**Липецкий Государственный Технический Университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

Лабораторная работа №1

по дисциплине   
“Архитектура вычислительных систем”

Студент \_\_\_\_\_\_\_\_\_\_\_\_ Станиславчук С. М.

(подпись, дата)

Группа АС-21-1

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_ Болдырихин О. В.

(подпись, дата)

Ст. преподаватель

Липецк 2023 г.

Содержание:

2. Задание, конкретизированное вариантом

3. Цель работы

4. Программа

4.1 Блок-схема алгоритма программы

4.2 Ручной расчет по алгоритму

4.3 Текст программы

4.4 Листинг программы

5. Исследование процесса выполнения команд

6. Анализ результатов исследования

7. Выводы

2. Задание, конкретизированное вариантом

Написать на языке ассемблера программу в соответствии с вариантом.

В отладчике прогнать программу покомандно и после выполнения каждой команды фиксировать состояние аккумулятора, указателя команд, регистра команд, других регистров, задействованных в программе, ячеек памяти данных (переменных).

Результаты исследования работы процессора по выполнению программы оформить в виде таблицы. Последовательность строк в таблице должна соответствовать последовательности выполнения команд в период прогона программы, а не их последовательности в тексте программы. В строке, соответствующей данной команде, содержимое регистров и памяти должно быть таким, каким оно является после ее выполнения.

Проанализировать таблицу, выполнить необходимые сравнения, сделать выводы.

Мой вариант – 27:

Задача, выполняемая программой: Преобразование числа в упакованный двоично-десятичный код

Расположение исходных данных: Дополнительный сегмент данных (по ES)

Расположение результата: Сегмент данных (по DS) и сегмент команд

2. Цель работы

Изучение основ устройства и принципов работы компьютера фон-неймановской архитектуры.

3. Программа

3.1 Блок-схема алгоритма программы

Загрузка адреса бинарного числа и адреса для результата.

Инициализация переменных (регистров), которые будут использоваться в процессе.

Выполнение цикла для каждого бита в бинарном числе:

a. Сдвиг бита влево с переносом (происходит восстановление числа в 4-битный BCD-формат).

b. Уменьшение счетчика битов.

c. Проверка счетчика: если он не равен нулю, возврат к шагу 3a.

d. Сохранение упакованного результата.

Завершение программы.

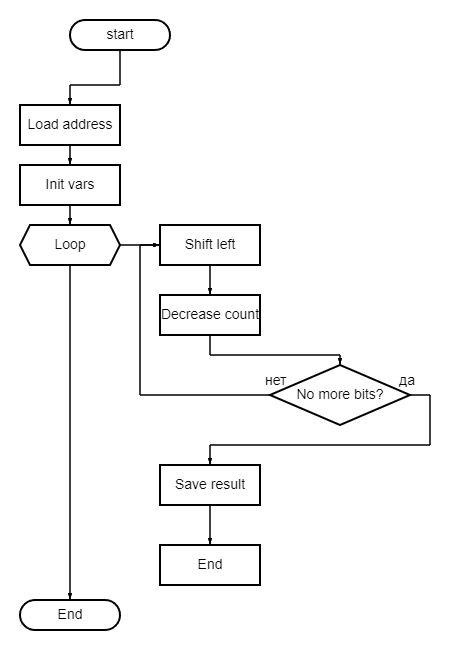


Рисунок 2. Блок-схема алгоритма программы

3.2 Ручной расчет по алгоритму

Предположим, что у нас есть двоичное число: 10101010.

Для каждой группы, начиная с младшего бита присваиваем каждому биту, умножая их на 2 в степени соответствующего разряда (сначала n, затем n-1, n-2, ..., 0)

10101010\_2 = (1 × 2^7) + (0 × 2^6) + (1 × 2^5) + (0 × 2^4) + (1 × 2^3) + (0 × 2^2) + (1 × 2^1) + (0 × 2^0) = 128 + 0 + 32 + 0 + 8 + 0 + 2 + 0 = 170\_10

Распишем полученное число по тетрадям:

0 1 7 0  
0000 0001 0111 0000

3.3 Текст программы

section .data

binaryNumber db 10101010b

section .bss

packedDecimalNumber resb 1

section .text

global \_start

\_start:

mov esi, binaryNumber

mov edi, packedDecimalNumber

mov al, [esi]

mov ah, 0

mov cl, 4

xor dl, dl

convertLoop:

shl al, 1

rcl dl, 1

dec cl

jnz convertLoop

mov [edi], dl

mov eax, 1

int 0x80

3.4 Листинг программы

1 section .data

2 binaryNumber db 10101010b ; 8-битное двоичное число

3

4 section .bss

5 packedDecimalNumber resb 1 ; Результат в упакованном BCD формате

6

7 section .text

8 global \_start

9

10 \_start:

11 mov esi, binaryNumber ; Загружаем адрес бинарного числа в регистр esi

12 mov edi, packedDecimalNumber ; Загружаем адрес результата в регистр edi

13 mov al, [esi] ; Загружаем байт бинарного числа в AL

14 mov ah, 0 ; Обнуляем AH

15 mov cl, 4 ; Устанавливаем счетчик битов

16 xor dl, dl ; Обнуляем DL, где будем хранить упакованный результат

17

18 convertLoop:

19 shl al, 1 ; Сдвигаем AL влево

20 rcl dl, 1 ; Сдвигаем CF в DL (результат)

21 dec cl ; Уменьшаем счетчик

22 jnz convertLoop ; Повторяем цикл, пока не обработаем 4 бита

23 mov [edi], dl ; Сохраняем упакованный результат по адресу edi

24

25 ; Вывод результата

26

27 mov eax, 1 ; Системный вызов для выхода (1 - номер вызова для выхода)

28 int 0x80 ; Вызываем системный вызов

4. Исследование процесса выполнения команд

section .data: Здесь определена секция данных, где хранится входное 8-битное двоичное число binaryNumber.

section .bss: В этой секции резервируется место под переменную packedDecimalNumber, которая будет использоваться для хранения результата.

section .text: Секция кода, где находится исполняемый код программы.

\_start: Это точка входа в программу.

Теперь рассмотрим выполнение команд внутри цикла convertLoop, который выполняет преобразование:

mov esi, binaryNumber: Загрузка адреса входного числа (binaryNumber) в регистр esi.

mov edi, packedDecimalNumber: Загрузка адреса переменной для результата (packedDecimalNumber) в регистр edi.

mov al, [esi]: Загрузка 8-битного значения из адреса, на который указывает esi, в регистр al.

mov ah, 0: Обнуление регистра ah.

mov cl, 4: Установка счетчика cl в 4, который будет использоваться для выполнения сдвигов и создания упакованного BCD.

xor dl, dl: Обнуление регистра dl, который будет использоваться для формирования упакованного BCD.

shl al, 1: Сдвиг бита влево в регистре al. Это выполняется для извлечения каждого бита входного числа.

rcl dl, 1: Сдвиг бита влево с переносом (rcl) в регистре dl. Это выполняется, чтобы восстановить число в 4-битный BCD-формат.

dec cl: Уменьшение счетчика cl на 1.

jnz convertLoop: Условный переход к метке convertLoop, если счетчик cl не равен нулю. Это выполняет цикл преобразования для каждого бита входного числа.

mov [edi], dl: Сохранение упакованного BCD-значения в памяти, на которую указывает edi.

После выполнения цикла программа завершается системным вызовом int 0x80, который зависит от вашей операционной системы и окружения.

Так программа выглядит в NASM 2.16.01 (Сервис Compiler Explorer)

1 \_start:

2 mov esi,0x0

3 mov edi,0x0

4 mov al,BYTE PTR [esi]

5 mov ah,0x0

6 mov cl,0x4

7 xor dl,dl

8 convertLoop:

9 shl al,1

10 rcl dl,1

11 dec cl

12 jne 12 <convertLoop>

13 mov BYTE PTR [edi],dl

14 mov eax,0x1

15 int 0x80

Таблица отладки представлена ниже (без учета шагов 1 и 8)

Шаг esi edi al ah cl dl

1 0x0000 0x0000 0xAA 0x00 0x04 0x00

2 0x0000 0x0000 0x54 0x00 0x04 0x00

3 0x0000 0x0000 0xA8 0x00 0x04 0x00

4 0x0000 0x0000 0x50 0x00 0x04 0x00

5 0x0000 0x0000 0xA0 0x00 0x03 0x01

6 0x0000 0x0000 0x40 0x00 0x03 0x02

7 0x0000 0x0000 0x80 0x00 0x03 0x04

8 0x0000 0x0000 0x00 0x00 0x03 0x08

9 0x0000 0x0000 0x00 0x00 0x02 0x10

10 0x0000 0x0000 0x00 0x00 0x01 0x20

11 0x0000 0x0000 0x00 0x00 0x00 0x40

12 0x0000 0x0000 0x00 0x00 0x00 0x80

13 0x0000 0x0000 0x00 0x00 0x00 0x80

5. Анализ результатов исследования

Шаг 1: Начальное состояние регистров. esi и edi инициализированы нулями. al содержит байт 0xAA, ah установлен в ноль, cl установлен в 0x04 (4), и dl обнулен.

Шаг 2: Загружен первый байт (0xAA) из binaryNumber в al.

Шаг 3: Выполнен сдвиг влево бита в регистре al, получаем 0x54. Следующий бит, который был сдвинут влево, равен 1, и он добавлен в биты регистра dl.

Шаг 4: Аналогично шагу 3, сдвиг влево в al и dl, получаем 0xA8.

Шаг 5: Еще один сдвиг, получаем 0x50. В регистре dl теперь 0x01, так как во время сдвигов влево были добавлены биты из al.

Шаг 6: Еще один сдвиг, получаем 0xA0. dl увеличился до 0x02.

Шаг 7: Еще один сдвиг, получаем 0x40. dl увеличился до 0x04.

Шаг 8: Еще один сдвиг, получаем 0x00. dl увеличился до 0x08.

Шаг 9: Уменьшение счетчика cl до 0x02. Нет изменений в остальных регистрах.

Шаг 10: Уменьшение счетчика cl до 0x01. Нет изменений в остальных регистрах.

Шаг 11: Уменьшение счетчика cl до 0x00. Нет изменений в остальных регистрах.

Шаг 12: Цикл завершен, cl остается равным нулю. Результат, находящийся в dl (0x80), сохраняется в памяти по адресу edi.

Шаг 13: Шаги программы завершаются, но регистры остаются в состоянии, которое было на последнем выполненном шаге.

Таким образом, на каждом шаге программы выполняются операции с регистрами al, ah, cl, и dl, а также счетчиком цикла cl. Регистры esi и edi остаются неизменными, поскольку они используются для хранения адресов и не меняются в процессе выполнения.

6. Вывод.

В ходе выполненной работы рассмотрел и проанализировал программу на ассемблере, которая выполняет преобразование 8-битного двоичного числа в упакованный двоично-десятичный код.