МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФГБОУ ВО «ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ И.С.ТУРГЕНЕВА»

Кафедра «Программная инженерия»

**Отчет**

По лабораторной работе №2  
на тему:  
“Линейное исполнение программ. Арифметические и поразрядные логические операции над целыми двоичными числами  
по дисциплине: “Архитектура ЭВМ и систем”

Работу выполнил:  
Студенты группы 01-ИТ

Колчев Д.Н.  
Галькеев А. Ш.

Проверил:  
Доцент кафедры  
программной инженерии

Конюхова О.В.

Орёл, 2020

1. **Структура программы:**

s\_s segment

b db 20 DUP(?) ; Выделение в сегменте Стека 20 байт

s\_s ends

d\_s1 segment

num1 db 00000100b ; Создание числа в яч. памяти (4)

d\_s1 ends

d\_s2 segment

a dd pt1 ; Создание ячейки с адресом переноса

d\_s2 ends

d\_s3 segment

num2 db 00011000b ; Создание числа в яч. памяти (24)

d\_s3 ends

c\_s1 segment

assume cs:c\_s1

pt1 label far ; Определение типа точки pt1

mov ax, d\_s1

mov ds, ax

assume ds:d\_s1

mov ax,0

mov bx,0

mov al, num1 ; Передача в регистр ‘al’ числа из яч. памяти num1

ror al, 3 ; Циклический сдвиг вправо (bx 0001)

rcl bl,1 ; Циклический сдвиг влево с переносом числа из флага ‘cf’

jmp far ptr pt2 ; Переход в точку pt2 в другой сегмент кода

c\_s1 ends

c\_s2 segment

assume cs:c\_s2

pt2 label far ; Определение типа точки pt2

mov ax, d\_s3

mov ds, ax

assume ds:d\_s3

mov ax, 0

mov bx, 0

mov al, num2 ; Передача в регистр ‘al’ числа из яч. памяти num2

ror al, 5 ; Циклический сдвиг вправо (bx 0100)

rcl bh, 1 ; Циклический сдвиг влево с переносом числа из флага ‘cf’

jmp far ptr exit ; Переход в точку exit в другой сегмент кода

c\_s2 ends

c\_s3 segment

assume cs:c\_s3

begin: ; Начало программы

mov ax, d\_s1

mov ds, ax

assume ds:d\_s1

mov ax,0

mov al, num1 ; Передача в рег ‘al’ число из яч. num1

shl al, 1 ; Логический сдвиг влево (умножение на 2) (al = 8)

mov ax, d\_s3

mov ds, ax

assume ds:d\_s3

mov ax,0

mov al, num2 ; Передача в рег ‘al’ число из яч. num2

shr al, 2 ; Логический сдвиг вправо (деление на 4) (al = 6)

mov ax, d\_s2

mov ds, ax ; Передаём процессоры данные из data-сегмента d\_s2

assume ds:d\_s2

jmp a ; Переход по адресу, записанному в ячейке памяти ‘a’

exit label far ; Определение типа точки exit

mov ah, 4ch

int 21h ; Вызов системного завершения программы

c\_s3 ends

end begin

1. **Контрольные вопросы:**
2. **Внутрисегментные и межсегментные переходы. Способы вычисления адресов переходов.** Командой JMP Модификатор адрес\_перехода задаются внутрисегментные и межсегментные переходы.  
    Команда перехода нарушает естественный порядок выполнения команд посредством загрузки в программный счетчик адреса команды, к которой осуществляется переход. Команды, изменяющие содержимое регистров CS и IP, называются межсегментными переходами, а модифицирующие лишь содержимое IP - внутрисегментными. Межсегментные передачи управления реализуются только командами безусловных переходов.  
    Модификатор указывает вид перехода (внутрисегментный прямой, внутрисегментный косвенный, межсегментный прямой, межсегментный косвенный) и принцип изменения содержимого регистров CS и IP. Модификатор не всегда указывается в команде JMP.  
    Что касается режимов адресации, то во внутрисегментных прямых переходах и межсегментных прямых переходах используется прямой режим. Во внутрисегментных косвенных переходах не допускается непосредственный режим, а в межсегментных косвенных переходах должна адресоваться область памяти.  
    Внутрисегментные переходы реализуются с помощью 8-битного смещения от текущего содержимого IP, 16-битного смещения от IP или получения адреса перехода из ячейки памяти, адресуемой командой. Вычисление 8-битного смещения реализуется так же, как в командах условных переходов, и имеет те же ограничения. Вычисление 16-битного смещения выполняется аналогично, но переход производится в любую точку текущего сегмента. Благодаря игнорированию переноса из 16-го бита текущий сегмент интерпретируется как кольцо.
3. **Разновидности внутрисегментных переходов и их особенности.** Прямой короткий внутрисегментный переход применятся, когда расстояние от команды JMP до адреса перехода находится в диапазоне от -128 байт (адрес перехода расположен до команды JMP в программе) до +127 байт (адрес перехода расположен после команды JMP в программе). В последнем случае для указания короткого перехода в команде JMP используется модификатор SHORT PTR. При выполнении короткого перехода длина команды безусловного перехода составляет два байта.  
    Прямой внутрисегментный переход отличается от предыдущего варианта перехода тем, что расстояние между адресом перехода и командой JMP находится в диапазоне от 128 байт до 64 Кбайт, т.е. переходы между командами могут осуществляться в пределах всего сегмента кода. Для уточнения вида перехода может использоваться  
    модификатор NEAR PTR. При выполнении внутрисегментного прямого перехода длина команды безусловного перехода составляет три байта.  
    В команде косвенного внутрисегментного перехода указывается не сам адрес перехода, а его местоположение, т.е. смещение (эффективный адрес) в сегменте данных. Если адрес ячейки памяти, где хранится адрес перехода, задаётся транслятору через регистр (с помощью команды LEA), то в команде перехода необходимо использовать модификатор WORD PTR для дополнительного сообщения о том, что переход является внутрисегментным.
4. **Флаги процессора и их использование в условиях.** Регистр флагов FLAGS, или слово состояния процессора (PSW – Processor State Word), имеет размер два байта и содержит одноразрядные признаки или флаги.  
    Всего в регистре девять флагов: шесть из них условные, или статусные, отражают результаты операций, выполненных ОУ, остальные три – управляющие, определяют режим исполнения программы.  
    Флаг PF был введён для совместимости с другими микропроцессорными архитектурами и по прямому назначению используется редко. Более распространено его использование совместно с остальными флагами состояния в арифметике с плавающей запятой[3]: инструкции сравнения (FCOM, FCOMP и т. п.) в математическом сопроцессоре устанавливают в нём флаги-условия C0, C1, C2 и C3 и эти флаги можно скопировать в регистр флагов. Для этого рекомендуется использовать инструкцию FSTSW AX для сохранения слова состояния сопроцессора в регистре AX и инструкцию SAHF для последующего копирования содержимого регистра AH в младшие 8 битов регистра флагов, при этом C0 попадает во флаг CF, C2 — в PF, а C3 — в ZF. Флаг C2 устанавливается, например, в случае несравнимых аргументов (NaN или неподдерживаемый формат) в инструкции сравнения FUCOM.
5. **Команды линейного логического и арифметического сдвигов. В чем заключается разница их выполнения? Области применении этих команд.** Команды арифметического линейного сдвига отличаются от команд логического сдвига тем, что они воспринимают сдвигаемые значения как числа со знаком и особым образом работают со знаковым битом (седьмым) числа. Но арифметический сдвиг вправо не помещает слева нули, а дублирует в старшие биты знак операнда.  
    Команды арифметического сдвига позволяют выполнить «быстрое» умножение и деление операнда на степени двойки. Например, сдвиг числа влево на один разряд аналогичен его умножению на 2 , а сдвиг числа вправо на один разряд аналогичен делению его на 2. Преимущество этих команд посравнению с традиционными командами умножения и деления заключается в скорости исполнения: команды сдвига выполняются быстрее.
6. **Особенности выполнения команд циклического сдвига. Сферы применения этих команд.** Команды циклического сдвига отличаются от команд сдвига тем, что операнд считается «кольцом», в котором выдвигаемые с одной стороны биты вдвигаются с другой стороны.  
    В командах простого циклического сдвига сдвигаемый бит одновременно вдвигается в операнд с другого конца и становится значением флага переноса CF.  
    В командах циклического сдвига через перенос сдвигаемый бит сначала помещается во флаг переноса CF. Только при следующем выполнении той же команды находящийся во флаге CF бит вталкивается с другой стороны операнда, а во флаг CF помещается следующий бит сдвигаемого числа.
7. **Что указывает директива ASSUME в программе?** С помощью директивы ASSUME ассемблеру сообщается информация о соответствии между сегментными регистрами, и программными сегментами. Директива имеет следующий формат:

ASSUME <пара>[[, <пара>]]  
ASSUME NOTHING  
  
 где <пара> - это <сегментный регистр> :<имя сегмента>  
 либо <сегментный регистр> :NOTHING  
  
 Например, директива ASSUME ES:A, DS:B, CS:C сообщает ассемблеру, что для сегментирования адресов из сегмента А выбирается регистр ES, для адресов из сегмента В – регистр DS, а для адресов из сегмента С – регистр CS.  
 Таким образом, директива ASSUME дает право не указывать в командах (по крайней мере, в большинстве из них) префиксы – опущенные префиксы будет самостоятельно восстанавливать ассемблер.