

Информатика

Занятие 6

Грозов Владимир Андреевич

va_groz@mail.ru

Блок-схемы





Руководящий документ:

- ГОСТ 19.701-90 (ИСО 5807-85)

Основные элементы блок-схем:

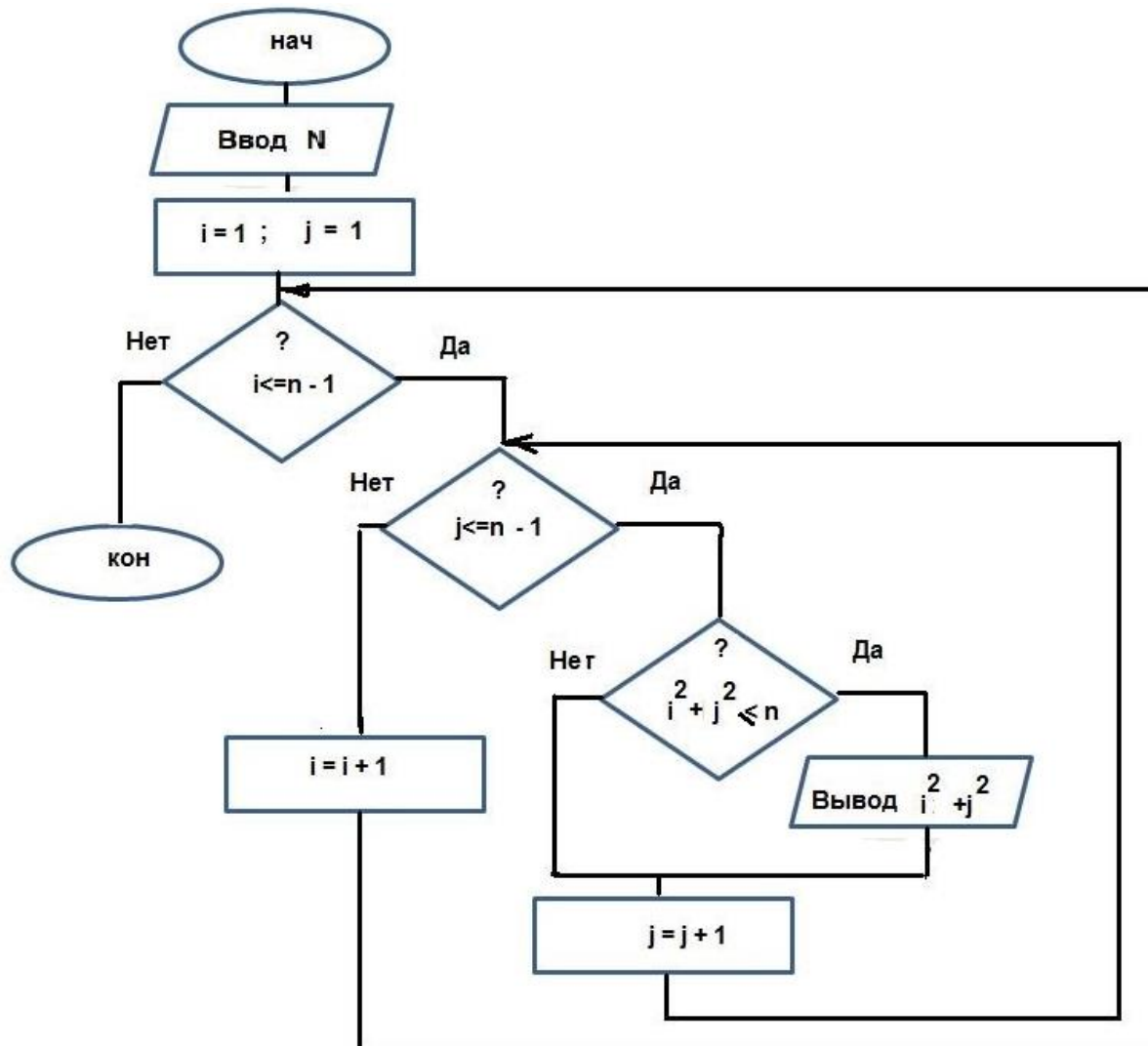
- Процесс
- Данные
- Решение
- Границы цикла
- Подготовка
- ...

Блок-схемы. Основные элементы

Элемент	Обозначение	Суть
Процесс		Обработка данных (вычисления и т.д.)
Терминатор		Начало и завершение программы
Данные		Операции ввода и вывода данных
Решение		Ветвления, выбор (в нашем случае – условный оператор)
Границы цикла		Подходят для любых циклов
Подготовка		Счётные циклы (цикл for)

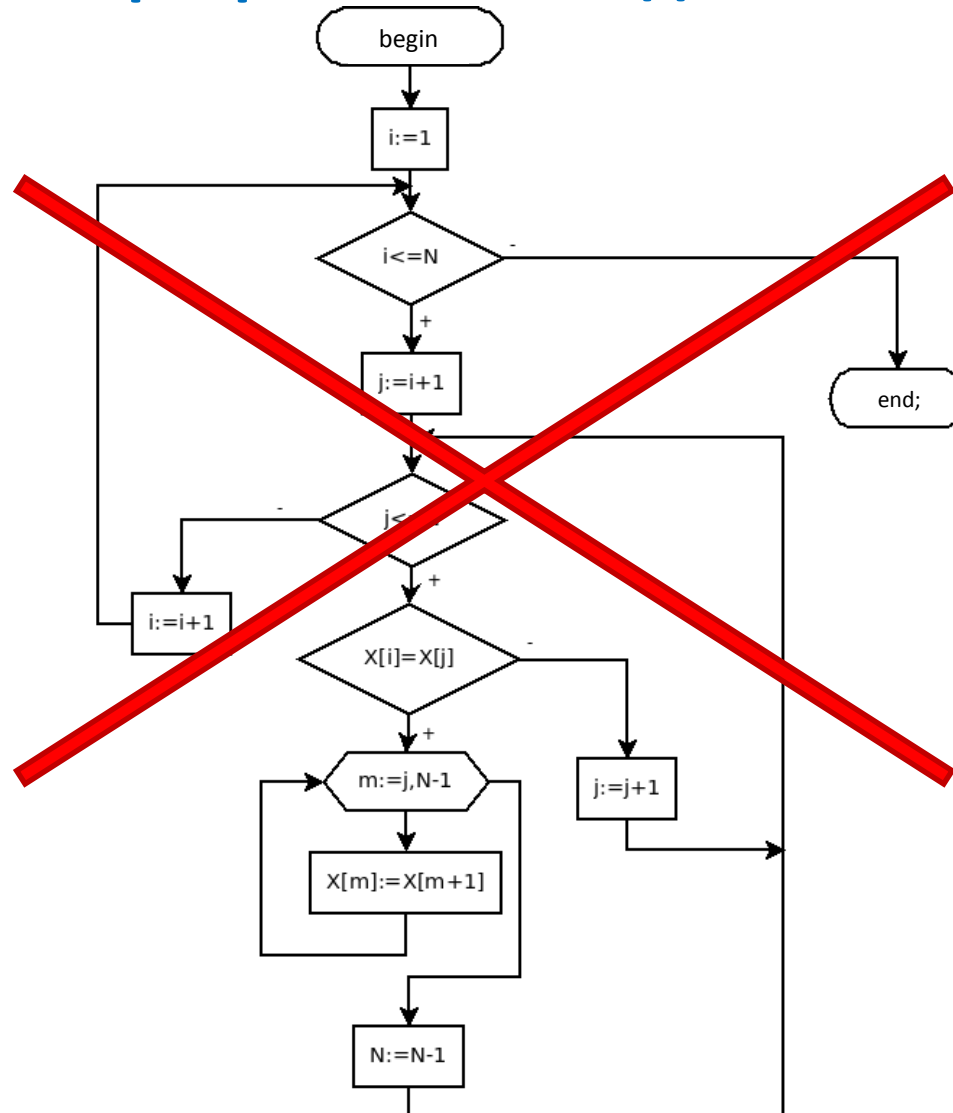
Блок-схемы

Пример блок-схемы программы на языке С:



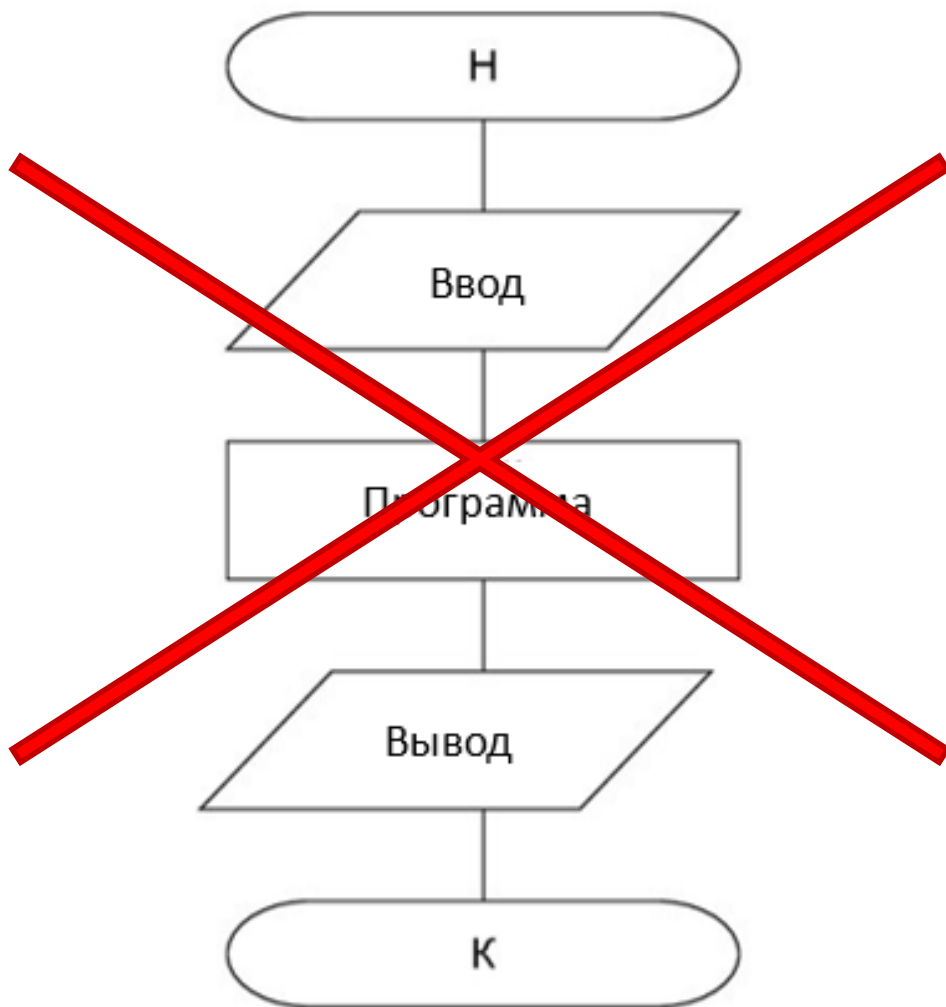
Блок-схемы. Требования

Блок-схема != Программный код



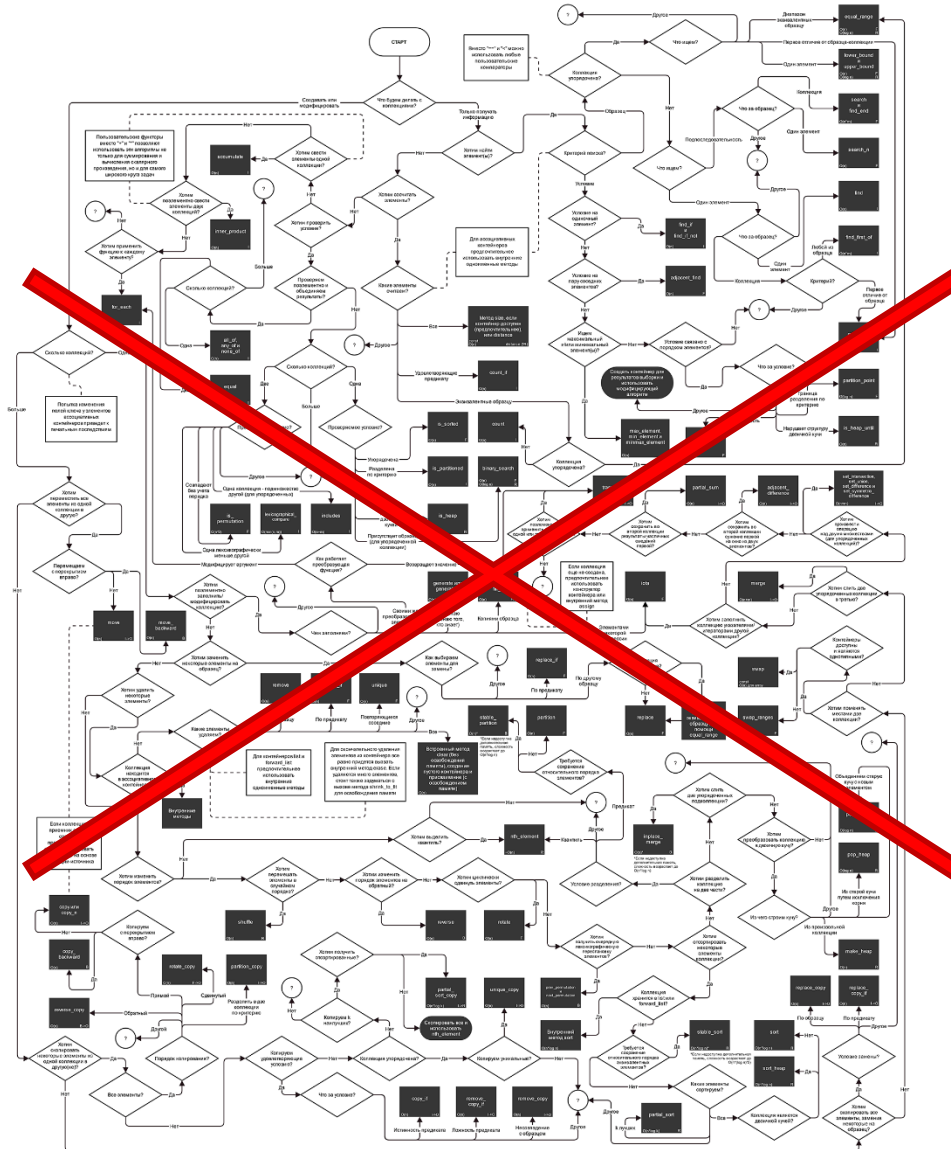
Блок-схемы. Требования

Блок-схема не должна состоять из 2-3-4 элементов!



Блок-схемы. Требования

Блок-схема не должна быть размером в 100500 элементов!



Assembler

Достоинства:

- Возможность сделать программу более эффективной
- Возможность получения прямого доступа к аппаратным средствам
- Возможность лучше понять принципы работы программы

Недостатки:

- Непривычность
- Могут возникать проблемы с переносимостью на компьютеры с другой архитектурой

Assembler. Регистры

- Регистров - ограниченное количество
- Часто используются для краткосрочного хранения данных
- Нужные нам виды регистров:
 - Регистры общего назначения (РОНЫ)
 - «Старые»
 - «Новые»
 - Специальные регистры

Assembler. Регистры

Регистры общего назначения

- Могут быть 1-байтными, 2-байтными, 4-байтными,...
- Могут разделяться на части, которые тоже являются регистрами
- Выполняются в арифметических, логических, ... операциях

Assembler. Регистры

«Старые» регистры (не совсем) общего назначения

32 бита	64 бита	Составные части 64-битных регистров
eax	rax	eax, ax, ah, al
ebx	rbx	ebx, bx, bh, bl
ecx	rcx	ecx, cx, ch, cl
edx	rdx	edx, dx, dh, dl
esi	rsi	esi, si
edi	rdi	edi, di
ebp	rbp	ebp, bp
esp	rsp	esp, sp

Assembler. Регистры

«Новые» регистры Assembler x86-64 общего назначения

Регистр	Составные части регистра
r8	r8d, r8w, r8b
r9	r9d, r9w, r9b
r10	r10d, r10w, r10b
r11	r11d, r11w, r11b
r12	r12d, r12w, r12b
r13	r13d, r13w, r13b
r14	r14d, r14w, r14b
r15	r15d, r15w, r15b

Assembler. Регистры

Специальные регистры

- Регистр флагов (**rflags**)
 - CF
 - SF
 - ZF
 - OF
 - ...
- Регистр счётчика команд (**rip**): 64 бита
 - eip: 32 бита
 - ip: 16 бит

Assembler. Hello World

```
global _start
section .text
_start:  mov rax, sys_write
         mov rdi, 1          ; 1 = stdout
         mov rsi, msg
         mov rdx, msg_len
         syscall
         mov rax, sys_exit
         xor rdi, rdi        ; exit code
         syscall

section .data
msg      db    'Hello, world!', 10 ; 10 – символ перевода строки
msg_len  equ   $ - msg ; $ - последний байт строки msg
sys_write equ   1
sys_exit equ   60
```

Assembler. Секции

- Ключевое слово – section
- Секция кода (.text)
- Сегмент данных
 - Секция данных (.data)
 - Секция неинициализированных данных (.bss)
- Секция стека

Assembler. Выделение памяти

Директивы задания исходных данных:

- db – 1 байт
- dw – 1 слово (2 байта)
- dd – 1 двойное слово (4 байта)
- dq – 1 учетверённое слово (8 байт)
- dt – 10 байт

Пример:

variable dq 1234321 ; задана восьмибайтная переменная variable,
; равная 1234321

Имена переменных – по сути, метки!

Assembler. Выделение памяти

Директивы инициализирования неинициализированной памяти:

- `resb` ;Резервирование 1 байта
- `resw` ;Резервирование 1 слова (2 байт)
- `resd` ;Резервирование 1 двойного слова (4 байт)
- `resq` ;Резервирование 1 учетверённого слова (8 байт)
- `rest` ;Резервирование 10 байт

Пример:

`x resw 100` ; по адресу, связанному с меткой `x`, расположен
; массив из 100 2-байтовых ячеек

Assembler. Задание чисел в разных системах счисления

Запись числа в двоичной системе счисления:

- Суффикс b (например, 010111101b)

Запись числа в восьмеричной системе счисления:

- Суффиксы q или o (например, 347o или 2137q)

Запись числа в шестнадцатеричной системе счисления:

- Суффикс h, или префикс \$, или префикс 0x (например, 3746h или \$234 или 0x8AF3)

Assembler. Инициализация переменных

- В Assembler имя переменной – по сути, метка, ссылающаяся на начало какой-то области памяти

```
section .data  
a    dw    'AbraCadabra', 10  
b    equ    100  
c    db     1  
d    dq     0x0ABCD123
```

Assembler. Задание чисел в разных системах счисления

- В Assembler запись числа всегда начинается с цифры!
- **Правильно:** 0x0F12D, \$0DEADBEEF, 0xA2B3C1D, 0D12BA7h
- **Ошибка:** \$D102D3A, AB15CDh

Assembler. Команда mov

- Выполняет копирование данных из одного места в другое
- Пример: `mov eax, ecx`
- `eax` – куда будут занесены данные
- `ecx` – откуда будут занесены данные

Assembler. Операнды

Виды операндов

- Регистровые операнды (rax, rbx, ...)
- Непосредственные операнды (54, 0x0AB, operand, x)
- Операнды типа «память» (адресные операнды) ([x], [operand], [rax])

Assembler. Адресация

Виды адресации

- **Прямая**
 - Берётся содержимое регистра/переменной
 - `mov rax, rbx`
- **Косвенная**
 - Содержимое регистра/переменной используется в качестве адреса
 - `mov rax, [rbx]`

Assembler. Размеры операндов

- Размеры регистровых операндов задавать не нужно!
- Для операндов типа «память» используются **спецификаторы размера**:
 - byte
 - word
 - dword
 - ...
- Примеры:
 - `mov byte [a], 100`
 - `mov [d], word 1000`

Assembler. Возможные комбинации операндов

- **Допустимые комбинации операндов команды mov**
mov eax, ebx
mov rax, 0x0A
mov eh, 12
mov [address], r10w
mov r10w, [address]
- **Недопустимые комбинации операндов команды mov**
mov [address1], [address2]
mov x, rax
mov 5, rcx
mov [vr], 120 ; правильный вариант: mov [vr], word 120
; или: mov word [vr], 120

Assembler. Арифметические операции с целыми числами

- **Сложение**

- add eax, ebx
- add [variable1], edx

- **Вычитание**

- sub rax, rbx
- sub rcx, [variable2]

Assembler. Сложение и вычитание с переносом

Учитывается значение флага CF

- **Сложение с переносом**
 - `adc eax, edx`
 - `adc [variable1], edx`
- **Вычитание с переносом**
 - `sbb rcx, rbx`
 - `sbb rbx, [variable2]`

Assembler. Целочисленное умножение и деление

- **Умножение**
 - Беззнаковое: **mul**
 - `mov ax, 10`
`mov r8w, 20`
`mul r8w`
 - Со знаком: **imul**
 - `mov ax, -10`
`mov r12w, 20`
`imul r12w`

Assembler. Целочисленное умножение и деление

- **Деление**

- Беззнаковое: `div`

- `mov eax, 110`
`mov r9d, 20`
`div r9d`

- Со знаком: `idiv`

- `mov eax, 120`
`mov r8d, -30`
`idiv r8d`

Assembler. Другие полезные команды

- **cmp** – сравнение двух операндов
 - `cmp eax, ebx`
- **inc** – инкремент
 - `x db 0`
`inc x`
- **dec** – декремент
 - `y db 100`
`dec y`
- **neg** – изменение знака числа
 - `t db 100`
`neg t`

Assembler. Переходы

- **Условные**

- Проверяется тот или иной флаг, полученный после выполнения какой-либо операции. Если значение этого флага равно 1, осуществляется переход по заданному адресу (часто – по метке)

- **Безусловный**

- Переход на заданный адрес (метку) осуществляется гарантированно, без всякой проверки

Assembler. Переходы

- Безусловный переход:
 - jmp
- Условные переходы (j (+ <not>) + <флаг>):
 - jz (ZF = 1)
 - js (SF = 1)
 - jc (CF = 1)
 - jnz (ZF = 0)
 - jnc (CF = 0)
 - ...

Assembler. Переходы по результатам сравнения

Беззнаковые числа

Переход	Назначение
jg	Переход, если первый операнд > второго
jge	Переход, если первый операнд >= второму
jl	Переход, если первый операнд < второго
jle	Переход, если первый операнд <= второму
jng	Переход, если первый операнд не больше второго
jnge	Переход, если первый операнд < второго
jnl	Переход, если первый операнд не меньше второго
jnle	Переход, если первый операнд > второго

Assembler. Переходы по результатам сравнения

Числа со знаком

Переход	Назначение
ja	Переход, если первый операнд > второго
jae	Переход, если первый операнд >= второму
jb	Переход, если первый операнд < второго
jbe	Переход, если первый операнд <= второму
jna	Переход, если первый операнд не больше второго
jnae	Переход, если первый операнд < второго
jnb	Переход, если первый операнд не меньше второго
jnbe	Переход, если первый операнд > второго

Assembler. Циклы

Способ 1

```
mov r8b, 0
```

```
mov r10b, 10
```

```
lp1:  ;...
```

```
    inc r8b          ; или dec r10b
```

```
    cmp r10b, r8b
```

```
    jne lp1
```

Assembler. Циклы

Способ 2

- Использование команды `loop`

```
mov ecx, 10
```

```
lp1:  ;...
```

```
      loop lp1
```

- У команды `loop` есть некоторые ограничения

Assembler. Побитовые операции

- `and` – побитовое «И»
 - `and rax, rbx`
- `or` – побитовое «ИЛИ»
 - `or bx, dx`
- `xor` – исключающее «ИЛИ»
 - `xor eax, eax` ; обнуление значения `eax`
- `not` – побитовое отрицание
 - `not si`

Assembler. Побитовые сдвиги

Простой побитовый сдвиг

- shr – простой побитовый сдвиг вправо
 - shr eax, 2
- shl – простой побитовый сдвиг влево
 - shl eax, 8

Арифметический сдвиг

- sal – арифметический сдвиг влево
- sar – арифметический сдвиг вправо

Assembler. Побитовые сдвиги

Циклический сдвиг

- ror (циклический сдвиг вправо)
 - ror `el`, 1
- rol (циклический сдвиг влево)
 - rol `eh`, 2

И некоторые другие сдвиги

Пример кода NASM

```
section .data

str1    db  'Here is string 1', 0xA

str1_len    equ $ - str1

pi        dw  0x123d

ksi       dd  0x12345678

ksi2      dd  'acde'

; -----
section .bss

mem       resb    12800

; -----
section .text

global _start

_start:
    mov eax, 4
    mov ebx, 1
    mov ecx, str1
    mov edx, str1_len
    int 80h

    mov ecx, str1_len

.loop:
    mov esi, ecx
    mov al, byte [str1+esi]
    mov byte [mem+esi], al
    loop .loop

    mov eax, 4
    mov ebx, 1
    mov ecx, mem
    mov edx, str1_len
    int 80h

    push 1234abc1h
    call print_hex

    mov eax, 1
    mov ebx, 0
    int 80h
```