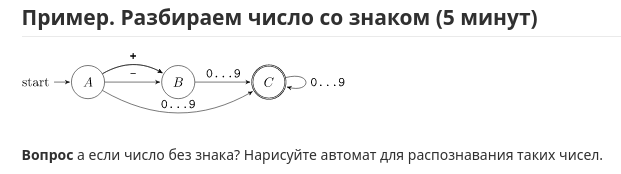
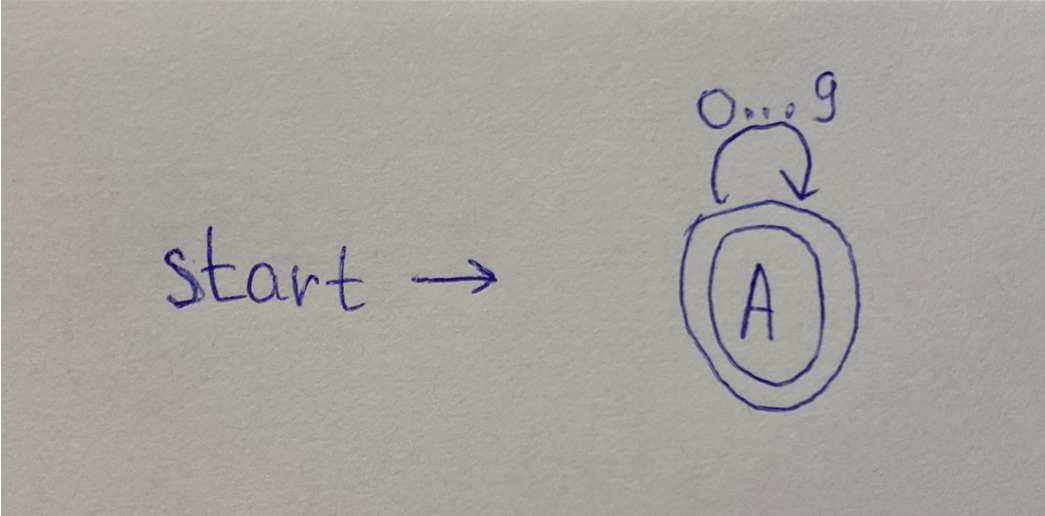


[start] (0)->A (1)->B (0)->B (0)->B (1)->A [end]

т.е. суть этого автомата в том, что если в строке четное кол-во единиц, то его конечным состоянием будет A, иначе B.

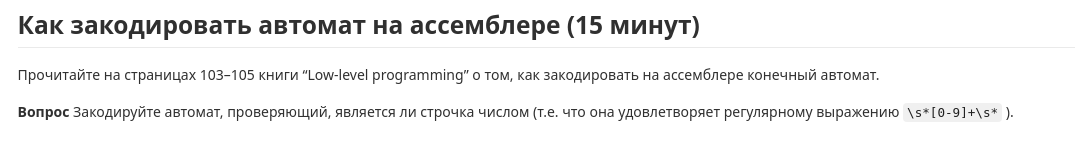


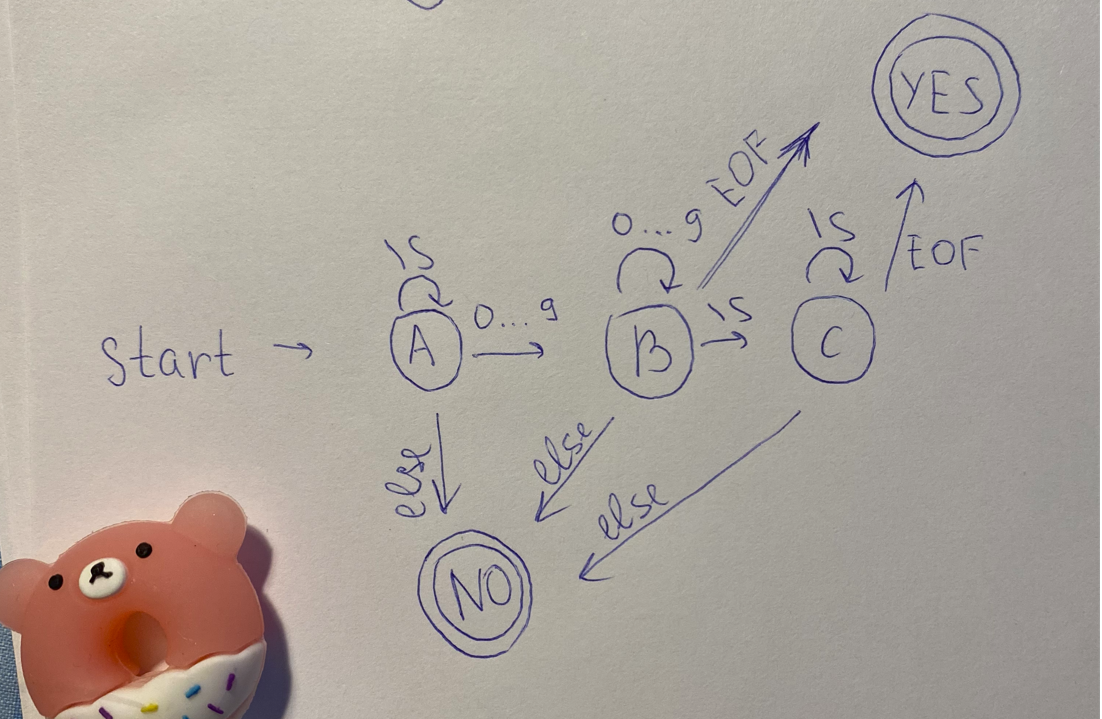


Двойной кружочек вокруг состояния А означает, что это же состояние и является конечным.



На диаграмме изображен автомат, описывающий жизненный цикл [процесса](https://ru.bmstu.wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B)). Так или иначе все операционные системы подчиняются этой модели. В конкретных операционных системах состояния процесса могут быть еще более детализированы, могут появиться некоторые новые варианты переходов из одного состояния в другое.





Сначала нарисуем автомат, соответствующий этому регулярному выражению.

\s - любой пробельный символ

else - символ что не является ни числом, ни пробельным символом

EOF - символ окончания строки (в данном случае 0)

**Код и комментарии 1.asm тута:**

section .data

str: db " 101", 0

section .text

global \_start

is\_num:

;состояния YES и NO считаем конечными, состояние NO говорит о некорректном символе

;в строке или о ее конце, а состояние YES о том, что всё супер.

.A:

;в r8b мы кладем текущий символ, т.е. берем байт по адресу rdi, который

;затем также на байт увеличиваем, чтоб в нем уже был указан адрес следующего

;символа

mov r8b, [rdi]

inc rdi

;далее мы сравниваем текущий символ со всеми пробельными символами,

;чтобы при распознавании оного перейти в состояние A

cmp r8b, 0xA ;0xA - код символа переноса строки

je .A

cmp r8b, 0x10 ;0x10 - код символа табуляции

je .A

cmp r8b, 0x20 ;0x20 - код символа пробела

je .A

;сравниваем r8b с '0'. jb - jump below, т.е. переходим в состояние NO,

;если r8b меньше, чем '0'. По сути, NO - это неопределенное состояние, если

;автомат оказывается в нем, значит в строке есть какие-то символы помимо

;пробельных и цифр

cmp r8b, '0'

jb .NO

;сравниваем r8b с '9'. ja - jump above, т.е. переходим в состояние NO,

;если r8b больше, чем '9'.

cmp r8b, '9'

ja .NO

;тут мы оказываемся если в r8b символ от 0 до 9, так что смело прыгаем в

;состояние B

jmp .B

.B:

;опять кладем в r8b текущий символ и сдвигаем указатель

mov r8b, [rdi]

inc rdi

cmp r8b, 0

je .YES

;при встрече с пробельным символом прыгаем в состояние C

cmp r8b, 0xA ;0xA - код символа переноса строки

je .C

cmp r8b, 0x10 ;0x10 - код символа табуляции

je .C

cmp r8b, 0x20 ;0x20 - код символа пробела

je .C

;повторям процедуру с определением разных неуместных символов

cmp r8b, '0'

jb .NO

cmp r8b, '9'

ja .NO

jmp .B

.C:

;опять кладем в r8b текущий символ и сдвигаем указатель

mov r8b, [rdi]

inc rdi

cmp r8b, 0

je .YES

;при встрече с пробельным символом прыгаем в состояние C

cmp r8b, 0xA ;0xA - код символа переноса строки

je .C

cmp r8b, 0x10 ;0x10 - код символа табуляции

je .C

cmp r8b, 0x20 ;0x20 - код символа пробела

je .C

;так как в состоянии C мы ожидаем только пробельные символы, то идем в

;неопределенность при любом раскладе, если в предыдущих строчках не

;распознали их.

jmp .NO

.NO:

ret

.YES:

ret

exit:

mov rax, 60

xor rdi, rdi

syscall

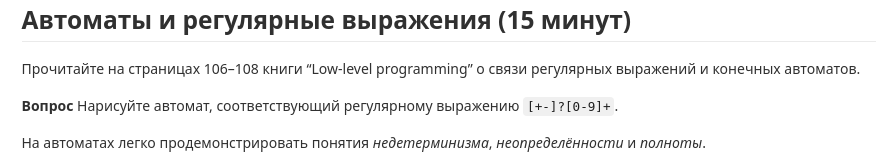
\_start:

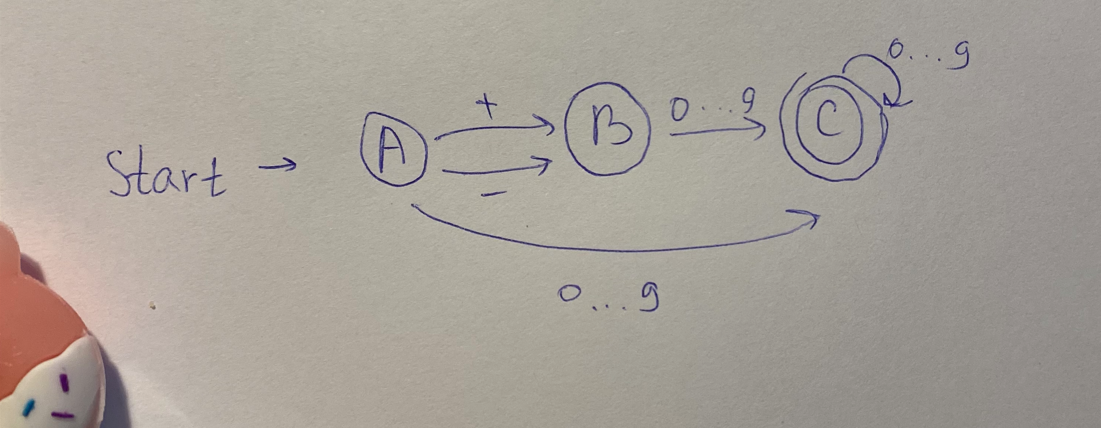
mov rdi, str

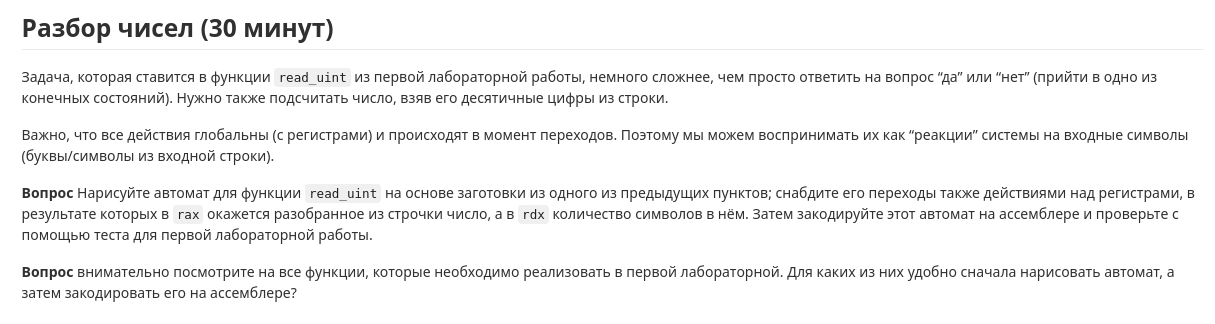
call is\_num

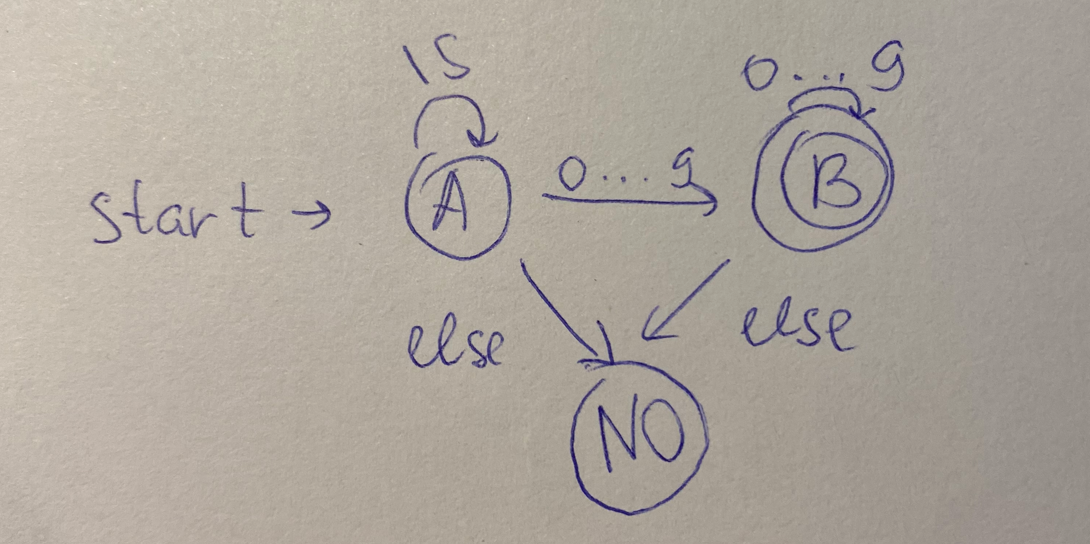
call exit

Добавив состояния YES и NO, мы достроили автомат до полного. В YES он придет в случае, если разберет число, иначе придет в NO.

****

****

****

****

Нарисуем автомат для функции, а дальше модифицируем программу is\_num\_machine.asm.

**Код и комментарии 2.asm тута:**

section .data

str: db " 101as23 ", 0

section .text

global \_start

read\_uint:

;состояния B считаем конечными, состояние NO говорит о некорректном символе

;в строке или о ее конце

;rdx - счетчик цифр в строке

xor rdx, rdx

;так же зануляем rax, в нем будет храниться само разобранное число

xor rax, rax

;обнуляем r9, этот регистр будет служить нам указателем на символ относительно

;начала строки

xor r9, r9

;храним в регистре 10 в качестве множителя rax

mov r11, 10

.A:

;в r8b мы кладем текущий символ, т.е. берем байт по адресу rdi+r9, где

;rdi - указатель на начало строки

;r9 - счетчик

mov r8b, [rdi+r9]

inc r9

;далее мы сравниваем текущий символ со всеми пробельными символами,

;чтобы при распознавании оного перейти в состояние A

cmp r8b, 0xA ;0xA - код символа переноса строки

je .A

cmp r8b, 0x10 ;0x10 - код символа табуляции

je .A

cmp r8b, 0x20 ;0x20 - код символа пробела

je .A

;сравниваем r8b с '0'. jb - jump below, т.е. переходим в состояние NO,

;если r8b меньше, чем '0'.

cmp r8b, '0'

jb .NO

;сравниваем r8b с '9'. ja - jump above, т.е. переходим в состояние NO,

;если r8b больше, чем '9'.

cmp r8b, '9'

ja .NO

;тут мы оказываемся если в r8b символ от 0 до 9, так что

;мы должны добавить к числу rax эту цифру

;т.е. умножить его на 10 и прибавить значение цифры

;умножаем rax на 10

;десятка лежит у нас в регистре r11

;на время умножения rdx мы кидаем в стек, так как при умножении rax

;результат располагается в двух регистрах rdx:rax, но rdx у нас является

;счетчиком, поэтому нам нужно его восстановить

push rdx

mul r11

pop rdx

;вычитаем из r8b код символа '0', чтоб в нем оказалась реальная цифра

; это не магия, просто коды цифр в таблице ascii лежат последовательно

; '0'~48 '1'~49 '2'~50 и тд

; таким образом '3'-'0' ~ 51-48=3

sub r8b, '0'

;наконец прибавляем к rax циферку

add rax, r8

;добавим единичку к счетчику цифр

inc rdx

jmp .B

.B:

;опять кладем в r8b текущий символ и сдвигаем указатель

mov r8b, [rdi+r9]

inc r9

;повторям процедуру с определением разных неуместных символов

cmp r8b, '0'

jb .NO

cmp r8b, '9'

ja .NO

;добавляем цифру к числу в rax

push rdx

mul r11

pop rdx

sub r8b, '0'

add rax, r8

;добавим единичку к счетчику цифр

inc rdx

jmp .B

.NO:

ret

exit:

mov rax, 60

xor rdi, rdi

syscall

\_start:

mov rdi, str

call read\_uint

call exit