

第三章 基本指令实验

§ 3.1 [实验 3.1] 循环操作

一、实验目的

1. 掌握循环操作指令的运用；
2. 掌握用汇编语言编写 DSP 程序的方法。

二、实验设备

1. 一台装有 CCS 软件的计算机；
2. DSP 实验箱的 TMS320C5416 主控板；
3. DSP 硬件仿真器。

三、实验原理

TMS320C54x 具有丰富的程序控制与转移指令，利用这些指令可以执行分支转移、循环控制以及子程序操作。本实验要求编写一程序完成 $y = \sum_{i=1}^5 x_i$ 的计算。这个求和运算可以通过一个循环操作指令 BANZ 来完成。BANZ 的功能是当辅助寄存器的值不为 0 时转移到指定标号执行。

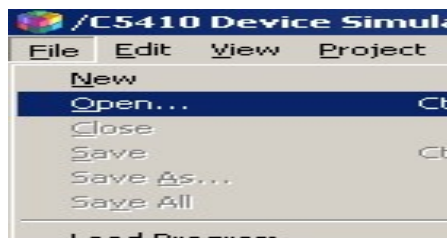
例如：


```
STM    #4,    AR2
loop:  ADD    *AR3+, A
      BANZ   loop,    *AR2-    ; 当 AR2 不为零时转移到 loop 行执行。
```

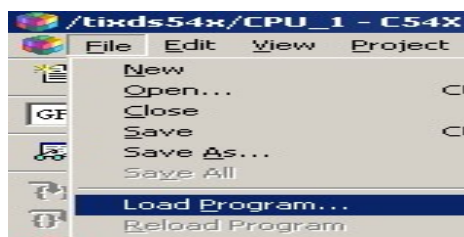
假设 AR3 中存有 x_1 到 x_5 五个变量的地址，则上述简单的代码就完成了这五个数的求和。


四、实验步骤

1. 学习有关指令的使用方法；
2. 在 CCS 环境中打开本实验的工程（Ex3_1.pjt），阅读源程序；

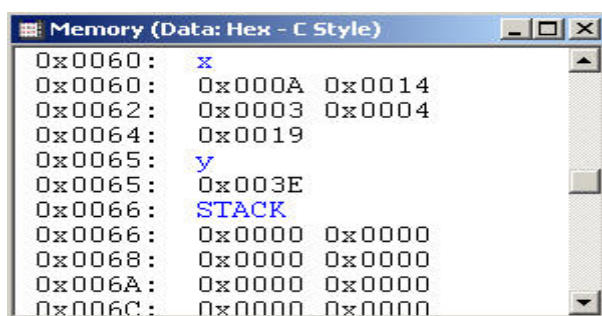


3. 编译并重建 .out 输出文件（Rebuild All...），然后通过仿真器把执行代码(.out 的文件)下载到 DSP 芯片中；




4. 在“end: B end”代码行设置断点（当光标置于改行时，单击工具条上的 Toggle Breakpoint 图标，此时该行代码左端会出现一个小红点或双击此行），单击运行 ；

5. 选择“View”->“memory”，起始地址设为“0x0060”，观察内存数值的变化。应能看到五个加数的值及其求和值。



注意查看 0X0060--0X0065 单元中计算值显示的十六进制结果。

6. 停止程序的运行（单击 ）；

7. 尝试改变对变量的初始赋值，或者增加或减少变量数目，重复上述 3~6 步过程，验证程序运行结果。

五、思考题

1. 总结迭代次数与循环计数器初值的关系（在本实验的代码中，用 AR2 作为循环计数器，设其初值为 4，共执行了 5 次加法。）；
2. 学习其它转移指令。

§ 3.2 [实验 3.2] 双操作数乘法

一、实验目的

1. 掌握 TMS320C54x 中的双操作数指令；
2. 掌握用汇编语言编写 DSP 程序的方法。

二、实验设备

1. 一台装有 CCS 软件的计算机；
2. DSP 实验箱的 TMS320C5416 主控板；
3. DSP 硬件仿真器。

三、实验原理

TMS320C54 x 片内的多总线结构，允许在一个机器周期内通过两个 16 位数据总线（C 总线和 D 总线）寻址两个数据和系数。双操作数指令是用间接寻址方式获得操作数的，并且只能用 AR2 到 AR5 的辅助寄存器。双操作数指令占用较少的程序空间，而获得更快的运行速度。

现举一例说明双操作数指令的用法。试求 $y=mx+b$ ，则用单操作数指令的代码应该如下：

```
LD      @m,    T
MPY     @x,     A      ; 单操作数乘法指令
ADD     @b,     A
STL     A,      @y
```

若用双操作数乘法指令则改为：

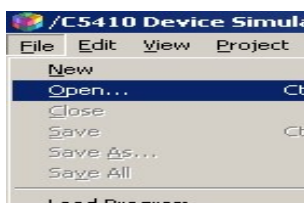
```
STM     @m,     AR2
STM     @x,     AR3
MPY     *AR2,   *AR3,  A ; 双操作数乘法指令
ADD     @b,     A
STL     A,      @y
```


表面上从代码的行数来看，用双操作数乘法指令似乎没有什么显著优势，但是双操作数指令可以节省机器周期，这在某些迭代运算过程中是十分有用的；迭代次数越多，节省的机器时间越多。

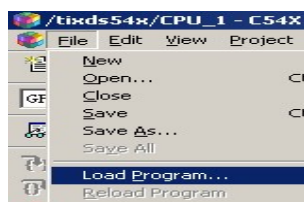
本实验要计算的乘法累加 $z = \sum_{i=1}^{10} a_i x_i$ 就是双操作数指令的一种应用场合。



四、实验步骤

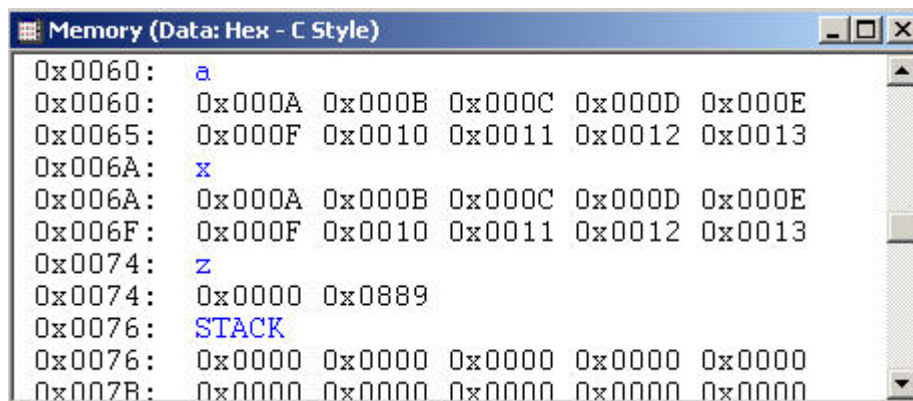
1. 学习有关双操作数乘法指令的使用方法；
2. 在 CCS 环境中打开本实验的工程（Ex3_2.pjt），阅读源程序；




3. 编译  并重建 .out 输出文件，然后通过仿真器把执行代码(.out 的文件)下载到 DSP 芯片中；



4. 在“end: B end”代码行设置断点  (或双击此行便可完成断点设制),单击运行 ;
5. 选择“View”->“memory”，起始地址设为“0x0060”，观察内存数值 a, x 和 z 的变化。



注意查看 0X0060--0X0075 单元中计算值显示的十六进制结果。

6. 停止程序的运行（单击 ）;
7. 改变对变量 a_i 和 x_i 的初始赋值，重复上述过程，验证程序运行结果。

五、思考题

1. 试用单操作数指令完成上述计算；
2. 学习其它双操作数指令。

§ 3.3 [实验 3.3] 并行运算

一、实验目的

1. 掌握 TMS320C54x 中的并行运算指令；
2. 掌握用汇编语言编写 DSP 程序的方法。

二、实验设备

1. 一台装有 CCS 软件的计算机；
2. DSP 实验箱的 TMS320C5416 主控板；
3. DSP 硬件仿真器。

三、实验原理

TMS320C54x 片内有 1 条程序总线，3 条数据总线和 4 条地址总线。这 3 条数据总线（CB、DB 和 EB）将内部各单元连接在一起。其中，CB 和 DB 总线传送从数据存储器读出的操作数，EB 总线传送写到存储器中的数据。并行运算就是同时利用 D 总线和 E 总线。其中，D 总线用来执行加载或算术运算，E 总线用来存放先前的结果。

并行指令有并行加载和乘法指令，并行加载和存储指令，并行存储和乘法指令，以及并行存储和加/减法指令 4 种。所有并行指令都是单字单周期指令。并行运算时存储的是前面的运算结果，存储之后再执行加载或算术运算。这些指令都工作在累加器的高位，且大多数并行运算指令都受 ASM（累加器移位方式位）影响。

现举一个并行指令为例：

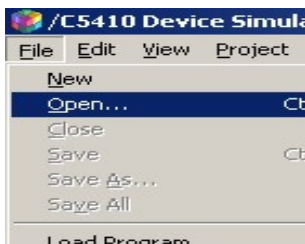
```
ST    src, Ymem          ; Ymem = src << (ASM-16)
|| LD  Xmem, dst          ; dst = Xmem << 16
```


这个并行加载和存储指令实现了存储 ACC 和加载累加器并行执行。其它的并行指令请读者查阅相关资料。

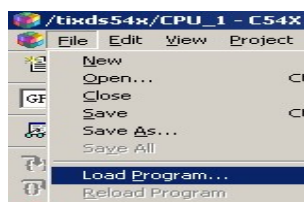
本实验中要求用并行运算指令编写程序完成 $z = x + y$ 和 $f = e + d$ 的计算。

四、实验步骤

1. 学习有关并行指令的使用方法；
2. 在 CCS 环境中打开本实验的工程（Ex3_3.pjt），阅读源代码 Ex3_3.asm；

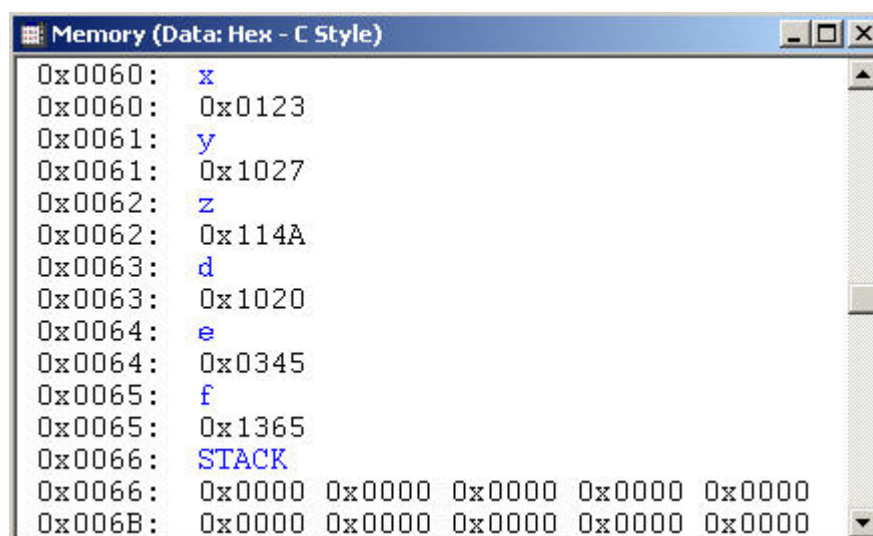


3. 编译  并重建 .out 输出文件，然后通过仿真器把执行代码(.out 的文件)下载到 DSP 芯片中；




4. 单击运行 ;

5. 选择“View”->“memory”，起始地址设为“0x0060”，观察内存数值的变化。应能看到 $z=x+y$ 和 $f=d+e$ 的结果。



注意查看 0X0060--0X0065 单元中计算值显示的十六进制结果。

6. 停止程序的运行（单击 ）；
7. 尝试改变对变量 x, y, d 和 e 的初始赋值，重复上述过程，验证程序运行结果。

五、思考题

1. 学习其它并行指令，理解其工作原理。

§ 3.4 [实验 3.4] 小数运算

一、实验目的

1. 掌握 TMS320C54x 中小数的表示和处理方法；
2. 掌握用汇编语言编写 DSP 程序的方法。

二、实验设备

1. 一台装有 CCS 软件的计算机；
2. DSP 实验箱的 TMS320C5416 主控板；
3. DSP 硬件仿真器。

三、实验原理

两个 16 位整数相乘，乘积总是“向左增长”，这意味着多次相乘后乘积将会很快超出定点器件的数据范围。而且要将 32 位乘积保存到数据存储器，就要开销 2 个机器周期以及 2 个字的程序和 RAM 单元；并且，由于乘法器都是 16 位相乘，因此很难在后续的递推运算中，将 32 位乘积作为乘法器的输入。然而，小数相乘，乘积总是“向右增长”，这就使得超出定点器件数据范围的是我们不太感兴趣的部分。在小数乘法下，既可以存储 32 位乘积，也可以存储高 16 位乘积，这就允许用较少的资源保存结果，也便于用于递推运算中。这就是为什么定点 DSP 芯片都采用小数乘法的原因。

小数的表示方法：

TMS320C54x 采用 2 的补码表示小数，其最高位为符号位，数值范围为 $(-1 \sim 1)$ 。一个十进制小数（绝对值）乘以 32768 后，再将其十进制整数部分转换成十六进制数，就能得到这个十进制小数的 2 的补码表示，例如：0.5 乘以 32768 得 16384，再转换成十六进制就得到 4000H，这就是 0.5 的补码表示形式。在汇编语言程序中，由于不能直接写入十进制小数，因此如果要定义一个小数 0.707，则应该写成 word 32768*707/1000，而不能写成 32768*0.707。

在进行小数乘法时，应事先设置状态寄存器 ST1 中的 FRCT 位（小数方式位）为“1”，这样，在乘法器将结果传送至累加器时就能自动地左移 1 位，从而自动消除两个带符号数相乘时产生的冗余符号位。使用的语句是

SSBX FRCT

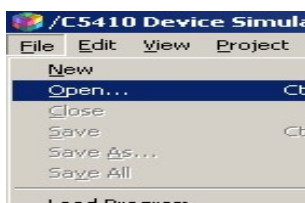
在本实验中，要求编写程序完成 $y = \sum_{i=1}^4 a_i x_i$ 的计算，其中的数据均为小数：a1=0.1 a2=0.2 a3=-0.3 a4=0.2


x1=0.8 x2=0.6 x3=-0.4 x4=-0.2

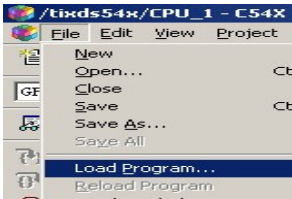
注意源代码中小数的表示。


四、实验步骤

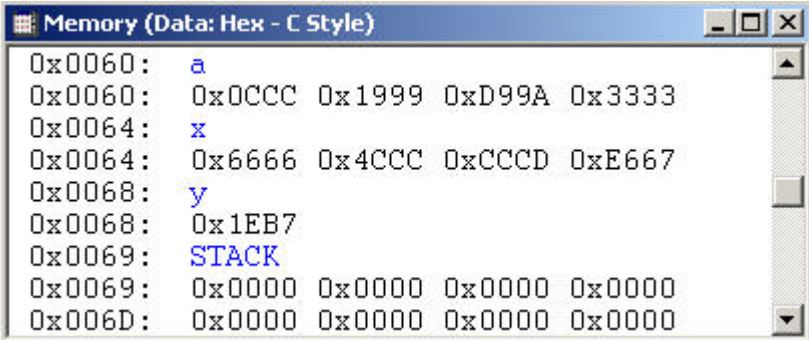
1. 在 CCS 环境中打开本实验的工程（Ex3_4.pjt），阅读源代码 Ex3_4.asm；




2. 编译并重建 .out 输出文件，然后通过仿真器把执行代码(.out 的文件)下载到 DSP 芯片中；



3. 单击运行；
4. 选择“View” -> “memory”，起始地址设为“0x0060”，观察内存数值的变化。



注意查看 0X0060--0X0068 单元中计算值显示的十六进制结果。其中计算结果为 $y=0.24=1EB7H$ ；

5. 停止程序的运行（单击）；
6. 尝试改变变量的赋值，重复上述过程，验证程序运行结果。

五、思考题

- a) 以 0.5×-0.375 为例分析两个带符号数相乘时的冗余符号位是如何产生的，理解为什么要设定 FRCT(小数)位。
- (注解：冗余符号位是两个带符号数相乘时存储器自动留出符号位的空间因此而产生;设置 FRCT 是为了减去多出来的一个符号位.)

操作	二进制		十进制
	0000000000000000	0100000000000000	0.5
*	1111111111111111	1101000000000000	-0.375
=	11110100000000000000000000000000		0.09375
左移一位	11101000000000000000000000000000		0.1875

§ 3.5 [实验 3.5] 长字运算

一、实验目的

1. 掌握 TMS320C54x 中的长字指令；
2. 掌握用汇编语言编写 DSP 程序的方法。

二、实验设备

1. 一台装有 CCS 软件的计算机；
2. DSP 实验箱的 TMS320C5416 主控板；
3. DSP 硬件仿真器。

三、实验原理

TMS320C54x 可以利用 32 位的长操作数进行长字运算。长字指令如下：

```
DLD    Lmem, dst
DST     src,    Lmem
DADD    Lmem, src [,dst]
DSUB    Lmem, src [,dst]
DRSUB   Lmem, src [,dst]
```

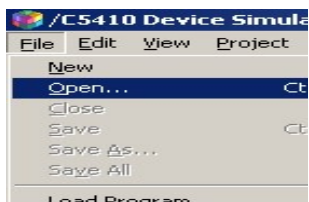
除了 DST 指令外，都是单字单周期指令，也就是在单个周期内同时利用 C 总线和 D 总线得到 32 位操作数。DST 指令存储 32 位数要用 E 总线 2 次，因此需要 2 个机器周期。


长操作数指令中的一个重要问题是，高 16 位和低 16 位操作数在存储器中如何排列。一般情况下，高 16 位操作数放在存储器中的低地址单元，低 16 位操作数放在存储器中的高地址单元。例如一个长操作数 16782345H，它在存储器中的存入方式是：(0060H) = 1678H（高字），(0061H) = 2345H（低字）。

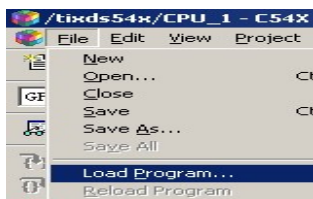
本实验利用长字指令完成两个 32 位数的相加。


四、实验步骤

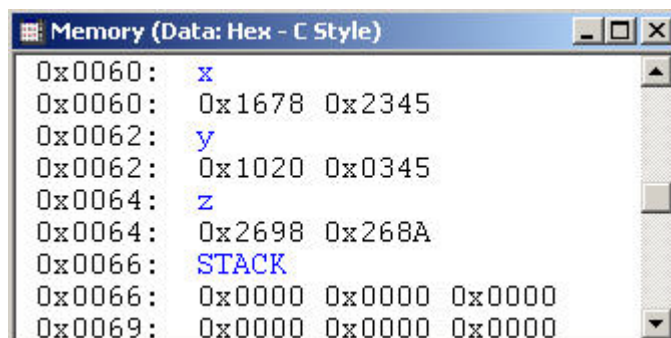
1. 在 CCS 环境中打开本实验的工程（Ex3_5.pjt），阅读源代码 Ex3_5.asm；




2. 编译  并重建 .out 输出文件，然后通过仿真器把执行代码(.out 的文件)下载到 DSP 芯片中；



- 单击运行 ;
- 选择 “View” -> “memory”，起始地址设为 “0x0060”，观察内存数值的变化。



注意查看 0X0060--0X0065 单元中计算值显示的十六进制结果。

- 停止程序的运行（单击 ）;
- 尝试改变变量的赋值，重复上述过程，验证程序运行结果。

五、思考题

- 试给出不用长字指令实现上述计算的代码。

§ 3.6 [实验 3.6] 浮点运算

一、实验目的

1. 掌握 TMS320C54x 中浮点数的表示和处理方法；
2. 掌握用汇编语言编写 DSP 程序的方法。

二、实验设备

1. 一台装有 CCS 软件的计算机；
2. DSP 实验箱的 TMS320C5416 主控板；
3. DSP 硬件仿真器。

三、实验原理

在数字信号处理中，为了扩大数据的范围和精度，需要采用浮点运算，TMS320C54x 虽然是个定点 DSP 器件，但它支持浮点运算。

浮点数的表示方法：

在 TMS320C54x 中浮点数用尾数和指数两部分组成，它与定点数的关系是“定点数 = 尾数 * $2^{\text{exp}(-\text{指数})}$ ”例如：定点数 0x2000（即 0.25）用浮点数表示时，尾数为 0x4000（即 0.5），指数为 1，即 $0.25 = 0.5 * 2^{-1}$ 。

定点数到浮点数的转换：

TMS320C54x 通过 3 条指令可将一个定点数转化成浮点数（设定点数存放在累加器 A 中）。这 3 条指令分别是“EXP A”，“ST T, EXPONENT”和“NORM A”。

（1）“EXP A”：这是一条提取指数的指令，指数保存在 T 寄存器中。

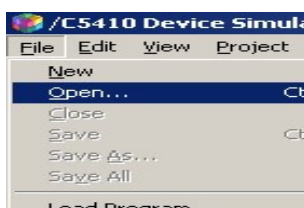
（2）“ST T, EXPONENT”：这条紧接在 EXP 后的指令是将保存在 T 寄存器中的指数存放到数据存储器的指定单元中。


（3）“NORM A”：这是一条按 T 寄存器中的内容对累加器 A 进行归一化处理（左移或右移），T 中为正数时左移，为负数时右移，移动的位数就是 T 中的指数值。但这条指令不能紧跟在 EXP 指令后面，因为此时 EXP 指令还没有来得及把指数值送至 T。

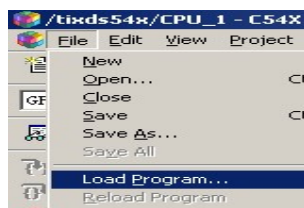
本实验中，要求编写浮点乘法程序，完成 $x1 * x2 = 0.3 * (-0.8)$ 的运算。源程序中保留了 10 个存储单元：x1（被乘数），x2（乘数），e1（被乘数的指数），m1（被乘数的尾数），e2（乘数的指数），m2（乘数的尾数），ep（乘积的指数），mp（乘积的尾数），product（乘积），temp（临时单元）。


四、实验步骤

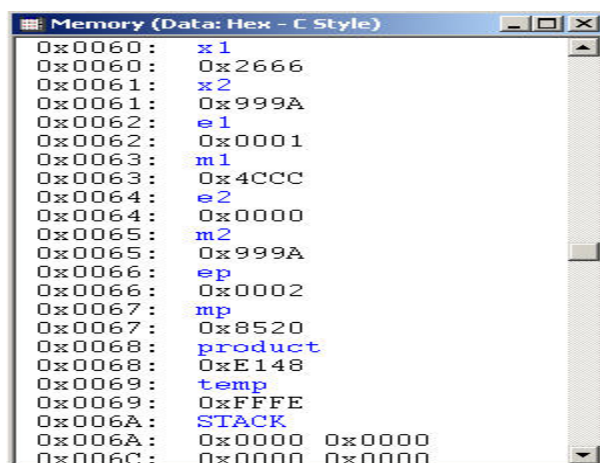
1. 在 CCS 环境中打开本实验的工程（Ex3_6.pjt），阅读源代码 Ex3_6.asm；




2. 编译  并重建 .out 输出文件，然后通过仿真器把执行代码(.out 的文件)下载到 DSP 芯片中；



3. 单击运行 ;
4. 选择 “View” -> “memory”，起始地址设为 “0x0060”，观察内存数值的变化。



注意查看 0X0060--0X0069 单元中计算值显示的十六进制结果。其中计算结果即乘积的尾数为 8520H，指数为 0002H，乘积的定点数为 E148H，对应的十进制数约等于 - 0.24；

5. 停止程序的运行（单击 ）；
6. 尝试改变变量的赋值，重复上述过程，验证程序的运行结果。

五、思考题

1. 试分析指数提取指令 “EXP A” 和归一化指令 “NORM A” 的工作原理；
2. 浮点数到定点数应如何转换。
3. 说明两个浮点数相乘的过程。