МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

#### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ

БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет автоматики и вычислительной техники

Кафедра радиоэлектронных средств

Отчет по дисциплине

«Цифровые устройства и микропроцессоры»

Лабораторная работа №2

«ПРИНЦИПЫ ВЫПОЛНЕНИЯ КОМАНД ВЕТВЛЕНИЯ, ОРГАНИЗАЦИЯ

ЦИКЛОВ И ПОДПРОГРАММ»

Вариант №12

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИНБс-3301-01-00 | \_\_\_\_\_\_\_\_\_\_\_\_ | П.С. Кривошеин |
| Проверил: доцент кафедры РЭС | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | М. А. Земцов |

#### Киров 2025

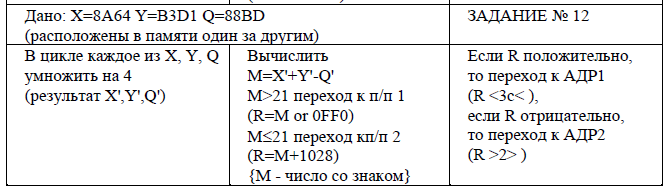
**Цель работы:** изучение принципов выполнения команд ветвления,

организации циклов и подпрограмм микропроцессоров с архитектурой x86.

**Ход работы:**

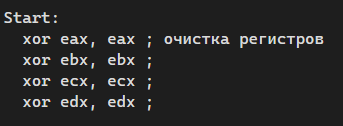
Исходные данные представлены в таблице 1.

Таблица 1 – Исходные данные



**1.** **В цикле каждое из X, Y, Q умножить на 4 (результат X’, Y’, Z’)**

Для начала работы необходимо очистить регистры, для упрощения работы.



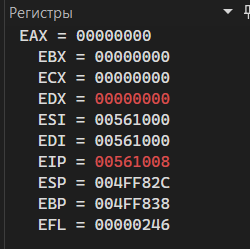


Рисунок 1 – Очистка регистров

Введем значения в регистры для последующего использования.

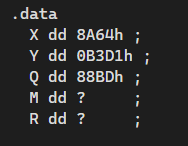


Рисунок 2 – Заданные значения

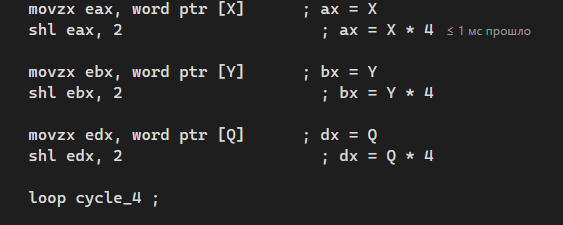


Рисунок 3 – Выполнение задания цикла

**Ход работы:**

1. С помощью команды **mov** загружаем значение X в EAX

8A6416 = 3546810



Рисунок 4 – Регистр EAX

1. С помощью команды **shl** выполняем умножение числа на 4 (сдвиг числа влево на 2 бита)

8A6416 \* 4 = 2299016 = 14171210



Рисунок 5 – Регистр EAX после выполнения команд

1. С помощью команды **mov** загружаем значение Y в EBX

B3D116 = 4603310



Рисунок 6– Регистр EBX

1. С помощью команды **shl** выполняем умножение числа на 4 (сдвиг числа влево на 2 бита)

B3D116 \* 4 = 2CF4416 = 18413210



Рисунок 7 – Регистр EBX после выполнения команд

1. С помощью команды **mov** загружаем значение Q в EDX

88BD16 = 3500510



Рисунок 8– Регистр EDX

1. С помощью команды **shl** выполняем умножение числа на 4 (сдвиг числа влево на 2 бита)

88BD16 \* 4 = 222F416 = 14002010



Рисунок 9 – Регистр EDX после выполнения команд

**2. Вычислить M = X' + Y' - Q'**

Если М > 21 переход к п/п 1 (R = M or 0FF0),

Если М < 21 переход к п/п 2 (R = M + 1028), где М – число со знаком

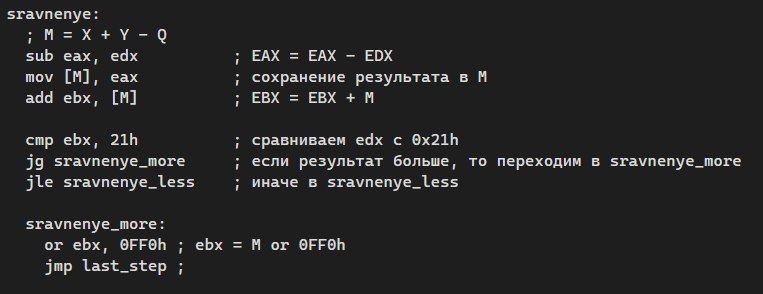


Рисунок 10 – Выполнение вычисления выражения

**Ход работы:**

1. С помощью команды **sub** выполняем вычитание 2 чисел X' и Q'

2299016 – 222F416 = 69C16

14171210 – 14002010 = 169210

|  |  |
| --- | --- |
| Число X'    Число Q' | Результат вычитания |

Рисунок 11 – Выполнение вычитания X' – Q'

1. С помощью команды **mov** загружаем значение EAX в M

M = 169210 = 69C16



Рисунок 12 – Сохранение результата вычитания в М

1. С помощью команды **add** выполняем сложение 2 чисел M и Y' (иначе (X' – Q') + Y’)

69C16 + 2CF4416 = 2D5E016

169210 + 18413210 = 18582410

|  |  |
| --- | --- |
| Число М    Число Y' | Результат сложения |

Рисунок 12 – Выполнение сложения M + Y'

1. Сравниваем с помощью команды **cmp** 2 числа, а именно значение в регистре *ebx* = M и число 2116 = 3310
2. Если M> 21, то с помощью команды **jg (больше)** переходим в **sravnenye\_more**

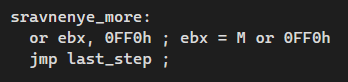


Рисунок 13 – Выполнение условия в случае, если число больше заданного

5.1) С помощью команды **or (дизъюнкция)** выполняем действие 2 чисел ebx = M и числа 0FF016 = 408010

2D5E016 = 0010 1101 0101 1110 0000

0FF016 = 1111 1111 0000

2D5E016 **v** 0FF016 = 2DFF0

0010 1101 0101 1110 0000

+ 1111 1111 0000

0010 1101 1111 1111 0000 = 2DFF0



Рисунок 14 – Результат выполнения действия

5.2) C помощью команды **jmp (переносит управление программой в указанное место в коде)** выполняем другое действие**.**

1. Если М < 21, то с помощью команды **jle (меньше равно)** переходим в **sravnenye\_less**

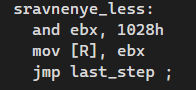


Рисунок 15 – Выполнение условия в случае, если число меньше заданного

6.1) С помощью команды **add** выполняем сложение 2 чисел ebx = M и 102816 = 413610

2D5E016 + 102816 = 2E60816

18582410 + 413610 = 185996010

6.2) С помощью команды **mov** загружаем значение EBX в R

6.3) C помощью команды **jmp (переносит управление программой в указанное место в коде)** выполняем другое действие

**3.** Если R положительно, то переход к АДР1 (R <3c< ),

Если R отрицательно, то переход к АДР2 (R >2 > ).

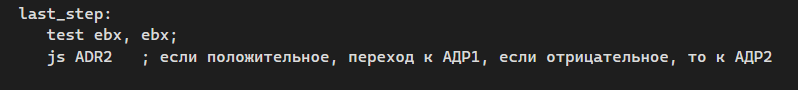


Рисунок 16 – Выполнение вычисления выражения

**Ход работы:**

1. С помощью команды **test (проверка)** проверяем знак EBX (устанавливаем флаги)
2. С помощью команды **js (результат отрицательный)** проверяем EBX = R.
3. Если R положительный (EBX = R), то переходим к ADR1.

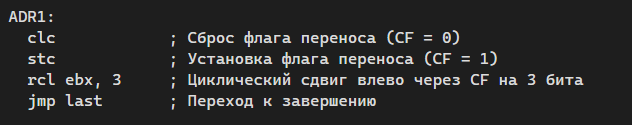


Рисунок 17 – Выполнение условия в случае, если число положительное

3.1) С помощью команды **clc (CLear CF)** делаем сброс флага переноса (CF = 0)

3.2) С помощью команды **stc (SeT Carry flag)** делаем установку флага переноса (CF = 1)

3.3) С помощью команды **rcl (циклический сдвиг влево через CF)** выполняем сдвиг влево на 3 бита

EBX = 002FF016 = 0010 1101 1111 1111 00002

1 сдвиг: 0101 1011 1111 1110 00012

2 сдвиг: 1011 0111 1111 1100 00102

3 сдвиг: 0001 0110 1111 1111 1000 01002

0001 0110 1111 1111 1000 01002 = 16FF8416



Рисунок 18 – Результат выполнения

3.4) С помощью команды **jmp** выполняем переход к завершению программы.

1. Если R отрицательное (EBX = R), то переходим к ADR2.

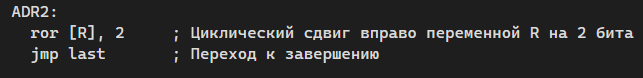


Рисунок 19 – Выполнение условия в случае, если число отрицательное

4.1) С помощью команды **ror (циклический сдвиг вправо)** выполняем сдвиг вправо на 2 бита

EBX = 002FF016 = 0010 1101 1111 1111 00002

1 сдвиг: 0001 0110 1111 1111 10002

2 сдвиг: 1011 0111 1111 11002

1011 0111 1111 11002 = B7FC16

4.2) С помощью команды **jmp** выполняем переход к завершению программы.

Вывод:

В данной работе я изучил принципы выполнения команд ветвления и организации циклов и подпрограмм микропроцессоров с архитектурой х86 в MASM.

* .686 – данная директива указывает ассемблеру использовать набор команд процессора Pentium Pro или Pentium II.
* .model flat - это несегментированная модель памяти, используемая в 32-разрядных операционных системах. В этой модели все данные и код программы размещаются в одном логическом адресном пространстве, что упрощает адресацию памяти. Сегментные регистры не используются для адресации данных, а вместо них используются 32-разрядные смещения
* .model stdcall - это соглашение о вызовах процедур, которое определяет порядок передачи параметров и очистки стека. При использовании stdcall, параметры передаются через стек в обратном порядке, а очистка стека производится вызываемой процедурой
* .stack100h – данная директива определяет размер стека программы. В данном случае, стек будет иметь размер 256 байт (100h — это шестнадцатеричное представление 256 в десятичной системе)
* ExitProcess PROTO STDCALL: DWORD:
  + ExitProcess:
    - Это имя функции из Windows API, которая завершает выполнение текущего процесса.
    - Она принимает один параметр — код завершения процесса (тип DWORD).
  + PROTO:
    - Ключевое слово MASM для объявления прототипа функции.
    - Прототип позволяет ассемблеру знать, как вызывать функцию, какие параметры она принимает и в каком порядке.
  + STDCALL:
    - Указывает соглашение о вызове функции.
    - В соглашении stdcall параметры передаются через стек в обратном порядке (сначала последний параметр), а очистка стека выполняется вызываемой функцией.
    - Это стандартное соглашение для большинства функций Windows API.
  + DWORD:
    - Описывает тип параметра функции. В данном случае функция принимает один параметр типа DWORD (32-битное беззнаковое целое число).
* API (Application Programming Interface) — это набор способов и правил, по которым различные программы общаются между собой и обмениваются данными.

**Код программы:**

.686

.model flat, stdcall

.stack 100h

.data

X dd 8A64h ;

Y dd 0B3D1h ;

Q dd 88BDh ;

M dd ? ;

R dd ? ;

.code

ExitProcess PROTO STDCALL :DWORD

Start:

xor eax, eax ; очистка регистров

xor ebx, ebx ;

xor ecx, ecx ;

xor edx, edx ;

mov cx, 4 ; loop 4 times

cycle\_4:

movzx eax, word ptr [X] ; ax = X

shl eax, 2 ; ax = X \* 4

movzx ebx, word ptr [Y] ; bx = Y

shl ebx, 2 ; bx = Y \* 4

movzx edx, word ptr [Q] ; dx = Q

shl edx, 2 ; dx = Q \* 4

loop cycle\_4 ;

sravnenye:

; M = X + Y - Q

sub eax, edx ; EAX = EAX - EDX

mov [M], eax ; сохранение результата в M

add ebx, [M] ; EBX = EBX + M

cmp ebx, 21h ; сравниваем edx с 0x21h

jg sravnenye\_more ; если результат больше, то переходим в sravnenye\_more

jle sravnenye\_less ; иначе в sravnenye\_less

sravnenye\_more:

or ebx, 0FF0h ; ebx = M or 0FF0h

jmp last\_step ;

sravnenye\_less:

and ebx, 1028h

mov [R], ebx

jmp last\_step ;

last\_step:

test ebx, ebx;

js ADR2 ; если положительное, переход к АДР1, если отрицательное, то к АДР2

ADR1:

clc ; Сброс флага переноса (CF = 0)

stc ; Установка флага переноса (CF = 1)

rcl ebx, 3 ; Циклический сдвиг влево через CF на 3 бита

jmp last ; Переход к завершению

ADR2:

ror [R], 2 ; Циклический сдвиг вправо переменной R на 2 бита

jmp last ; Переход к завершению

last:

Invoke ExitProcess, 0

End Start