки проекта

```
CC=g++

all: prog0 prog1 prog2 prog3 prog4

prog0: prog0.s
	$(CC) prog0.s -o prog0

prog1: prog1.s
	$(CC) prog1.s -o prog1

prog2: prog2.s
	$(CC) prog2.s -o prog2

prog3: prog3.s
	$(CC) prog3.s -o prog3

prog4: prog4.s
	$(CC) prog4.s -o prog4
```

Приложение 2. `tester.sh`

Файл для замера времени работы программы

```
#/bin/bash

echo "compiling programs..."
make >/dev/null
if [[ $? -ne 0 ]]; then
      exit
fi
echo "programs compiled"


echo "prog0" | tee report.txt
echo -e "#\treal\tuser\tsys" >>report.txt
for (( i = 0 ; i < 6 ; i++ ))
do
      \time -o report.txt -a -f "$i\t%e\t%U\t%S" ./prog0 \
                >/dev/null
      if [[ $? -ne 0 ]]; then
           exit
      fi
done
echo -e "\n" >>report.txt


echo "prog1" | tee -a report.txt
echo -e "#\treal\tuser\tsys" >>report.txt
for (( i = 0; i < 6 ; i++ ))
do
      \time -o report.txt -a -f "$i\t%e\t%U\t%S" ./prog1 \
                >/dev/null
```

```
	if [[ $? -ne 0 ]]; then
		exit
	fi
done
echo -e "\n" >>report.txt


echo "prog2" | tee -a report.txt
echo -e "#\treal\tuser\tsys" >>report.txt
for (( i = 0; i < 6 ; i++ ))
do
	\time -o report.txt -a -f "$i\t%e\t%U\t%S" ./prog2 \
			>/dev/null
	if [[ $? -ne 0 ]]; then
		exit
	fi
done
echo -e "\n" >>report.txt


echo "prog3" | tee -a report.txt
echo -e "#\treal\tuser\tsys" >>report.txt
for (( i = 0; i < 6 ; i++ ))
do
	\time -o report.txt -a -f "$i\t%e\t%U\t%S" ./prog3 \
			>/dev/null
	if [[ $? -ne 0 ]]; then
		exit
	fi
done
echo -e "\n" >>report.txt
```

```
echo "prog4" | tee -a report.txt
echo -e "#\treal\tuser\tsys" >>report.txt
for (( i = 0; i < 6 ; i++ ))
do
      \time -o report.txt -a -f "$i\t%e\t%U\t%S" ./prog4 \
                /dev/null
      if [[ $? -ne 0 ]]; then
           exit
      fi
done

echo "all done!"
```

Приложение 3. `report.txt`

Протокол результата замеров времени исполнения программ

```
prog0
#    real user sys
0    6.03 5.81 0.21
1    5.85 5.61 0.22
2    5.83 5.57 0.25
3    5.83 5.59 0.23
4    5.84 5.57 0.25
5    5.84 5.62 0.21


prog1
#    real user sys
0    5.53 5.35 0.16
1    5.53 5.31 0.21
2    5.52 5.35 0.17
3    5.52 5.26 0.25
4    5.53 5.32 0.19
5    5.52 5.33 0.18


prog2
#    real user sys
0    5.36 5.13 0.22
1    5.37 5.14 0.21
2    5.37 5.13 0.23
3    5.36 5.17 0.19
4    5.36 5.12 0.22
5    5.36 5.15 0.19
```

prog3

```
#    real user sys
0    5.37 5.14 0.22
1    5.37 5.09 0.26
2    5.36 5.12 0.23
3    5.37 5.11 0.25
4    5.37 5.11 0.25
5    5.38 5.11 0.25
```

prog4

```
#    real user sys
0    5.32 5.13 0.18
1    5.33 5.07 0.25
2    5.33 5.15 0.16
3    5.32 5.06 0.26
4    5.32 5.10 0.21
5    5.34 5.09 0.23
```

Приложение 4. *decompiled.cpp*
Декомпилированный вручную ассемблерный код

```
#include <stdio.h>
#include <cmath>
#include <climits>

void func1(double *arr1, double *arr2) {
    for(int index = 0; index < 50000000; index++) {
        arr1[index] = rand() * 100. / INT_MAX - 50;
        arr2[index] = rand() * 100. / INT_MAX - 50;
    }
}

double func2(const double *arr1, const double *arr2) {
    double res = .0;
    for(int index = 0; index < 50000000; index++)
        res += sin(arr1[index]) * 7.38906 * cos(arr2[index]);
    return res;
}

int main() {
    double *a = new double[50000000];
    double *b = new double[50000000];
    func1(a, b);
    printf("\n\n result=%f\n", func2(a, b));

    delete[] a;
    delete[] b;
    return 0;
}
```

Приложение 5. *prog0.s*

Исходный ассемблерный код

```
 .file     "prog.cpp"
 .text
 .globl    _Z5func1PdS_
 .type     _Z5func1PdS_, @function
_Z5func1PdS_:
.LFB2:
 pushq     %rbp
.LCFI0:
 movq %rsp, %rbp
.LCFI1:
 pushq     %rbx
 subq $40, %rsp
.LCFI2:
 movq %rdi, -40(%rbp)
 movq %rsi, -48(%rbp)
 movl $0, -20(%rbp)
.L3:
 cmpl $49999999, -20(%rbp)
 jg   .L2
 movl -20(%rbp), %eax
 cltq
 leaq 0(,%rax,8), %rdx
 movq -40(%rbp), %rax
 leaq (%rdx,%rax), %rbx
 call rand
 pxor %xmm0, %xmm0
 cvtsi2sd  %eax, %xmm0
 movsd     .LC0(%rip), %xmm1
 mulsd     %xmm1, %xmm0
```

```
 movsd     .LC1(%rip), %xmm1
 divsd     %xmm1, %xmm0
 movsd     .LC2(%rip), %xmm1
 subsd     %xmm1, %xmm0
 movsd     %xmm0, (%rbx)
 movl -20(%rbp), %eax
 cltq
 leaq 0(,%rax,8), %rdx
 movq -48(%rbp), %rax
 leaq (%rdx,%rax), %rbx
 call rand
 pxor %xmm0, %xmm0
 cvtsi2sd  %eax, %xmm0
 movsd     .LC0(%rip), %xmm1
 mulsd     %xmm1, %xmm0
 movsd     .LC1(%rip), %xmm1
 divsd     %xmm1, %xmm0
 movsd     .LC2(%rip), %xmm1
 subsd     %xmm1, %xmm0
 movsd     %xmm0, (%rbx)
 addl $1, -20(%rbp)
 jmp  .L3
.L2:
 movl $0, %eax
 addq $40, %rsp
 popq %rbx
 popq %rbp
.LCFI3:
 ret
.LFE2:
 .size     _Z5func1PdS_, .-_Z5func1PdS_
 .globl    _Z5func2PdS_
```

```
 .type     _Z5func2PdS_, @function
_Z5func2PdS_:
.LFB3:
 pushq     %rbp
.LCFI4:
 movq %rsp, %rbp
.LCFI5:
 subq $48, %rsp
 movq %rdi, -24(%rbp)
 movq %rsi, -32(%rbp)
 pxor %xmm0, %xmm0
 movsd     %xmm0, -8(%rbp)
 movl $0, -12(%rbp)
.L7:
 cmpl $49999999, -12(%rbp)
 jg   .L6
 movl -12(%rbp), %eax
 cltq
 leaq 0(,%rax,8), %rdx
 movq -24(%rbp), %rax
 addq %rdx, %rax
 movq (%rax), %rax
 movq %rax, -40(%rbp)
 movsd     -40(%rbp), %xmm0
 call sin
 movapd    %xmm0, %xmm1
 movsd     .LC4(%rip), %xmm0
 mulsd     %xmm0, %xmm1
 movsd     %xmm1, -40(%rbp)
 movl -12(%rbp), %eax
 cltq
 leaq 0(,%rax,8), %rdx
```

```
 movq -32(%rbp), %rax
 addq %rdx, %rax
 movq (%rax), %rax
 movq %rax, -48(%rbp)
 movsd     -48(%rbp), %xmm0
 call cos
 mulsd     -40(%rbp), %xmm0
 movsd     -8(%rbp), %xmm1
 addsd     %xmm1, %xmm0
 movsd     %xmm0, -8(%rbp)
 addl $1, -12(%rbp)
 jmp  .L7
.L6:
 movsd     -8(%rbp), %xmm0
 leave
.LCFI6:
 ret
.LFE3:
 .size     _Z5func2PdS_, .-_Z5func2PdS_
 .section  .rodata
.LC5:
 .string   "\n\n result = %lf\n"
 .text
 .globl    main
 .type     main, @function
main:
.LFB4:
 pushq     %rbp
.LCFI7:
 movq %rsp, %rbp
.LCFI8:
 subq $48, %rsp
```

```
pxor %xmm0, %xmm0
movsd     %xmm0, -24(%rbp)
movl $400000000, %edi
call _Znam
movq %rax, -16(%rbp)
movl $400000000, %edi
call _Znam
movq %rax, -8(%rbp)
movq -8(%rbp), %rdx
movq -16(%rbp), %rax
movq %rdx, %rsi
movq %rax, %rdi
call _Z5func1PdS_
movq -8(%rbp), %rdx
movq -16(%rbp), %rax
movq %rdx, %rsi
movq %rax, %rdi
call _Z5func2PdS_
movq %xmm0, %rax
movq %rax, -24(%rbp)
movq -24(%rbp), %rax
movq %rax, -40(%rbp)
movsd     -40(%rbp), %xmm0
movl $.LC5, %edi
movl $1, %eax
call printf
movq -16(%rbp), %rax
movq %rax, %rdi
call _ZdlPv
movq -8(%rbp), %rax
movq %rax, %rdi
call _ZdlPv
```

```
    movl $0, %eax
    leave
.LCFI9:
    ret
.LFE4:
    .size    main, .-main
    .section  .rodata
    .align 8
.LC0:
    .long    0
    .long    1079574528
    .align 8
.LC1:
    .long    4290772992
    .long    1105199103
    .align 8
.LC2:
    .long    0
    .long    1078525952
    .align 8
.LC4:
    .long    3100958126
    .long    1075678820
    .section  .eh_frame,"a",@progbits
.LEFDE1:
    .ident    "GCC: (Ubuntu 5.5.0-12ubuntu1~16.04) 5.5.0
20171010"
    .section  .note.GNU-stack,"",@progbits
```

Приложение 5. *prog4.s*

Ассемблерный код после всех оптимизаций

```
  .file    "prog.cpp"
 .text
 .globl    _Z5func1PdS_
 .type     _Z5func1PdS_, @function
_Z5func1PdS_:
.LFB2:
 pushq     %rbp
.LCFI0:
 movq %rsp, %rbp
.LCFI1:
 pushq     %rbx
 pushq   %r12
 pushq   %r13
 pushq   %r14
 subq $16, %rsp
.LCFI2:
 movq %rdi, %r13
 movq %rsi, %r14
 xorq %r12, %r12
.L3:
 cmpl $49999999, %r12d
 jg   .L2

 movl %r12d, %eax
 leaq 0(,%rax,8), %rdx
 leaq (%rdx,%r13), %rbx
 call rand
 pxor %xmm0, %xmm0
```

```
cvtsi2sd  %eax, %xmm0
movsd     .LC3(%rip), %xmm1
divsd     %xmm1, %xmm0

movsd     .LC2(%rip), %xmm1
subsd     %xmm1, %xmm0
movsd     %xmm0, (%rbx)

movl %r12d, %eax
leaq 0(,%rax,8), %rdx
leaq (%rdx,%r14), %rbx
call rand
pxor %xmm0, %xmm0

cvtsi2sd  %eax, %xmm0
movsd     .LC3(%rip), %xmm1
divsd     %xmm1, %xmm0

movsd     .LC2(%rip), %xmm1
subsd     %xmm1, %xmm0
movsd     %xmm0, (%rbx)


movl %r12d, %eax
leaq 8(,%rax,8), %rdx
leaq (%rdx,%r13), %rbx
call rand
pxor %xmm0, %xmm0

cvtsi2sd  %eax, %xmm0
movsd     .LC3(%rip), %xmm1
divsd     %xmm1, %xmm0
```

```
movsd     .LC2(%rip), %xmm1
subsd     %xmm1, %xmm0
movsd     %xmm0, (%rbx)

movl %r12d, %eax
leaq 8(,%rax,8), %rdx
leaq (%rdx,%r14), %rbx
call rand
pxor %xmm0, %xmm0

cvtsi2sd  %eax, %xmm0
movsd     .LC3(%rip), %xmm1
divsd     %xmm1, %xmm0

movsd     .LC2(%rip), %xmm1
subsd     %xmm1, %xmm0
movsd     %xmm0, (%rbx)


movl %r12d, %eax
leaq 16(,%rax,8), %rdx
leaq (%rdx,%r13), %rbx
call rand
pxor %xmm0, %xmm0

cvtsi2sd  %eax, %xmm0
movsd     .LC3(%rip), %xmm1
divsd     %xmm1, %xmm0

movsd     .LC2(%rip), %xmm1
subsd     %xmm1, %xmm0
```

```
movsd     %xmm0, (%rbx)

movl %r12d, %eax
leaq 16(,%rax,8), %rdx
leaq (%rdx,%r14), %rbx
call rand
pxor %xmm0, %xmm0

cvtsi2sd  %eax, %xmm0
movsd     .LC3(%rip), %xmm1
divsd     %xmm1, %xmm0

movsd     .LC2(%rip), %xmm1
subsd     %xmm1, %xmm0
movsd     %xmm0, (%rbx)


     movl %r12d, %eax
leaq 24(,%rax,8), %rdx
leaq (%rdx,%r13), %rbx
call rand
pxor %xmm0, %xmm0

cvtsi2sd  %eax, %xmm0
movsd     .LC3(%rip), %xmm1
divsd     %xmm1, %xmm0

movsd     .LC2(%rip), %xmm1
subsd     %xmm1, %xmm0
movsd     %xmm0, (%rbx)

movl %r12d, %eax
```

```
leaq 24(,%rax,8), %rdx
leaq (%rdx,%r14), %rbx
call rand
pxor %xmm0, %xmm0

cvtsi2sd  %eax, %xmm0
movsd     .LC3(%rip), %xmm1
divsd     %xmm1, %xmm0

movsd     .LC2(%rip), %xmm1
subsd     %xmm1, %xmm0
movsd     %xmm0, (%rbx)


     movl %r12d, %eax
leaq 32(,%rax,8), %rdx
leaq (%rdx,%r13), %rbx
call rand
pxor %xmm0, %xmm0

cvtsi2sd  %eax, %xmm0
movsd     .LC3(%rip), %xmm1
divsd     %xmm1, %xmm0

movsd     .LC2(%rip), %xmm1
subsd     %xmm1, %xmm0
movsd     %xmm0, (%rbx)

movl %r12d, %eax
leaq 32(,%rax,8), %rdx
leaq (%rdx,%r14), %rbx
call rand
```

```
 pxor %xmm0, %xmm0

 cvtsi2sd  %eax, %xmm0
 movsd     .LC3(%rip), %xmm1
 divsd     %xmm1, %xmm0

 movsd     .LC2(%rip), %xmm1
 subsd     %xmm1, %xmm0
 movsd     %xmm0, (%rbx)



 addl $5, %r12d
 jmp  .L3
.L2:
 xorq %rax, %rax
 addq $16, %rsp
 popq    %r14
 popq    %r13
 popq    %r12
 popq %rbx
 popq %rbp
.LCFI3:
 ret
.LFE2:
 .size     _Z5func1PdS_, .-_Z5func1PdS_
 .globl    _Z5func2PdS_
 .type     _Z5func2PdS_, @function
_Z5func2PdS_:
.LFB3:
 pushq     %rbp
.LCFI4:
```

```
 movq %rsp, %rbp
.LCFI5:
 pushq     %r12
 pushq     %r13
 pushq     %r14
 subq $32, %rsp
 movq %rdi, %r13
 movq %rsi, %r14
 pxor %xmm0, %xmm0
 movsd     %xmm0, -8(%rbp)
 xorq %r12, %r12
.L7:
 cmpl $49999999, %r12d
 jg   .L6

 leaq 0(,%r12,8), %rdx
 movq %r13, %rax
 addq %rdx, %rax
 movq (%rax), %rax
 movq %rax, %xmm0
 call sin
 movsd     %xmm0, -24(%rbp)

 leaq 0(,%r12,8), %rdx
 movq %r14, %rax
 addq %rdx, %rax
 movq (%rax), %rax
 movq %rax, %xmm0
 call cos

 mulsd     -24(%rbp), %xmm0
 movsd     -8(%rbp), %xmm1
```

```
 addsd     %xmm1, %xmm0
 movsd     %xmm0, -8(%rbp)

 addl $1, %r12d
 jmp  .L7
.L6:
 movsd     -8(%rbp), %xmm0
 movsd     .LC4(%rip), %xmm1
 mulsd     %xmm1, %xmm0
 popq %r14
 popq %r13
 popq %r12
 leave
.LCFI6:
 ret
.LFE3:
 .size     _Z5func2PdS_, .-_Z5func2PdS_
 .section  .rodata
.LC5:
 .string   "\n\n result = %lf\n"
 .text
 .globl    main
 .type     main, @function
main:
.LFB4:
 pushq     %rbp
.LCFI7:
 movq %rsp, %rbp
.LCFI8:
 subq $48, %rsp
 pxor %xmm0, %xmm0
 movsd     %xmm0, -24(%rbp)
```

```
movl $400000000, %edi
call _Znam
movq %rax, -16(%rbp)
movl $400000000, %edi
call _Znam
movq %rax, -8(%rbp)
movq -8(%rbp), %rdx
movq -16(%rbp), %rax
movq %rdx, %rsi
movq %rax, %rdi
call _Z5func1PdS_
movq -8(%rbp), %rdx
movq -16(%rbp), %rax
movq %rdx, %rsi
movq %rax, %rdi
call _Z5func2PdS_
movq %xmm0, %rax
movq %rax, -24(%rbp)
movq -24(%rbp), %rax
movq %rax, -40(%rbp)
movsd     -40(%rbp), %xmm0
movl $.LC5, %edi
movl $1, %eax
call printf
movq -16(%rbp), %rax
movq %rax, %rdi
call _ZdlPv
movq -8(%rbp), %rax
movq %rax, %rdi
call _ZdlPv
movl $0, %eax
leave
```

```
.LCFI9:
 ret
.LFE4:
 .size     main, .-main
 .section  .rodata
 .align 8
.LC0:
 .long     0
 .long     1079574528
 .align 8
.LC1:
 .long     4290772992
 .long     1105199103
 .align 8
.LC2:
 .long     0
 .long     1078525952
 .align 8
.LC3:
 .long     1199906488
 .long     1098152673
 .align 8
.LC4:
 .long     3100958126
 .long     1075678820
 .section  .eh_frame,"a",@progbits
.LEFDE1:
 .ident    "GCC: (Ubuntu 5.5.0-12ubuntu1~16.04) 5.5.0
20171010"
 .section  .note.GNU-stack,"",@progbits
```