Лабораторная работа № 5
Программирование ветвлений

Цель работы: Практическое овладение навыками разработки программного кода на языке Ассемблер. Изучение команд условного и безусловного перехода. Исследование организации переходов.

Задачи: Разработка простой программы использующей операторы передачи управления и приемов программирования арифметических выражений, содержащих разветвления.

Порядок выполнения работы

1. Создать рабочую папку для текстов программ на ассемблере и записать в нее файлы tasm.exe, tlink.exe, rtm.exe и td.exe. из пакета tasm, а также файл с исходным текстом программы на ассемблере, который сохранить с именем prog5.asm.
2. Разработка простой программы использующей операторы передачи управления и приемов программирования арифметических выражений, содержащих разветвления.
3. Создать загрузочный модуль, загрузить его в отладчик и выполнить программу в пошаговом режиме.

Содержание отчета:

1. Цель и задачи работы.
2. Постановка задачи (вариант).
3. Листинг программы.
4. Пояснения к программе.
5. Результат работы программы в 16-тиричном и 10-тичном виде (3 окна TD).
6. Вывод.

Теоретическая часть

Безусловные переходы

JMP(JuMP)

Переход безусловный

```
------------------------------------
            jmp   метка
------------------------------------
```

Назначение:

Используется в программе для организации безусловного перехода как внутри текущего сегмента команд, так и за его пределы. При определенных условиях в защищенном режиме работы команда jmp может использоваться для переключения задач.

Применение:

Команду jmp применяют для осуществления ближних и дальних безусловных переходов без сохранения контекста точки перехода.

Алгоритм работы:

Команда **jmp** в зависимости от типа своего операнда изменяет содержимое либо только одного регистра eip, либо обоих регистров cs и eip:

- если операнд в команде jmp — метка в текущем сегменте команд (a8, 16, 32), то ассемблер формирует машинную команду, операнд которой является значением со знаком, являющимся смещением перехода относительно следующей за **jmp** команды. При этом виде перехода изменяется только регистр eip/ip;
- если операнд в команде jmp — символический идентификатор ячейки памяти (m16, 32, 48), то ассемблер предполагает, что в ней находится адрес, по которому необходимо передать управление. Этот адрес может быть трех видов:
    - значением абсолютного смещения метки перехода относительно начала сегмента кода. Размер этого смещения может быть 16 или 32 бит в зависимости от режима адресации;
    - дальним указателем на метку перехода в реальном и защищенном режимах, содержащим два компонента адреса — сегментный и смещение. Размеры этих компонентов также зависят от установленного режима адресации (use16 или use32). Если текущим режимом является use16, то адрес сегмента и смещение занимают по 16 бит, причем смещение располагается в младшем слове двойного слова, отводимого под этот полный адрес метки перехода. Если текущим режимом является use32, то адрес сегмента и смещение занимают, соответственно, 16 и 32 бит, — в младшем двойном слове находится смещение, в старшем — адрес сегмента;
    - адресом в одном из 16 или 32-разрядных регистров — этот адрес представляет собой абсолютное смещение метки, на которую необходимо передать управление, относительно начала сегмента команд.

Состояние флагов после выполнения команды (за исключением случая переключения задач): выполнение команды не влияет на флаги

Условные переходы

Инструкция условного перехода может осуществлять или нет переход на целевую (указанную в ней) метку, в зависимости от состояния регистра флагов.

CMP (CoMPare operands)

Сравнение операндов

```
------------------------------------
          cmp   операнд1, операнд2
------------------------------------
```

Назначение*:*
Сравнение двух операндов.

Алгоритм работы:

- выполнить вычитание (операнд1-операнд2), при этом ни один из операндов не модифицируется;
- в зависимости от результата установить флаги, операнд 1 и операнд 2 не изменять (то есть результат не запоминать).

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF

Применение:

Данная команда используется для сравнения двух операндов методом вычитания, при этом операнды не изменяются. По результатам выполнения команды устанавливаются флаги. Команда cmp применяется с командами условного перехода и командой установки байта по значению setcc.

```
len     equ     10
...
        cmp     ax,len
        jne     m1      ;переход если (ax)<>len
              jmp     m2      ;переход если (ax)=len
```

JCC/JCXZ/JECXZ(Jump if condition)

(Jump if CX=Zero/ Jump if ECX=Zero)

Переход, если выполнено условие

Переход, если CX/ECX равен нулю

Схема команды:	jcc метка jcxz метка jecxz метка

Назначение: переход внутри текущего сегмента команд в зависимости от некоторого условия.

Алгоритм работы команд (кроме jcxz/jecxz):

Проверка состояния флагов в зависимости от кода операции (оно отражает проверяемое условие):

- если проверяемое условие истинно, то перейти к ячейке, обозначенной операндом;
- если проверяемое условие ложно, то передать управление следующей команде.

Алгоритм работы команды jcxz/jecxz:

Проверка условия равенства нулю содержимого регистра *ecx/cx*:

- если проверяемое условие истинно, то есть содержимое *ecx/cx* равно 0, то перейти к ячейке, обозначенной операндом метка;
- если проверяемое условие ложно, то есть содержимое *ecx/cx* не равно 0, то передать управление следующей за jcxz/jecxz команде программы.

Состояние флагов после выполнения команды:

11	07	06	05	04	03	02	01	00
OF	SF	ZF	0	AF	0	PF	1	CF
?	?	?		r		?		r

Применение (кроме jcxz/jecxz):

Команды условного перехода удобно применять для проверки различных условий, возникающих в ходе выполнения программы. Как известно, многие команды формируют признаки результатов своей работы в регистре eflags/flags. Это обстоятельство и используется

командами условного перехода для работы. Ниже приведены перечень команд условного перехода, анализируемые ими флаги и соответствующие им логические условия перехода.

Таблица 1 - Инструкции условных переходов

Название	**Состояние проверяемых флагов**[1]	**Проверяемые флаги**
JB/JNAE	Перейти, если меньше / перейти, если не больше или равно	CF = 1
JAE/JNB	Перейти, если больше или равно / перейти, если не меньше	CF = 0
JBE/JNA	Перейти, если меньше или равно / перейти, если не больше	CF = 1 или ZF = 1
JA/JNBE	Перейти, если больше / перейти, если не меньше или равно	CF = 0 и ZF = 0
JE/JZ	Перейти, если равно	ZF = 1
JNE/JNZ	Перейти, если не равно	ZF = 0
JL/JNGE	Перейти, если меньше чем / перейти, если не больше чем или равно	SF = OF
JGE/JNL	Перейти, если больше чем или равно /перейти, если не меньше чем	SF = OF
JLE/JNLE	Перейти, если меньше чем или равно / перейти, если не больше, чем	ZF = 1 или SF = OF
JG/JNLE	Перейти, если больше чем / перейти, если не меньше чем или равно	ZF = 0 или SF = OF
JP/JPE	Перейти по четности	PF = 1
JNP/JPO	Перейти по нечетности	PF = 0
JS	Перейти по знаку	SF = 1
JNS	Перейти, если знак не установлен	SF = 0
JC	Перейти при наличии переноса	CF = 1
JNC	Перейти при отсутствии переноса	CF = 0
JO	Перейти по переполнению	OF = 1
JNO	Перейти при отсутствии переполнения	OF = 0

Логические условия "больше" и "меньше" относятся к сравнениям целочисленных значений со знаком, а "выше и "ниже" — к сравнениям целочисленных значений без знака. Если внимательно посмотреть, то у многих команд можно заметить одинаковые значения флагов для

[1] CF - флаг переноса, SF - флаг знака, OF - флаг переполнения, ZF - флаг нуля, PF - флаг четности.

перехода. Это объясняется наличием нескольких ситуаций, которые могут вызвать одинаковое состояние флагов. В этом случае с целью удобства ассемблер допускает несколько различных мнемонических обозначений одной и той же машинной команды условного перехода. Эти команды ассемблера по действию абсолютно равнозначны, так как это одна и та же машинная команда. Изначально в микропроцессоре i8086 команды условного перехода могли осуществлять только короткие переходы в пределах -128...+127 байт, считая от следующей команды. Начиная с микропроцессора i386, эти команды уже могли выполнять любые переходы в пределах текущего сегмента команд. Это стало возможным за счет введения в систему команд микропроцессора дополнительных машинных команд. Для реализации межсегментных переходов необходимо комбинировать команды условного перехода и команду безусловного перехода jmp. При этом можно воспользоваться тем, что практически все команды условного перехода парные, то есть имеют команды, проверяющие обратные условия.

Применение jcxz/jecxz:

Команда	Состояние флагов в eflags/flags	Условие перехода
JCXZ	не влияет	если регистр CX=0
JECXZ	не влияет	если регистр ECX=0

Команду **jcxz/jecxz** удобно использовать со всеми командами, использующими регистр *ecx/cx* для своей работы. Это команды организации цикла и цепочечные команды. Очень важно отметить то, что команда jcxz/jecxz, в отличие от других команд перехода, может выполнять только близкие переходы в пределах -128...+127 байт, считая от следующей команды. Поэтому для нее особенно актуальна проблема передачи управления далее чем в указанном диапазоне. Для этого можно привлечь команду безусловного перехода jmp. Например, команду jcxz/jecxz можно использовать для предварительной проверки счетчика цикла в регистре cx для обхода цикла, если его счетчик нулевой.

Пример1.

```
...
        jcxz    m1      ;обойти цикл, если cx=0
cycl:
;некоторый цикл
        loop    cycl
m1:     ...
```

Варианты

1	$y = y1 + y2;\ y1 = \begin{cases} a + x, & \text{если } x > a \\ 2a - x, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a * x, & \text{если } x > 10 \\ x, & \text{если } x <= 10 \end{cases}.$
2	$y = y1 - y2;\ y1 = \begin{cases} x - 2, & \text{если } x >= 2 \\ 8, & \text{если } x < 2 \end{cases};\ y2 = \begin{cases} 4, & \text{если } x = 0 \\ a - x, & \text{если } x <> 0 \end{cases}.$
3	$y = y1 * y2;\ y1 = \begin{cases} x - a, & \text{если } x > a \\ 5, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a, & \text{если a} > x \\ a * x, & \text{если a} <= x \end{cases}.$
4	$y = y1 + y2;\ y1 = \begin{cases} 2 - x, & \text{если } x < 2 \\ a + 3, & \text{если } x >= 2 \end{cases};\ y2 = \begin{cases} a - 1, & \text{если } x < a \\ a * x - 1, & \text{если } x >= a \end{cases}.$
5	$y = y1 - y2;\ y1 = \begin{cases} \lvert x \rvert, & \text{если } x < 0 \\ x - a, & \text{если } x >= 0 \end{cases};\ y2 = \begin{cases} a + x, & \text{если } x \bmod 3 = 1 \\ 7, & \text{в остальных случаях} \end{cases}.$
6	$y = y1 + y2;\ y1 = \begin{cases} x \bmod 4, & \text{если } x > a \\ a, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a * x, & \text{если } x / a > 3 \\ x, & \text{если } x / a <= 3 \end{cases}.$
7	$y = y1 + y2;\ y1 = \begin{cases} 4 - x, & \text{если } \lvert x \rvert < 3 \\ a + x, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} 2, & \text{если } x \text{ четное} \\ a + 2, & \text{в остальных случаях} \end{cases}.$
8	$y = y1 + y2;\ y1 = \begin{cases} 4 * x, & \text{если } x <= 4 \\ x - a, & \text{если } x > 4 \end{cases};\ y2 = \begin{cases} 7, & \text{если } x \text{ нечетное} \\ x / 2 + a, & \text{в остальных случаях} \end{cases}.$
9	$y = y1 * y2;\ y1 = \begin{cases} a * x, & \text{если } x \bmod 3 = 2 \\ 9, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} a - x, & \text{если } a > x \\ a + 2, & \text{если } a <= x \end{cases}.$
10	$y = y1 - y2;\ y1 = \begin{cases} a + \lvert x \rvert, & \text{если } x > a \\ a - 7, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a * 3, & \text{если } a > 3 \\ 11, & \text{если a} <= 3 \end{cases}.$
11	$y = y1 \bmod y2;\ y1 = \begin{cases} 10 + x, & \text{если } x > 1 \\ \lvert x \rvert + a, & \text{если } x <= 1 \end{cases};\ y2 = \begin{cases} 2, & \text{если } x > 4 \\ x, & \text{если } x <= 4 \end{cases}.$
12	$y = y1 / y2;\ y1 = \begin{cases} 15 + x, & \text{если } x > 7 \\ \lvert a \rvert + 9, & \text{если } x <= -7 \end{cases};\ y2 = \begin{cases} 3, & \text{если x} > 2 \\ \lvert x \rvert - 5, & \text{если } x <= 2 \end{cases}.$
13	$y = y1 * y2;\ y1 = \begin{cases} 3 + x, & \text{если } x = a \\ a - x, & \text{если } x <> a \end{cases};\ y2 = \begin{cases} \lvert a \rvert, & \text{если } a < x \\ \lvert a \rvert - x, & \text{если ```a} >= x \end{cases}.$
14	$y = y1 - y2;\ y1 = \begin{cases} 2 * x + a, & \text{если } x > 2 \\ 2 * x + 1, & \text{если } x <= 2 \end{cases};\ y2 = \begin{cases} \lvert x \rvert + 1, & \text{если x} > 0 \\ a - 1, & \text{если } x <= 0 \end{cases}.$

	$\begin{cases} x-a, & \text{если } x <= 4 \end{cases}$ $\begin{cases} 9, & \text{если } x <= a \end{cases}$
20	$y = y1 * y2;\ y1 = \begin{cases} 2*x, & \text{если } x < 5 \\ \lvert a \rvert + x, & \text{если } x >= 5 \end{cases};$
21	$y = y1 + \lvert y2 \rvert;\ y1 = \begin{cases} 3, & \text{если } x \bmod 3 = 1 \\ x-a, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} a/x, & \text{если } x <> 0 \\ 4, & \text{если } x = 0 \end{cases}.$
22	$y = y1 - y2;\ y1 = \begin{cases} \lvert x \rvert + \lvert a \rvert, & \text{если } x <= 3 \\ x*a, & \text{если } x > 3 \end{cases};\ y2 = \begin{cases} 3, & \text{если } a = x \\ a-x, & \text{если } a <> x \end{cases}.$
23	$y = y1 + y2;\ y1 = \begin{cases} 2*x, & \text{если } \lvert x \rvert > 4 \\ 4+a, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} 9, & \text{если } x = 0 \\ a/x, & \text{если } x <> 0 \end{cases}.$
24	$y = y1 * y2;\ y1 = \begin{cases} x, & \text{если } x \bmod 4 <> 2 \\ a+x, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} a-x, & \text{если } x < a \\ a*x, & \text{если } x >= a \end{cases}.$
25	$y = y1 / y2;\ y1 = \begin{cases} 12, & \text{если } x < 12 \\ x+1, & \text{если } x >= 12 \end{cases};\ y2 = \begin{cases} 2, & \text{если } x > 2 \\ a+x, & \text{если } x <= 2 \end{cases}.$

на №5