**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 CS:Code

Main PROC FAR

mov ax, DATASG ;ds setup

mov ds, ax

jmp f1

f1\_end:

mov di,offset VARI1

pop [di]

jmp f2

f2\_end:

mov di,offset VARI2

pop [di]

jmp f3

f3\_end:

mov di,offset VARRES

pop [di]

mov ah, 4ch ;exit

int 21h

f1:

mov si,OFFSET VARA ;si = &a

mov ax,[si] ;ax = \*si (a)

mov si,OFFSET VARB ;si = &b

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

jg f1\_1 ;if a>b

jmp f1\_2

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

mov si,offset VARI

mov bx,[si] ;bx = i

shl bx,1 ;bx \*= 2

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\_2: ;else (6i-10)

mov si,offset VARI

mov bx,[si] ;bx = i

shl bx,1 ;bx \*= 2

shl bx,1 ;bx \*= 2

mov ax,[si] ;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\_c:

jmp f1\_end

f2:

mov si,OFFSET VARA ;si = &a

mov ax,[si] ;ax = \*si (a)

mov si,OFFSET VARB ;si = &b

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

jg f2\_1 ;if a>b

jmp f2\_2

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

mov si,offset VARI

mov bx,[si] ;bx = i

shl bx,1 ;bx \*= 2

shl bx,1 ;bx \*= 2

mov ax,[si] ;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 si,offset VARI

mov ax,[si] ;ax = i

shl ax,1 ;ax \*= 2

add ax,[si] ;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:

mov si,OFFSET VARK ;si = &k

mov ax,[si] ;ax = \*si (k)

cmp ax,0 ;cmp k,0

je f3\_1 ;if k=0

jmp f3\_2

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

mov si,offset VARI1

mov ax,[si] ;ax = i1

mov si,offset VARI2

add ax,[si] ;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\_2: ;else min(i1,i2)

mov si,offset VARI1

mov ax,[si] ;ax = i1

mov si,offset VARI2

cmp ax,[si] ;cmp i1,i2

jg f3\_2\_c ;unless i1 > i2

push ax ;ret i2

jmp f3\_c

f3\_2\_c: ;else

push [si] ;ret i1

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 9/25/19 24:04:53

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 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 BF 0008 R mov di,offset VARI1

000B 8F 05 pop [di]

000D EB 4D 90 jmp f2

0010 f2\_end:

0010 BF 000A R mov di,offset VARI2

0013 8F 05 pop [di]

0015 E9 009C R jmp f3

0018 f3\_end:

0018 BF 000C R mov di,offset VARRES

001B 8F 05 pop [di]

001D B4 4C mov ah, 4ch

;exit

001F CD 21 int 21h

0021 f1:

0021 BE 0000 R mov si,OFFSET VARA ;si = &a

0024 8B 04 mov ax,[si] ;ax = \*si

#Microsoft (R) Macro Assembler Version 5.10 9/25/19 24:04:53

Page 1-2

(a)

0026 BE 0002 R mov si,OFFSET VARB ;si = &b

0029 3B 04 cmp ax,[si] ;cmp ax,\*si

(a,b)

002B 7F 03 jg f1\_1 ;if a>b

002D EB 15 90 jmp f1\_2

0030 f1\_1: ;then -

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

0030 BE 0004 R mov si,offset VARI

0033 8B 1C mov bx,[si] ;bx = i

0035 D1 E3 shl bx,1 ;bx \*=

2

0037 D1 E3 shl bx,1 ;bx \*=

2

0039 2B C0 sub ax,ax ;ax = 0

003B 2B C3 sub ax,bx ;ax = -

bx (-4i)

003D 2D 0003 sub ax,3 ;ax -=

3

0040 50 push ax ;ret ax

0041 EB 17 90 jmp f1\_c

0044 f1\_2: ;else (

6i-10)

0044 BE 0004 R mov si,offset VARI

0047 8B 1C mov bx,[si] ;bx = i

0049 D1 E3 shl bx,1 ;bx \*=

2

004B D1 E3 shl bx,1 ;bx \*=

2

004D 8B 04 mov ax,[si] ;ax = i

004F D1 E0 shl ax,1 ;ax \*=

2

0051 03 C3 add ax,bx ;ax +=

bx (ax = 6i)

0053 2D 000A sub ax,10 ;ax -=

10

0056 50 push ax ;ret ax

0057 EB 01 90 jmp f1\_c

005A f1\_c:

005A EB AC jmp f1\_end

005C f2:

005C BE 0000 R mov si,OFFSET VARA ;si = &a

005F 8B 04 mov ax,[si] ;ax = \*si

(a)

0061 BE 0002 R mov si,OFFSET VARB ;si = &b

0064 3B 04 cmp ax,[si] ;cmp ax,\*si

(a,b)

0066 7F 03 jg f2\_1 ;if a>b

0068 EB 1B 90 jmp f2\_2

006B f2\_1: ;then -

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

006B BE 0004 R mov si,offset VARI

#Microsoft (R) Macro Assembler Version 5.10 9/25/19 24:04:53

Page 1-3

006E 8B 1C mov bx,[si] ;bx = i

0070 D1 E3 shl bx,1 ;bx \*=

2

0072 D1 E3 shl bx,1 ;bx \*=

2

0074 8B 04 mov ax,[si] ;ax = i

0076 D1 E0 shl ax,1 ;ax \*=

2

0078 03 C3 add ax,bx ;ax +=

bx (ax = 6i)

007A 2B DB sub bx,bx ;bx = 0

007C 2B D8 sub bx,ax ;bx = -

ax (bx = -6i)

007E 83 EB 08 sub bx,8 ;bx -=

8

0081 53 push bx ;ret bx

0082 EB 15 90 jmp f2\_c

0085 f2\_2: ;else 9

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

0085 BE 0004 R mov si,offset VARI

0088 8B 04 mov ax,[si] ;ax = i

008A D1 E0 shl ax,1 ;ax \*=

2

008C 03 04 add ax,[si] ;ax +=

i (ax = 3i)

008E 2B DB sub bx,bx ;bx = 0

0090 2B D8 sub bx,ax ;bx = -

ax (bx = -3i)

0092 83 C3 0C add bx,12 ;bx +=

12

0095 53 push bx ;ret bx

0096 EB 01 90 jmp f2\_c

0099 f2\_c:

0099 E9 0010 R jmp f2\_end

009C f3:

009C BE 0006 R mov si,OFFSET VARK ;si = &k

009F 8B 04 mov ax,[si] ;ax = \*si

(k)

00A1 3D 0000 cmp ax,0 ;cmp k,0

00A4 74 03 je f3\_1 ;if k=0

00A6 EB 16 90 jmp f3\_2

00A9 f3\_1: ;then a

bs(i1+i2)

00A9 BE 0008 R mov si,offset VARI1

00AC 8B 04 mov ax,[si] ;ax = i

1

00AE BE 000A R mov si,offset VARI2

00B1 03 04 add ax,[si] ;ax +=

i2 (ax = i1 + i2)

00B3 3D 0000 cmp ax,0 ;cmp i1

+i2,0

#Microsoft (R) Macro Assembler Version 5.10 9/25/19 24:04:53

Page 1-4

00B6 7D 02 jnl f3\_1\_c ;unless

ax >= 0

00B8 F7 D8 neg ax ;neg ax

00BA f3\_1\_c: ;else d

ont

00BA 50 push ax ;ret ax

00BB EB 16 90 jmp f3\_c

00BE f3\_2: ;else m

in(i1,i2)

00BE BE 0008 R mov si,offset VARI1

00C1 8B 04 mov ax,[si] ;ax = i

1

00C3 BE 000A R mov si,offset VARI2

00C6 3B 04 cmp ax,[si] ;cmp i1

,i2

00C8 7F 04 jg f3\_2\_c ;unless

i1 > i2

00CA 50 push ax ;ret i2

00CB EB 06 90 jmp f3\_c

00CE f3\_2\_c: ;else

00CE FF 34 push [si] ;ret i1

00D0 EB 01 90 jmp f3\_c

00D3 f3\_c:

00D3 E9 0018 R jmp f3\_end

00D6 Main ENDP

00D6 CODE ENDS

END Main

;ENDS C

ODE

#Microsoft (R) Macro Assembler Version 5.10 9/25/19 24:04:53

Symbols-1

Segments and Groups:

N a m e Length Align Combine Class

CODE . . . . . . . . . . . . . . 00D6 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 0030 CODE

F1\_2 . . . . . . . . . . . . . . L NEAR 0044 CODE

F1\_C . . . . . . . . . . . . . . L NEAR 005A CODE

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

F2 . . . . . . . . . . . . . . . L NEAR 005C CODE

F2\_1 . . . . . . . . . . . . . . L NEAR 006B CODE

F2\_2 . . . . . . . . . . . . . . L NEAR 0085 CODE

F2\_C . . . . . . . . . . . . . . L NEAR 0099 CODE

F2\_END . . . . . . . . . . . . . L NEAR 0010 CODE

F3 . . . . . . . . . . . . . . . L NEAR 009C CODE

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

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

F3\_2 . . . . . . . . . . . . . . L NEAR 00BE CODE

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

F3\_C . . . . . . . . . . . . . . L NEAR 00D3 CODE

F3\_END . . . . . . . . . . . . . L NEAR 0018 CODE

MAIN . . . . . . . . . . . . . . F PROC 0000 CODE Length = 00D6

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 9/25/19 24:04:53

Symbols-2

143 Source Lines

143 Total Lines

35 Symbols

47992 + 457218 Bytes symbol space free

0 Warning Errors

0 Severe Errors