Практическая работа №4

Программирование ветвлений

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

ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Команда безусловного перехода

Безусловный переход в программе на ассемблере производится по команде JMP. Полный формат команды следующий:

JMP [модификатор] ***адрес_перехода*****.**

Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода.

В системе команд микропроцессора существуют несколько кодов машинных команд безусловного перехода. Их различия определяются дальностью перехода и способом задания целевого адреса. Дальность перехода определяется местоположением операнда *адрес_перехода*. Этот адрес может находиться в текущем сегменте кода или в некотором другом сегменте. В первом случае переход называется внутрисегментным или близким, а во втором случае – межсегментным или дальним.

Внутрисегментный переход предполагает, что изменяется только содержимое регистра IP. Можно выделить три варианта внутрисегментного перехода: *прямой короткий, прямой, косвенный.*

Прямой короткий внутрисегментный переход применяется, когда расстояние от команды JMP до адреса перехода не более чем 127 байтов выше или ниже. В этом случае транслятор языка формирует машинную команду безусловного перехода длиной 2 байта: первый байт – код операции, второй байт – смещение. В коде операции заложена информация о том, что второй байт интерпретируется как смещение. Здесь нужно отметить одну особенность транслятора ассемблера – он является однопроходным, иными словами, машинный код программы получается за один просмотр команд от начала программы до ее окончания. В связи с этим обстоятельством, если безусловный переход должен происходить на адрес до команды JMP, то транслятор может легко вычислить смещение. Если же переход короткий, но на метку после команды JMP, то транслятору нужно подсказать, что он должен сформировать команду безусловного короткого перехода. С этой целью в команде JMP используется модификатор SHORT PTR (полностью - SHORT POINTER или короткий указатель):

```
        JMP  SHORT PTR M1
        . . . . . .  не более 35-40 команд
M1:
        MOV AL, 34H.
```

Прямой внутрисегментный переход отличается от короткого тем, что длина машинной команды составляет 3 байта, в которой два последних байта интерпретируются

как смещение. Нетрудно определить, что в этом варианте можно осуществлять переход в пределах 64 Кбайт памяти относительно следующей за JMP команды.

Косвенный внутрисегментный переход означает, что в команде JMP указывается не сам адрес перехода, а место, где этот адрес записан.

Например:

```
LEA BX, M1
JMP BX
. . . . . .
M1:          MOV AL, 34H
```

В командах косвенного перехода рекомендуется применять модификатор NEAR, т.к. при косвенном переходе не всегда транслятору удается определить, находится адрес перехода в текущем сегменте кода или нет.

Команда прямого межсегментного перехода имеет длину 5 байт, из которых 2 байта составляет смещение адреса перехода, а другие 2 байта – значение сегментной составляющей (CS) того кодового сегмента, где находится адрес перехода.

Например:

```
SEG1         SEGMENT  PARA  PUBLIC  'CODE'
ASSUME    CS:SEG1, DS:DSEG1, SS:STACK
. . . . . .. .
JMP FAR PTR M2
. . . . . ..
M1           LABEL FAR
. . . . . ..
SEG1  ENDS
SEG2         SEGMENT  PARA  PUBLIC  'CODE'
ASSUME    CS:SEG2, DS:DSEG2, SS:STACK
. . . . . .. .
M2           LABEL FAR
             JMP M1.
```

Во втором случае FAR необязательно, но если модификатор примените, то ошибки не будет. Необязательность объясняется тем, что метка находится раньше команды перехода и транслятор может самостоятельно определить, что переход является межсегментным.

Команда косвенного межсегментного перехода в качестве операнда имеет адрес области памяти, в которой содержится смещение и сегментная часть целевого адреса перехода.

Например:

```
DSEG         SEGMENT   PARA  PUBLIC  'DATA'
ADDRDD       M1
. . . . . .
CSEG         SEGMENT  PARA  PUBLIC  'CODE'
             ASSUME    CS:CSEG, DS:DSEG, SS:STACK
. . . . . .
             JMP ADDR
CSEG         ENDS
CS1          SEGMENT  PARA  PUBLIC  'CODE'
             ASSUME    CS:CS1, DS:DS1, SS:ST1
M1     LABEL FAR
             MOV          AX, BX
```

```
. . . .. ..
CS1         ENDS.
```

Одним из вариантов рассматриваемого перехода является **косвенный регистровый межсегментный переход**. Адрес перехода в этом варианте указывается в регистре, что удобно при программировании динамических переходов, когда конкретный адрес перехода определяется в процессе выполнения программы и помещается в регистр:

```
DSEG        SEGMENT   PARA  PUBLIC  'DATA'
ADDRDD      M1
. . . . . . . .
CSEG        SEGMENT   PARA  PUBLIC  'CODE'
            ASSUME    CS:CSEG, DS:DSEG, SS:STACK
 . . . . . .
            LEA BX, ADDR
JMP DWORD PTR [BX]
 . . . . . .
CSEG        ENDS
CS1         SEGMENT   PARA  PUBLIC  'CODE'
            ASSUME    CS:CS1, DS:DS1, SS:ST1
M1    LABEL FAR
            MOV         AX, BX
. . . . . . . .
CS1         ENDS.
```

В двойное слово ADDR помещается смещение адреса и начала сегмента кода, включающего метку M1, в нашем случае, начало сегмента CS1.

Т.о. модификаторы SHORT PTR, NEAR PTR и WORD PTR применяют при организации внутрисегментных переходов, а FAR PTR и DWORD PTR – при межсегментных переходах.

Приведѐм фрагмент программы с различными видами команд безусловного перехода, в этом фрагменте описаны два кодовых сегмента (для иллюстрации дальних переходов) и один сегмент данных (для команд косвенного перехода):

```
data segment
   .  . .
A1     dw L2;          Смещение команды с меткой L2  в своѐм сегменте
A2     dd Code1:L1;  Это m32=seg:off
    . . .
data   ends

code1 segment
       . . .
L1:    mov ax,bx
       . . .
code1 ends

code2 segment
       assume cs:code2,
       mov ax,@data
       mov ds,ax;          загрузка сегментного регистра DS
L2:    jmp far ptr L1;  дальний прямой абсолютный переход, op1=i32=seg:off
       jmp L1;             ошибка т.к. без far ptr
       jmp L2;             близкий относительный переход, op1=i8  или i16
       jmp A1;             близкий абсолютный косвенный переход, op1=m16
```

```
        jmp A2;             дальний абсолютный косвенный переход, op1=m32
        jmp bx;             близкий абсолютный косвенный переход, op1=r16
        jmp [bx];           ошибка, нет выбора между: op1=m16 или op1=m32
        mov bx,offset A2
        jmp dword ptr [bx]; дальний абсолютный косвенный переход op1=m32
        . . .
code2 ends
```

Вывод: если в команде перехода задаѐтся метка, то при прямом переходе она описана в программе с двоеточием (это метка команды), а при косвенном – без двоеточия (это метка области памяти). Отметим здесь также одно важное преимущество относительных переходов перед абсолютными переходами. Значение `i8` или `i16` в команде относительного перехода зависит только от рас- стояния в байтах между командой перехода и точкой, в которую производится переход. При любом изменении в сегменте кода *вне* этого диапазона команд значения `i8` или `i16` не меняются. Это полезное свойство относительных переходов позволяет, например, при необходимости достаточно легко *склеивать* два сегмента кода в один, что используется в системной программе редактора внешних связей (эта про- грамма будет изучаться позже).

2. Команды условного перехода

Команды передачи управления реализуют изменение естественного порядка выполнения команд программы. Их можно разделить на 3 подгруппы, описание которых приведено в приложении 1 В мнемонические обозначения команд условного перехода входят буквы, которые определяют условия в соответствии с таблицей 1.

В этой таблице операнд “метка перехода” или “близкая метка” отражает тот факт, что метка помеченной команды должна находиться в пределах текущего сегмента кода и на относительном расстоянии от команды перехода >-128 и < 127 байтов. Ограничение (–128..127) байтов снято у процессоров, начиная с модели 80386, однако ограничение передачи управления в пределах текущего сегментного кода действует и в моделях PENTIUM.

Таблица 1

Мнемокоды команд условного перехода

Буква мнемокода	Условие	Тип операндов
E	равно	любые
N	Не равно	любые
G	больше	Числа со знаком
L	меньше	Числа со знаком
A	Выше, в смысле “больше”	Числа без знака
B	ниже, в смысле “меньше”	Числа без знака

Решение о том, куда будет передано управление командой условного перехода, принимается на основании условия. Источниками таких условий могут быть:

- любая команда, изменяющая состояние арифметических флагов (ниже эти флаги будут перечислены);

- команда сравнения CMP.

Формат команды CMP:

CMP приемник, источник **или** ***CMP операнд1, операнд2.***

Эта команда осуществляет вычитание (операнд1 - операнд2) или (приемник – источник), однако результат никуда не записывается, а только устанавливает флаги в соответствие с таблицей 2 =.

Таблица 2

Значения флагов, устанавливаемые командой CMP

Сравниваемые операнды	Флаги			
	OF	SF	ZF	CF
Операнды без знака Источник < приемник	H	H	0	0
Источник = приемник	H	H	1	0
Источник > приемник	H	H	0	1
Операнды со знаком				
Источник < приемник	0/1	0	0	H
Источник = приемник	0	0	1	H
Источник > приемник	0/1	1	0	H

В этой таблице приняты следующие обозначения:

- “H” означает, что ‘не имеет значения” или иначе, на этот флаг операция не влияет;

- 0/1 означает, что флаг устанавливается или в 1 или в 0 в зависимости от значений операндов (отрицательные или положительные или разнознаковые операнды сравниваются).

Приведем еще одну таблицу 3, в которой отражается действие команд условного перехода по значениям анализируемых этими командами флагов. В этой таблице через слеш ‘/’ перечисляются идентичные команды, действие которых совершенно одинаково, и применение конкретной из них зависит от пристрастий программиста. Наличие идентичных команд объясняется тем фактом, что если число_1 >число_2, то можно с уверенностью утверждать, что число_1 не (меньше или равно) число_2.

Таблица 3

Логика команд условного перехода

Тип операндов	Мнемокод команды	Критерий перехода	Значения флагов для перехода
любые	JE	Операнд_1=операнд_2	ZF=1
Любые	JNE	Операнд_1<>операнд_2	ZF=0
Со знаком	JL/JNGE	Операнд_1<операнд_2	SF<>OF
Со знаком	JLE/JNG	Операнд_1<=операнд_2	SF<>OF или ZF=1
Со знаком	JG/JNLE	Операнд_1>операнд_2	SF=OF и ZF=0
Со знаком	JGE/JNL	Операнд_1>=операнд_2	SF=OF
Без знака	JB/JNAE	Операнд_1<операнд_2	CF=1
Без знака	JBE/JNA	Операнд_1<=операнд_2	CF=1 или ZF=1
Без знака	JA/JNBE	Операнд_1>операнд_2	CF=0 и ZF=0
Без знака	JAE/JNB	Операнд_1=>операнд_2	CF=0

Условный оператор ЕСЛИ_ТО

Условный оператор if-else используется для принятия решения о дальнейшем пути исполнения программы. Синтаксис условного оператора в нотации :

ЕСЛИ выражение ТО
    оператор1;
ИНАЧЕ

оператор2

Алгоритм работы условного оператора – вычисляется логическое значение выражения: если оно истинно, то выполняется оператор1, в противном случае – оператор2.

На ассемблере данные варианты условного оператора можно реализовать следующим образом:

; полный вариант оператора ЕСЛИ выражение ТО оператор1 ИНАЧЕ оператор2

```
cmp op1,op2                ; вычисление выражения
        jne else1
        ; …                ; последовательность команд, соответствующая оператор1
        jmp endif
else1:
        ;…                 ; последовательность команд, соответствующая оператор2
        endif:             ; конец полного условного оператора
        ; …
```

Вместо команды CMP можно использовать любую команду, изменяющую флаг, который будет анализироваться последующим оператором условного перехода. Аналогично и с командой JNE, вместо которой может стоять требуемая в данном вычислительном контексте команда условного перехода.

3. ЗАДАНИЯ

5.1 Для данных ниже команд определить тип перехода (прямой или косвенный), для неверных команд объяснить ошибку. Считать, что выше команды перехода были описаны имена: A — байт; B — двойное слово; K — константа со значением 4000FFFFh; L, M — метки.

а) `JMP L`
б) `JMP A`
в) `JMP BX`
г) `JMP B`
д) `JMP ECX`
е) `JMP M`
ж) `JMP EIP`
з) `JMP K`

5.2 `X DB 206 ; (-50)`

Определить, будет ли сделан переход на метку MET при выполнении следующих команд:

а)
```
CMP X,100
JA MET
```
б)
```
CMP X,100
JG MET
```
в)
```
CMP X,210
JA MET
```
г)
```
CMP X,210
  JE LAB
  JB MET
LAB:
```
д)
```
CMP X,-40
JG MET
```
е)
```
CMP X,-40
JL MET
```
ж)
```
CMP X,216
JL MET
```

5.3 `X DW ? ; число со знаком`

Определить, какие из следующих фрагментов эквивалентны условному оператору

```
if X>80 then X:=X-1 else X:=X+1
```

а)
```
  CMP X,80
  JG M
  INC X
  JMP L
M: DEC X
L:
```
б)
```
  CMP X,80
  JLE M
  DEC X
M: INC X
```
в)
```
  CMP X,80
  JG M
  DEC X
  JMP L
M: INC X
L:
```
г)
```
  CMP X,80
  JLE M
  ADD X,2
M: DEC X
```
д)
```
  CMP X,80
  JBE M
  DEC X
  JMP L
M: INC X
L:
```
е)
```
  CMP X,80
  JLE M
  DEC X
  JMP L
M: INC X
L:
```

5.4 Упростить (записать меньшим числом команд) следующий фрагмент:

а)
```
CMP AH,0
JE M
JG M
NEG AH
M:
```
б)
```
CMP AH,0
JL M1
JMP M
M1: NEG AH
M:
```
в)
```
L: ADD EAX,EAX
DEC EBX
CMP EBX,0
JE M
JMP L
M:
```
г)
```
L: ADD EAX,EAX
DEC ECX
CMP ECX,0
JNE L
```

4.

5.5 Реализовать следующие последовательности операторов (числа со знаком)

а) `if ESI>20 then AL:= 1 else AL:= 0;`

б)
```
AL:=0;
  if ESI>20 then AL:=1
```

Сравнить полученные фрагменты кода.

5.6 Реализовать команды

а) `JECXZ M` б) `LOOP M`

если расстояние от команды перехода до метки `M` больше 128 байтов.

5.7
```
C DB ?
K DB ?
```

Проверить, является ли символ C 16-ричной цифрой. Если является, записать в `K` значение 1, иначе записать 0.

5.8 Пусть `X`, `Y` и `Z` — знаковые байтовые переменные. Реализовать наименьшим числом команд следующее действие:

а) `if (X<2) and (Y<4) and (Z<10) then EAX:=102 else EAX:=-3;`

б) `if (X<2) or (Y<4) or (Z<10) then EAX:=102 else EAX:=-3`

5.9 Пусть `X`, `Y`, `Z` и `W` — знаковые переменные-слова. Реализовать следующее присваивание:

а) `Y:=abs(X)`

б) `Z:=min(X,Y)`

в) `W:=min(X,Y,Z)`

г) `W:=max(X+10,2*Y,8-Z)`

Считать, что все промежуточные результаты умещаются в слово.

5.10 Пусть `X` — четверное слово, `S` — байт. Записать в `S` знак числа `X`:

$$S := \begin{cases} 1, & \text{если } X > 0 \\ 0, & \text{если } X = 0 \\ -1, & \text{если } X < 0 \end{cases}$$

5.11 Переменные x, y — двойные слова, числа со знаком. Записать в байтовую переменную k номер четверти координатной плоскости, в которой лежит точка (x, y).

5.12 Считая, что `Y` — двойное слово, число без знака, определить, является ли год `Y` високосным

а) реализовать полный условный оператор

```
if (Y mod 400 = 0) or (Y mod 4 = 0) and (Y mod 100 <> 0)
   then AL:=1
   else AL:=0;
```

Приложение 1

Формат команд передачи управления

Мнемокод	Формат
Команды безусловной передачи управления JMP	JMP имя
Команды условной передачи управления	
JA / JNBE	JA / JNBE близкая метка
JAE / JNB	JAE / JNB близкая метка
JNC	JNC близкая метка
JB / JNAE	JB / JNAE близкая метка
JC	JC близкая метка
JBE / JNA	JBE / JNA близкая метка
JCXZ	JCXZ близкая метка
JE / JZ	JE / JZ близкая метка
JG / JNLE	JG / JNLE близкая метка
JGE / JNL	JGE / JNL близкая метка
JL / JGNE	JL / JGNE близкая метка
JLE / JNG	JLE / JNG близкая метка
JNE / JNZ	JNE / JNZ близкая метка
JNO	JNO близкая метка
JNP / JPO	JNP / JPO близкая метка
JNS	JNS близкая метка
JO	JO близкая метка
JP / JPE	JP / JPE близкая метка
JS	JS близкая метка