Национальный исследовательский университет “Высшая школа экономики”.

Факультет компьютерных наук. Программная инженерия.

Архитектура вычислительных систем.

Домашнее задание №4 студента группы БПИ213 Абрамова Александра Сергеевича.

При выполнении задания были разработаны программы `1.asm` и `2.asm`, а также shell-скрипты для удобной сборки и запуска программ с помощью утилит `as` и `gcc`.

Программа `1.asm` вычисляет наибольшее значение аргумента, при котором значение факториала помещается в 64-х разрядном машинном слове.

Для этого программа использует одноопернадную команду `mul`, которая умножает значение регистра `rax` на значение указанного регистра (`rcx`) и записывает результат в пару регистров `rdx:rax`. Очевидно, что результат умножения двух 64-х битных регистров всегда поместится в 128 бит - пару регистров, а если значение не превышает 2^{64}, то оно будет полностью размещено в регистре `rax` и значение `rdx` будет равно нулю. Таким образом, результат вычисления факториала размещается в 64-х разрядном машинном слове тогда и только тогда, когда регистр `rdx` не используется для хранения результата вычисления факториала (то есть равен нулю).

Исходя из этого, для решения задачи будем последовательно вычислять факториалы чисел, начиная с 1, пока регистр `rdx` не будет использован для хранения результата. Установим начальные значения `rax`, `rdx` и `rcx` в `1`, `0` и `1` соответственно. Будем поддерживать значения так, что

$$rdx{:}rax \ = \ rcx!$$

Организуем цикл `check_next`, который на каждой итерации увеличивает значение `rcx` на `1` и обновляет значение факториала, умножая предыдущее значение на новое значение `rcx`. По свойствам факториала,

$$(n \ + \ 1)! \ = \ n! \ * \ (n \ + \ 1)$$

Учитывая, что предыдущее значение в начале итерации не превышает 64 разряда в длину, то есть полностью содержится в регистре `rax` (`rdx = 0`), команда `mul rcx` выполнит корректное вычисление значения.

Если значение факториала после умножения не помещается в 64-х разрядном машинном слове, для его хранения будет использован регистр `rdx`. Таким образом, если `rdx != 0`, ответ был найден, и необходимо завершить цикл (команды `cmp rdx, 0` и `jne print_answer`). В ином случае необходимо перейти к следующей итерации командой `jmp check_next`.

После выхода из цикла регистр `rcx` содержит первое значение, при котором произошло переполнение 64-х разрядного машинного слова при вычислении факториала. Значит, искомое значение равно `rcx - 1`. Для его вывода воспользуемся функцией `printf` стандартной библиотеки языка C. Для этого вызовем её с помощью `call printf`, передав `output_template[rip]` и `rcx` как первый и второй параметры в регистрах `rdi` и `rsi` соответственно. Здесь `output_template` -

шаблонная строка, указывающая на вывод ровно одного целочисленного значения. Для корректной работы `printf` необходимо также установить значение регистра `rax` в `0`, что указывает `printf`, что все необходимые значения находятся в регистрах или на стеке, а не в сопроцессорах (например, SSE).

Программа `2.asm` реализовывает вычисление факториала в виде подпрограммы, и выводит значения факториалов чисел от 1 до 20 на стандартный поток вывода.

Для этого реализована подпрограмма `factorial`, которая, в соответствии с соглашением о вызовах, принимает в качестве входных данных в регистре `rdi` единственное значение - аргумент. По аналогии с программой `1.asm` функция вычисляет значение факториала, перемножая все числа от `1` до значения `rdi` в паре регистров `rdx:rax`, в которых и располагается возвращаемое значение.

Основная программа организует цикл `check_next` на основе значения регистра `r12`, который относится к группе регистров, сохраняемых вызываемое функцией. На каждой итерации значение `r12` увеличивается на `1` и передаётся в `factorial`. После этого, аналогично программе `1.asm`, происходит проверка на переполнение 64-х разрядного машинного слова значением факториала. Если произошло переполнение, происходит выход из цикла и завершение программы; иначе - вывод значения с помощью `printf` и переход на следующую итерацию.

В результате запуска разработанных программ получены следующие ответы:

```
debian@vps-d19579e5:~/as$ ./1.sh
20
debian@vps-d19579e5:~/as$ ./2.sh
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
debian@vps-d19579e5:~/as$
```