**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Организация ЭВМ и систем»**

Тема: Представление и обработка целых чисел. Организация ветвящихся процессов

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 |  | Ларин А. |
| Преподаватель |  | Ефремов М.А, |

Санкт-Петербург

2019

**Цель работы.**

Научится организовать ветвление в языке ассемблера. Изучить операторы сравнения, условного и безусловного перехода. Научится описывать функции с условием используя ветвление.

**Основные теоретические положения.**

Обычно в программе есть точки, в которых нужно принять решение о том, какая команда будет выполняться следующей. Это решение может быть

* безусловным — в данной точке необходимо передать управление не той команде, которая идет следующей, а другой, которая находится на некотором удалении от текущей команды;
* условным — решение о том, какая команда будет выполняться следующей, принимается на основе анализа некоторых условий или данных.

CMP

Данная команда ассемблера сравнивает содержимое двух полей данных.

Фактически команда CMP вычитает второй операнд из первого, но содержимое полей  не изменяет.  Операнды должны иметь одинаковую длину: байт или слово. Команда CMP может сравнивать  содержимое  регистра,  памяти или непосредственное значение  с  содержимым  регистра; или    содержимое регистра или непосредственное значение с содержимым памяти.

Команда воздействует на такие флаги процессора как AF, CF, OF, PF, SF и ZF.

Безусловные переходы

Предыдущее обсуждение выявило некоторые детали механизма перехода. Команды перехода модифицируют регистр указателя команды eip/ip и, возможно, сегментный регистр кода cs. Что именно должно подвергнуться модификации, зависит:

* от типа операнда в команде безусловного перехода (ближний или дальний);
* от указания перед адресом перехода (в команде перехода) модификатора; при этом сам адрес перехода может находиться либо непосредственно в команде (прямой переход), либо в регистре или ячейке памяти (косвенный переход).

Модификатор может принимать следующие значения:

* near ptr — прямой переход на метку внутри текущего сегмента кода. Модифицируется только регистр eip/ip (в зависимости от заданного типа сегмента кода use16 или use32) на основе указанного в команде адреса (метки) или выражения, использующего символ извлечения значения СчА — $;
* far ptr — прямой переход на метку в другом сегменте кода. Адрес перехода задается в виде непосредственного операнда или адреса (метки) и состоит из 16-битного селектора и 16/32-битного смещения, которые загружаются, соответственно, в регистры cs и ip/eip;
* word ptr — косвенный переход на метку внутри текущего сегмента кода. Модифицируется (значением смещения из памяти по указанному в команде адресу, или из регистра) только eip/ip. Размер смещения 16 или 32 бит;
* dword ptr — косвенный переход на метку в другом сегменте кода. Модифицируются (значением из памяти — и только из памяти, из регистра нельзя) оба регистра, cs и eip/ip. Первое слово/двойное слово этого адреса представляет смещение и загружается в ip/eip; второе/третье слово загружается в cs.

Команда безусловного перехода jmp

Синтаксис команды безусловного перехода   
jmp [модификатор] адрес\_перехода - безусловный переход без сохранения информации о точке возврата.

Адрес\_перехода представляет собой адрес в виде метки либо адрес области памяти, в которой находится указатель перехода.

Всего в системе команд микропроцессора есть несколько кодов машинных команд безусловного перехода jmp.   
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, во втором — межсегментным, или дальним.   
Внутрисегментный переход предполагает, что изменяется только содержимое регистра eip/ip.   
Можно выделить три варианта внутрисегментного использования команды jmp:

* прямой короткий;
* прямой;
* косвенный.

Условные переходы

Микропроцессор имеет 18 команд условного перехода. Эти команды позволяют проверить:

* отношение между операндами со знаком (“больше — меньше”);
* отношение между операндами без знака (“выше — ниже”)2;
* состояния арифметических флагов zf, sf, cf, of, pf (но не af).

Команды условного перехода имеют одинаковый синтаксис:   
jcc метка\_перехода

Как видно из табл. 1, мнемокод всех команд начинается с “j” — от слова jump (прыжок), cc — определяет конкретное условие, анализируемое командой.   
Что касается операнда метка\_перехода, то эта метка может находится только в пределах текущего сегмента кода, межсегментная передача управления в условных переходах не допускается. В связи с этим отпадает вопрос о модификаторе, который присутствовал в синтаксисе команд безусловного перехода.

Таблица 1. Перечень команд условного перехода для команды cmp

|  |  |  |  |
| --- | --- | --- | --- |
| Типы операндов | Мнемокод команды условного перехода | Критерий условного перехода | Значения флагов для осществления 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 or zf = 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| Со знаком | jg/jnle | операнд\_1 > операнд\_2 | sf = of and 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 or zf=1 |
| Без знака | ja/jnbe | операнд\_1 > операнд\_2 | cf = 0 and zf = 0 |
| Без знака | jae/jnb | операнд\_1 => операнд\_2 | cf = 0 |

**Задание**

Разработать на языке Ассемблера программу, которая по заданным целочисленным значениям параметров a, b, i, k вычисляет:

а) значения функций i1 = f1(a,b,i) и i2 = f2(a,b,i);

b) значения результирующей функции res = f3(i1,i2,k),

где вид функций f1 и f2 определяется из табл. 2, а функции f3 - из табл.3 по цифрам шифра индивидуального задания (n1,n2,n3), приведенным в табл.4.

Значения a, b, i, k являются исходными данными, которые должны выбираться студентом самостоятельно и задаваться в процессе исполнения программы в режиме отладки. При этом следует рассмотреть всевозможные комбинации параметров a, b и k, позволяющие проверить различные маршруты выполнения программы, а также различные знаки параметров a и b.

|  |  |
| --- | --- |
| Таблица 2 | Таблица 3 |
| / 15-2\*i , при a>b  f1 = <  \ 3\*i+4 , при a<=b  / - (4\*i+3) , при a>b  f2 = <  \ 6\*i -10 , при a<=b  / 7 - 4\*i , при a>b  f3 = <  \ 8 -6\*i , при a<=b  / -(6\*i - 4) , при a>b  f4 = <  \ 3\*(i+2) , при a<=b  / 20 - 4\*i , при a>b  f5 = <  \ -(6\*I - 6), при a<=b  / 2\*(i+1) -4 , при a>b  f6 = <  \ 5 - 3\*(i+1), при a<=b  / -(4\*i -5) , при a>b  f7 = <  \ 10 - 3\*i , при a<=b  / - (6\*i+8) , при a>b  f8 = <  \ 9 -3\*(i-1), при a<=b | / min(i1,i2), при k=0  f1 = <  \ max(i1,i2), при k/=0  / max(i1,10-i2), при k<0  f2 = <  \ |i1 - i2| , при k>=0  / |i1 + i2|, при k=0  f3 = <  \ min(i1,i2), при k/=0  / min (|i1 - i2|, 2), при k<0  f4 = <  \ max( -6, -i2), при k>=0  / min(|i1|, 6), при k=0  f5 = <  \ |i1|+|i2|, при k/=0  / |i1 - i2|, при k<0  f6 = <  \ max(7, |i2|), при k>=0  / |i1| + |i2|, при k<0  f7 = <  \ max(6, |i1|), при k>=0  / |i1| - |i2|, при k<0  f8 = <  \ max(4,|i2|-3), при k>=0 |

Таблица 4

|  |  |  |  |
| --- | --- | --- | --- |
| № студента | Шифр задания | № студента | Шифр задания |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | 1.2.1  1.3.2  1.4.3  1.5.4  1.6.5  1.7.6  1.8.7  2.3.8  2.4.7  2.5.6  2.6.5  2.7.4  2.8.3 | 14  15  16  17  18  19  20  21  22  23  24  25  26 | 3.4.2  3.5.3  3.6.4  3.7.5  3.8.6  4.5.7  4.6.8  4.7.2  4.8.3  5.6.4  5.7.5  5.8.6  6.8.1 |

Замечания:

1. при разработке программы нельзя использовать фрагменты, представленные на ЯВУ, в частности, для ввода-вывода данных. Исходные данные должны вводиться, а результаты контролироваться в режиме отладки;
2. при вычислении функций f1 и f2 вместо операции умножения следует использовать арифметический сдвиг и, возможно, сложение;
3. при вычислении функций f1 и f2 нельзя использовать процедуры;
4. при разработке программы следует минимизировать длину кода, для чего, если надо, следует преобразовать исходные выражения для вычисления функций.

**Выполнение**

Вариант 13 Шифр задания 2.8.3

/ - (4\*i+3) , при a>b

f2 = <

\ 6\*i -10 , при a<=b

/ - (6\*i+8) , при a>b

f8 = <

\ 9 -3\*(i-1), при a<=b

/ |i1 + i2|, при k=0

f3 = <

\ min(i1,i2), при k/=0

Сегменты в программе размещены в порядке:

1. Стек
2. Данные
3. Код

В сегменте данных заданы метки для переменных *a, b, i, k, i1, i2*, res использующихся в программе.

В связи с запретом на использование процедур в программе, функции были реализованы при помощи фрагментов кода, размеченных метками, с безусловными переходами на них. “Вызов” функции безусловным переходом jmp к метке данной функции. Функции используют значения переменных в сегменте данных, непосредственной передачи аргументов в функцию не происходит. Возврат значения происходит посредством заталкивания результата в стек, и выталкивание оного в переменную в основном фрагменте программы.

Функции содержат ветвление. Их поведение зависит от состояния переменных *a, b,* следовательно функции *f1* и *f2* логически подразделяются на три части:

сравнение переменных *a* и *b* и две ветви, переход в которые осуществляется непосредственно после сравнения. После расчета значения функции, оно помещается в стек, а управление передается в основной фрагмент кода, по заранее заданной метке.

Исполнение функции *f3* происходит аналогичным образом, но использует значения рассчитанные в двух предыдущих функциях

**Тестирование.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| a | b | i | k | i1 = f1 | i2 = f2 | res = f3 |
| 1 | 2 | 2 | 0 | 2 | 6 | 8 |
| 2 | 1 | 2 | 1 | -11 | -20 | -20 |
| 2 | 1 | 2 | 0 | -11 | -20 | 31 |

**Выводы.**

В результате работы были разобраны некоторые базовые концепции языка ассемблера. Были изучены команды сравнения, условного и безусловного перехода. Была написана рабочая программа, рассчитывающая значения кусочных математических функций.

Приложение

LR3.ASM

STACKSG SEGMENT PARA STACK 'Stack'

DW 32 DUP(?)

STACKSG ENDS

DATASG SEGMENT PARA 'Data' ;SEG DATA

VARA DW 1h

VARB DW 1h

VARI DW 1h

VARK DW 1h

VARI1 DW 1h

VARI2 DW 1h

VARRES DW 1h

DATASG ENDS ;ENDS DATA

CODE SEGMENT ;SEG CODE

ASSUME DS:DataSG, CS:Code

Main PROC FAR

mov ax, DATASG ;ds setup

mov ds, ax

jmp f1

f1\_end:

mov ax,VARI1

pop VARI1

jmp f2

f2\_end:

pop VARI2

jmp f3

f3\_end:

pop VARRES

mov ah, 4ch ;exit

int 21h

f1: ;dw f1(VARA,VARB,VARI)

mov ax,VARA ;ax = \*si (a)

mov si,VARB ;si = &b

mov bx,VARI ;bx = i

shl bx,1 ;bx \*= 2

cmp ax,si ;cmp ax,\*si (a,b)

jg f1\_1 ;if a>b

f1\_2: ;else (6i-10)

shl bx,1 ;bx \*= 2

mov ax,VARI ;ax = i

shl ax,1 ;ax \*= 2

add ax,bx ;ax += bx (ax = 6i)

sub ax,10 ;ax -= 10

push ax ;ret ax

jmp f1\_c

f1\_1: ;then -(4i+3) = (-4i - 3)

shl bx,1 ;bx \*= 2

sub ax,ax ;ax = 0

sub ax,bx ;ax = -bx (-4i)

sub ax,3 ;ax -= 3

push ax ;ret ax

jmp f1\_c

f1\_c:

jmp f1\_end

f2: ;dw f2(VARA,VARB,VARI)

mov ax,VARA ;ax = \*si (a)

cmp ax,VARB ;cmp ax,\*si (a,b)

jg f2\_1 ;if a>b

jmp f2\_2

f2\_1: ;then -(6i + 8) = (-6i - 8)

mov bx,VARI ;bx = i

shl bx,1 ;bx \*= 2

shl bx,1 ;bx \*= 2

mov ax,VARI ;ax = i

shl ax,1 ;ax \*= 2

add ax,bx ;ax += bx (ax = 6i)

sub bx,bx ;bx = 0

sub bx,ax ;bx = -ax (bx = -6i)

sub bx,8 ;bx -= 8

push bx ;ret bx

jmp f2\_c

f2\_2: ;else 9-3(i-1) = -3i + 12

mov ax,VARI ;ax = i

shl ax,1 ;ax \*= 2

add ax,VARI ;ax += i (ax = 3i)

sub bx,bx ;bx = 0

sub bx,ax ;bx = -ax (bx = -3i)

add bx,12 ;bx += 12

push bx ;ret bx

jmp f2\_c

f2\_c:

jmp f2\_end

f3: ;dw f3(VARI1,VARI2,VARK)

mov ax,VARK ;ax = \*si (k)

cmp ax,0 ;cmp k,0

je f3\_1 ;if k=0

; jmp f3\_2

f3\_2: ;else min(i1,i2)

mov ax,VARI1 ;ax = i1

cmp ax,VARI2 ;cmp i1,i2

jg f3\_2\_c ;unless quiri1 > i2

push ax ;ret i2

jmp f3\_c

f3\_2\_c: ;else

push VARI2 ;ret i1

jmp f3\_c

f3\_1: ;then abs(i1+i2)

mov ax,VARI1 ;ax = i1

add ax,VARI2 ;ax += i2 (ax = i1 + i2)

cmp ax,0 ;cmp i1+i2,0

jnl f3\_1\_c ;unless ax >= 0

neg ax ;neg ax

f3\_1\_c: ;else dont

push ax ;ret ax

jmp f3\_c

f3\_c:

jmp f3\_end

Main ENDP

CODE ENDS

END Main ;ENDS CODE

LR3.Lst

Microsoft (R) Macro Assembler Version 5.10 10/2/19 23:57:13

Page 1-1

0000 STACKSG SEGMENT PARA STACK 'Stack'

0000 0020[ DW 32 DUP(?)

????

]

0040 STACKSG ENDS

0000 DATASG SEGMENT PARA 'Data'

;SEG DATA

0000 0001 VARA DW 1h

0002 0001 VARB DW 1h

0004 0001 VARI DW 1h

0006 0001 VARK DW 1h

0008 0001 VARI1 DW 1h

000A 0001 VARI2 DW 1h

000C 0001 VARRES DW 1h

000E DATASG ENDS

;ENDS DATA

0000 CODE SEGMENT

;SEG CODE

ASSUME DS:DataSG, CS:Code

0000 Main PROC FAR

0000 B8 ---- R mov ax, DATASG

;ds setup

0003 8E D8 mov ds, ax

0005 EB 1A 90 jmp f1

0008 f1\_end:

0008 A1 0008 R mov ax,VARI1

000B 8F 06 0008 R pop VARI1

000F EB 40 90 jmp f2

0012 f2\_end:

0012 8F 06 000A R pop VARI2

0016 EB 75 90 jmp f3

0019 f3\_end:

0019 8F 06 000C R pop VARRES

001D B4 4C mov ah, 4ch

;exit

001F CD 21 int 21h

0021 f1: ;dw f1(VARA,VARB,VARI)

0021 A1 0000 R mov ax,VARA ;ax = \*si

(a)

0024 8B 36 0002 R mov si,VARB ;si = &b

0028 8B 1E 0004 R mov bx,VARI ;bx = i

002C D1 E3 shl bx,1 ;bx \*=

Microsoft (R) Macro Assembler Version 5.10 10/2/19 23:57:13

Page 1-2

2

002E 3B C6 cmp ax,si ;cmp ax,\*si

(a,b)

0030 7F 10 jg f1\_1 ;if a>b

0032 f1\_2: ;else (

6i-10)

0032 D1 E3 shl bx,1 ;bx \*=

2

0034 A1 0004 R mov ax,VARI ;ax = i

0037 D1 E0 shl ax,1 ;ax \*=

2

0039 03 C3 add ax,bx ;ax +=

bx (ax = 6i)

003B 2D 000A sub ax,10 ;ax -=

10

003E 50 push ax ;ret ax

003F EB 0E 90 jmp f1\_c

0042 f1\_1: ;then -

(4i+3) = (-4i - 3)

0042 D1 E3 shl bx,1 ;bx \*=

2

0044 2B C0 sub ax,ax ;ax = 0

0046 2B C3 sub ax,bx ;ax = -

bx (-4i)

0048 2D 0003 sub ax,3 ;ax -=

3

004B 50 push ax ;ret ax

004C EB 01 90 jmp f1\_c

004F f1\_c:

004F EB B7 jmp f1\_end

0051 f2: ;dw f2(VARA,VARB,VARI)

0051 A1 0000 R mov ax,VARA ;ax = \*si

(a)

0054 3B 06 0002 R cmp ax,VARB ;cmp ax,\*si

(a,b)

0058 7F 03 jg f2\_1 ;if a>b

005A EB 1B 90 jmp f2\_2

005D f2\_1: ;then -

(6i + 8) = (-6i - 8)

005D 8B 1E 0004 R mov bx,VARI ;bx = i

0061 D1 E3 shl bx,1 ;bx \*=

2

0063 D1 E3 shl bx,1 ;bx \*=

2

0065 A1 0004 R mov ax,VARI ;ax = i

0068 D1 E0 shl ax,1 ;ax \*=

Microsoft (R) Macro Assembler Version 5.10 10/2/19 23:57:13

Page 1-3

2

006A 03 C3 add ax,bx ;ax +=

bx (ax = 6i)

006C 2B DB sub bx,bx ;bx = 0

006E 2B D8 sub bx,ax ;bx = -

ax (bx = -6i)

0070 83 EB 08 sub bx,8 ;bx -=

8

0073 53 push bx ;ret bx

0074 EB 15 90 jmp f2\_c

0077 f2\_2: ;else 9

-3(i-1) = -3i + 12

0077 A1 0004 R mov ax,VARI ;ax = i

007A D1 E0 shl ax,1 ;ax \*=

2

007C 03 06 0004 R add ax,VARI ;ax += i

(ax = 3i)

0080 2B DB sub bx,bx ;bx = 0

0082 2B D8 sub bx,ax ;bx = -

ax (bx = -3i)

0084 83 C3 0C add bx,12 ;bx +=

12

0087 53 push bx ;ret bx

0088 EB 01 90 jmp f2\_c

008B f2\_c:

008B EB 85 jmp f2\_end

008D f3: ;dw f3(VARI1,VARI2,VARK)

008D A1 0006 R mov ax,VARK ;ax = \*si

(k)

0090 3D 0000 cmp ax,0 ;cmp k,0

0093 74 14 je f3\_1 ;if k=0

; jmp f3\_2

0095 f3\_2: ;else m

in(i1,i2)

0095 A1 0008 R mov ax,VARI1 ;ax = i

1

0098 3B 06 000A R cmp ax,VARI2 ;cmp i1,i2

009C 7F 04 jg f3\_2\_c ;unless

quiri1 > i2

009E 50 push ax ;ret i2

009F EB 1A 90 jmp f3\_c

00A2 f3\_2\_c: ;else

00A2 FF 36 000A R push VARI2 ;ret i1

00A6 EB 13 90 jmp f3\_c

00A9 f3\_1: ;then a

bs(i1+i2)

00A9 A1 0008 R mov ax,VARI1 ;ax = i

1

00AC 03 06 000A R add ax,VARI2 ;ax +=

i2 (ax = i1 + i2)

00B0 3D 0000 cmp ax,0 ;cmp i1

Microsoft (R) Macro Assembler Version 5.10 10/2/19 23:57:13

Page 1-4

+i2,0

00B3 7D 02 jnl f3\_1\_c ;unless

ax >= 0

00B5 F7 D8 neg ax ;neg ax

00B7 f3\_1\_c: ;else d

ont

00B7 50 push ax ;ret ax

00B8 EB 01 90 jmp f3\_c

00BB f3\_c:

00BB E9 0019 R jmp f3\_end

00BE Main ENDP

00BE CODE ENDS

END Main

;ENDS C

ODE

Microsoft (R) Macro Assembler Version 5.10 10/2/19 23:57:13

Symbols-1

Segments and Groups:

N a m e Length Align Combine Class

CODE . . . . . . . . . . . . . . 00BE PARA NONE

DATASG . . . . . . . . . . . . . 000E PARA NONE 'DATA'

STACKSG . . . . . . . . . . . . 0040 PARA STACK 'STACK'

Symbols:

N a m e Type Value Attr

F1 . . . . . . . . . . . . . . . L NEAR 0021 CODE

F1\_1 . . . . . . . . . . . . . . L NEAR 0042 CODE

F1\_2 . . . . . . . . . . . . . . L NEAR 0032 CODE

F1\_C . . . . . . . . . . . . . . L NEAR 004F CODE

F1\_END . . . . . . . . . . . . . L NEAR 0008 CODE

F2 . . . . . . . . . . . . . . . L NEAR 0051 CODE

F2\_1 . . . . . . . . . . . . . . L NEAR 005D CODE

F2\_2 . . . . . . . . . . . . . . L NEAR 0077 CODE

F2\_C . . . . . . . . . . . . . . L NEAR 008B CODE

F2\_END . . . . . . . . . . . . . L NEAR 0012 CODE

F3 . . . . . . . . . . . . . . . L NEAR 008D CODE

F3\_1 . . . . . . . . . . . . . . L NEAR 00A9 CODE

F3\_1\_C . . . . . . . . . . . . . L NEAR 00B7 CODE

F3\_2 . . . . . . . . . . . . . . L NEAR 0095 CODE

F3\_2\_C . . . . . . . . . . . . . L NEAR 00A2 CODE

F3\_C . . . . . . . . . . . . . . L NEAR 00BB CODE

F3\_END . . . . . . . . . . . . . L NEAR 0019 CODE

MAIN . . . . . . . . . . . . . . F PROC 0000 CODE Length = 00BE

VARA . . . . . . . . . . . . . . L WORD 0000 DATASG

VARB . . . . . . . . . . . . . . L WORD 0002 DATASG

VARI . . . . . . . . . . . . . . L WORD 0004 DATASG

VARI1 . . . . . . . . . . . . . L WORD 0008 DATASG

VARI2 . . . . . . . . . . . . . L WORD 000A DATASG

VARK . . . . . . . . . . . . . . L WORD 0006 DATASG

VARRES . . . . . . . . . . . . . L WORD 000C DATASG

@CPU . . . . . . . . . . . . . . TEXT 0101h

@FILENAME . . . . . . . . . . . TEXT LR3

@VERSION . . . . . . . . . . . . TEXT 510

Microsoft (R) Macro Assembler Version 5.10 10/2/19 23:57:13

Symbols-2

132 Source Lines

132 Total Lines

35 Symbols

48070 + 457140 Bytes symbol space free

0 Warning Errors

0 Severe Errors