Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебная дисциплина «Встраиваемые микропроцессорные системы»

Инструкция

по выполнению лабораторной работы
«Программирование на языке Ассемблер. Составление и выполнение разветвляющихся программ»

Минск
2017

Лабораторная работа № 11

Тема работы: «Программирование на языке Ассемблер. Составление и выполнение разветвляющихся программ»

1. Цель работы:

Получение практических навыков составления и выполнения разветвляющихся программ.

2. Задание

Написать на языке ассемблера и выполнить ветвящуюся программу для вычисления заданного алгебраического выражения. Программа должна работать с двухбайтовыми числами.

3. Оснащение работы

Техническое задание, ПК, эмулятор DOSBox.

4. Основные теоретические сведения

Прежде рассматривались линейные программы, где выполнение процессором команд шло в строгой последовательности: каждая следующая команда выполнялась сразу после команды предыдущей. Так, при рассмотрении программы

```
. . .
MOV   AX,[BaseCount]
ADD   AX,2
. . .

PUSH   AX
. . .
```

была полная уверенность, что команда ADD выполнится непосредственно после команды MOV, а позднее будет выполнена команда PUSH.

Если бы этим все ограничивалось, то программы были бы довольно примитивны. Весьма важно наличие команд, которые могут выполнять в программе переходы (ветвления) на команды, не следующие за текущей командой непосредственно. Столь же важна возможность выполнения в программе условного перехода – в зависимости от состояния процессора или от результата операции.

Набор команд процессора содержит команды для обоих видов переходов. К тому же предусмотрены специальные команды переходов для обеспечения повторяемой обработки определенной части кода.

Безусловные переходы. Основная команда перехода в наборе таких команд – команда JMP. Она указывает, что вслед за ней нужно выполнить команду, указанную меткой. Например, после завершения выполнения фрагмента программы:

```
. . .
MOV   AX,1
JMP   AddTwoToAX
```

```
AddOneToAx:
            INC   AX
            JMP   AXIsSet
AddTwoToAX:
            ADD   AX,2
AXIsSet:
            . . .
```

регистр AX будет содержать значение 3, а команды ADD и JMP, следующие за меткой AddOneToAX, никогда не будут выполняться. Здесь команда

JMP AddTwoToAX

указывает процессору, что нужно установить указатель команд IP в значение смещения метки AddTwoToAX; поэтому следующей выполняемой командой будет команда

ADD AX,2

Иногда совместно с командой JMP используется оператор SHORT. Для указания на целевую метку команда JMP обычно использует 16-битовое смещение. Оператор SHORT указывает ассемблеру, что нужно использовать не 16-битовое, а 8-битовое смещение (что позволяет сэкономить в команде JMP один байт). Например, последний фрагмент программы можно переписать так, что он станет на два байта короче:

```
            . . .
            MOV   AX,1
            JMP   SHORT AddTwoToAX
AddOneToAx:
            INC   AX
            JMP   SHORT AXIsSet
AddTwoToAX:
            ADD   AX,2
AXIsSet:
            . . .
```

Недостаток использования оператора SHORT состоит в том, что короткие переходы могут передавать управление на метки, отстоящие от команды JMP не далее чем на 128 байтов, поэтому иногда ассемблер может сообщать, что метка недостижима с помощью короткого перехода. К тому же оператор SHORT имеет смысл использовать лишь для ссылок вперед, поскольку для переходов назад (на предшествующие метки) ассемблер автоматически использует короткие переходы, если на метку можно перейти с помощью короткого перехода, и длинные - в противном случае.

Команду JMP можно использовать для перехода в другой сегмент кода, загружая в одной команде регистр CS и регистр IP. Например, в программе:

```
            . . .
CSeg1       SEGMENT
            ASSUME   CS:Cseg1
            . . .
FarTarget             LABEL   FAR
            . . .
CSeg1       ENDS
            . . .
CSeg2       SEGMENT
            ASSUME   CS:CSeg2
            . . .
```

```
            JMP   FarTarget  ; переход дальнего типа
            . . .
CSeg2       ENDS
            . . .
```

выполняется переход дальнего типа.

Если нужно, чтобы метка принудительно понималась как метка дальнего типа, можно использовать оператор FAR PTR. Например, во фрагменте программы:

```
            . . .
            JMP   FAR PTR NearLabel
            NOP
NearLabel:
            . . .
```

выполняется переход дальнего типа на метку NearLabel, хотя эта метка находится в том же сегменте кода, что команда JMP.

Наконец, можно выполнить переход по адресу, записанному в регистре или в переменной памяти. Например:

```
            . . .
            MOV   AX,OFFSET TestLabel
            JMP   AX
            . . .
TestLabel:
            . . .
```

Здесь выполняется переход на метку TestLabel, так же как в следующем фрагменте:

```
            . .
            .DATA
JumpTarget        DW   TestLabel
            . .
            .CODE
            . . .
            JMP   [JumpTarget]
            . . .
TestLabel:
            . . .
```

Примечание. Здесь использованы директивы TASM для режима Ideal.

Условные переходы. Описанные команды переходов – лишь часть того, что требуется для написания программ. Необходима возможность писать программы, могущие принимать решения.

Это делается с помощью команд условных переходов.

Команда условного перехода может выполнять или не выполнять переход на целевую метку – в зависимости от состояния регистра признаков (флагов). Рассмотрим пример:

```
            . .
            MOV   AH,1           ; функция DOS ввода с клавиатуры
            INT   21H            ; получить следующую нажатую клавишу
            CMP   AL,'A'         ; была нажата буква "A"?
            JE    AWasTyped      ; да, обработать ее
            MOV   [TampByte], AL ; нет, сохранить символ
            . . .
AWasTyped:
            PUSH   AX            ; сохранить символ в стеке
            . . .
```

Сначала в программе с помощью функции DOS (по прерыванию INT 21H) воспринимается нажатая клавиша. Затем для сравнения введенного символа с символом A используется команда CMP. Признак нуля устанавливается, лишь если регистр AL содержит символ A.

Команда JE есть команда условного перехода, которая выполняет передачу управления, лишь если признак нуля равен 1. Иначе выполняется команда, непосредственно следующая за JE (здесь – команда MOV). Признак нуля будет установлен лишь при нажатии клавиши A, и лишь тогда процессор перейдет к выполнению команды с меткой AWasTyped - команды PUSH.

Набор команд процессора предусматривает разнообразные команды условных переходов, что позволяет выполнить переход почти по любому признаку или комбинации признаков. Можно выполнить условный переход по состоянию нуля, переноса, по знаку, по переполнению и по комбинации признаков, показывающих результаты операций чисел со знаками.

Несмотря на гибкость, команды условного перехода имеют серьезные ограничения, ибо переходы в них всегда короткие: указанная метка должна отстоять от команды перехода не более чем на 128 байтов. Например, невозможно ассемблировать:

```
                . . .
    JumpTarget:
                . . .
                DB   1000 DUP (?)
                . . .
                DEC   AX
                JNZ   JumpTarget
                . . .
```

ибо метка JumpTarget отстоит от команды JNZ более чем на 1000 байтов. В таком случае нужно сделать следующее:

```
                . . .
    JumpTarget:
                . . .
                DB   1000 DUP (?)
                . . .
                DEC   AX
                JNZ   SkipJump
                JMP   JumpTarget
    SkipJump:
                . . .
```

где условный переход применяется, чтобы определить, нужно ли выполнить длинный безусловный переход.

Пример программы для 16-битового разветвленного математического вычисления. Программные тексты вставлены в обрамление.

```
TITLE   ALGBRAN (EXE)                    ; вызов программы
;---------------------------------------
STACKSG SEGMENT PARA STACK 'Stack'
      DW    32 DUP(?)
STACKSG    ENDS
;---------------------------------------
DATASG  SEGMENT PARA 'Data'
```

```
        UVAR    DW (?)                  ; переменная U
        XVAR    DW ???                  ; переменная X
        YVAR    DW ???                  ; переменная Y
        ASCVAL  DB '      ','$'         ; шаблон вывода
        AUX     DW (?)                  ; вспомогательная
        XY      DW (?)                  ; для произведения XY
        CON     DW 1                    ; константа-единица
        EXIT    DW (?)                  ; выход по любой клавише
DATASG  ENDS
;---------------------------------------
CODESG  SEGMENT PARA 'Code'
BEGIN   PROC    FAR
        ASSUME  CS:CODESG, DS:DATASG, SS:STACKSG
        PUSH    DS
        XOR     AX,AX                   ; записать в стек
        PUSH    AX                      ;    нулевой адрес
        MOV     AX,DATASG               ; поместить адрес
        MOV     DS,AX                   ;    DATASG в регистр DS
        LEA     SI,ASCVAL
;---------------------------------------
; Программа вычисления выражения
;
; U = (X + Y)^2 - XY,  если XY > 0
; U = (X + Y)^2 + XY,  если XY < 0
; U = (X + Y)^2 + 1,   если XY = 0
;
;---------------------------------------
; вычисление квадрата суммы X и Y
;---------------------------------------
        MOV     AX,XVAR                 ; вычисление X+Y
        ADD     AX,YVAR
        MOV     AUX,AX                  ; запоминание суммы
        IMUL    AUX                     ; возведение в квадрат
        MOV     AUX,AX                  ; запоминание результата
;---------------------------------------
; вычисление произведения X и Y
;---------------------------------------
        MOV     AX,XVAR                 ; вычисление XY
        MOV     XY,AX
        MOV     AX,YVAR
        IMUL    XY
        MOV     XY,AX                   ; запоминание XY
;---------------------------------------
; проверка значения произведения
;---------------------------------------
        JZ      ZERO
        JS      MINUS
        JMP     PLUS
;---------------------------------------
; вычисление результата
;---------------------------------------
MINUS:
        MOV     AX,AUX                  ; восстановление значения
        ADD     AX,XY                   ; сложение с произведением
        JMP     RES                     ; переход к завершению
PLUS:
        MOV     AX,AUX                  ; восстановление значения
        SUB     AX,XY                   ; вычитание произведения
        JMP     RES                     ; переход к завершению
```

```
ZERO:
        MOV     AX,AUX                  ; восстановление значения
        ADD     AX,CON                  ; сложение с константой
RES:                                    ; завершение вычисления
        MOV     UVAR,AX                 ; запоминание результата
;---------------------------------------
; вызов подпрограммы вывода результата
;---------------------------------------
        CALL    SCRLIN                  ; вызов подпрограммы
        RET                             ; завершение программы
BEGIN   ENDP
;
;***************************************
SCRLIN  PROC    NEAR
;---------------------------------------
; очистка экрана
;---------------------------------------
        MOV     AX,0600H
        MOV     BH,07
        MOV     CX,0000
        MOV     DX,184FH
        INT     10H
;---------------------------------------
; установка курсора
;---------------------------------------
        MOV     AH,02
        MOV     BH,00
        MOV     DX,0000
        INT     10H
;---------------------------------------
; преобразование BIN в ASCII
;---------------------------------------
        MOV     CX,10
        MOV     AX,UVAR
      CMP   AX,0000
        JNS     L10
        MOV     BH,2DH
        MOV     [SI],BH
        NEG     AX
L10:    ADD     SI,5
L15:    CMP     AX,10
        JB      L20
        XOR     DX,DX
        DIV     CX
        OR      DL,30H
        MOV     [SI],DL
        DEC     SI
        JMP     L15
L20:    OR      AL,30H
        MOV     [SI],AL
;---------------------------------------
; вывод значения ASCVAL на экран
;---------------------------------------
        MOV     AH,09
        LEA     DX,ASCVAL
        INT     21H
;---------------------------------------
; задержка до нажатия клавиши
;---------------------------------------
```

```
        MOV     AH,00
        INT     16H
        RET
SCRLIN  ENDP
;***************************************
CODESG  ENDS
        END     BEGIN
```

5. Порядок выполнения работы

1. Скопировать из указанного каталога текст обрамления программы EXE_FORM.ASM, изменив ему имя по усмотрению, в свой рабочий каталог.
2. Ввести в этот текст нужные дополнения и изменения.
3. Вставить в текст программы значения нужных переменных по собственному выбору. Если программа содержит команды деления, следует подобрать эти значения так, чтобы деление давало целочисленный результат. Нужно следить, чтобы все исходные и промежуточные значения не превышали пределов двухбайтовых чисел.
4. Ассемблировать программу и убедиться в отсутствии ошибок.
5. Скомпоновать программу.
6. Выполнить полученный EXE-модуль и проверить результат вычислений. Записать результат вычислений в отчет.

Варианты заданий:

01. $(X + Y)^2 - 2*X*Y$, если $X*Y > 0$
    $(X + Y)^2 + X*Y$, если $X*Y < 0$
    $(X + Y) + 1$, если $X*Y = 0$
02. $A*X*Y + (X^2 - Y)^2$, если $X/Y > 0$
    $A*X*Y + (X*Y^2)$, если $X/Y < 0$
    $(X^2 + Y^2)$, если $X = 0$
03. $X^2 + Y^2 + 3*X$, если $X - Y = 0$
    $(X - Y)^2 + 2*X$, если $X - Y > 0$
    $(Y - X)^2 + X$, если $X - Y < 0$
04. $X + Y/2$, если Y четное
    $2*X + Y$, если Y нечетное
    $3*X + 7$, если $Y = 0$
05. $(X - Y)^2 + X + Y$, если $X > Y$
    $(Y - X)^2 - X + Y$, если $X < Y$
    $X + Y + 2$, если $X = Y$
06. $2*X^3 + 3*Y^2$, если $X < Y$
    $3*X^2 + 2*Y^3$, если $X > Y$
    $3*X^3 + 12$, если $X = Y$
07. $X^2 * Y^2$, если $X < 10$ и $Y < 10$
    $X^2 * Y$, если $X < 10$ и $Y > 10$
    $X*Y^2$, если $X > 10$ и $Y < 10$
08. $X^2 + Y^2$, если $10 < X*Y < 20$
    $X + Y$, если $X*Y > 20$ или $X*Y < 10$

09. 3*X*Y, если X < 10 или Y < 10
2*(X + Y), если X > 10 и Y > 10
6*X + 7*Y, иначе
10. $3*X^3 + 3*Y^2$, если X > |Y|
$3*X^2 + 3*Y^3$, если X =< |Y|
11. X + Y + 1, если X не делится на Y нацело
X/Y + 7, иначе
12. $(X + Y)^2$, если X >= 0 и Y >= 0
(|X| + |Y|), иначе
13. $2*X^2 + |Y|$, если Y < 0
$|X| + Y^2$, если X < 0
3*X*Y + |Y|, если X = 0
14. |X + Y| - max(X,Y), если X > Y
|X + Y| - min(X,Y), если X <= Y
15. U = (X + Y) - X*Y, если X + Y > 0
U = (X + Y) + X*Y, если X + Y < 0
U = (X + Y)^2 + 1, если X + Y = 0

6. Форма отчета о работе

Лабораторная работа № ___
Номер учебной группы _____
Фамилия, инициалы учащегося _____
Дата выполнения работы _____
Тема работы: _____
Цель работы: _____
Оснащение работы: _____
Индивидуальное задание на работу.
Указание имен исходного и исполняемого файлов
Результат выполнения работы: _____
_____
_____

Отчет представляется в виде текстового файла. К отчету должны прилагаться файл исходного кода Программы и рабочий исполняемый файл.

7. Контрольные вопросы и задания

1. Назовите три типа команды безусловного перехода.
2. Какой может быть длина перехода в разных типах команды JMP?
3. Содержимое каких регистров модифицируется при выполнении
4. безусловных переходов разных типов?
5. Какова максимальная длина условного перехода?
6. Каким образом может быть указан адрес перехода? Какие флаги могут быть использованы в командах условного перехода после выполнения команды

сложения?

7. Приведите возможные команды условных переходов, если после сравнения беззнаковых чисел D1 и D2 оказалось: а) D1=D2, б) D1≤D2, в) D1>D2.

8. Приведите возможные команды условных переходов, если после сравнения чисел со знаками P1 и P2 оказалось:

а) P1≠P2,

б) P1<P2,

в) P1≥P2.

8. Рекомендуемая литература

Финогенов, К. Г. Основы языка Ассемблера [Текст] / К. Г. Финогенов. – М.: Радио и связь, 2000.

Финогенов, К. Г. Использование языка Ассемблера [Текст]: учеб. пособие для вузов / К.Г. Финогенов. – М.: Горячая линия Телеком, 2004.

Юров, В. И. Assembler [Текст]: учеб. пособие для вузов / В. И. Юров. 2-е изд. – СПб.: Питер, 2007.