Отчет по индивидуальному домашнему заданию №2 по дисциплине “Архитектура вычислительных систем”.
Владимиров Дмитрий Андреевич, БПИ218.
Вариант 23.

Условие. Разработать программу, которая ищет в ASCII-строке заданную подстроку и возвращает индекс первого символа первого вхождения подстроки в строке. Подстрока вводится как параметр.

Важные замечания. Ограничения по длине строки и искомой подстроки условны и нужны лишь для избежания ошибок, связанных с переполнениями буфера. Если параметры не передаются через командную строку – они генерируются рандомно, иначе необходимо вводить первым параметром имя входного файла, а дальше собственно подстроку которую нужно найти (по условию ее обязательно передавать именно как параметр, она может быть пустой).

На 4 балла:

1. Решение задачи на C – файлы main.c и find_substring.c
2. Оба файла main.c и find_substring.c скомпилированы без каких-либо оптимизирующих опций в файлы main.s и find_substring.s путем команд > gcc -masm=intel -S main.c -o main.s и > gcc -masm=intel -S find_substring.c -o find_substring.s. В этих файлах в комментариях указана связь между переменными в коде на C с действиями с ними в коде на ассемблере. Далее файлы main.s и find_substring.s докомпилированы в исполняемый файл main при помощи команды > gcc main.s find_substring.c -o main.
3. Файлы main.c и find_substring.c скомпилированы с флагами оптимизации по размеру в файлы main2.s и find_substring2.s путем команд > gcc -masm=intel -fno-asynchronous-unwind-tables -fno-jump-tables -fno-stack-protector -fno-exceptions -S main.c -o main2.s и > gcc -masm=intel -fno-asynchronous-unwind-tables -fno-jump-tables -fno-stack-protector -fno-exceptions -S find_substring.c -o find_substring2.s.
4. Файлы main2.s и find_substring2.s докомпилированы в исполняемый файл main2 при помощи команды > gcc main2.s find_substring2.s -o main2.
5. Оба файла были протестированы при помощи команд > ./main input.txt <входные данные> и > ./main2 input.txt <входные данные> и сравнения результатов в файле output.txt.

Входные данные	main	main2
is	82	82
. Some day you will look at your friend	25518	25518
"evening dew";	274914	274914

"evening dew;”	-1	-1
chef	146197	146197
chef	146198	146198
The end	-1	-1
THE END	438335	438335
THE END2	-1	-1

Видим, что выходные данные не отличаются, а значит обе программы функционируют одинаково.
Были использованы следующие опции компиляции:

- -masm=intel для использования необходимого вида ассемблера.
- -fno-asynchronous-unwind-tables гарантирует точность таблиц раскрутуки только в пределах функции.
- -fno-jump-tables – не генерировать инструкций табличного перехода
- -fno-stack-protector – отключает защиту компилятор
- -fno-exceptions позволяет отключить генерацию таблиц обработки исключений.

На 5 баллов:

1. Функции с передачей данных через параметры присутствуют
2. Локальные переменные используются
3. Комментарии по передаче фактических и формальных параметров в функцию и возвращаемого результата добавлены в файлы main.s и find_substring.s к остальным комментариям.

На 6 баллов:
Задача: использовать регистра процессора rbx и r12-r15 везде где можно, то есть минимизировать использование стека.

1. Где было возможно, сохранил переменные в регистры процессора. Файлы с ассемблерным кодом после рефакторинга и комментариями – main_refactored.s и find_substring_refactored.s.
2. Файл find_substring_refactored.s оказался больше чем find_substring.s, а файл main_refactored оказался меньше чем main.s. Такое случилось из за того, что требуется добавить много строчек в программу для первичного выделения регистров (и соответственно освобождения), но при этом они экономят на выполнении операций, поэтому на маленьких файлов оптимизации по размеру не происходит, а в таком крупном файле как main.s уже обратный эффект
3. Программа протестирована на тех же тестовых данных, что и в первый раз. Предварительно программа была скомпилирована при помощи команды > gcc main_refactored.s find_substring_refactored.s -o main_refactored TODO: табличка с тестами

Входные данные	main	main_refactored
is	82	82
. Some day you will look at your friend	25518	25518
"evening dew";	274914	274914
"evening dew;”	-1	-1
chef	146197	146197
chef	146198	146198
The end	-1	-1
THE END	438335	438335
THE END2	-1	-1

На 7 баллов:

1. Программа реализована в виде двух единиц компиляции. (ассемблерные файлы main.s и find_substring.s)
2. Исходные данные лежат в каком-то файле (при указании этого файла, иначе – рандом) и выводятся в файл output.txt. Имя входного файла указывается первым аргументом командной строки, если файл не удается открыть – выводится надпись “Wrong filename” и программа завершается. => командная строка проверяется на корректное открытие файла. На корректность числа аргументов тоже проверка присутствует (если их нет, то запускается рандомная генерация, если это только файл, то ничего не мешает найти пустую подстроку в файле, а если их больше, то ограничения требуются только на размер выделенной памяти, а именно 1000 символов, что тоже отсекается при достижении)
3. Файл input2.txt помогает протестировать программу на небольших данных с большей гибкостью, его легко менять и подстраивать под необходимые условия, а файл input.txt помогает тестировать программу на больших данных, где текст примерно на полмиллиона символов. Результаты работы с тестовыми файлами можно посмотреть в тестовом покрытии на 4 балла.
4. Так как программа не менялась с момента критериев на 4 балла, то необходимость тестировать ее на данном этапе отсутствует.

На 8 баллов:

1. При отсутвии входных данных они генерируются рандомно – сгенерированный входной файл – random_input.txt, сгенерированная подстрока для поиска выводится в консоль, а выходной файл в данном случае – random_output.txt

2. Выбрать в командной строке ввод из файла или рандомной генерации возможно. Как я отмечал ранее, если необходимо сгенерировать данные рандомно, то не нужно передавать ничего, просто запустить исполняемый файл с программой
3. Время работы программы без ввода и вывода замеряется и выводится каждый раз в консоль. Для чистоты эксперимента было приянто решение находить результат 1000 раз, так можно объективнее сравнивать скорости выполнения двух программ. К примеру поиск подстроки “THE END” в файле input.txt 1000 раз занимает примерно 1 секунду, а при рандомной генерации это значение может достигать и 2,5 секунд.
4. Для тестирования времени работы программы подсчет результата в ней выполняется 1000 раз, буду запускать по 5 раз файл main.s на каждых входных данных и брать среднее значение из них.

Входные данные	Время в миллисекундах
is	0
. Some day you will look at your friend	65
"evening dew";	907
"evening dew;”	1417
chef	321
chef	344
The end	976
THE END	951
THE END2	937

На 9 баллов:
1) Для начала скомпилируем ассемблерные файлы с флагом оптимизации -O3 путем команд > gcc -S -O3 main.c -o main_opt.s и > gcc -S -O3 find_substring.c -o find_substring_opt.s. Оба файла по размеру стали меньше чем обычные main.s и find_substring.s без флагов оптимизации.
Далее получим исполняемый файл main_opt путем команды > gcc main_opt.s find_substring_opt.s -o main_opt
Исполняемый файл main_opt также по размеру оказался меньше чем main
Теперь сравним время выполнения программ на тех же входных данных, что и обычно. Данные для первого файла у нас уже есть, осталось только высчитать их для второго.

Входные данные	main (время в мс)	main_opt (время в мс)
is	0	0

. Some day you will look at your friend	65	21
"evening dew";	907	487
"evening dew;”	1417	794
chef	321	96
chef	344	129
The end	976	293
THE END	951	301
THE END2	937	300

2) Для начала скомпилируем ассемблерные файлы с флагом оптимизации -Os путем команд > gcc -S -Os main.c -o main_optsize.s и > gcc -S -Os find_substring.c -o find_substring_optsize.s. Оба файла по размеру стали ожидаемо НАМНОГО меньше чем обычные main.s и find_substring.s без флагов оптимизации.
Далее получим исполняемый файл main_optsize путем команды > gcc main_optsize.s find_substring_optsize.s -o main_optsize
Исполняемый файл main_optsize также по размеру оказался меньше чем main
Теперь сравним время выполнения программ на тех же входных данных, что и обычно. Данные для первого файла у нас уже есть, осталось только высчитать их для второго.

Входные данные	main (время в мс)	main_optsize (время в мс)
is	0	0
. Some day you will look at your friend	65	30
"evening dew";	907	517
"evening dew;”	1417	843
chef	321	139
chef	344	167
The end	976	429
THE END	951	423
THE END2	937	411

Вывод: Флаг -O3 больше оптимизирует по скорости, но меньше по размеру кода (хотя какая-то оптимизация и происходит), а флаг -Os сильно оптимизирует по размеру кода, но по производительности уступает флагу -O3