Коми Республикаса велӧдан, наука да том йӧз политика министерство
Министерство образования, науки и молодежной политики Республики Коми
Государственное профессиональное образовательное учреждение
«Сыктывкарский целлюлозно – бумажный техникум»

РАССМОТРЕНО
на заседании ПЦК информационных
дисциплин ГПОУ «СЦБТ»
Протокол № __
«__» _____ г.
Председатель ПЦК _____ В.А. Шулепов
Преподаватель _____ Д.Д. Расов

УТВЕРЖДАЮ
Зам.директора по УПР
ГПОУ «СЦБТ»
_____ Е.В. Соколова
«____» _____ 20__ г.

Методические указания по выполнению лабораторных работ

по учебной дисциплине МДК02.01 «Микропроцессорные системы»
для специальности
09.02.01 Компьютерные системы и комплексы

Программирование микропроцессора i8086 на языке Ассемблер

Сыктывкар 2018

Лабораторная работа № 7

КОМАНДЫ ЛОГИЧЕСКИХ ОПЕРАЦИЙ

Цель работы: *ознакомиться с работой команд логических операций: and, or, xor, test, not и реализацией их работы на практике.*

1. Краткие теоретические сведения.

Логические операции являются важным элементом в проектировании микросхем и имеют много общего в логике программирования. Команды **AND**, **OR**, **XOR** и **TEST** - являются командами логических операций. Эти команды используются для сброса и установки бит и для арифметических операций в коде ASCII. Все эти команды обрабатывают один байт или одно слово в регистре или в памяти, и устанавливают флаги **CF**, **OF**, **PF**, **SF**, **ZF**.

1.1. Команда AND.

Команда **AND** (Логическое И) осуществляет логическое (побитовое) умножение первого операнда на второй. Исходное значение первого операнда (приемника) теряется, замещаясь результатом умножения. В качестве первого операнда команды and можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме сегментного), ячейку памяти или непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды могут быть байтами или словами. Команда воздействует на флаги **SF**, **ZF** и **PF**.

Правила побитового умножения:

Первый операнд-бит 0101	Бит результата 0001
Второй операнд-бит 0011	

Пример 1

mov AX,0FFEh

and AX,5555h ; AX=0554h

Пример 2

mov ax,00101001b

add ax,11110111b ; AX=00100001b

1.2. Команда OR.

Команда **OR** (Логическое ВКЛЮЧАЮЩЕЕ ИЛИ) выполняет операцию логического (побитового) сложения двух операндов. Результат замещает первый операнд (приемник); второй операнд (источник) не изменяется. В качестве первого операнда можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме сегментного), ячейку памяти или непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды команды **OR** могут быть байтами или словами. Команда воздействует на флаги **OF**, **SF**, **ZF**, **PF** и **CF**, при этом флаги **CF** и **OF** всегда сбрасываются в 0.

Правила побитового сложения:

Первый операнд-бит 0101	Бит результата 0111
Второй операнд-бит 0011	

Пример 1

mov AX,000Fh

mov BX,00F0h

or AX,BX ; AX=00FFh, BX=00F0h

Пример 2

mov AX,00101001b

mov BX,11110111b

or AX,BX ; mov dx,11111111b

Пример 3

mov AX,000Fh

or AX,8001h ; AX=800Fh

1.3. Команда XOR.

Команда **XOR** (Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ) выполняет операцию логического (побитового) ИСКЛЮЧАЮЩЕГО ИЛИ над своими двумя операндами. Результат операции замещает первый операнд; второй операнд не изменяется. Каждый бит результата устанавливается в 1, если соответствующие биты операндов различны, и сбрасывается в 0, если соответствующие биты операндов совпадают.

В качестве первого операнда команды **XOR** можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме сегментного), ячейку памяти или непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды могут быть байтами или словами. Команда воздействует на флаги **OF**, **SF**, **ZF**, **PF** и **CF**, причем флаги **OF** и **CF** всегда сбрасываются, а остальные флаги устанавливаются в зависимости от результата.

Правила побитового исключающего или:

Первый операнд-бит 0101	Бит результата 0110
Второй операнд-бит 0011	

Пример 1

mov AX,0Fh

xor AX,0FFFFh ; AX=FFF0h

Пример 2

mov AX,00101001b

mov BX,11110111b

xor ax,bx ; 11011110b

Пример 3

mov SI,0AAAAh

mov BX,5555h

xor SI,BX ; SI=FFFFh,BX=5555h

Пример 4

xor BX, BX ; Обнуление BX

1.4. Команда TEST.

Команда TEST (Логическое сравнение) выполняет операцию логического умножения И над двумя операндами и, в зависимости от результата, устанавливает флаги **SF**, **ZF** и **PF**. Флаги **OF** и **CF** сбрасываются, а **AF** имеет неопределенное значение. Состояние флагов можно затем проанализировать командами условных переходов. *Команда **TEST** не изменяет ни один из операндов.*

В качестве первого операнда команды **TEST** можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме

сегментного), ячейку памяти или непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды могут быть байтами или словами и представлять числа со знаком или без знака.

Правила побитового умножения:

Первый операнд-бит 0101	**Бит результата 0001**
Второй операнд-бит 0011	

Флаг **SF** устанавливается в 1, если в результате выполнения команды образовалось число с установленным знаковым битом.

Флаг **ZF** устанавливается в 1, если в результате выполнения команды образовалось число, состоящее из одних двоичных нулей.

Флаг **PF** устанавливается в 1, если в результате выполнения команды образовалось число с четным количеством двоичных единиц в его битах.

Пример 1

test AX,1

jne label2: ; Переход, если бит 0 в AX установлен

je label1: ; Переход, если бит 0 в AX сброшен

Пример №1.4.1.1

.model tiny	**; модель памяти в которой сегменты кода, данных и стека объединены.**
.code	**; сегмент кода, который содержит данные.**
org 100h	**; начало COM-файла**
begin:	**; метка начала кода программы**
mov CX,<число1 >	**; загружаем в CX число1 <любое число1>**
mov BX,<число2>	**; загружаем в BX число2 <любое число2>**
test cx,bx	**; логически сравниваем числа в регистрах cx с bx**
jne label2	**; если одно из значений не равно 0 то переходим на метку label2**

je label1	**; если одно из значений равно 0 то переходим на метку label1**
ret	**; функция DOS "завершить программу" (не выполняется)**
label1:	**; начало блока метки Label1**
mov ah,9	**; помещаем номер функции DOS "вывод строки (9)" в регистр AH.**
mov dx,offset string	**помещает в регистр DX смещение метки String относительно начала сегмента данных**
int 21h	**; функция DOS "вывод строки"**
ret	**; функция DOS "завершить программу"**
String db 'одно из чисел равно 0$'	**; строка с содержащая выводимые данные.**
label2:	**; начало блока метки Label2**
mov ah,9	**; помещаем номер функции DOS "вывод строки (9)" в регистр AH.**
mov dx,offset string1	**помещает в регистр DX смещение метки String1 относительно начала сегмента данных**
int 21h	**; функция DOS "вывод строки"**
ret	**; функция DOS "завершить программу"**
string1 db 'не равны 0$'	**; строка с содержащая выводимые данные.**
end begin	**; метка окончания кода программы**

Данный пример сравнивает два значения (строка (6)), если одно из двух значений равно нулю тогда переходим на метку label1 (строка (10)) далее выполняются команды, следующие после этой метки, в случае если одно из двух значений равно нулю тогда переходим на метку label2.

Пример 2

test SI,8

jne bityes ; Переход, если бит 3 в SI установлен

je bitno ; Переход, если бит 0 в AX сброшен

Пример 3

test DX,0FFFFh

jz null ; Переход, если DX=0

jnz smth ; Переход, если DX не 0

1.5. Команда NOT.

Команда **NOT** (NOT Инверсия, дополнение до 1, логическое отрицание) выполняет инверсию битов указанного операнда, заменяя 0 на 1 и наоборот. В качестве операнда можно указывать регистр (кроме сегментного) или ячейку памяти размером как в байт, так и в слово. Не допускается использовать в качестве операнда непосредственное значение. Команда не воздействует на флаги процессора.

Правила побитовой инверсии:

Операнд-бит 0 1	Бит результата 1 0

Пример 1

mov AX,0FFFFh

not AX ; AX=0000h

Пример 2

mov SI,5551h

not SI ; SI=AAAEh

2. Характерные примеры работы команд логических операций.

Для следующих несвязанных примеров, предположим, что:

AL содержит **1100 0101**

BH содержит **0101 1100**:

1. **AND AL,BH** ; Устанавливает в AL 0100 0100
2. **OR BH,AL** ; Устанавливает в BH 1101 1101
3. **XOR AL,AL** ; Устанавливает в AL 0000 0000
4. **AND AL,00** ; Устанавливает в AL 0000 0000
5. **AND AL,0FH** ; Устанавливает в AL 0000 0101
6. **OR CL,CL** ; Устанавливает флаги SF и ZF

Примеры **3 и 4 демонстрируют способ очистки регистра**. В примере **5 обнуляются левые четыре бита регистра AL**.

Можно применить команду OR для следующих целей:

1. **OR CX,CX** ; Проверка CX на нуль
   JZ ; Переход, если нуль
2. **OR CX,CX** ; Проверка знака в CX
   JS ; Переход, если отрицательно

3. Задание для выполнения.

3.1. Запустить эмулятор EMU8086. Пользуясь правилами оформления ассемблерных программ, наберите код примера №1.4.1.1, запустите код на выполнение.

3.2. Проанализируйте работу кода примера №1.4.1.1

3.3. С помощью справки эмулятора EMU8086 выясните работу команд (jne, je, js, jz)

3.4. Получите задание у преподавателя (один из четырех вариантов (команды (and, or, xor, test, not))) и, пользуясь правилами оформления ассемблерных программ, напишите три программы характеризующие (показывающие) работу их с числами

(двоичной, десятеричной, шестнадцатеричной систем счисления) согласно перечислению приведенных примеров.

3.5. Результаты работы продемонстрируйте преподавателю.

4. Контрольные вопросы

4.1 Назначение команд логических операций?

4.2 Команда and основное назначение?

4.3 Команда or основное назначение?

4.4 Команда xor основное назначение?

4.5 Команда test основное назначение?

4.6 Команда not основное назначение?

4.7 Альтернативная работа команд (test, xor, and)?

4.8 Допустим что будит

4.9 В чем заключается работа связки команд jne, je?

4.10 В чем заключается работа связки команд js, jz?

4.11 Что произойдет, если в примере №1.4.1.1 мы изменим строку (8) на следующую (jne label1)?