算术函数标准计算程序设计

钱念祖　肖定柏　著

前 言

近几年来，微型计算机的发展迅猛异常，已经在各工业交通、通信广播、军事科学、航天技术、海洋工程、科技教育、文艺体育、银行商业、服务事业以及家用电器等各行各业得到应用，深刻地影响着国民经济，冲击着各种工业产品的结构，推动了社会生产力的发展。其作用完全可以和应用蒸汽机、电力、原子能的“工业革命”相媲美。因此很多人指出：微型机的出现和应用是“新的一次工业革命”，是第四次工业革命！

在进行四个现代化建设的我国人民，将大规模集成电路和计算机的发展象“两弹一箭”来抓，把使用计算机实现企事业管理和自动化控制作为四个现代化的标志。不少人认为：微型机的出现，给计算技术比较后进的国家提供了一个迎头赶上的良机，並且是一个稍纵即逝的机会，不可等闲视之。正因为这样，我国各行各业的专业人员在各级领导支持和关怀下，日夜工作于数以万计的各类微型机上，几年来取得了显著的成绩。

发展计算机工业，推广计算机的应用，硬件是基础、软件是灵魂，应用是目的。可见软件是很重要的一个方面，每个微机应用人员必须做好程序设计工作。一个好的软件应该成为社会的共同财富，这样可以避免许多人的重复劳动。本书介绍的算术、函数计算程序是一个移植分析和创新改造相结合产生的标准通用程序，是一个可以在以Z—80为CPU或以I8080、I8085为CPU的单板机上运行的汇编程序，是作者献给广大软件工作者的一份程序设计资料，给软件工作者提供了一条进行各类计算程序设计的捷径。我们这样肯定本书的作用，是因为对于过程控制、数据采集和处理、仪器仪表、机械产品智能化都少不了进行算术和函数运算。而对于实时性要求较高或使用存储器容量小的单板机的地方，一般没有可供存放高级语言的缓存器，即使能使用上它，高级语言也很难满足速度的要求，无法精确定时和准确描写控制要求。因此很多使用者必须根据自己的需要，用汇编语言编制程序，特别是要建立一套算术和函数计算程序，为此，要投入大量的时间。本书所介绍的算术、函数运算程序就可以帮助你解决这个问题，为您节约宝贵的时间。本书所介绍的程序已编辑成若干个子程序库的形式，为叙述方便将算术、函数标准计算程序简称**AFSP**(Arithmetical and Functional Standardized Program)，本算术、函数标准计算程序(**AFSP**)具有如下特点：

1. **AFSP** 具有占用内存少、执行速度快、运算精度高、使用方便，除占用一定内存外无需硬件支持。

2. **AFSP** 是以浮点单精度加、减、乘、除计算程序为核心的结构程序，外层调用内存，相辅相成，结构严密，长期运行可靠。

3. **AFSP** 在函数计算中使用了独到的数学模型、使用了“动态程序”设计和“单字节进入”等方法，使整个程序简洁、高速。

4. **AFSP** 包含54个计算程序，占用3.82K字节的空间，数据区占用84个字节，在作函数计算时相对误差达到10^{-7}。

5. **AFSP** 以ASCII码输入，并以ASCII码输出，计算结果可直接送CRT显示，或打印机打印。

6. **AFSP** 作浮点单精度运算时内部按7位数字运算，以6位有效数字输出，在作浮点双精度运算时内部按17位数字运算，以16位有效数字输出。

7. **AFSP** 运算范围在[-1.7×10^{38}，$+1.7\times10^{38}$]。如果数值的绝对值小于5.8×10^{-39}时，作为0值处理。

8. **AFSP** 除54个计算程序外，还提供28个可供直接调用的小子程序和若干个机内常数。

9. **AFSP** 的源程序是用Z80汇编语言编写，只作部分改动就能方便地运行于以I8080和I8085为CPU的微机上。

本书还包含一些其它与之有关的内容：如Z—80指令与I8080指令的兼容性，如何将Z—80指令写成的**AFSP**改为I8080指令的**AFSP**；第八章通过从简单到复杂的各种应用计算程序编制方法举列，可以启迪读者在各行各业的应用；本书虽然是一本“程序设计”，但还就程序的固化、复制和选择地址、测试程序运行时间的方法上，软件与硬件相结合的形式上作了具体介绍，相信这对各类单板机用户有一定帮助。

这里发表的程序清单是经改进和验证的新型清单。新型除了开方程序按牛顿迭代法重新设计外，主要改动是在排列次序上的变化，其功能、用法与原型相同。

“算术、函数标准计算程序设计”一书是在“算术函数标准程序”以固体软件产品形式向社会推广后，应广大用户的要求，並得到多方支持和鼓励下写成的。其中第2—7章及第八章的第一节由上海蓄电池厂助工钱念祖执笔，第八章的第二节由申联软件开发中心助工徐开明提供，其余由上海第二工业大学讲师肖定柏执笔和负责全书的审订与编辑。作者在此向广大用户和读者表示衷心感谢，不当之处，请提宝贵意见。

目　录

第一章　汇编程序分析和使用的基本知识…… 1
一、汇编语言引论…… 1
二、汇编语言程序的格式…… 1
三、Z80指令系统的特点及与I 8080指令系统的兼容性 …… 3
四、AFSP改写为I8080指令程序 …… 9
五、字符编码……10
第二章　关于AFSP的综述 ……11
一、AFSP是用Z80汇编语言编写的 ……11
二、AFSP中数的类型及其在内存中的表示 ……11
三、AFSP中关于出错的处理 ……12
四、AFSP使用的数据区 ……13
五、AFSP中各类型数的算术和函数运算的原始数据(或自变量)和计算结果(或函数值)存放地址表 ……15
六、AFSP各子程序的名称，标号和入口地址表 ……16
第三章　算术计算程序……19
一、关于源程序清单解释的说明……19
二、浮点单精度型数的加法(RADD)，减法(RSUB)程序……19
三、浮点单精度型数的乘法(RMU)程序 ……27
四、浮点单精度型数的除法(RDIU)程序 ……30
五、整型数的加法 (IADD)、减法(ISVB)程序 ……35
六、整型数的乘法(IMU)程序 ……39
七、整型数的除法(IDIV)程序 ……42
八、浮点双精度型数的加法(DADD)、减法(DSUB)程序…… ……42
九、浮点双精度型数的乘法 (DMU) 程序……50
十、浮点双精度型数的除法(DDIV)程序 ……52
第四章　特殊函数计算程序……56
一、转换为浮点双精度型数(CDBL)程序……57
二、转换为浮点单精度型数(CSNG)程序……58
三、取自变量的绝对值(ABS)程序 ……58
四、取自变量的符号(SGN)程序 ……61
五、取不大于自变量的最大整型数①(CINT)程序 ……62
六、取不大于自变量的最大整数②(INT)程序……65
七、简单截尾(FIX)程序……68
八、取伪随机数(RND)程序 ……66

九、 类型转换对精度的影响……72
十、 特殊函数的自变量和函数值的存放地址表……73
第五章 十进制数与二进制数相互转换程序……74
一、 十进制数转换为二进制数(TENTWO)程序……74
二、 二进制数转换为十进制数(TWOTEN)程序……86
第六章 常用函数计算程序……103
一、 求自变量的平方(SQUARE)程序……103
二、 求自变量的平方根(SQR)程序……103
三、 计算以e为底的指数(EXP)程序……107
四、 计算以e为底的对数(LN)程序……112
五、 计算以10为底的指数(TENXP)程序……116
六、 计算以10为底的对数(LOG)程序……116
七、 乘方(POWER)程序……116
八、 计算正弦(SIN)和余弦(COS)函数程序……120
九、 计算正切(TAN)函数程序……124
十、 计算反正切(ATN)函数程序……124
十一、 取自变量的倒数(REC)程序……126
第七章 复杂函数程序和小子程序……127
一、 说明……127
二、 计算各复杂函数的公式表……127
三、 程序请单和注释……128
四、 供编程时直接调用的28个小子程序……132
第八章 使用AFSP编制应用程序……137
一、 简单应用举例……137
二、 复杂计算程序举例……142
(一)、 用龙贝格公式求数值积分……142
(二)、 用无回代消去法解线性方程组……151
(三)、 用牛顿迭代法解一元方程……164
第九章 程序的固化和运行时间的测定……170
一、 存储空间的划分……170
二、 存储器译码器……170
三、 EPROM芯片……173
四、 EPROM的编程方法……175
五、 运行时间的测定……179

第一章　汇编程序分析和使用的基本知识

假若你已经学习过微型计算机的指令系统和初步掌握了一些程序设计技巧，本章《汇编程序分析和使用的基本知识》你就可能具备了。在这里仅对其基本知识提纲挈领和列表形式汇编于后。

一、汇编语言引论

一切数字计算机能直接识别的指令和数据都是二进制形式数的组合。我们把这种二进制形式的指令和数据叫机器语言。如果用它们编成一段程序(叫机器语言程序或叫目标程序)，存在程序长、难看懂、容易错等缺点。微计算机对二进制数表示方法的一个改进是使用十六进制数，但上述缺点並未解决。

为了帮助记忆和使用机器语言指令，微处理器设计师给每条指令都取了一个合适的名称，使用英文"助记符"，如：AND、OR、CMP、NEG、XOR、ADD、SUB等，给CPU中常用寄存器也取了便于记忆的名称，如：AF、B、C、D、E、H、L、SP、IX、IY等。使用这种助记符写的指令、数据、连同汇编所用的标号、伪指令及使用规则构成了汇编语言。用汇编语言写的程序(叫汇编语言程序或叫源程序)就基本克服了机器语言的缺点。

汇编语言程序不能直接为计算机所认识，需要一种汇编语言程序的翻译器——汇编器或叫汇编程序——来区别指令、伪指令、标号、数据，并逐条进行翻译成机器语言。这一工作可以是人手工完成，但方便且广泛使用的是微计算机系统提供的编辑/汇编文件(EDTASM/CMD)，用它先完成用助记符书写的用户程序的编辑，然后用汇编命令(A)将源程序翻译成机器语言(目标程序)，如命令后还带有选项/WE/LP，(WE—表示出错选项，LP—表示打印清单)，进行汇编后，就会一边显示，一边打印程序清单，並且遇错自动停止显示和打印。

上面指出了汇编语言比机器语言有进步，但它与高级语言相比又如何？可以有三点优于高级语言的地方：

1)．汇编程序经汇编后是以机器语言形式存放(固化)，可直接运行，占用内存少，比高级语言编译后的目标程序短。

2)．程序运行速度快，比执行高级语言程序要快上几倍至几十倍，这是因为用有些高级语言所编的程序必须在编译系统(或解释程序)存在的条件下进行。

3)．编写程序时可直接调用计算机的全部资源，能准确掌握程序执行时间，提供描述精细的控制。

目前尚未出现比汇编语言占内存少、执行速度快、标准化强的高级语言。尽管不同微处理器的汇编语言不同，不象高级语言通用，但在工业过程控制和智能仪器的应用设计中，仍广泛使用它。

二、汇编语言程序的格式：

汇编语言程序被分成四段：**标号、操作码/伪指令、操作数/地址、注释**。操作码段是唯一不可缺的段，不是微处理器的指令，就是汇编程序使用的"伪指令"或"伪操作"。操作数/地址段，可以是数据或地址，或空着(象某些一字节指令那样)。标号和注释可以有，也可以无。见下表1—1

表 1—1　　汇编语言指令的各段

标　号	操作码/伪指令	操作数/地址	注　释
START ：	L D	A，(VAL1)	；取第一数至A
	L D	B，　A	；转存于 B
	L D	A，(VAL2)	；取第二数至A
	ADD	A，　B	；二数相加
	L D	(SUM)，A	；存和
	HALT		；停机
NEXT ：	⋮		
VAL1 ：	DEFS	1	
VAL2 ：	DEFS	1	
SUM ：	DEFS	1	

由表可见，各段开始或结束使用了特定的符号或界符，标号后用“:”，操作码和操作数间用**空格**，地址或操作数间用“,”，注释(用英文)前用“;”，园括号内写访内地址或口地址。在我们下面介绍的**AFSP**清单中，都加有标号和注释(用中文)，就这两部分用法罗列如下：

1. 标号

一个源程序中有标号、伪指令和注释，这些只对阅读和汇编起作用的部分，汇编后不进入目标程序。因此，如由一个目标程序反汇编得到它的源程序，就再也没有标号和注释了，读起来就非常困难。

标号的作用在于：1). 使程序单元容易查找；2). 可以通过移动标号来修改程序，而不必修改后面使用该标号的任何指令，因为汇编程序能自动进行全部必要的修改；3). 如给每个标号加一个常数(相对地址偏移量)，汇编程序可使整个程序浮动，从而可实现插入其它程序，重新安排存储器空间，和从偏移标号某一值的程序进入；4). 使程序容易作为库程序，供其它用户调用某标号；5). 由标号自动计算存储器地址，避免了由于微处理器指令长短不一，计算地址麻烦和出错的弊端。

为了将某些指令的地址作为程序转移和调用的目标地址，便于分析和认识，我们在编制功能块文件时，对其加有一助记性标号，如TENTWO (10进数转换为2 进数)，SIN (正弦)等。有些属程序内部的转移，我们则在标准子程序标号后加上序号来减少标号的种类，如OADD nn。要将标号使用恰当是需要消化程序后，按有利于阅读和区别为原则来拟定。对于标号后与EQU、DEFL和MACRO等伪操作间可不用冒号，只要空一格。

2. 注释

对源程序加上注释是程序设计时常用的方法，它不影响目标码，而可帮助阅读程序及编写文件。良好的注释是来自对程序功能的深刻了解，是一件很不容易做好的事。这里遵循的准则是：

1) 用注释来说明程序做什么，而不是指令本身做什么?如注释：“整型数计算溢出，转单

精度型数处理”。

2)注释应简明扼要，详细内容放到文件别处去说。

3)在所有关键地方加上注释。

4)注释全部定义、常数，指出它们的目的，标明全部表格和数据存放区。

5)可能是对一条指令，也可能是对一个程序段进行注释。

6)发现容易混淆的地方留下说明，例如“记住上一条指令，使进位置1”。

三、Z80指令系统的特点及与I8080指令系统的兼容性

1. Z80指令系统的特点

Z80 CPU由Zilog公司公布它可执行158条指令，由于有些指令有8位与16位之分，还有102条关于IX和IY的Z80指令未正式发表(见附)，所以实际可执行的指令更多。其中包括全部78条I8080指令，不过与I8080指令有相同意义的那些指令，符号和写法是有些不同，(详见Z80和I8080助记符对照表)，所以说Z80指令是对Intel8080的机器码一级向上兼容。

Z80指令主要扩充利用了Intel8080没有用的指令码：CB、ED、DD、FD形成多字节指令。有二字节、三字节、还有四字节指令。不管它是几字节指令，对于今天单地址计算机，一定是将进入的第一字节作为进入该指令的开始，即以此时地址所指的内容为某指令码的第一字节，如果有意或无意、或错误地从多字节指令的其它字节进入，就要从这个进入字节来确定其指令的含义了。程序设计，调试中要注意这点，有时则可能利用这一点，实现改变操作的“单字节进入”，使程序发生变化，得到多功能的通用程序。(具体使用技巧见第三章四、2节)。

Z80对I8080的扩充不仅在指令的数量上，而且表现在寻址方式上，有了多种寻址方式，加强了指令的功能，方便了程序设计。这些寻址方式有：①隐含寻址；②立即(数)寻址；③扩展立即寻址；④寄存器寻址；⑤寄存器间接寻址；⑥扩展寻址；⑦可修改的0页寻址；⑧相对寻址；⑨变址寻址；⑩位寻址。

Z80指令按类型分为：八位传送类指令，十六位传送类指令，交换、数据块传送和数据查找指令，八位算术运算和逻辑运算指令，十六位算术运算指令，通用运算和CPU控制类指令，循环和移位指令，位置0、置1和位测试类指令，有条件和无条件的转移、转子和返回类指令，输入和输出类指令。详细见“Z80汇编语言程序设计”一书。

2. Z80与I8080的兼容性

i.两者的兼容性包括以下几个方面：

①I 8080的所有指令的机器码，都是Z80指令的机器码，但助记符不同，故是机器码的兼容，没有扩大到汇编语言源程序；

②I8080的所有寄存器，Z80 CPU中也全具有；

③几乎所有的I8080的程序都可在Z80机上运行。(微小的差别部分将在后面指出)，但反过来就不能保证，除非你全用Z80中8080级指令来编写你的程序。对于使用SIM、RIM的8085程序，不能在Z—80上运行。

ii.不兼容性在下列几个方面：

1)它们状态标志略有不同，它们的对应关系如下：

I8080的标志位

F:	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
	S	Z	0	AC	0	P	1	C

Z80的标志位

F:	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
	S	Z	×	H	×	p/v	N	C

可见标志名称和所在位基本一致，但有如下三点差别：

① Z80用标志P/V作多用途标志：在输入、循环和逻辑运算时，它是一奇偶标志(1表示偶，0表示奇)；在算术运算时，它是算术运算后的对2补码溢出标志；在数据传送和搜索操作中，恒为1，直到字节数为0，才复位为0；在执行LD A，I或LD A，R时，为中断开放触发器(IF/F_2)的现行状态。而在I8080中它仅仅是一个奇偶标志P。

② 增加了一个减法标志N，N = 1减，N = 0加，这是为了增强DAA指令的功能。在I 8080中只有十进制加法的调整功能，而Z80中既可调整十进制加，又可调整十进制减。(DAA指令总紧接在+/-类指令后，用来转换累加器内容为BCD形式)。这是因为BCD校正对加法和减法的校正是不同的，为了增加Z80的这一功能，使用这个减法标志(N)，就可以方便进行。

③ 半进位(H)或叫辅助进位(AC)，在Z80中执行循环指令时，是清除其标志(H = 0)，而8080中执行循环指令不影响它的辅助进位标志(即AC不变)。

2) 有些指令的附加功能不同，如IN，OUT指令，在Z80中除完成对第二字节指定的口址将A中内容输出或将口址的内容输入到A中外，同时还将原A口内容转至A 8～A15地址线上。而I8080中，执行该二指令，只是简单地把地址的低8位也输出至高8位地址线上，Z80还有以C寄存器作口址的IN，OUT指令及成组IN，OUT指令组，使数据在外设口上的传送和可编程接口芯片的初始化编程十分方便。

3) Z80机与I8080、I8085机的指令时序完全不同，因此依赖精确指令定时的程序，不能简单使用Z80指令程序转变为I8080、I8085指令的程序，应在所用微处理器的系统上运行修改确定。

4) Z80与I8085系统增加的两条指令：RIM(读中断屏蔽，机器码20)、SIM_(置中断屏蔽，机器码30)是不相容的。因此I8085用到这种中断结构的程序是不能在Z80机上使用。

下面对I8085这两条指令作一说明：

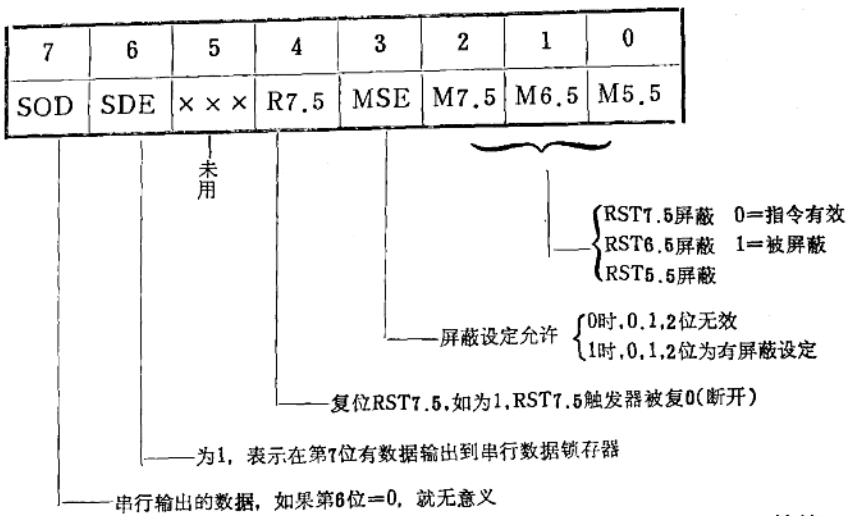
RIM指令是将一个8位数装入累加器的多用途指令，它按位指示：给定中断屏蔽、中断标志、中断挂起，D7表示串行输入的一位数据。机器码20。

7	6	5	4	3	2	1	0
SID	17	16	15	IE	7.5	6.5	5.5

中断挂起为1挂起　　中断屏蔽，为1屏蔽

串行输入数据位　　中断允许标志，为1允许(对应INTE功能)

SIM 指令是一条与RIM具有某些相反功能的多用途指令，用现时累加器的内容完成下列功能：中断屏蔽与/或写出一位串行数据。机器码30。



在Z80中，机器码20为 JR NZ，disp 的第一字节，机器码 30为JR NC，disp 的第一字节。

在Z80和I8080助记符对照表中主要使用的缩记符含义如下：

addr——16位存贮器地址；

data——8位二进数据；

data16——16位二进数据；

disp——8位带符号的二进地址位移量；

pr ——寄存器对BC、DE、HL、AF之一；

rp ——寄存器对BC、DE、HL、SP之一；

XY ——变址寄存器IX或IY。

Z 80	I 8080	Z 80	I 8080
ADC A, data	ACI data	AND (xy+disp)	—
ADC A, (HL)	ADC M	BIT b, (HL)	—
ADC A, reg	ADC reg	BIT b, reg	—
ADC A, (xy+disy)	—	BIT b, (xy+disp)	—
ADC HL, rp	—	CALL addr	CALL addr
ADD A, data	ADI data	CALL C, addr	CC addr
ADD A, (HL)	ADD M	CALL M,addr	CM addr
ADD A, reg	ADD reg	CALL NC, addr	CNC addr
ADD A, (xy+disp)	—	CALL NZ, addr	CNZ addr
ADD HL, rp	DAD rp	CALL P,addr	CP addr
ADD IX, rp	—	CALL PE, addr	CPE addr
ADD IY, rp	—	CALL PO, addr	CPO addr
AND data	ANI data	CALL Z, addr	CZ addr
AND (HL)	ANA M	CCF	CMC
AND reg	ANA reg	CP data	CPI data

Z80	I8080
CP (HL)	CMP M
CP reg	CMP reg
CP (xy+disp)	—
CPD	—
CPDR	—
CPI	—
CPIR	—
CPL	CMA
DAA	DAA
DEC (HL)	DCR M
DEC reg	DCR reg
DEC rp	DCX rp
DEC xy	—
DEC (xy+disp)	—
DI	DI
DJNZ disp	—
EI	EI
EX AF, AF′	—
EX DE, HL	XCHG
EX (SP), HL	XTHL
EX (SP), XY	—
EXX	—
HALT	HLT
IM m	—
IN A, (port)	IN port
IN reg, (C)	—
INC (HL)	INR M
INC reg	INR reg
INC rp	INX rp
INC xy	—
INC (xy+disp)	—
IND	—
INDR	—
INI	—
INIR	—
JP addr	JMP addr
JP C, addr	JC addr
JP (HL)	PCHL
JP M,addr	JM addr
JP NC,addr	JNC addr
JP NZ,addr	JNZ addr

Z80	I8080
JP P, addr	JP addr
JP PE, addr	JPE addr
JP PO, addr	JPO addr
JP Z, addr	JZ addr
JP XY	—
JR C, disp	—
JR disp	—
JR NC, disp	—
JR NZ, disp	—
JR Z, disp	—
LD A, (addr)	LDA addr
LD A,(BC)或(DE)	LDAX B或D
LD A, I	—
LD A,R	—
LD (addr), A	STAX addr
LD (addr),BC或DE	—
LD (addr),HL	SHLD addr
LD (addr),SP	—
LD (addr),XY	—
LD (BC)或(DE),A	STAX B或D
LD BC或DE,(aedr)	—
LD HL,(addr)	LHLD addr
LD (HL),data	MVI M, data
LD (HL), reg	MOV M, reg
LD I,A	—
LD R,A	—
LD reg,data	MVI reg, data
LD reg, (HL)	MOV reg, M
LD reg,reg	MOV reg,reg
LD reg, (xy+disp)	—
LD rp,data 16	LXI rp,data16
LD SP,(addr)	—
LD SP,HL	SPHL
LD SP,XY	—
LD XY,dada 16	—
LD XY,(addr)	—
LD (xy+disp), data	—
LD (xy+disp),reg	—
LDD	—
LDDR	—
LDI	—

Z 80	I 8080
LDIR	—
NEG	—
NOP	NOP
OR data	ORI data
OR (HL)	ORA M
OR reg	ORA reg
OR (xy+disp)	—
OTDR	—
OTIR	—
OUT (C), reg	—
OUT (port), A	OUT port
OUTD	—
OUTI	—
POP pr	POP pr
POP xy	—
PUSH pr	PUSH pr
PUSH xy	—
RES b, (HL)	—
RES b, reg	—
RES b,(xy+disp)	—
RET	RET
RET C	RC
RET M	RM
RET NC	RNC
RET NZ	RNZ
RET P	RP
RET PE	RPE
RET PO	RPO
RET Z	RZ
RET I	—
RET N	—
RL (HL)	—
RL reg	—
RL (xy+disp)	—
RLA	RAL
RLC (HL)	—
RLC reg	—
RLC (xy+disp)	—
RLCA	RLC
RLD	—
RR (HL)	—

Z 80	I 8080
RR reg	—
RR (xy+disp)	—
RRA	RAR
RRC (HL)	—
RRC reg	—
RRC (xy+disp)	—
RRCA	RRC
RRD	—
RST n	RST n
SBC A, data	SBI data
SBC A,(HL)	SBB M
SBC A,reg	SBB reg
SBC A,(xy+disp)	—
SBC HL, rp	—
SCF	STC
SET b, (HL)	—
SET b,reg	—
SET b,(xy+disp)	—
SLA (HL)	—
SLA reg	—
SLA (xy+disp)	—
SRA (HL)	—
SRA reg	—
SRL (xy+disp)	—
SRL (HL)	—
SRL reg	—
SRL (xy+disp)	—
SUB data	SUI data
SUB (HL)	SUB M
SUB reg	SUB reg
SUB (xy+disp)	—
XOR data	XRI data
XOR (HL)	XRA M
XOR reg	XRA reg
XOR (xy+disp)	—

附　未正式发表的Z—80指令清单：（102条）

DD67	LD	IXH,A	FD67	LD	IYH,A
DD60	LD	IXH,B	FD60	LD	IYH,B
DD61	LD	IXH,C	FD61	LD	IYH,C
DD62	LD	IXH,D	FD62	LD	IYH,D
DD63	LD	IXH,E	FD63	LD	IYH,E
DD6F	LD	IXL,A	FD6F	LD	IYL,A
DD68	LD	IXL,B	FD68	LD	IYL,B
DD69	LD	IXL,C	FD69	LD	IYL,C
DD6A	LD	IXL,D	FD6A	LD	IYL,D
DD6B	LD	IXL,E	FD6B	LD	IYL,E
DD7C	LD	A,IXH	FD7C	LD	A,IYH
DD44	LD	B,IXH	FD44	LD	B,IYH
DD4C	LD	C,IXH	FD4C	LD	C,IYH
DD54	LD	D,IXH	FD54	LD	D,IYH
DD5C	LD	E,IXH	FD5C	LD	E,IYH
DD7D	LD	A,IXL	FD7D	LD	A,IYL
DD45	LD	B,IXL	FD45	LD	B,IYL
DD4D	LD	C,IXL	FD4D	LD	C,IYL
DD55	LD	D,IXL	FD55	LD	D,IYL
DD5D	LD	E,IXL	FD5D	LD	E,IYL
DD64	LD	IXH,IXH	FD64	LD	IYH,IYH
DD6C	LD	IXL,IXH	FD6C	LD	IYL,IYH
DD65	LD	IXH,IXL	FD65	LD	IYH,IYL
DD6D	LD	IXL,IXL	FD6D	LD	IYL,IYL
DD84	ADD	A,IXH	FD84	ADD	A,IYH
DD8C	ADC	A,IXH	FD8C	ADC	A,IYH
DD94	SUB	A,IXH	FD94	SUB	A,IYH
DD9C	SBC	A,IXH	FD9C	SBC	A,IYH
DDA4	AND	IXH	FDA4	AND	IYH
DDAC	XOR	IXH	FDAC	XOR	IYH
DDB4	OR	IXH	FDB4	OR	IYH
DDBC	CP	IXH	FDBC	CP	IYH
DD24	INC	IXH	FD24	INC	IYH
DD25	DEC	IXH	FD25	DEC	IYH
DD85	ADD	A,IXL	FD85	ADD	A,IYL
DD8D	ADC	A,IXL	FD8D	ADC	A,IYL
DD95	SUB	A,IXL	FD95	SUB	A,IYL
DD9D	SBC	A,IXL	FD9D	SBC	A,IYL
DDA5	AND	IXL	FDA5	AND	IYL
DDAD	XOR	IXL	FDAD	XOR	IYL
DDB5	OR	IXL	FDB5	OR	IYL
DDBD	CP	IXL	FDBD	CP	IYL
DD2C	INC	IXL	FD2C	INC	IYL

DD2D	DEC	IXL	FD2D	DEC	IYL
DD26 n	LD	IXH,n	FD26 n	LD	IYH,n
DD2En	LD	IXL,n	FD2En	LD	IYL,n
DDCB d 36	SLL	(IX+d)	FDCB d 36	SLL	(IY+d)
CB30	SLL	B	CB34	SLL	H
CB31	SLL	C	CB35	SLL	L
CB32	SLL	D	CB26	SLL	(HL)
CB33	SLL	E	CB37	SLL	A

其中IXL、IXH、IYL、IYH分别为变址寄存器IX、IY的低、高8位，IX指令操作码的首字节为DD,IY指令操作码的首字节为FD。通用寄存器移位指令操作码首字节为CB,变址寄存器所指单元移位指令操作码前二字节分别为 DDCB，FDCB。一共补充了三类指令：寄存器间相互传送、取立即数n及移位指令。移位指令是左移，CY←7—0←1，1移入第0位，第7位进入CY。了解这些指令能对机器码给出的某些系统软件或应用程序进行正确剖析，因为国外有些产品用了这些指令。

四、 AFSP程序由Z-80指令改写为I8080指令程序

搞清了Z—80与I8080指令的异同，再去修改用Z—80指令编写的标准算术函数程序是不难的。在TRS—80的BASIC Ⅱ固化的运算程序中，大概是借用了I8080系统中所用的BASIC程序的缘故，Z—80强化、扩充指令用得並不多，没有用到成组检索、位测指令，没有用到两个变址寄存器和辅助寄存器等，主要需要改动的部分有：

1)Z80的相对转移、相对条件转移指令改为I8080无条件转移和条件转移指令：

Z80指令		对应的8080指令
JR disp	——→	JMP addr
JR C,disp	——→	JC addr
JR Z,disp	——→	JZ addr
JR NC,disp	——→	JNC addr
JR NZ,disp	——→	JNZ addr

2)成组传送指令的改写

Z80程序	I8080程序
LD HL,addr$_1$	LXI H,addr$_1$
LD DE,addr$_2$	LXI D,addr$_2$
LD BC,count	LXI B,count
LDIR	LOOP: MOV A,M
	STAX D
	INX H
	INX D
	DCX B
	MOV A,B
	OR C
	JNZ LOOP

五、字符编码

标准子程序的输入数据和输出数据，符合计算机的输入、输出数据习惯，即与键盘、电传打字机、通讯设备和CRT字符显示终端使用的字符代码相同，每一字节均使用最高位为0的7位ASCII码。是打印字符还是控制字符由输入输出设备来区分，计算机对他们是一视同仁的。

从ASCII码表可以知道，十进数0～9表示为30～39，英文大写字母A～Z表示为41～5A。在本子程序中要用到的ASCII码字符有：

00——NULL（无效），用于表示数组的结束

0D——CR(回车)　　20——SPACE(空格)

2A——＊(乘)　　2B——＋（加）

2D——－(减)　　2E——．（点）

2F——/（除）

44——D，用于表示以10为底的双精度型数的指数

45——E，用于表示以10为底的单精度型数的指数

第二章　关于 AFSP 的综述

一．　AFSP是用Z80汇编语言编写的。

原型是TRS—80 BASIC Ⅱ中的内部函数即所谓标准子程序。笔者将这些子程序从BASIC Ⅱ解释程序中分离出来，根据单板机的实际情况，吸取了其它程序的优点进行了修改、编辑，并编制了一些复杂函数计算程序。考虑到各种单板机的存储区不同，以及应用对象不同，将Z80CPU中的辅助寄存器A′、F′、B′、C′、D′、E′、H′、和L′留给使用者，将RST n指令亦留给使用者。重新分配了数据存储区。考虑到本程序要能应用于以I 8080和I8085为CPU的微型机上，尽量地使用单字节和双字节指令，没有使用Z80特有的变址寄存器IX和IY。

二．　AFSP中数的类型及其在内存中的表示

在 **AFSP** 中，为了保证所需精度，快速运算，少占内存，它将数分成三类：整型、浮点单精度型、浮点双精度型，使用一个类型标志单元

①整型：

一个整数在内存中占两个字节，正数以原码表示，负数以补码表示。高字节（MSB）的最高位为符号位，当数值为正时MSB的最高位为“0”，数值为负时MSB的最高位为“1”。这样能表示的最大正数为7FFFH＝32767₊，能表示的最小负数为8000H＝－32768₊，因此整型数的范围为[-32768，＋32767]

例1：　＋375在内存中表示为：　MSB　LSB

0 1　7 7

－375在内存中表示为：　FE　8 9

②浮点单精度型：

在内存中占4个字节，其中阶码一个字节，尾数3个字节。所有浮点单精度型数均以规格化形式存放，所谓规格化就是小数点放在第一个不为0的数的前面。零值简单地存储为：指数为0，其它字节没有意义。尾数的最高位为符号位，但它不占用有效位。对于正数，尾数的最高位一定为“0”，对于负数，该位一定为“1”。**AFSP**中为了要在尾数字节中放入表示该数的符号，又不占去尾数的有效位，即不影响数的精度，对这形式上看是矛盾的两个要求作了巧妙地处理，这一处理的依据是浮点规格化形式已使尾数的最高字节的最高位必定是1。只要记住这一事实，就可以把这最高字节的最高位作为表示该数的符号位，规定正数的最高位为“0”，负数的最高位为“1”。在运算过程中，由程序来判别符号和对正数的尾数恢复最高位为“1”。这样做就可以：既能解决数的符号表示，又不减少数的有效位数。

数的浮点表示中，不管是单精度型还是双精度型，都是取用一个字节阶码，以80H（十进制128）为中间阶码，在规格化过程中将小数点移动的位数（16进制数）加上80H后作为此数的阶码。小数点向左移的位数为正，向右移的位数为负（补码）。

例1：　$A=1024.5_{+}=10000000000.1_{-}=.100000000001\times 2^{0BH}$

这个A的指数，即它的阶码为80H＋0BH＝8BH　A尾数的MSB是10000000，由于值是正的，最高位应改为“0”，因此A在内存中是这样存储的：

指数	MSB	NEXT MSB	LSB
8B	00	10	00

例2：　$B=-1024.5_{+}=-10000000000.1=-.100000000001\times 2^{0BH}$

同样B的阶码为8BH，B尾数的MSB是10000000，由于值是负的，最高位应为"1"。因此B在内存中是这样存储的：

指数	MSB	NEXT MSB	LSB
8 B	80	10	00

例3：　$C=0.05_{+}=0.00001100110011001100110\cdots\cdots=$

$$\doteq .11001100110011001100110\times 2^{FCH}$$

C的阶码为80H＋FCH＝7CH（不考虑进位）

C的MSB是11001100，由于值是正的，最高位为"0"，因此C在内存中是这样存储的：

指数	MSB	NEXT MSB	LSB
7C	4C	CC	CD

这样做既可以判别数的正负，又不减少有效数的表示范围。浮点单精度型在内存中以7位（指十进制的7位）有效数字存放，转换成十进制数以6位有效数字输出。一字节阶码中，7位二进制数表示范围最大为±127。而$2^{127}=1.70141\times 10^{38}$，$2^{-127}=5.87747\times 10^{-39}$，所以浮点单精度型在内存中能表示的最大范围为$[-1.70141\times 10^{38},\ +1.70141\times 10^{38}]$，而绝时值小于$5.87747\times^{-}10^{-39}$的数均作为0值，欲要取得更多位有效数可用双精度型表示。

③浮点双精度型：

在内存中占8个字节，其中阶码1个字节，尾数7个字节，阶码和尾数的表示方法与浮点单精度型相同。在内存中以17位有效数字存储，转换为十进制数以16位有效数字输出。浮双点精度型数在内存中能表示的最大范围为：$[-1.701411834544556\times 10^{38},+1.701411834544556\times 10^{38}]$，由于它与单精度型数都使用一个字节阶码，所以数的最大范围相同，只是数的有效位不同。

④类型标志：

上面已经说明了**AFSP**中将数据分为整型、浮点单精度型、浮点双精度型三种。在算术运算中两个运算数必须为相同类型，否则运算结果要出错。

AFSP中将RAM中的**TYPE**单元作为数的类型存放单元，存放目前正在进行计算的数(或自变量)、或转换结果(10进制转2进制)、或函数值的类型，也就是字节数。对于整型(TYPE)＝2；对于浮点单精度型(TYPE)＝4；对于浮点双精度型(TYPE)＝8，在后面的程序清单中笔者将2F48H单元作为TYPE单元，存放类型标志。因此，通过检验2F48H单元的内容就可以知道计算结果时数的类型。

三．　AFSP中关于出错的处理

在**AFSP**中将可能遇到的出错情况分为两类：以**ERR**和**EFF**表示。**ERR表示溢出**，包括下列情况：

①输出或导出的数值对计算机来说太大，例如：在调用10转2程序时，被转换的数超出了机内允许表示的范围，在做加、减、乘、除时得到的结果超出了机内允许表示的范围。

②自变量的定义域超出规定范围。例如：在调用计算e^{x}程序时自变量大于88.72284。

EFF表示类型错误或有非法调用函数