
Koyo

Value & Technology

可编程序控制器

NK0/NK1 系列 编程手册

[第四版]

光洋电子(无锡)有限公司

目录

第 1 章 简介.....	1
第 2 章 逻辑指令.....	3
2.1 逻辑指令的使用方法.....	3
2.2 逻辑运算开始 NO 接点 (LD)	9
2.3 逻辑运算开始 NC 接点 (LDN)	9
2.4 位逻辑运算开始 NO 接点 (BLD)	10
2.5 位逻辑运算开始 NC 接点 (BLDN)	10
2.6 逻辑或运算 NO 接点 (OR)	11
2.7 逻辑或运算 NC 接点 (ORN)	11
2.8 位逻辑或运算 NO 接点 (BOR)	12
2.9 位逻辑或运算 NC 接点 (BORN)	12
2.10 逻辑与运算 NO 接点 (AND)	13
2.11 逻辑与运算 NC 接点 (ANDN)	13
2.12 位逻辑与运算 NO 接点 (BAND)	14
2.13 位逻辑与运算 NC 接点 (BANDN)	14
2.14 逻辑组串联指令 (ANDLD)	15
2.15 逻辑组并联指令 (ORLD)	15
2.16 输出指令 (后面优先) (ZOUT)	16
2.17 位输出指令 (BOUT)	17
2.18 动作输出指令 (OUT)	18
2.19 高速输出指令 (ZOUTH)	19
2.20 非指令 (NOT)	20
2.21 一次扫描输出指令 (PD)	20
2.22 逻辑运算开始上升沿接点 (LDPD)	21
2.23 逻辑运算开始下降沿接点 (LDND)	21
2.24 逻辑或运算上升沿接点 (ORPD)	22
2.25 逻辑或运算下降沿接点 (ORND)	22
2.26 逻辑与运算上升沿接点 (ANDPD)	23
2.27 逻辑与运算开始下降沿接点 (ANDND)	23
2.28 线圈置位指令 (SET)	24
2.29 线圈复位指令 (RST)	24
2.30 BIT 置位指令 (BSET)	25
2.31 BIT 复位指令 (BRST)	25
第 3 章 比较接点指令.....	26
3.1 逻辑运算开始比较一致接点 (LDEQ/DLDEQ)	26
3.2 逻辑运算开始比较不一致接点 (LDNEQ/DLDNEQ)	26
3.3 逻辑或运算比较一致接点 (OREQ/DOREQ)	27
3.4 逻辑或运算比较不一致接点 (ORNEQ/DORNEQ)	27
3.5 逻辑与运算比较一致接点 (ANDEQ/DANDEQ)	28
3.6 逻辑与运算比较不一致接点 (ANDNEQ/DANDNEQ)	28
3.7 比较大等于逻辑运算开始接点 (LDGE/DLDGE)	29
3.8 比较小于逻辑运算开始接点 (LDNGE/DLDNGE)	29
3.9 比较大等于逻辑或运算接点 (ORGE/DORGE)	30
3.10 比较小于逻辑或运算接点 (ORNGE/DORNGE)	30
3.11 比较大等于逻辑与运算接点 (ANDGE/DANDGE)	31
3.12 比较小于逻辑与运算接点 (ANDNGE/DANDNGE)	31
第 4 章 直接指令.....	32
4.1 逻辑运算开始 NO 直接接点 (LDDI)	32
4.2 逻辑运算开始 NC 直接接点 (LDNDI)	32

4.3 逻辑或运算 NO 直接接点 (ORDI)	33
4.4 逻辑或运算 NC 直接接点 (ORNDI)	33
4.5 逻辑与运算 NO 直接接点 (ANDDI)	34
4.6 逻辑与运算 NC 直接接点 (ANDNDI)	34
4.7 直接输出指令 (ZDI)	35
4.8 OR 直接输出指令 (OUTDI)	35
4.9 线圈直接置位指令 (SETDI)	36
4.10 线圈直接复位指令 (RSTDI)	36
4.11 直接 16 位读入指令 (LDDW)	37
4.12 直接读入任意位指令 (LDDF)	38
4.13 直接 16 位输出指令 (OUTDW)	39
4.14 直接输出任意位指令 (OUTDF)	40
第 5 章 定时器、计数器、移位寄存器指令	41
5.1 定时器的使用方法	41
5.2 0.1 秒定时器 (TMR) 和 0.01 秒定时器 (HTMR)	42
5.3 0.1 秒累积定时器 (ATMR) 和 0.01 秒累积定时器 (AHTMR)	45
5.4 计数器的使用方法	48
5.5 加算计数器(带复位端) (CNT)	49
5.6 加算计数器(不带复位端) (GCNT)	52
5.7 加减计数器 (UDCNT)	55
5.8 定时器、计数器复位指令 (RSTTC)	58
5.9 移位寄存器指令 (SR)	59
第 6 章 累加器/数据堆栈和输出指令	60
6.1 累加器、堆栈的使用方法	60
6.2 读入指令 (LDW)	66
6.3 读入指令 (LDS)	67
6.4 读入指令 (LDD)	68
6.5 读入指令 (LDC)	69
6.6 任意位读入指令 (LDF)	70
6.7 寄存器地址读入指令 (LDR)	72
6.8 索引 16 位读入指令 (LDIX)	73
6.9 程序块索引读入指令 (LDSIX)	74
6.10 实数读入指令 (RLDD)	76
6.11 实数读入指令 (RLDC)	76
6.12 16 位写入指令 (OUTW)	77
6.13 32 位写入指令 (OUTD)	78
6.14 任意位写入指令 (OUTF)	79
6.15 索引 16 位写入指令 (OUTIX)	80
6.16 低 8 位写入指令 (OUTL)	81
6.17 高 8 位写入指令 (OUTM)	81
6.18 数据堆栈弹出指令 (POP)	82
第 7 章 累加器逻辑指令	84
7.1 16 位逻辑与指令 (ANDW)	84
7.2 32 位逻辑与指令 (ANDD)	86
7.3 32 位常数逻辑与指令 (ANDC)	88
7.4 任意位长逻辑与指令 (ANDF)	89
7.5 堆栈逻辑与指令 (SAND)	91
7.6 16 位逻辑或指令 (ORW)	92
7.7 32 位逻辑或指令 (ORD)	93
7.8 32 位常数逻辑或指令 (ORC)	95
7.9 任意位长逻辑与指令 (ORF)	96

7.10 堆栈逻辑或指令 (SOR)	98
7.11 16 位逻辑异或指令 (XORW)	99
7.12 32 位逻辑异或指令 (XORD)	100
7.13 32 位常数逻辑异或指令 (XORC)	102
7.14 任意位长逻辑异或指令 (XORF)	103
7.15 堆栈逻辑异或指令 (SXOR)	105
7.16 16 位比较指令 (CMPR)	106
7.17 16 位二进制比较指令 (BCMPR)	106
7.18 32 位比较指令 (CMPRD)	107
7.19 32 位二进制比较指令 (BCMPRD)	107
7.20 8 位常数比较指令 (CMPRC)	108
7.21 8 位常数二进制比较指令 (BCMPRC)	108
7.22 任意位长比较指令 (CMPRF)	109
7.23 堆栈比较指令 (SCMPR)	111
7.24 堆栈二进制比较指令 (BSCMPR)	111
7.25 实数比较指令 (RCMPR)	112
7.26 实数常数比较指令 (RCMPRC)	113
第 8 章 数据处理指令的解说.....	115
8.1 数据的形态.....	115
8.2 数据的指定.....	117
8.3 数据处理的基本形式.....	121
8.4 浮点数的形式.....	130
第 9 章 算术运算指令.....	131
9.1 4 位 BCD 加法指令 (ADD)	131
9.2 8 位 BCD 加法指令 (ADDD)	133
9.3 8 位 BCD 常数加法指令 (ADDC)	135
9.4 实数加法指令 (RADD)	136
9.5 实数常数加法指令 (RADDC)	137
9.6 4 位 BCD 减法指令 (SUB)	139
9.7 8 位 BCD 减法指令 (SUBD)	141
9.8 8 位 BCD 常数减法指令 (SUBC)	143
9.9 实数减法指令 (RSUB)	144
9.10 实数常数减法指令 (RSUBC)	145
9.11 4 位 BCD 乘法指令 (MUL)	147
9.12 4 位 BCD 常数乘法指令 (MULS)	148
9.13 8 位 BCD 乘法指令 (MULD)	149
9.14 8 位 BCD 常数乘法指令 (MULC)	151
9.15 实数乘法指令 (RMUL)	152
9.16 实数常数乘法指令 (RMULC)	153
9.17 4 位 BCD 除法指令 (DIV)	155
9.18 4 位 BCD 常数除法指令 (DIVS)	156
9.19 8 位 BCD 除法指令 (DIVD)	157
9.20 8 位 BCD 常数除法指令 (DIVC)	158
9.21 实数除法指令 (RDIV)	159
9.22 实数常数除法指令 (RDIVC)	160
9.23 BCD 加 1 指令 (INCR)	162
9.24 BCD 减 1 指令 (DECR)	162
9.25 16 位 (BIN) 加法指令 (BADD)	163
9.26 16 位常数 (BIN) 加法指令 (BADDS)	165
9.27 32 位 (BIN) 加法指令 (BADDD)	166
9.28 32 位常数 (BIN) 加法指令 (BADDC)	167

9.29 16 位 (BIN) 减法指令 (BSUB)	168
9.30 16 位常数 (BIN) 减法指令 (BSUBS)	169
9.31 32 位 (BIN) 减法指令 (BSUBD)	170
9.32 32 位常数 (BIN) 减法指令 (BSUBC)	171
9.33 16 位 (BIN) 乘法指令 (BMUL)	172
9.34 16 位 (BIN) 常数乘法指令 (BMULS)	173
9.35 32 位 (BIN) 乘法指令 (BMULD)	174
9.36 16 位 (BIN) 除法指令 (BDIV)	175
9.37 16 位 (BIN) 常数除法指令 (BDIVS)	176
9.38 32 位 (BIN) 除法指令 (BDIVD)	177
9.39 BIN 加 1 指令 (BINC)	178
9.40 BIN 减 1 指令 (BDEC)	178
9.41 任意位长 (BCD) 加法指令 (ADDF)	179
9.42 任意位长 (BCD) 减法指令 (SUBF)	181
9.43 任意位长 (BCD) 乘法指令 (MULF)	183
9.44 任意位长 (BCD) 除法指令 (DIVF)	185
9.45 堆栈加法指令 (SADD)	187
9.46 堆栈减法指令 (SSUB)	189
9.47 堆栈乘法指令 (SMUL)	191
9.48 堆栈除法指令 (SDIV)	193
9.49 堆栈二进制加法指令 (SBADD)	195
9.50 堆栈二进制减法指令 (SBSUB)	197
9.51 堆栈二进制乘法指令 (SBMUL)	199
9.52 堆栈二进制除法指令 (SBDIV)	201
9.53 实数正弦指令 (RSIN)	203
9.54 实数余弦指令 (RCOS)	203
9.55 实数正切指令 (RTAN)	203
9.56 实数反正弦指令 (RASIN)	203
9.57 实数反余弦指令 (RACOS)	203
9.58 实数反正切指令 (RATAN)	203
9.59 实数平方根指令 (RSQRT)	203
9.60 BCD 数正弦指令 (SIN)	205
9.61 BCD 数余弦指令 (COS)	205
9.62 BCD 数正切指令 (TAN)	205
9.63 BCD 数反正弦指令 (ASIN)	205
9.64 BCD 数反余弦指令 (ACOS)	205
9.65 BCD 数反正切指令 (ATAN)	205
9.66 BCD 数平方根指令 (SQRT)	205
9.67 幂指令 (RPOW)	206
9.68 指数指令 (REXP)	207
9.69 自然对数指令 (RLN)	208
9.70 常用对数指令 (RLG)	209
第 10 章 位操作指令	210
10.1 0N 位求和指令 (SUM)	210
10.2 左移指令 (SHFL)	211
10.3 右移指令 (SHFR)	213
10.4 循环左移指令 (ROTL)	215
10.5 循环右移指令 (ROTR)	217
10.6 编码指令 (ENCO)	219
10.7 译码指令 (DECO)	221
第 11 章 数据变换指令	222

11.1 BIN 码变换指令 (BIN)	222
11.2 BCD 码变换指令 (BCD)	224
11.3 取反指令 (INV)	226
11.4 十进制补码变换 (BCDCPL)	227
11.5 二进制到实数变换指令 (REAL)	228
11.6 实数到二进制数变换指令 (INT)	230
11.7 实数弧度变换指令 (RRAD)	232
11.8 实数角度变换指令 (RDEG)	232
11.9 BCD 数弧度变换指令 (RAD)	233
11.10 BCD 数角度变换指令 (DEG)	233
11.11 ASC II → HEX 码变换指令 (ATH)	234
11.12 HEX → ASC II 码变换指令 (HTA)	236
11.13 7 段译码指令 (SEG)	238
11.14 GRAY → BCD 码变换指令 (GRAY)	240
11.15 位替换指令 (SFLDGT)	242
第 12 章 表检索指令	244
12.1 数据组传送指令 (MOVE)	244
12.2 程序存储器—数据寄存器之间的数据传送指令 (MOVMC)	245
12.3 数据标记地址读出指令 (LDLBL)	245
12.4 登记数据寄存器传送指令 (MOVAS)	248
12.5 任意位置位指令 (BITSET)	249
12.6 任意位复位指令 (BITRST)	249
12.7 同一数据成组写入指令 (FILL)	251
12.8 同一数据检索指令 (SRCH)	253
12.9 数据级别分类指令 (CLASS)	255
12.10 指针加法取出指令 (TTD)	257
12.11 指针减法取出指令 (RFB)	260
12.12 指针加法存入指令 (STT)	263
12.13 堆栈上托取出指令 (RFT)	266
12.14 堆栈下推存入指令 (ATT)	269
12.15 表左移指令 (TSHFL)	272
12.16 表右移指令 (TSHFR)	272
12.17 ACC 逻辑与传送指令 (ANDMOV)	274
12.18 ACC 逻辑或传送指令 (ORMOV)	274
12.19 ACC 逻辑异或传送指令 (XORMOV)	274
12.20 多字节数据检索指令 (BSRCH)	276
12.21 交换指令 (SWAP)	277
第 13 章 日历/时钟指令 (NK0 不支持)	278
13.1 日期设定指令 (DATE)	278
13.2 时钟设定指令 (TIME)	279
第 14 章 CPU 控制指令	280
14.1 空操作指令 (NOP)	280
14.2 程序结束指令 (END)	280
14.3 扫描停止指令 (STOP)	280
14.4 监视定时器复位指令 (WDOGR)	281
第 15 章 程序控制指令	282
15.1 跳转指令 (GOTO)	282
15.2 跳转目标指令 (GLBL)	282
15.3 循环执行开始指令 (FOR)	283
15.4 循环体最后指令 (NEXT)	283
15.5 子程序调用指令 (CAL)	284

15.6 子程序开始标记 (CLBL)	284
15.7 无条件复归指令 (CEND)	284
15.8 条件复归指令 (RET)	284
15.9 新母线开始指令 (MLS)	286
15.10 母线复归指令 (MLR)	286
第 16 章 中断指令 (NK0 不支持)	288
16.1 中断子程序指令 (ILBL)	288
16.2 中断子程序无条件复归指令 (IEND)	289
16.3 中断子程序条件复归指令 (RETI)	289
16.4 中断许可指令 (INE)	289
16.5 中断禁止指令 (INH)	289
16.6 硬件中断举例.....	290
16.7 软件中断举例.....	291
第 17 章 通讯指令.....	292
17.1 读出指令 (RX)	292
17.2 写入指令 (WX)	294
第 18 章 信息指令.....	296
18.1 数据区标号指令 (DLBL)	296
18.2 ASCII 数据登记指令 (ACON)	296
18.3 数值数据登记指令 (NCON)	296
第 19 章 Modbus RTU 指令.....	297
19.1 Modbus 网络读指令 (MRX)	297
19.2 Modbus 网络写指令 (MWX)	299
19.3 KPPsoft 软件中使用 MRX/MWX 的例子	301
第 20 章 ASCII 指令.....	304
20.1 ASCII 输入指令 (AIN)	304
20.2 ASCII 码打印指令 (PRINTV)	308
第 21 章 定位指令 (NK0 不支持)	309
21.1 原点搜索指令 (MT_ORG) (IB-2200)	309
21.2 绝对位置定位指令 (MT_ABSC) (IB-2201)	315
21.3 相对位置定位指令 (MT_INCC) (IB-2202)	319
21.4 速度控制指令 (MT_SPDC) (IB-2203)	322
21.5 中断定位指令 (MT_INTC) (IB-2204)	324
21.6 当前位置变更指令 (MT_PSET) (IB-2212)	326
21.7 复位指令 (MT_RST) (IB-2210)	327
21.8 停止指令 (MT_STPM) (IB-2211)	328
21.9 多段定位指令 (MT_MTL) (IB-2214)	329
第 22 章 级式编程方法.....	333
22.1 级式编程介绍.....	333
22.2 画状态转移图.....	334
22.3 级转移指令的使用方法.....	338
22.4 级式语言编程举例：开/关灯控制.....	339
22.5 级式语言四步编程.....	340
22.6 级式编程举例：车库门开关控制器.....	341
22.7 级式程序设计注意事项.....	346
22.8 并行进程的概念.....	350
22.9 大型程序的管理.....	352
第 23 章 级式指令.....	354
23.1 级登记指令 (SG)	354
23.2 初始级登记指令 (ISG)	355
23.3 级转移指令 (JMP)	355

23.4 条件不成立时级转移指令 (NJMP)	356
23.5 合流级登记指令 (CV)	357
23.6 合流级转移指令 (CVJMP)	357
23.7 级组请求指令 (BREQ)	359
23.8 级组开始指令 (BSTART)	359
23.9 级组结束指令 (BEND)	359
第24章 凸轮控制介绍.....	361
24.1 简介.....	361
24.2 步转移.....	363
24.3 凸轮操作概述.....	366
24.4 凸轮控制技术.....	368
第25章 凸轮控制指令.....	370
25.1 时间驱动型凸轮控制指令 (DRUM)	370
25.2 时间/事件驱动型凸轮控制指令 (EDRUM)	373
25.3 带掩膜事件驱动型凸轮控制指令 (MDRMD) (离散点输出)	376
25.4 带掩膜事件驱动型凸轮控制指令 (MDRMW) (字输出)	379
第26章 IBOX 指令.....	382
26.1 16 位复制指令 (MOVEW) (IB-200)	382
26.2 32 位复制指令 (MOVED) (IB-201)	383
26.3 反转开关指令 (PONOFF) (IB-300)	384
26.4 ON 延迟定时器指令 (ONDTMR) (IB-301)	385
26.5 OFF 延迟定时器指令 (OFFDTMR) (IB-302)	386
26.6 1 周期输出指令 (ONESHOT) (IB-303).....	387
26.7 高/低报警-BIN 指令 (HILOALB) (IB-401)	388
26.8 过滤超时时间 (BIN) 指令 (FILTERB) (IB-402)	390
26.9 12 位模拟量二进制变换指令 (ANSCLB) (IB-403)	391
26.10 高/低报警设定 (BCD) 指令 (HILOAL) (IB-421)	392
26.11 过滤超时时间 (BCD) 指令 (FILTER) (IB-422)	394
26.12 12 位模拟量 BCD 转换指令 (ANSCL) (IB-423)	395
26.13 BIN 算术运算指令 (MATHBIN (IB-501) /MATHBIND (IB-551))	396
26.14 BIN 数求和指令 (SUMBIN) (IB-502)	397
26.15 BIN 平方指令 (SQUAREB) (IB-503)	398
26.16 BCD 算术运算指令 (MATHBCD (IB-521) /MATHBCDD (IB-550)	399
26.17 BCD 数求和指令 (SUMBCD) (IB-522)	400
26.18 BCD 平方指令 (SQUARE) (IB-523)	401
26.19 实数算术运算指令 (MATHR) (IB-541)	402
26.20 实数求和指令 (SUMR) (IB-542)	403
26.21 实数平方指令 (SQUARER) (IB-543)	404
26.22 16 位 BCD-带小数点实数转换指令 (BCDTOR) (IB-560)	405
26.23 实数-16 位 BCD 转换指令 (RTOBCD) (IB-561)	406
26.24 32 位 BCD-带小数点实数转换指令 (BCDTORD) (IB-562)	407
26.25 实数-32 位 BCD 转换指令 (RTOBCDD) (IB-563)	408
26.26 CRC 校验码运算指令 (CRCXOR) (IB-552)	409
附录 1 NK1 功能存储器表.....	410
附录 2 ASCII 转换表.....	411

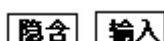
第 1 章 简介

感谢购买使用本公司 NK0/NK1 系列 PLC 产品。

NK0/NK1 指令集丰富，功能完善，可以实现逻辑控制、数据运算（包括浮点数/三角函数运算）、程序执行流程控制、通信控制、数据表格、PLC 功能设定等多种不同类型的功能。本手册向您介绍了 NK0/NK1 指令集指令的使用方法。本资料后述的各指令介绍内容中，括号中的内容就是各指令的名称。

NK0/NK1 的编程软件是 KPPsoft，各指令在 KPPsoft 中的输入/使用方式，请参看 KPPsoft 的有关说明资料和软件在线帮助文件。

在 KPPsoft 软件中输入 NK0/NK1 指令的方式有 2 种：隐含使用和直接输入。本手册的每个指令都被标示出在 KPPsoft 中是隐含使用还是直接输入使用。“隐含”表示隐含使用，“输入”表示可以直接输入，见下图。



本手册介绍的许多指令虽然在 KPPsoft 中不能使用键盘输入，而是隐含在梯形图符号里面；但是，当程序编写和编译完成后，可以在语句视图中看到这些指令。

本手册前面的 A 到 C 版本是专门针对 NK1 系列 PLC 指令的说明资料。从 D 版本开始，增加了对 NK0 系列产品支持的描述。

NK0 系列 PLC 是基于 NK1 系列 PLC 的简易微型化 PLC 系列，其指令体系基本兼容于 NK1 的指令体系，2 者相同的指令其功能是完全一样的，在 KPPsoft 中的输入方式也是完全一样的。但请注意，下面列出的这些指令，NK0 不支持，使用时请十分注意！

序号	指令	NK0	序号	指令	NK0
1	DATE	×	9	MT_ORG	×
2	TIME	×	10	MT_ABSC	×
3	ILBL	×	11	MT_INCC	×
4	RETI	×	12	MT_SPDC	×
5	IEND	×	13	MT_INTC	×
6	INE	×	14	MT_PSET	×
7	INH	×	15	MT_RSTC	×
8	EMAIL *注	×	16	MT_STPM	×
			17	MT_TABL	

注：1) ‘×’ 表示不支持。2) 有关 Email 指令的说明，请参见《NK1 用户手册》资料。

另外，请注意，需要在版本 V1.6.5.0 后的 KPPsoft 软件，才支持 NK0 系列 PLC。

本手册使用的提示标记：



左边的“记事本”图标表明紧随其后的段落将是一条特别的**注意事项**。



左边的“感叹号”图标表示**警告信息**。此信息可以防止伤害、财产损失，甚至人员伤亡。

如果你有有关本手册的情况需要与我们联系，请首先确定手册的版本号！

资料名称：《NK0/NK1 编程手册》（原《NK1 编程手册》）

资料编号	编制日期	内容说明
KEW-M2512A	2015 年 3 月	原稿第一版试用
KEW-M2512A1	2015 年 6 月	增加 4 条二进制比较指令：（固件 20150603） BCMPR, BCMPRD, BSCMPR, BCMPRC （需要 V1.0.1.10 以上版本 KPPSoft 软件支持）
KEW-M2512B	2016 年 3 月	增加新支持的 IBOX 指令（需要 V1.1.0.9 以上版本 KPPSoft 软件支持）
KEW-M2512B1	2016 年 5 月	增加浮点数形式的介绍；Modbus RTU 指令及定位指令相关内容的修正
KEW-M2512C	2019 年 7 月	增加 32 位比较接点指令 12 条（DLD/DAND/DOR 系指令），32 位数据乘除运算指令 4 条（BMULD/MDIVD/BMULC/BDIVC），IBOX 指令 3 条（MATHBCDD/MATHBIND/CRCXOR）（固件 20190716） （需要 V1.6.4.7 以上版本 KPPSoft 软件支持）
KEW-M2512D	2020 年 5 月	增加对 NK0 系列 PLC 的支持说明

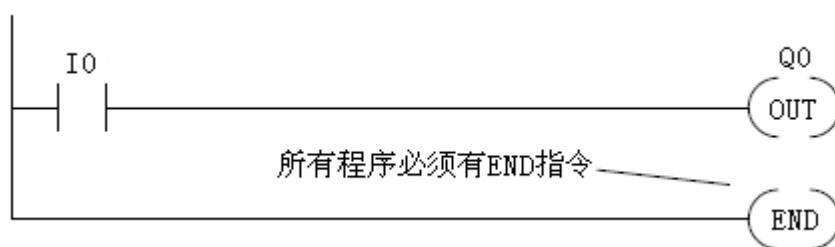
第 2 章 逻辑指令

2.1 逻辑指令的使用方法

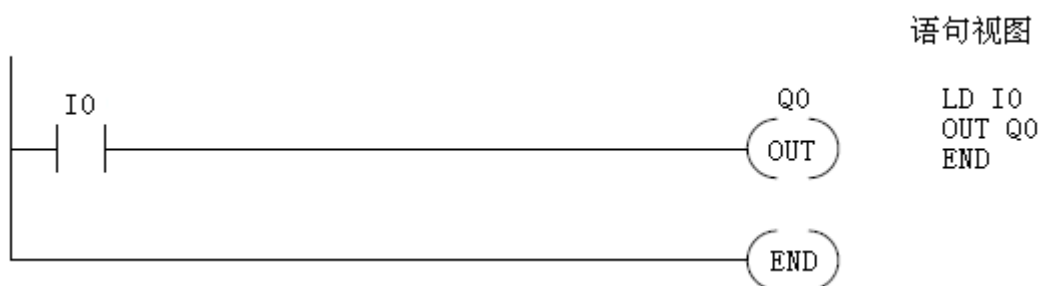
为什么大多数PLC厂家提供的扫描时间都是1K逻辑程序的扫描时间呢？这是因为大多数程序使用大量的逻辑指令。这些指令通常是非常简单的指令，用于在各种串联和并联回路中添加输入和输出接点。我们的KPPsoft软件也是这样，它使用图形符号来编程，因此不需要记住所有的指令。

2.1.1 END 指令

所有NK1程序需要以END指令结束，以此来通知CPU程序已结束。通常情况下，END指令后的指令不会被执行，子程序、中断程序等除外，相关内容后面会有详细介绍。

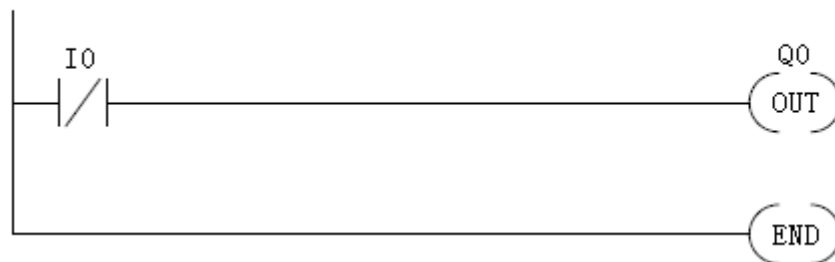


下面的例子是一行包含接点和线圈的程序。接点是 LD 指令，线圈是 OUT 指令。



2.1.2 常闭接点

常闭接点也很常用，常闭接点指令是 LDN，见下面的例子。

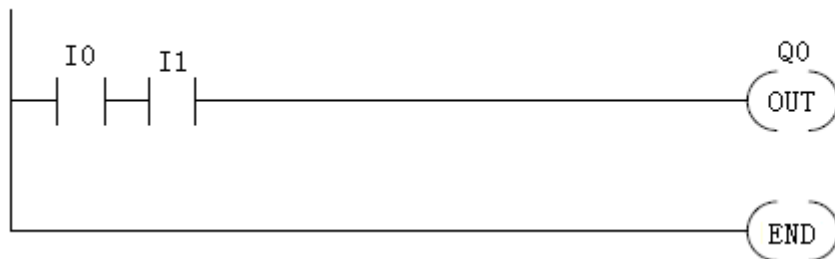


语句视图

```
LDN I0
OUT Q0
END
```

2.1.3 接点串联

使用 AND 指令可将两个及以上的接点串联，见下面的例子。

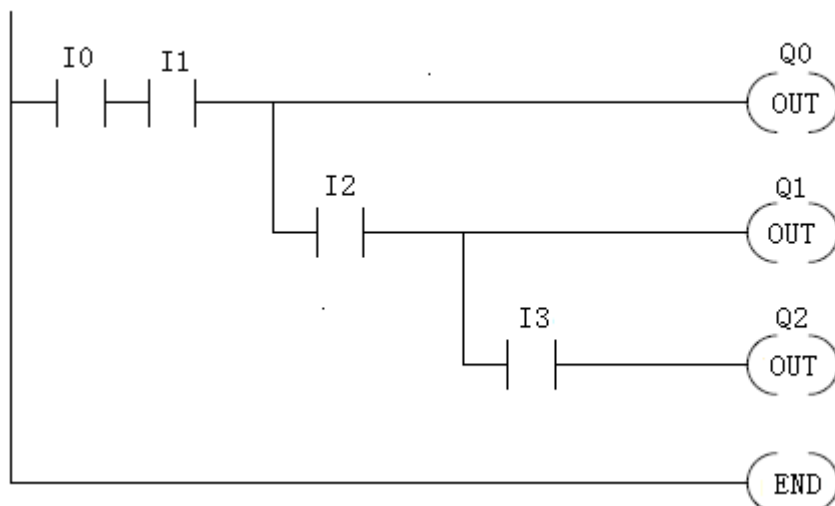


语句视图

```
LD I0
AND I1
OUT Q0
END
```

2.1.4 中间线输出

有时需要使用中间线输出得到其它条件的额外输出，见下面的例子，用 AND 指令实现。

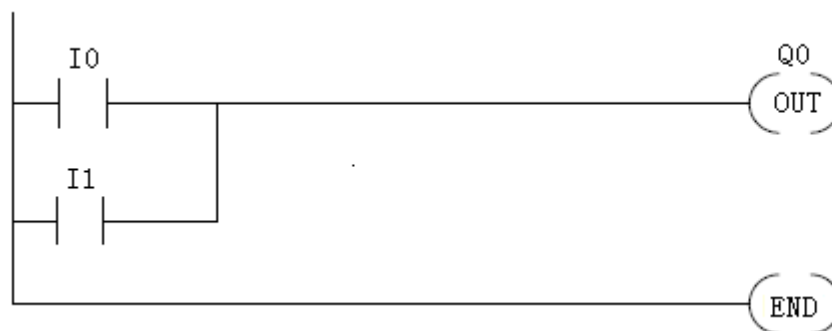


语句视图

```
LD I0
AND I1
OUT Q0
AND I2
OUT Q1
AND I3
OUT Q2
END
```

2.1.5 接点并联

使用 OR 指令可将接点并联，见下面的例子。

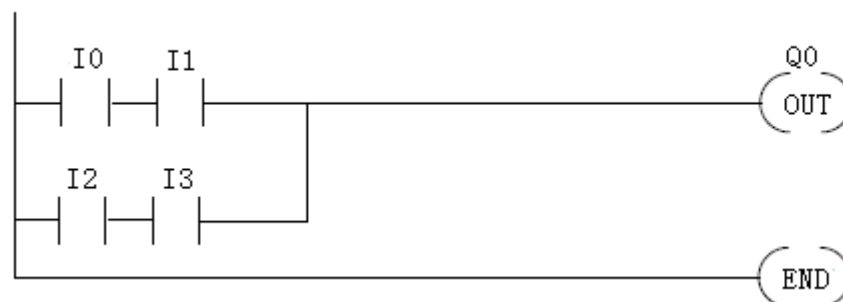


语句视图

```
LD I0
OR I1
OUT Q0
END
```

2.1.6 串联支路并联

使用 ORLD 指令可将串联的支路并联，见下面的例子。

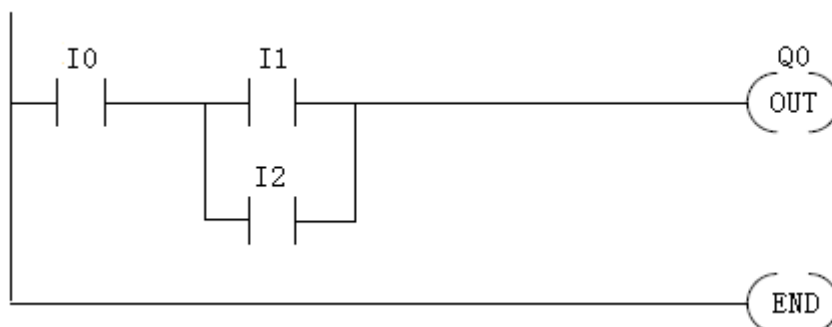


语句视图

```
LD I0
AND I1
LD I2
AND I3
ORLD
OUT Q0
END
```

2.1.7 串联中加入并联支路

使用 ANDLD 指令可在串联回路中加入并联支路，见下面的例子。

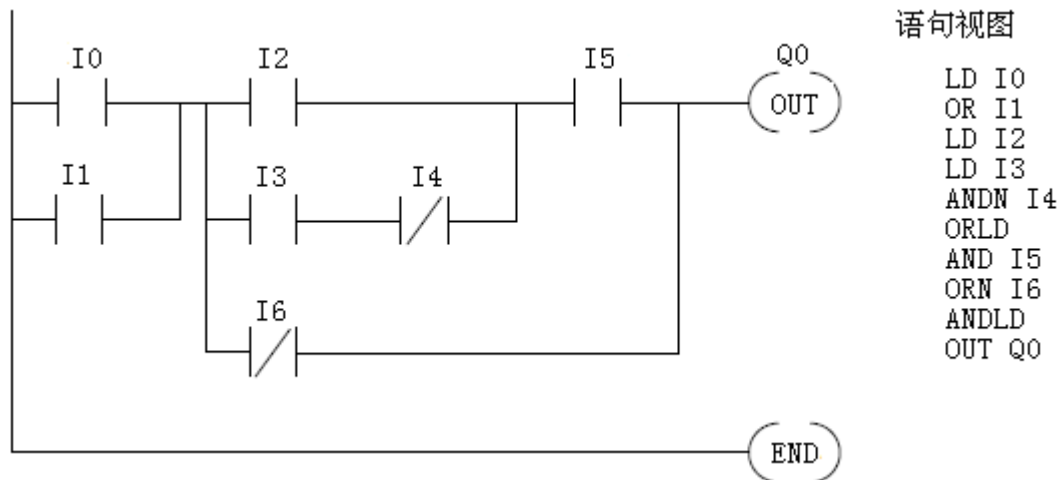


语句视图

```
LD I0
LD I1
OR I2
ANDLD
OUT Q0
END
```

2.1.8 网络组合

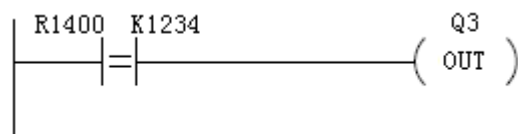
可以将各种串联和并联支路组合来解决各种应用问题，下面是一个简单的例子。



2.1.9 比较接点

比较接点是使用一个逻辑接点将两个 4 位 BCD/十六进制数进行比较，比较接点有：比较一致接点，比较不一致接点，大于等于接点和小于等于接点。

下面的例子中，当寄存器 R1400 中的数据等于 BCD 常数 1234 时，Q3 被接通。



2.1.10 逻辑堆栈

一程序的指令数是有限制的，这是因为 NK1 CPU 使用一个 8 级逻辑堆栈来存放逻辑指令，逻辑堆栈是一个逻辑指令的临时存放区域。每输入一个 LD 指令，这个指令就被存放在堆栈的顶端，堆栈中的其它级被压入下一级中。ANDLD 和 ORLD 指令将各级的指令组合起来。由于逻辑堆栈只有 8 级，如果 CPU 检测到程序使用了超出 8 级的逻辑堆栈，就会产生错误。

可能你会说：“我使用 KPPsoft 编程，这样我就没必要知道逻辑堆栈是如何工作的。”这样是不对的，KPPsoft 中，虽然可用图形标志创建程序，但是 CPU 对堆栈的限制也是一样的。如果超出这个限制，程序编译的时候将会产生错误。

下面的例子演示了逻辑堆栈是如何工作的。

The diagram shows a ladder logic network. It starts with a vertical line on the left. From this line, a branch goes down to a normally open contact labeled I0. The main line continues to a normally open contact labeled I1. After I1, the line branches again: one branch goes down to a normally open contact labeled I2, which then connects to another normally open contact labeled I3. The other branch from the point after I1 goes to a normally open contact labeled I4. The lines from I3 and I4 merge, then go to a normally open contact labeled I5. After I5, the line goes to a coil labeled Q0 (OUT). Various instructions are labeled along the lines: LD I0 at the start, LD I1 after the first branch, ORLD at the merge point after I4, AND I4 before the coil, ANDLD at the start of the I2 branch, LD I2 before the I3 contact, AND I3 before the I3 contact, and OR I5 before the I5 contact.

LD I0

1	LD I0
2	
3	
4	

LD I1

1	LD I1
2	LD I0
3	
4	

LD I2

1	LD I2
2	LD I1
3	LD I0
4	

AND I3

1	I2 AND I3
2	LD I1
3	LD I0
4	

ORLD

1	I1 OR (I2 AND I3)
2	LD I0
3	

AND I4

1	I4 AND [I1 OR (I2 AND I3)]
2	LD I0
3	

OR I5

1	NOT I5 OR I4 AND [I1 OR (I2 AND I3)]
2	LD I0
3	

ANDLD

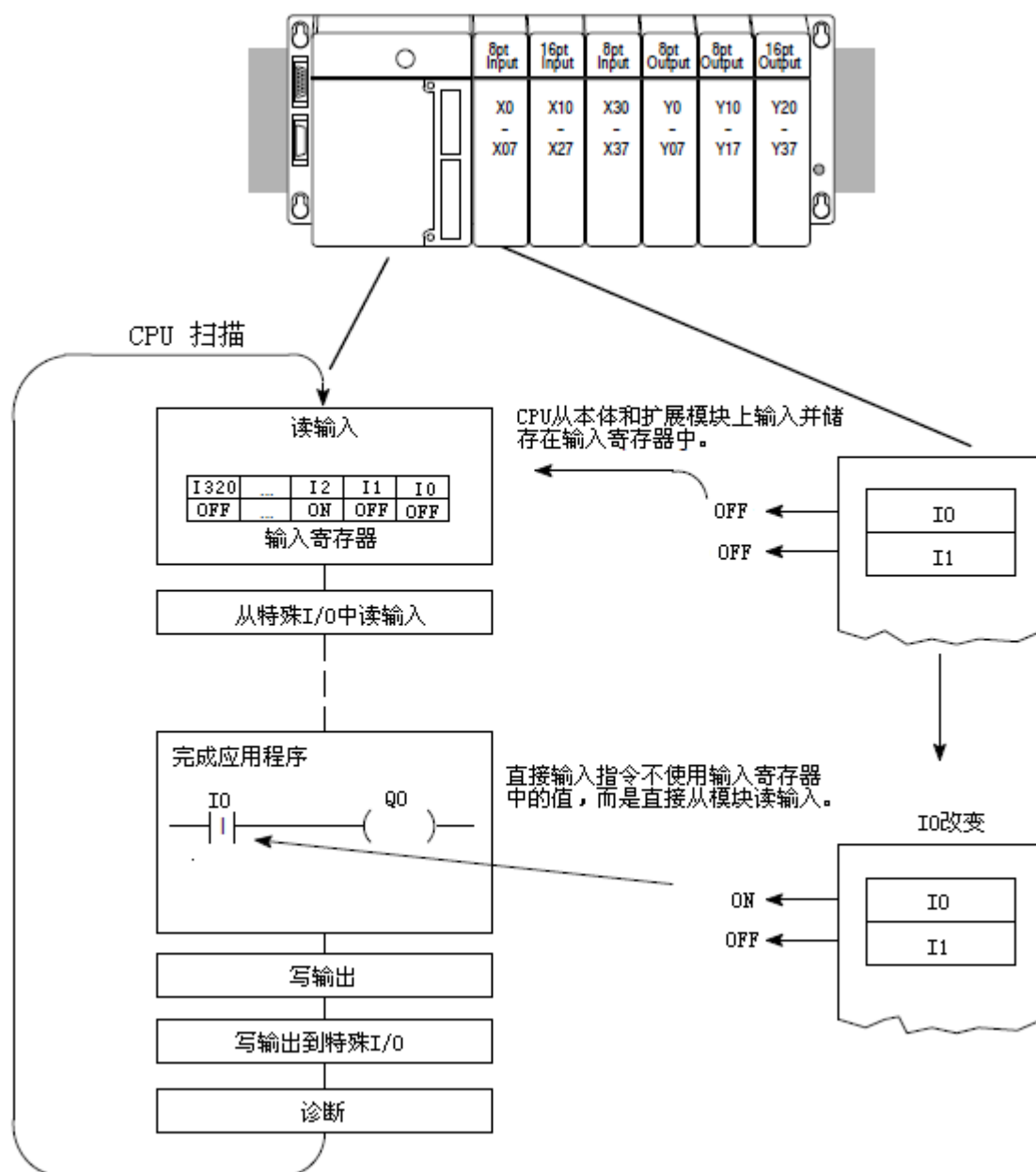
1	I0 AND (NOT I5 OR I4) AND [I1 OR (I2 AND I3)]
2	
3	

2.1.11 直接指令

NK1 CPU 完成一个操作循环通常只需要几毫秒，但是有些应用中，几毫秒都不能等待。NK1 CPU 提供了特殊的逻辑指令——直接输入和输出指令，在程序执行时允许立即读输入和写输出。直接指令会使程序执行时间加长，因为 CPU 读写输入输出时程序执行被中断。

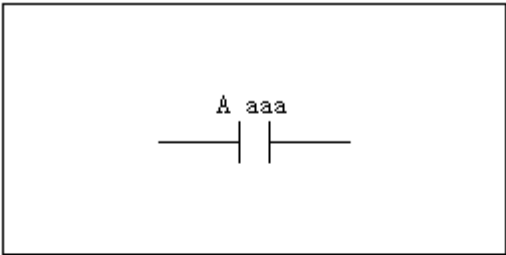


注意：虽然直接输入指令读取模块的最新状态值，但是只有一个指令是使用这个最新状态值，寄存器的数据并没有被更新，因此，其它常规指令仍然使用寄存器中的状态值，直接输入指令会再次采集到模块的最新状态值。直接输出指令将输出写入模块并且更新寄存器的数据。



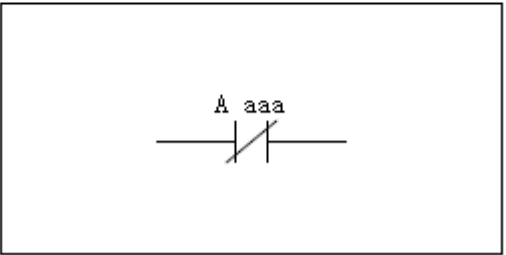
2.2 逻辑运算开始 NO 接点（LD）

隐含 LD 指令是一个常开触点，用于开始一程序或是增加一个支路。触点的状态同寄存器或存储地址中的相应位状态保持一致。



2.3 逻辑运算开始 NC 接点（LDN）

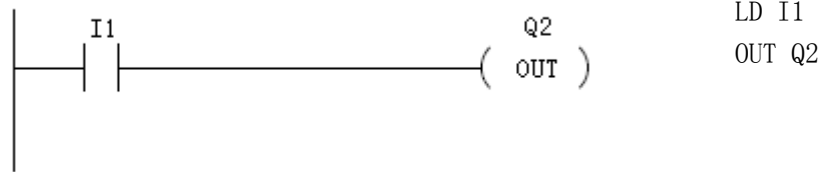
隐含 LDN 指令是一个常闭触点，用于开始一程序或是增加一个支路。触点的状态同寄存器或存储地址中的相应位状态相反。



操作数类型		NK1 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777
级	S	0-1777
定时器	T	0-777
计数器	C	0-777
特殊线圈	SP	0-1777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

在下面的 LD 指令例子中，当 I1 为 ON 时，输出 Q2 被接通。

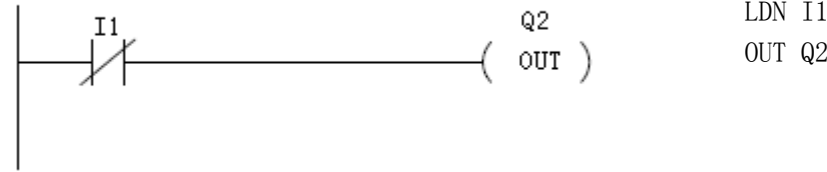
语句视图



LD I1
OUT Q2

在下面的 LDN 指令例子中，当 I1 为 OFF 时，输出 Q2 被接通。

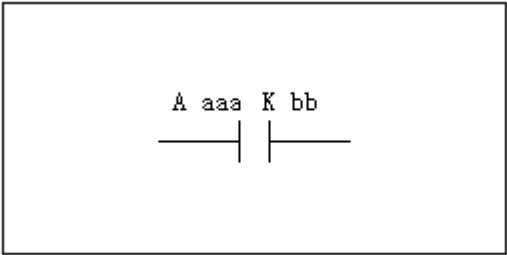
语句视图



LDN I1
OUT Q2

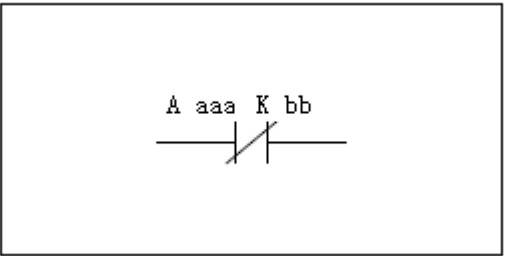
2.4 位逻辑运算开始 NO 接点（BLD）

隐含 BLD 指令是一个常开触点，用于开始一程序或是增加一个支路。触点的状态同存储地址中的相应位状态保持一致。



2.5 位逻辑运算开始 NC 接点（BLDN）

隐含 BLDN 指令是一个常闭触点，用于开始一程序或是增加一个支路。触点的状态同存储地址中的相应位状态相反。



操作数类型		NK1 范围	
	A	aaa	bb
寄存器	R	所有（附录 1）	BCD，0-15
指针	P	所有（附录 1）	BCD，0-15

在下面的 BLD 指令例子中，当寄存器 R1400 的位 12 为“1”时，输出 Q2 被接通。



语句视图

BLD R1400 K12
OUT Q2

在下面的 BLDN 指令例子中，当寄存器 R1400 的位 12 为“0”时，输出 Q2 被接通。



语句视图

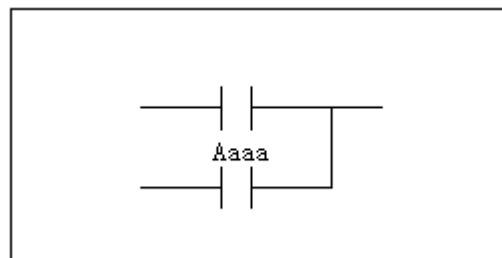
BLD R1400 K12
OUT Q2



注意：使用 KPPsoft 软件编程时，BLD 和 BLDN 的图标分别是  和 .

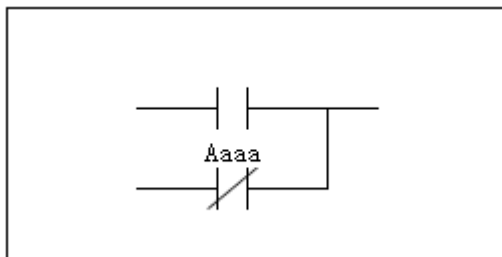
2.6 逻辑或运算 NO 接点（OR）

隐含 OR 指令将一个触点同另一个并联的常开接点逻辑或，触点的状态同指定功能存储器的状态相同。



2.7 逻辑或运算 NC 接点（ORN）

隐含 ORN 指令将一个触点同另一个并联的常闭触点逻辑或，触点的状态同指定功能存储器的状态相反。



操作数类型		NK1 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777
级	S	0-1777
定时器	T	0-777
计数器	C	0-777
特殊线圈	SP	0-1777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

下面的 OR 指令例子中，当输入 I1 或 I2 为 ON 时，输出 Q5 被接通。

语句视图



```
LD I1
OR I2
OUT Q5
```

下面的 ORN 指令例子中，当输入 I1 为 ON 或 I2 为 OFF 时，输出 Q5 被接通。

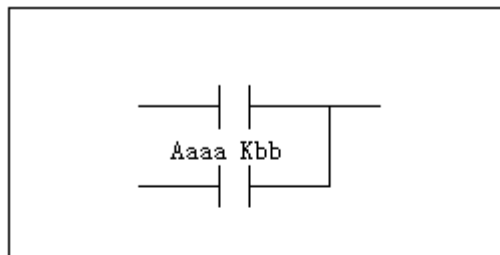
语句视图



```
LD I1
ORN I2
OUT Q5
```

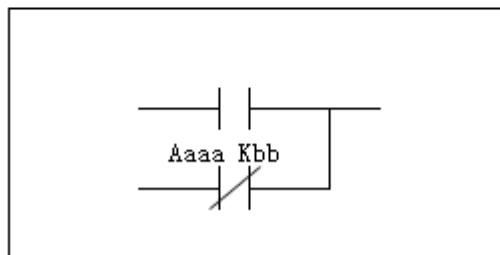
2.8 位逻辑或运算 NO 接点（BOR）

隐含 BOR 指令将一个触点同一个常开触点并联，常开触点的状态同存储器的指定位状态相同。



2.9 位逻辑或运算 NC 接点（BORN）

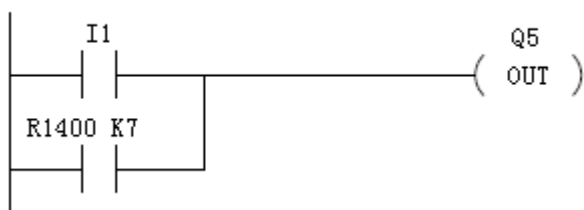
隐含 BORN 指令将一个触点同一个常闭触点并联，常闭触点的状态同存储器的指定位状态相反。



操作数类型	NK1 范围	
A	aaa	bb
寄存器	R	所有（附录 1）
指针	P	所有（附录 1）
		BCD, 0-15

下面的 BOR 指令例子中，当输入 I1 或 R1400 的位 7 为 ON 时，输出 Q5 被接通。

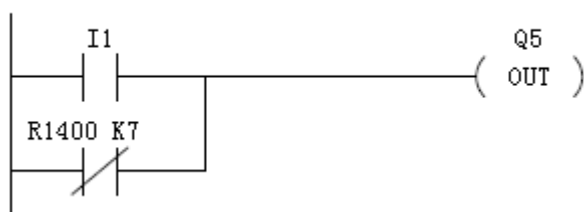
语句视图



```
LD I1
BOR R1400 K7
OUT Q5
```

下面的 BORN 指令例子中，当输入 I1 为 ON 或 R1400 的位 7 为 OFF 时，输出 Q5 被接通。

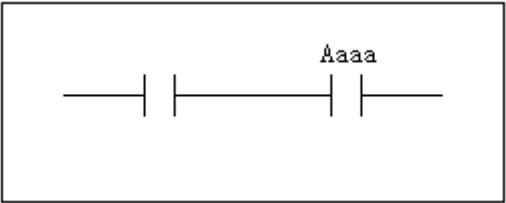
语句视图



```
LD I1
BORN R1400 K7
OUT Q5
```

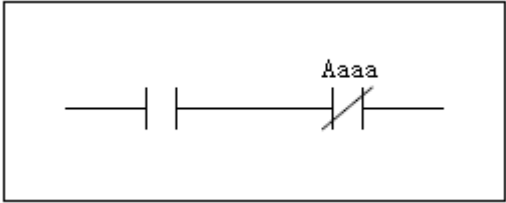
2. 10 逻辑与运算 NO 接点（AND）

隐含 AND 指令将一个触点同一个串联的常开接点逻辑与，触点的状态同指定存储器的状态相同。



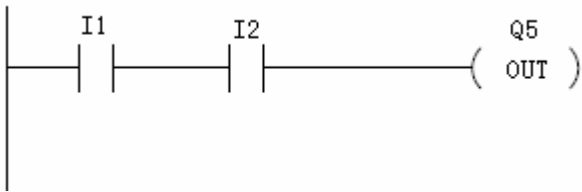
2. 11 逻辑与运算 NC 接点（ANDN）

隐含 ANDN 指令将一个触点同一个串联的常闭触点逻辑与，触点的状态同指定存储器的状态相反。



操作数类型		NK1 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777
级	S	0-1777
定时器	T	0-777
计数器	C	0-777
特殊线圈	SP	0-1777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

下面的 AND 指令例子中，当输入 I1 和 I2 均为 ON 时，输出 Q5 被接通。



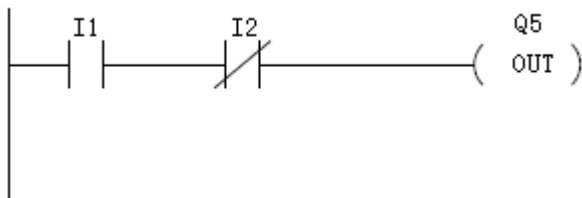
语句视图

LD I1

AND I2

OUT Q5

下面的 ANDN 指令例子中，当输入 I1 为 ON 并且 I2 为 OFF 时，输出 Q5 被接通。



语句视图

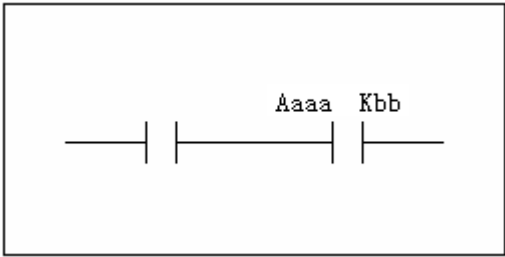
LD I1

ANDN I2

OUT Q5

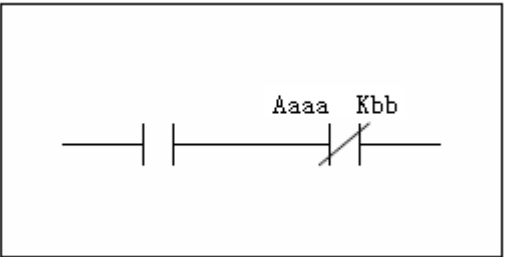
2.12 位逻辑与运算 NO 接点（BAND）

隐含 BAND 指令将一个触点同一个串联的常开触点逻辑与，触点的状态同功能存储器的指定位状态相同。



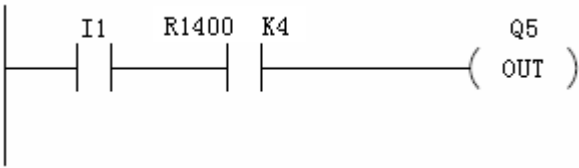
2.13 位逻辑与运算 NC 接点（BANDN）

隐含 BANDN 指令将一个触点同一个串联的常闭触点逻辑与，触点的状态同功能存储器的指定位状态相反。



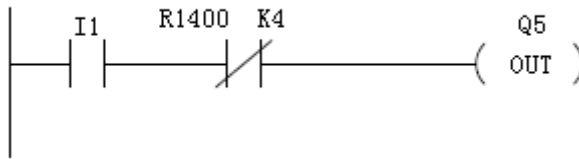
操作数类型	NK1 范围	
A	aaa	bb
寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

下面的 BAND 指令例子中，当输入 I1 和 R1400 的位 4 均为 ON 时，输出 Q5 被接通。



```
语句视图
LD I1
BAND R1400 K4
OUT Q5
```

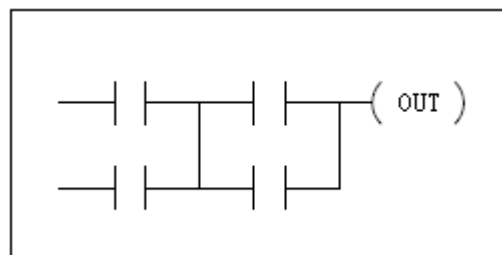
下面的 BANDN 指令例子中，当输入 I1 为 ON 并且 R1400 的位 4 为 OFF 时，输出 Q5 被接通。



```
语句视图
LD I1
BANDN R1400 K4
OUT Q5
```

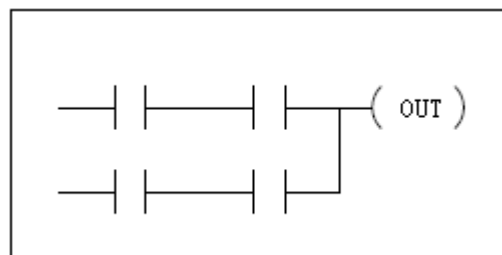
2.14 逻辑组串联指令（ANDLD）

隐含 ANDLD 指令将两个逻辑组串联，两个逻辑组都必须以逻辑运算开始指令开头。

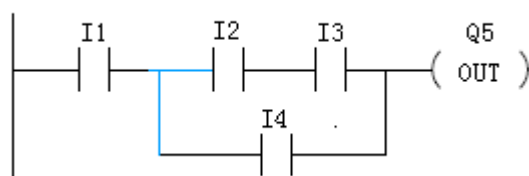


2.15 逻辑组并联指令（ORLD）

隐含 ORLD 指令将两个逻辑组并联，两个逻辑组都必须以逻辑运算开始指令开头。



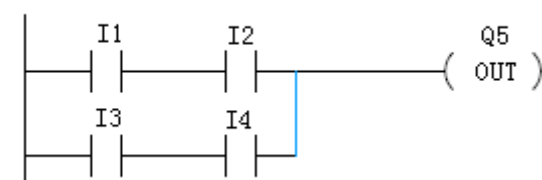
下面的 ANDLD 指令例子中，I2、I3 和 I4 组成的逻辑组同 I1 串联。



语句视图

```
LD I1
LD I2
AND I3
OR I4
ANDLD
OUT Q5
```

下面的 ORLD 指令例子中，I1、I2 组成的逻辑组同 I3、I4 组成的逻辑组并联。

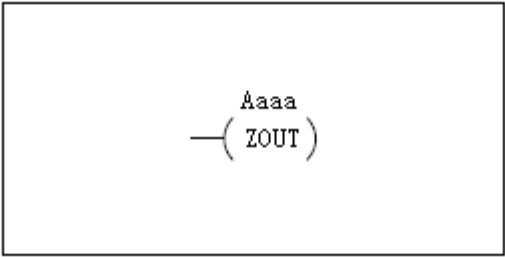


语句视图

```
LD I1
AND I2
LD I3
AND I4
ORLD
OUT Q5
```

2.16 输出指令（后面优先）（ZOUT）

输入 输出指令将程序结果输出到功能存储器，多个输出指令不可使用相同的功能存储器地址，因为只有最后一个指令控制物理输出点。



操作数类型		NK1 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

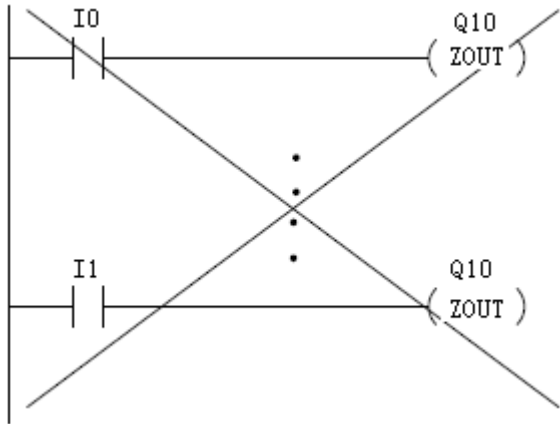
下面的输出指令例子中，当 I1 为 ON 时，输出 Q2 和 Q5 被接通。



语句视图

```
LD I1
ZOUT Q2
ZOUT Q5
```

下面的输出指令例子中，有两个输出指令使用了相同的地址（Q10），Q10 最终是由下面一条程序控制，I1 将覆盖 I0 的控制结果，因此编程时要避免多个输出指令使用同一个地址。



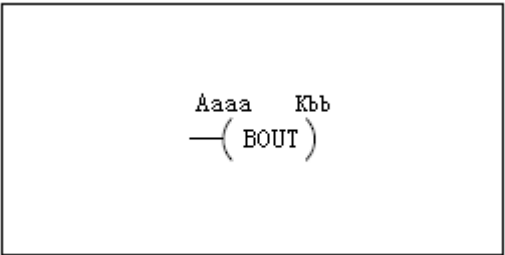
注意：

1. OUT 指令双重使用时，OR 动作。ZOUT 指令双重使用时，后面的优先动作。
2. 如果同一线圈没有双重使用，OUT 指令、ZOUT 指令动作相同。
3. 在通常的梯形图程序中，同一线圈不双重使用，所以没有必要区别 OUT 和 ZOUT。



2. 17 位输出指令（BOUT）

输入 输出指令将程序结果输出到功能存储器的指定位，多个输出指令不可使用相同的功能存储器位地址，因为只有最后一个指令控制物理输出点。



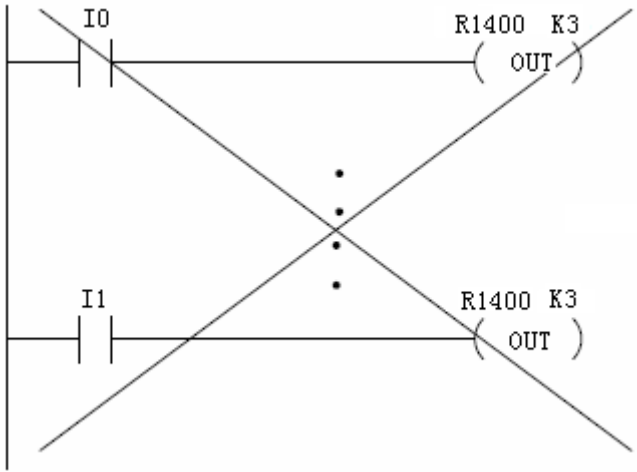
操作数类型		NK1 范围	
	A	aaa	bb
寄存器	R	所有（附录 1）	十进制，0-15
指针	P	所有（附录 1）	十进制，0-15

下面的位输出指令例子中，当 I1 为 ON 时，R1400 的位 3 和 R1401 的位 6 被置为 ON。



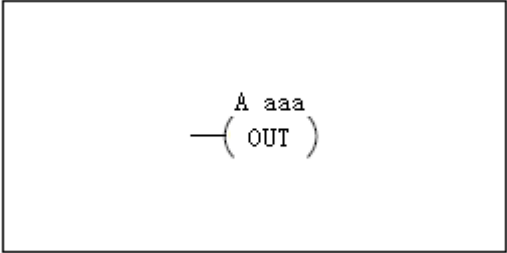
语句视图
LD I1
BOUT R1400 K3
BOUT R1401 K6

下面的输出指令例子中，有两个输出指令使用了相同的地址（R1400 的位 3），R1400 的位 3 最终是由下面一条程序控制，I1 将取代 I0 的控制结果，因此编程时要避免多个输出指令使用同一个地址。



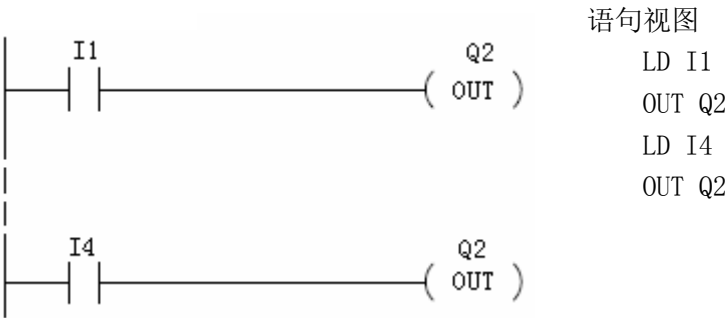
2. 18 动作输出指令（OUT）

输入 OUT 指令是或逻辑动作输出指令，用于多段程序控制单个输出的程序，任何一段程序为 ON 时，输出也为 ON。



操作数类型		NK1 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

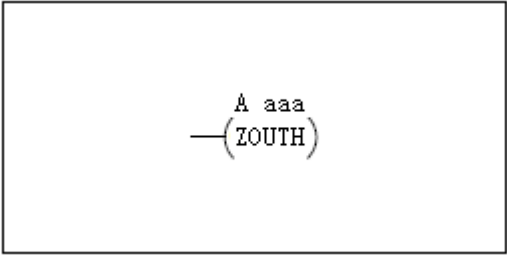
下面的例子中，当 I1 或 I4 为 ON 时，Q2 被接通。



- 注意：**
- OUT 指令双重使用时，或逻辑动作。ZOUT 指令双重使用时，后面的优先动作。
 - 如果同一线圈没有双重使用，OUT 指令、ZOUT 指令动作相同。
 - 在通常的梯形图程序中，同一线圈不双重使用，所以没有必要区别 OUT 和 ZOUT。

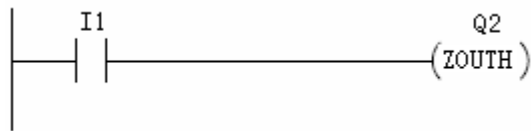
2. 19 高速输出指令（ZOUTH）

输入 ZOUTH 指令与 ZOUT 指令相似，但无视输出点的替代（override）参数状态。可以在需要快速扫描时间，而不需要替代（override）功能等调试功能的应用程序中使用该指令。ZOUTH 指令直接根据回路状态，向指定的存储器位置或者位输出状态，由于 ZOUTH 指令直接控制实际输出点，对于相同离散点，请不要使用复数个 ZOUT、ZOUTH 指令。



操作数类型		NK1 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777

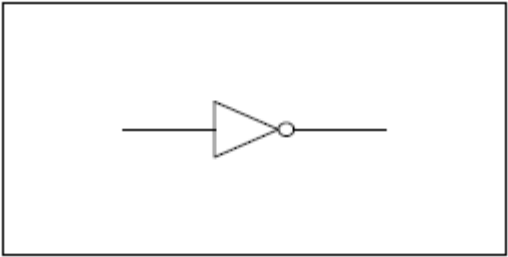
下面的输出指令例子中，当 I1 为 ON 时，输出 Q2 和 Q5 被接通。



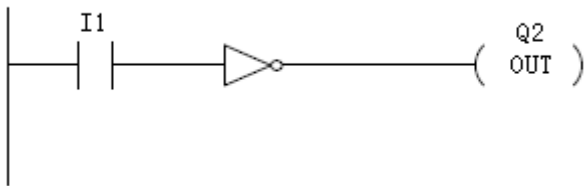
语句视图
LD I1
ZOUTH Q2

2. 20 非指令（NOT）

隐含 非指令改变当前回路的状态。



下面的例子中，当 I1 为 OFF 时，Q2 被接通。这是因为 NOT 指令改变了回路的状态。

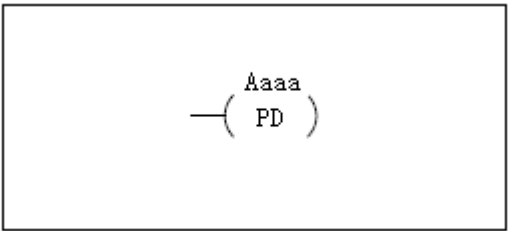


语句视图

```
LD I1
NOT
OUT Q2
```

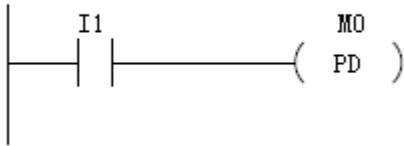
2. 21 一次扫描输出指令（PD）

输入 当开关量输入产生一个 OFF→ON 的脉冲时，在脉冲的上升沿输出将接通一个扫描周期，此后，其仍然保持 OFF 直到下一个 OFF→ON 脉冲的产生。



操作数类型		NK1 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777

下面的例子中，每次 I1 上产生一个 OFF→ON 的脉冲，MO 就接通一个扫描周期。



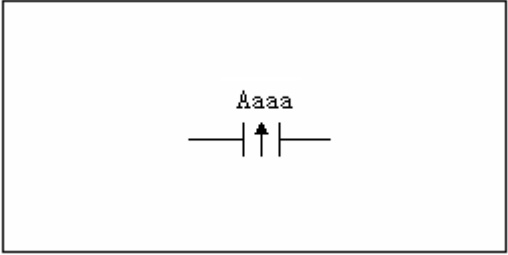
语句视图

```
LD I1
PD MO
```

注意，如果在 PD 指令前加入一个 NOT 指令，可在 ON→OFF 变化时产生一个扫描周期的脉冲。

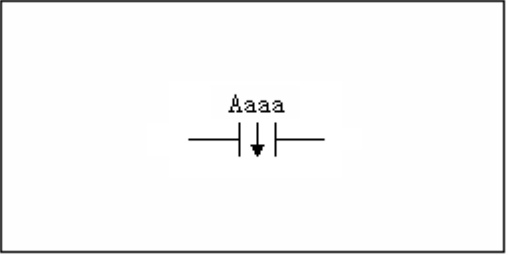
2.22 逻辑运算开始上升沿接点（LDPD）

隐含 LDPD 指令可直接连接在母线上，也可以连接在触点组上使用，是一个常开触点。当指定的功能存储器位发生 OFF→ON 的变化时，LDPD 触点将闭合一个扫描周期。此后，此触点将保持断开状态直到下一个 OFF→ON 变化的发生。



2.23 逻辑运算开始下降沿接点（LDND）

隐含 LDND 指令可直接连接在母线上，也可以连接在触点组上使用，是一个常闭触点。当指的功能存储器位发生 ON→OFF 的变化时，LDND 触点将闭合一个扫描周期。此后，此触点将保持断开状态直到下一个 ON→OFF 变化的发生。



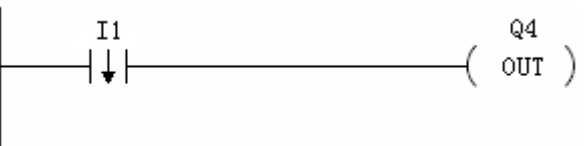
操作数类型		NK1 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777
级	S	0-1777
定时器	T	0-777
计数器	C	0-777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

下面的程序例子中，每次 I1 上发生 OFF→ON 的变化时，Q4 将接通一个周期。



语句视图
LDPD I1
OUT Q4

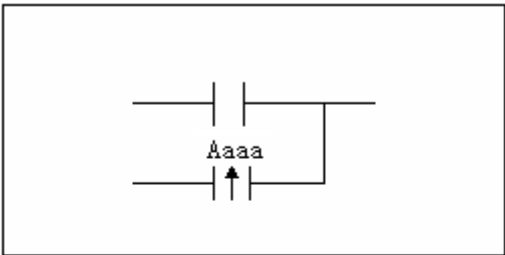
下面的程序例子中，每次 I1 上发生 ON→OFF 的变化时，Q4 将接通一个周期。



语句视图
LDND I1
OUT Q4

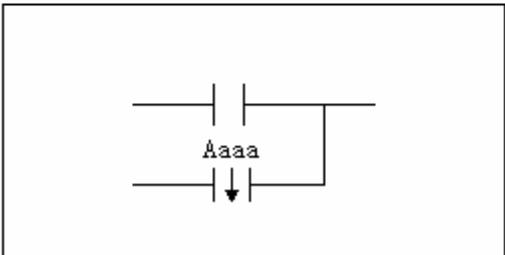
2.24 逻辑或运算上升沿接点（ORPD）

隐含 ORPD 指令将一个触点同一个 PD 触点并联。当指定的功能存储器位发生 OFF→ON 的变化时，ORPD 触点将闭合一个扫描周期。此后，此触点将保持断开状态直到下一个 OFF→ON 变化的发生。



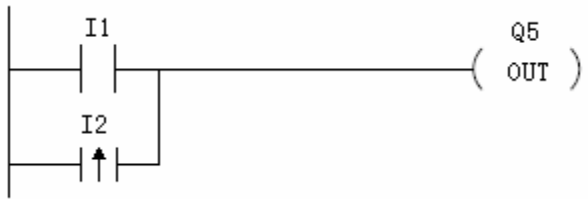
2.25 逻辑或运算下降沿接点（ORND）

隐含 ORND 指令将一个触点同一个 ND 触点并联。当指定的功能存储器位发生 ON→OFF 的变化时，ORND 触点将闭合一个扫描周期。此后，此触点将保持断开状态直到下一个 ON→OFF 变化的发生。



操作数类型		NK1 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777
级	S	0-1777
定时器	T	0-777
计数器	C	0-777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

下面的程序例子中，当 I1 为 ON，或是当 I2 上发生 OFF→ON 的变化时，Q5 被接通一个周期。



语句视图

```
LD I1
ORPD I2
OUT Q5
```

下面的程序例子中，每次 I1 为 ON，或是当 I2 上发生 ON→OFF 的变化时，Q5 被接通一个周期。

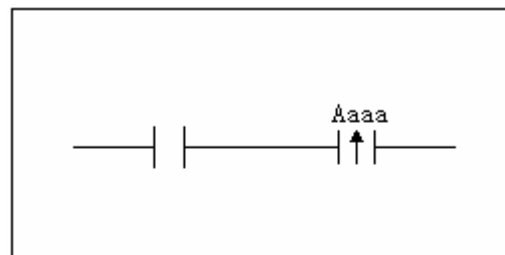


语句视图

```
LD I1
ORND I2
OUT Q5
```

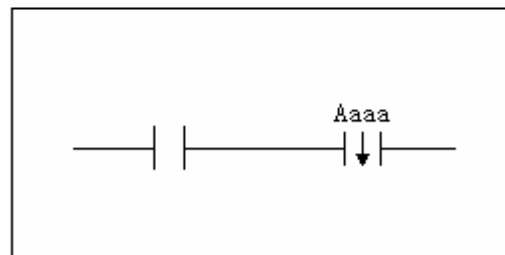
2.26 逻辑与运算上升沿接点（ANDPD）

隐含 ANDPD 指令将一个触点同个 PD 触点串联。当指定的功能存储器位发生 OFF→ON 的变化时，ANDPD 触点将闭合一个扫描周期。此后，此触点将保持断开状态直到下一个 OFF→ON 变化的发生。



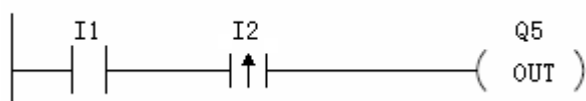
2.27 逻辑与运算开始下降沿接点（ANDND）

隐含 ANDND 指令将一个触点同个 ND 触点串联。当指定的功能存储器位发生 ON→OFF 的变化时，ANDND 触点将闭合一个扫描周期。此后，此触点将保持断开状态直到下一个 ON→OFF 变化的发生。



操作数类型		NK1 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777
级	S	0-1777
定时器	T	0-777
计数器	C	0-777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

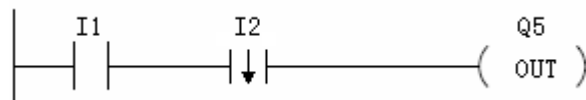
下面的程序例子中，当 I1 为 ON 并且 I2 上产生 OFF→ON 的变化时，Q5 被接通一个周期。



语句视图

```
LD I1
ANDPD I2
OUT Q5
```

下面的程序例子中，当 I1 为 ON 并且 I2 上产生 ON→OFF 的变化时，Q5 被接通一个周期。

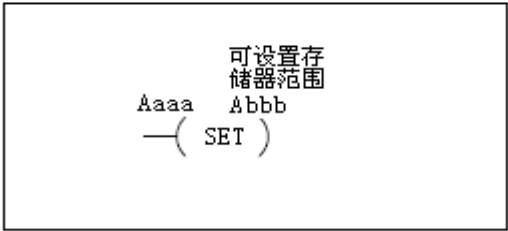


语句视图

```
LD I1
ANDND I2
OUT Q5
```

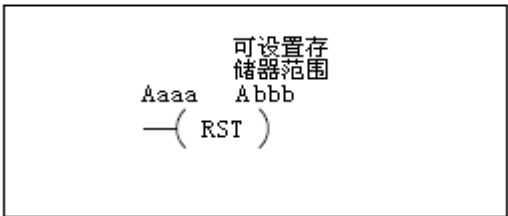
2.28 线圈置位指令（SET）

输入 SET 指令将指定的功能存储器置位，这些点一旦被置位就一直处于 ON 状态，直到 RST 指令将其复位。



2.29 线圈复位指令（RST）

输入 RST 指令将指定的功能存储器复位，这些点一旦被复位就一直处于 OFF 状态，直到 SET 指令将其置位。



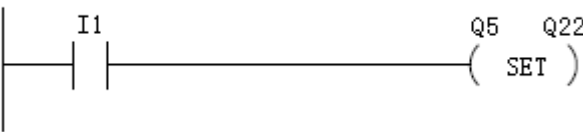
操作数类型		NK1 范围	
	A	aaa	bbb
输入	I	0-1777	0-1777
输出	Q	0-1777	0-1777
内部线圈	M	0-3777	0-3777
级	S	0-1777	0-1777
定时器*	T	0-777	0-777
计数器*	C	0-777	0-777
通讯输入	GI	0-3777	0-3777
通讯输出	GQ	0-3777	0-3777

*操作数类型定时器和计数器在使用 SET 指令时是无效的。



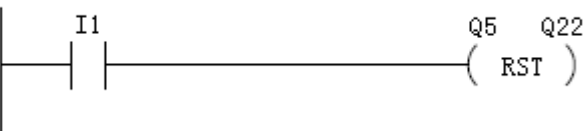
注意：不可将分配给输入模块的输入点（I）置位。

下面的程序例子中，当 I1 变为 ON 时，输出 Q5-Q22 被置位。



语句视图
LD I1
SET Q5 Q22

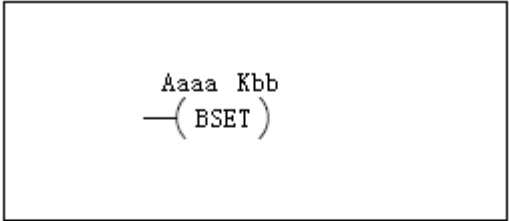
下面的程序例子中，当 I1 变为 ON 时，输出 Q5-Q22 被复位。



语句视图
LD I1
RST Q5 Q22

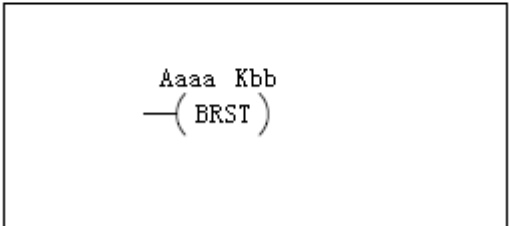
2.30 BIT 置位指令（BSET）

输入 BSET 指令将指定寄存器的指定位置位，指定位一旦被置位就一直处于 ON 状态，直到 BRST 指令将其复位。



2.31 BIT 复位指令（BRST）

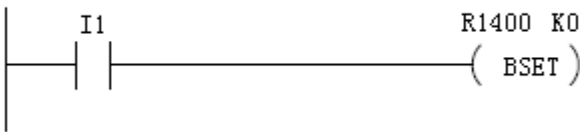
输入 BRST 指令将指定寄存器的指定位置复，指定位一旦被复位就一直处于 OFF 状态，直到 BSET 指令将其置位。



操作数类型		NK1 范围	
	A	aaa	bb
寄存器	B	所有（附录 1）	BCD, 0-15
指针	P	所有（附录 1）	BCD, 0-15

下面的程序例子中，当 I1 变为 ON 时，R1400 的 BIT 0 被置位。

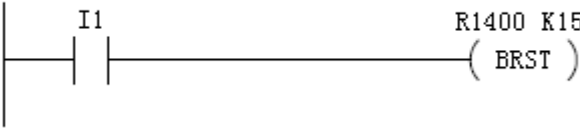
语句视图



LD I1
BSET R1400 K0

下面的程序例子中，当 I1 变为 ON 时，R1400 的 BIT 15 被复位。

语句视图



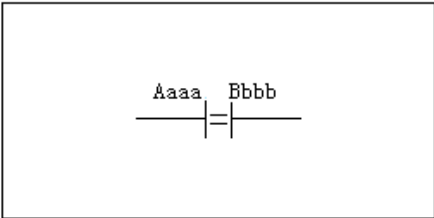
LD I1
BRST R1400 K15

第 3 章 比较接点指令

3.1 逻辑运算开始比较一致接点（LDEQ/DLDEQ）

隐含 LDEQ/DLDEQ 指令开始一个回路或添加一个支路，是常开比较接点，当 A aaa=B bbb 时，接点变为 ON。

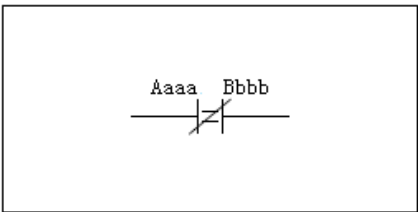
LDEQ 为 16 位数据比较接点指令。
DLDEQ 为 32 位数据比较接点指令。



3.2 逻辑运算开始比较不一致接点（LDNEQ/DLDNEQ）

隐含 LDNEQ/DLDNEQ 指令开始一个回路或添加一个支路，是常闭比较接点，当 Aaaa≠Bbbb 时，接点为 ON。

LDNEQ 为 16 位数据比较接点指令。
DLDNEQ 为 32 位数据比较接点指令。

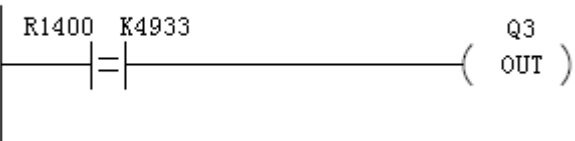


操作数类型		NK1 范围	
A/B		aaa	bbb
寄存器	R	所有(附录 1,)	所有(附录 1)
指针	P	所有(附录 1)	所有(附录 1)
常数	K	-	16 位指令：0-FFFF 32 位指令：0-FFFFFFFF

注：32 位数指令使用 R 数据类型时，不能使用每段数据寄存器的最后一个寄存器。

下面的例子程序中，当 R1400 中的数据等于 4933 时，Q3 被接通。

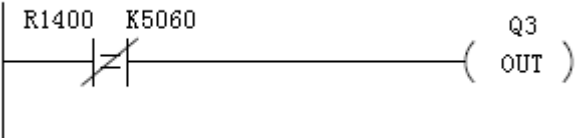
语句视图



LDEQ R1400 K4933
OUT Q3

下面的例子程序中，当 R1400 中的数据不等于 5060 时，Q3 被接通。

语句视图

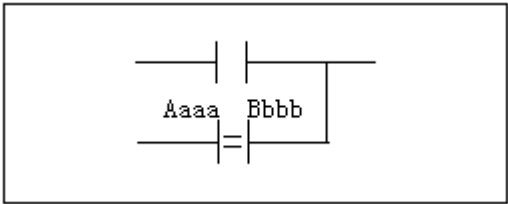


LDNEQ R1400 K5060
OUT Q3

3.3 逻辑或运算比较一致接点（OREQ/DOREQ）

隐含 OREQ/DOREQ 指令将一个触点同一个常开比较触点并联，当 Aaaa=Bbbb 时，常开比较触点为 ON。

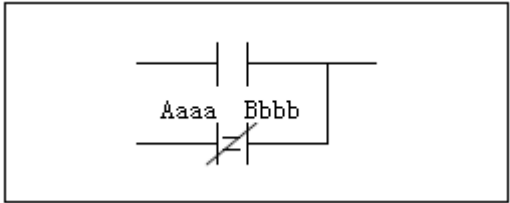
OREQ 为 16 位数据比较接点指令。
DOREQ 为 32 位数据比较接点指令。



3.4 逻辑或运算比较不一致接点（ORNEQ/DORNEQ）

隐含 ORNEQ/DORNEQ 指令将一个触点同一个常闭比较触点并联，当 Aaaa≠Bbbb 时，常闭比较触点为 ON。

ORNEQ 为 16 位数据比较接点指令。
DORNEQ 为 32 位数据比较接点指令。

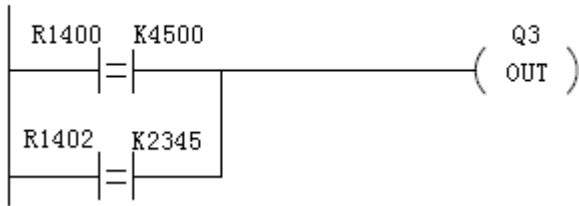


操作数类型		NK1 范围	
	A/B	aaa	bbb
寄存器	R	所有(附录 1)	所有(附录 1)
指针	P	所有(附录 1)	所有(附录 1)
常数	K	-	16 位指令：0~FFFF 32 位指令：0~FFFFFFFF

注：32 位数指令使用 R 数据类型时，不能使用每段数据寄存器的最后一个寄存器。

下面的例子中，当 R1400=4500 或 R1402=2345 时，Q3 被接通。

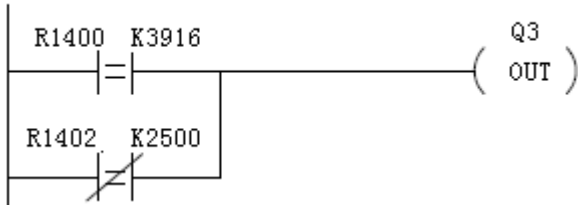
语句视图



```
LDEQ R1400 K4500
OREQ R1402 K2345
OUT Q3
```

下面的例子中，当 R1400=3916 或 R1402≠2500 时，Q3 被接通。

语句视图



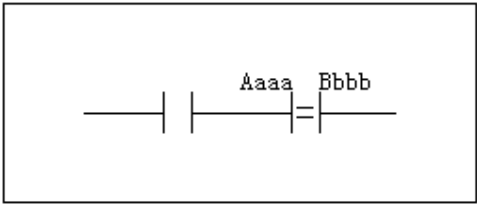
```
LDEQ R1400 K3916
ORNEQ R1402 K2500
OUT Q3
```

3.5 逻辑与运算比较一致接点（ANDEQ/DANDEQ）

隐含 ANDEQ/DANDEQ 指令将一个触点同一个常开比较触点串联，当 Aaaa=Bbbb 时，常开比较触点为 ON。

ANDEQ 为 16 位数据比较接点指令。

DANDEQ 为 32 位数据比较接点指令。

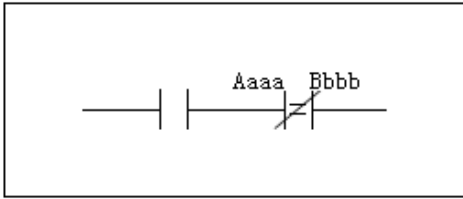


3.6 逻辑与运算比较不一致接点（ANDNEQ/DANDNEQ）

隐含 ANDNEQ/DANDEQ 指令将一个触点同一个常闭比较触点串联，当 Aaaa≠Bbbb 时，常闭比较触点为 ON。

ANDNEQ 为 16 位数据比较接点指令。

DANDNEQ 为 32 位数据比较接点指令。

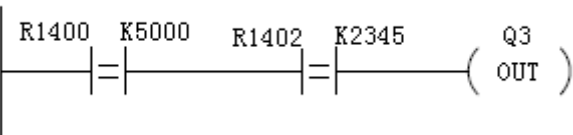


操作数类型		NK1 范围	
	A/B	aaa	bbb
寄存器	R	所有(附录 1)	所有(附录 1)
指针	P	所有(附录 1)	所有(附录 1)
常数	K	-	16 位指令：0-FFFF 32 位指令：0-FFFFFFFF

注：32 位数指令使用 R 数据类型时，不能使用每段数据寄存器的最后一个寄存器。

下面的例子中，当 R1400=5000 并且 R1402=2345 时，Q3 被接通。

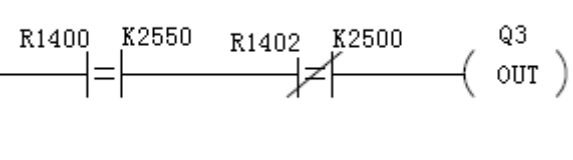
语句视图



```
LDEQ R1400 K5000
ANDEQ R1402 K2345
OUT Q3
```

下面的例子中，当 R1400=2550 并且 R1402≠2500 时，Q3 被接通。

语句视图

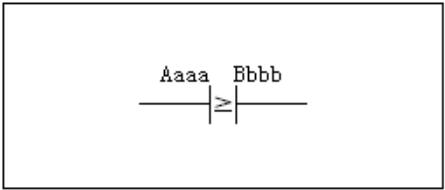


```
LDEQ R1400 K2550
ANDNEQ R1402 K2500
OUT Q3
```

3.7 比较大等于逻辑运算开始接点（LDGE/DLDGE）

隐含 LDGE/DLDGE 指令用一个常开比较触点开始一个新回路或一个新支路，当 $Aaaa \geq Bbbb$ 时，触点为 ON。

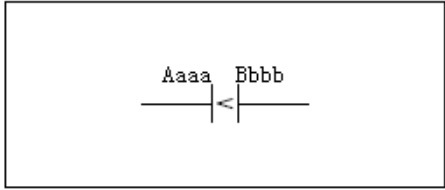
LDGE 为 16 位数据比较接点指令。
DLDGE 为 32 位数据比较接点指令。



3.8 比较小于逻辑运算开始接点（LDNGE/DLDNGE）

隐含 LDNGE/DLDNGE 指令用一个常闭比较触点开始一个新回路或一个新支路，当 $Aaaa < Bbbb$ 时，触点为 ON。

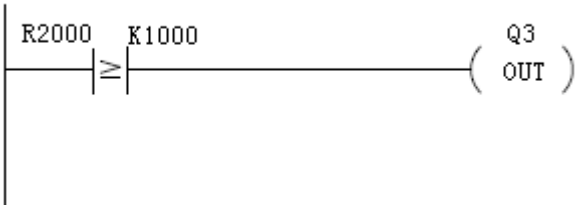
LDNGE 为 16 位数据比较接点指令。
DLDNGE 为 32 位数据比较接点指令。



操作数类型		NK1 范围	
	A/B	aaa	bbb
寄存器	R	所有(附录 1)	所有(附录 1)
指针	P	所有(附录 1)	所有(附录 1)
常数	K	-	16 位指令：0~FFFF 32 位指令：0~FFFFFFFF

注：32 位数指令使用 R 数据类型时，不能使用每段数据寄存器的最后一个寄存器。

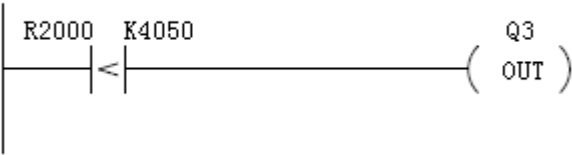
下面的例子中，当 $R2000 \geq 1000$ 时，Q3 被接通。



语句视图

```
LDGE R2000 K1000
OUT Q3
```

下面的例子中，当 $R2000 < 4050$ 时，Q3 被接通。



语句视图

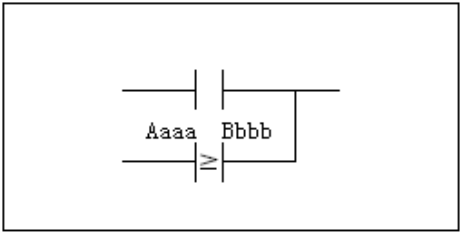
```
LDNGE R2000 K4050
OUT Q3
```

3.9 比较大等于逻辑或运算接点（ORGE/DORGE）

隐含 ORGE/DORGE 指令将一个常开比较触点同另一个触点并联，当 $Aaaa \geq Bbbb$ 时，触点为 ON。

ORGE 为 16 位数据比较接点指令。

DORGE 为 32 位数据比较接点指令。

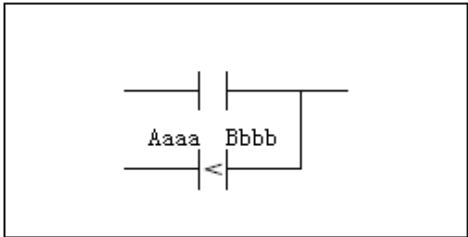


3.10 比较小于逻辑或运算接点（ORNGE/DORNGE）

隐含 ORNGE/DORNGE 指令将一个常开比较触点同另一个触点并联，当 $Aaaa < Bbbb$ 时，触点为 ON。

ORNGE 为 16 位数据比较接点指令。

DORNGE 为 32 位数据比较接点指令。

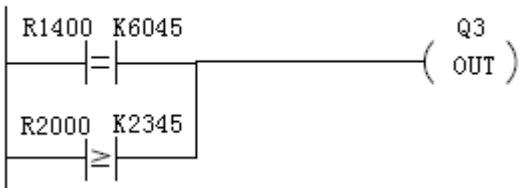


操作数类型		NK1 范围	
	A/B	aaa	bbb
寄存器	R	所有(附录 1)	所有(附录 1)
指针	P	所有(附录 1)	所有(附录 1)
常数	K	-	16 位指令：0-FFFF 32 位指令：0-FFFFFFFF

注：32 位数指令使用 R 数据类型时，不能使用每段数据寄存器的最后一个寄存器。

下面的例子中，当 $R1400=6045$ 或 $R2000 \geq 2345$ 时，Q3 被接通。

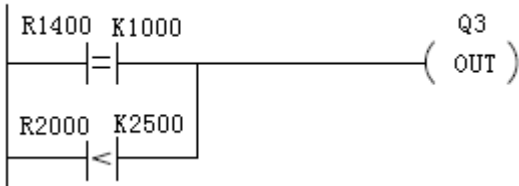
语句视图



```
LDEQ R1400 K6045
ORGE R2000 K2345
OUT Q3
```

下面的例子中，当 $R1400=1000$ 或 $R2000 < 2500$ 时，Q3 被接通。

语句视图



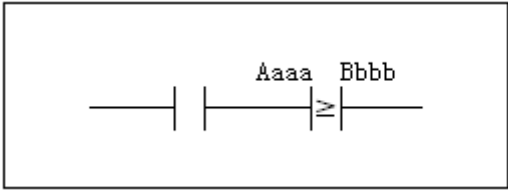
```
LDEQ R1400 K1000
ORNGE R2000 K2500
OUT Q3
```

3. 11 比较大于等于逻辑与运算接点（ANDGE/DANDGE）

隐含 ANDGE/DANDGE 指令将一个常开比较触点同另一个触点串联，当 $Aaaa \geq Bbbb$ 时，触点为 ON。

ANDGE 为 16 位数据比较接点指令。

DANDGE 为 32 位数据比较接点指令。

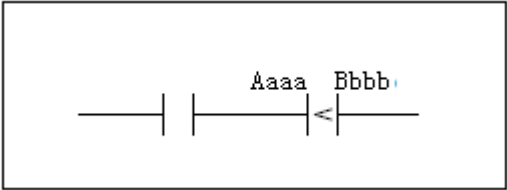


3. 12 比较小于逻辑与运算接点（ANDNGE/DANDNGE）

隐含 ANDNGE/DANDNGE 指令将一个常开比较触点同另一个触点串联，当 $Aaaa < Bbbb$ 时，触点为 ON。

ANDNGE 为 16 位数据比较接点指令。

DANDNGE 为 32 位数据比较接点指令。

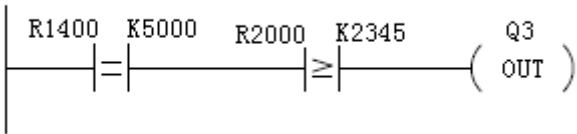


操作数类型		NK1 范围	
	A/B	Aaa	bbb
寄存器	R	所有(附录 1)	所有(附录 1)
指针	P	所有(附录 1)	所有(附录 1)
常数	K	-	16 位指令：0~FFFF 32 位指令：0~FFFFFFFF

注：32 位数指令使用 R 数据类型时，不能使用每段数据寄存器的最后一个寄存器。

下面的例子中，当 $R1400=5000$ 并且 $R2000 \geq 2345$ 时，Q3 被接通。

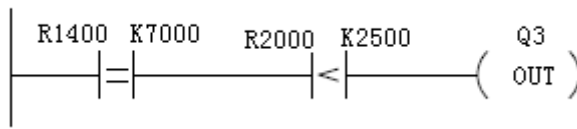
语句视图



```
LDEQ R1400 K5000
ANDGE R2000 K2345
OUT Q3
```

下面的例子中，当 $R1400=7000$ 并且 $R2000 < 2500$ 时，Q3 被接通。

语句视图



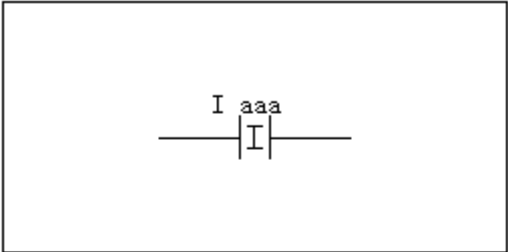
```
LDEQ R1400 K7000
ANDNGE R2000 K2500
OUT Q3
```

第 4 章 直接指令

注意：直接指令只对 NK0/NK1 本体单元上的输入/输出电有效。
对扩展单元上的输入/输出点使用直接指令，其功能和普通的输入/输出指令相同。

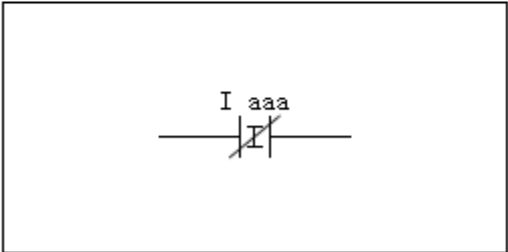
4.1 逻辑运算开始 NO 直接接点 (LDDI)

隐含 LDDI 指令开始一个新回路或一个新支路，指令执行时，触点的状态同指定的模块输入点状态相同，对应寄存器状态未更新。



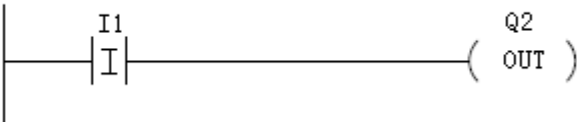
4.2 逻辑运算开始 NC 直接接点 (LDNDI)

隐含 LDNDI 指令开始一个新回路或一个新支路，指令执行时，触点的状态同指定的模块输入点状态相反，对应寄存器状态未更新。



操作数类型		NK1 范围
		aaa
输入	I	0-1777

下面的例子中，当 I1 为 ON 时，Q2 被接通。



语句视图

```
LDDI I1
OUT Q2
```

下面的例子中，当 I1 为 OFF 时，Q2 被接通。

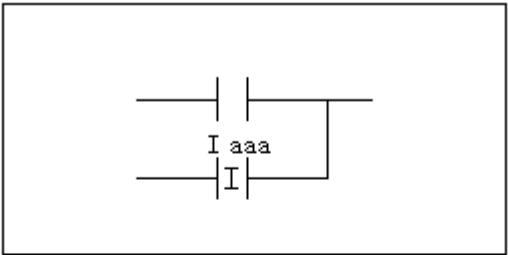


语句视图

```
LDNDI I1
OUT Q2
```

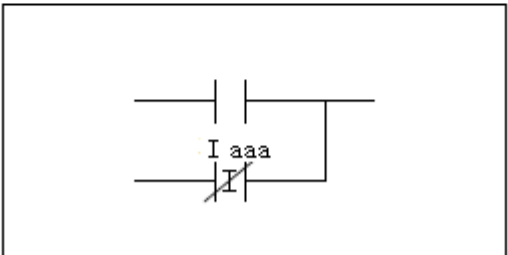

4.3 逻辑或运算 NO 直接接点（ORDI）

隐含 ORDI 指令将两个触点并联，指令执行时，接点状态同指定的模块输入点状态相同，对应寄存器状态未更新。



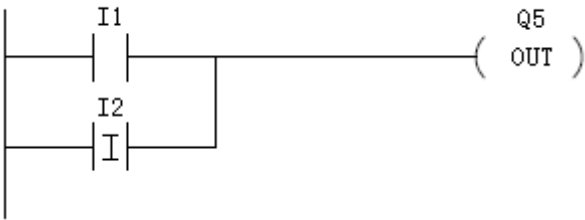
4.4 逻辑或运算 NC 直接接点（ORNDI）

隐含 ORNDI 指令将两个触点并联，指令执行时，接点状态同指定的模块输入点状态相反，对应寄存器状态未更新。



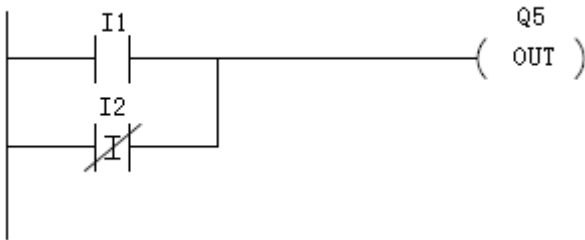
操作数类型		NK1 范围
		aaa
输入	I	0-1777

下面的例子中，当 I1 或 I2 为 ON 时，Q5 被接通。



语句视图
LD I1
ORDI I2
OUT Q5

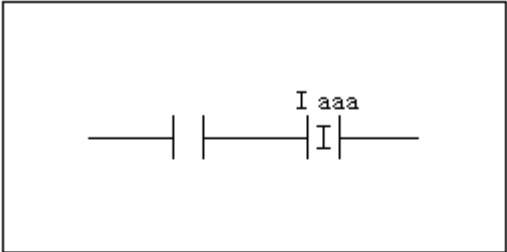
下面的例子中，当 I1 为 ON 或是 I2 为 OFF 时，Q5 被接通。



语句视图
LD I1
ORNDI I2
OUT Q5

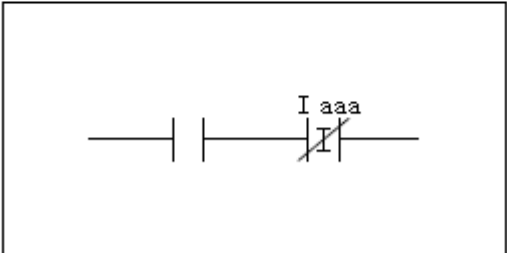
4.5 逻辑与运算 NO 直接接点 (ANDDI)

隐含 ANDDI 指令将两个触点串联，接点状态同指定的模块输入点状态相同，对应寄存器状态未刷新。



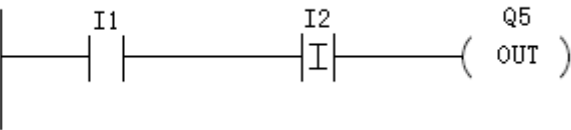
4.6 逻辑与运算 NC 直接接点 (ANDNDI)

隐含 ANDNDI 指令将两个触点串联，接点状态同指定的模块输入点状态相反，对应寄存器状态未刷新。



操作数类型		NK1 范围
		aaa
输入	I	0-1777

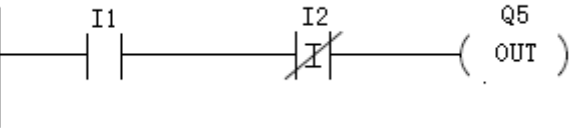
下面的例子中，当 I1 和 I2 均为 ON 时，Q5 被接通。



语句视图

```
LD I1
ANDDI I2
OUT Q5
```

下面的例子中，当 I1 为 ON 并且 I2 为 OFF 时，Q5 被接通。

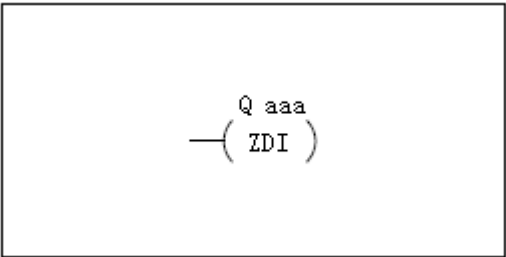


语句视图

```
LD I1
ANDNDI I2
OUT Q5
```

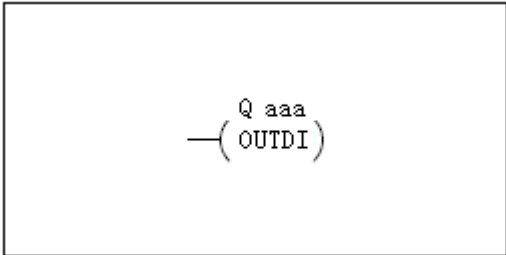
4.7 直接输出指令（ZDI）

输入 直接输出指令表现回路状态，并将回路状态输出到指定的模块输出点，指令执行时，对应寄存器状态也被更新。当有多个直接输出指令对同一个模块输出点进行输出时，那这个输出点的状态在一个扫描周期中可能变化多次，见 OUTDI 指令。



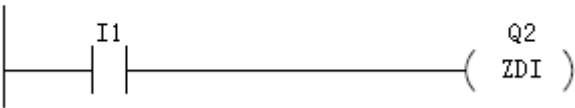
4.8 OR 直接输出指令（OUTDI）

输入 OUTDI 指令用于多个输出回路控制同一个输出点。由于多个 OUTDI 输出到同一个输出点，因此，输出结果是这些回路的逻辑或。如果一个回路是 ON，这个输出就是 ON。



操作数类型		NK1 范围
		aaa
输出	Q	0-1777

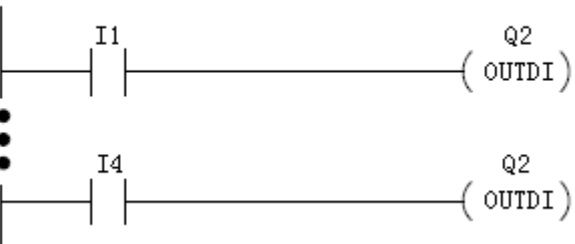
下面的例子中，当 I1 为 ON 时，Q2 被接通。



语句视图

```
LD I1
ZDI Q2
```

下面的例子中，当 I1 或 I4 为 ON 时，Q2 被接通。

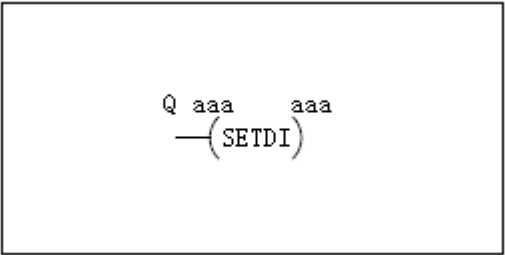


语句视图

```
LD I1
OUTDI Q2
...
LD I4
OUTDI Q2
```

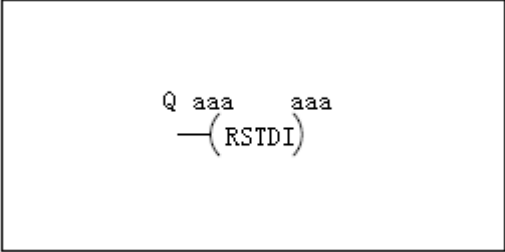
4.9 线圈直接置位指令（SETDI）

输入 SETDI 指令，当其执行时，将指定的一个或一定范围的输入点置位。对应的寄存器未被刷新。输出被置位后就保持 ON 状态，直到 RSTD I 指令将其复位。



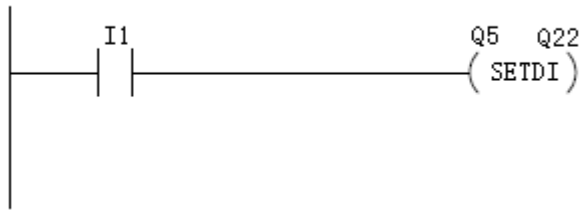
4.10 线圈直接复位指令（RSTD I）

输入 RSTD I 指令，当其执行时，将指定的一个或一定范围的输入点复位。对应的寄存器未被刷新。输出被复位后就保持 OFF 状态。



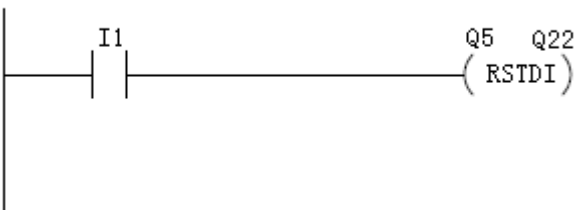
操作数类型		NK1 范围
		aaa
输出	Q	0-1777

下面的例子中，当 I1 为 ON 时，输出 Q5-Q22 被置位。



语句视图
LD I1
SETDI Q5 Q22

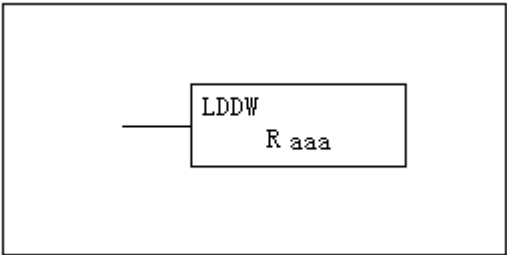
下面的例子中，当 I1 为 ON 时，输出 Q5-Q22 被复位。



语句视图
LD I1
RSTD I Q5 Q22

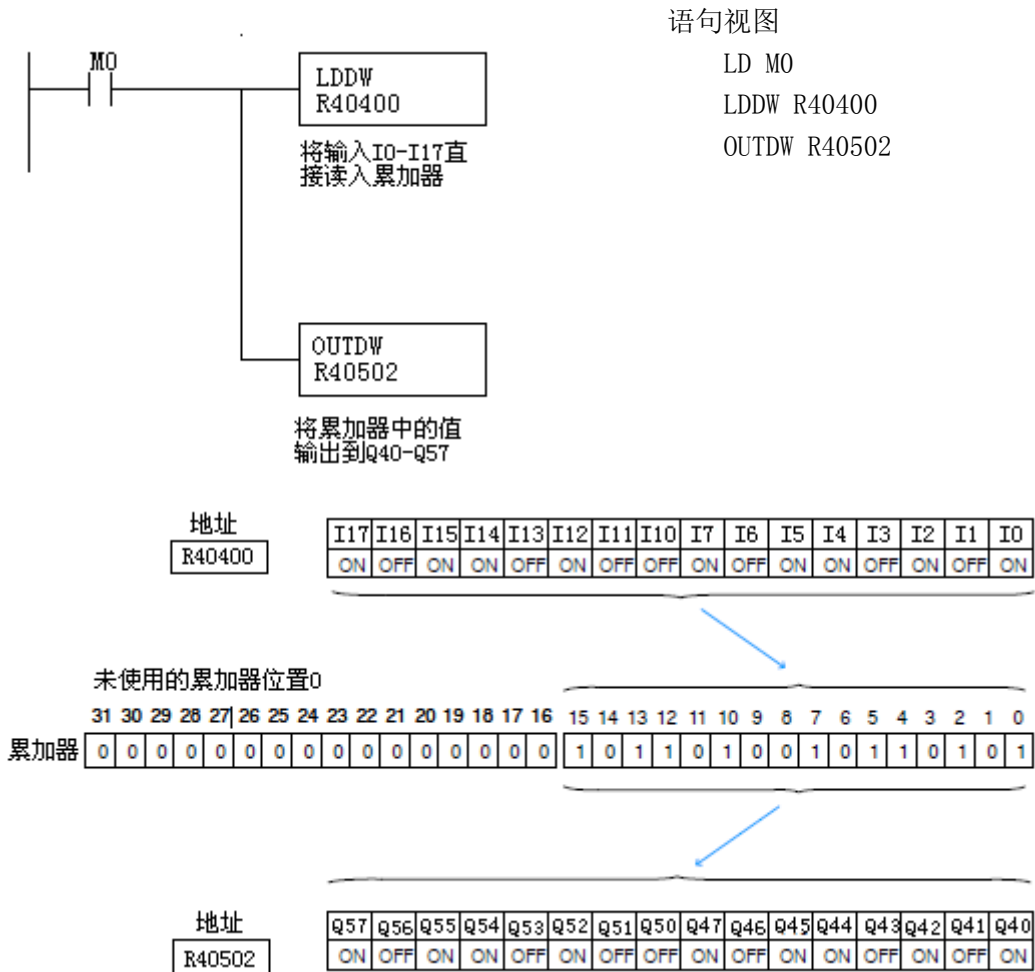
4.11 直接 16 位读入指令（LDDW）

输入 LDDW 指令将一个 16 位 R 寄存器数据读入累加器。本体框架上的所有输入点地址都是有效地址。此值是 LDDW 指令执行时输入点的当前值，此指令可替代 LDDF 指令。



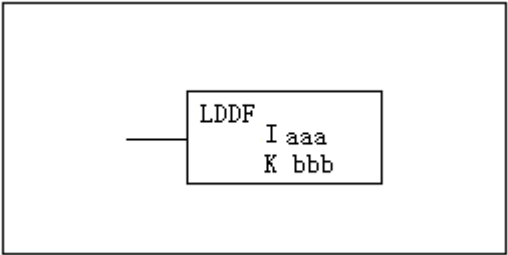
操作数类型		NK1 范围
		aaaaa
寄存器	R	40400-40477

下面的例子中，当 M0 为 ON 时，输入 I0-I17 被 LDDW 指令读入累加器，OUTDW 指令将累加器中的数据直接输出到输出点，本例中是 Q40-Q57。这种方法用于将输入快速输出到输出点（无需等待 CPU 扫描周期的完成）。



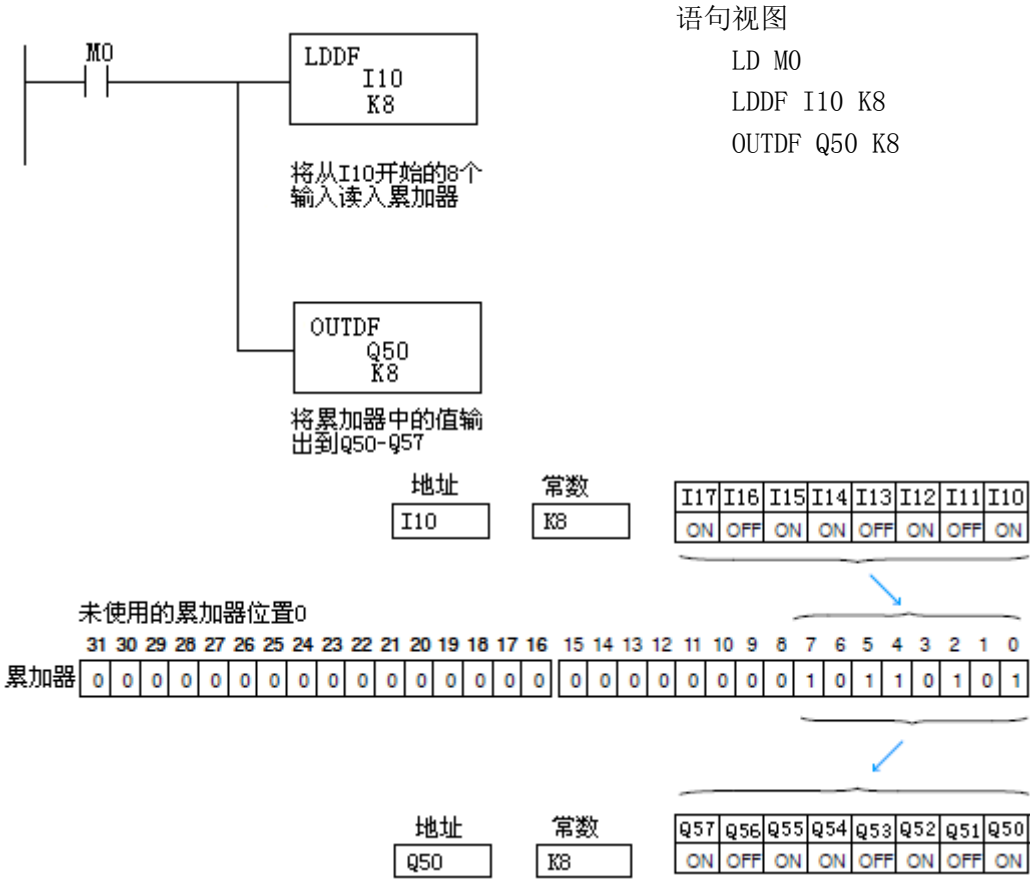
4.12 直接读入任意位指令（LDDF）

输入 LDDF 指令将 1-32 位的二进制数据读入累加器，此数据是 LDDF 指令执行时输入模块的当前值。累加器的未使用位被设置为 0。



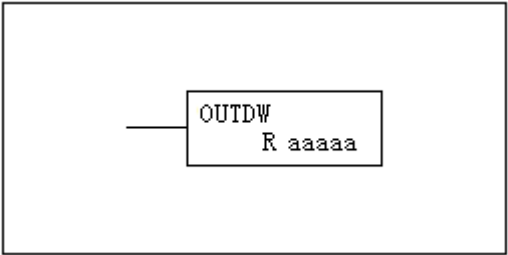
操作数类型		NK1 范围	
		aaa	bbb
输入	I	0-1777	-
常数	K	-	1-32

下面的例子中，当 M0 为 ON 时，输入 I10-I17 被 LDDF 指令读入累加器，OUTDF 指令将累加器中的数据输出到 Q50-Q57。这种方法用于将输入点快速输出到输出点（无需等待 CPU 扫描周期的完成）。



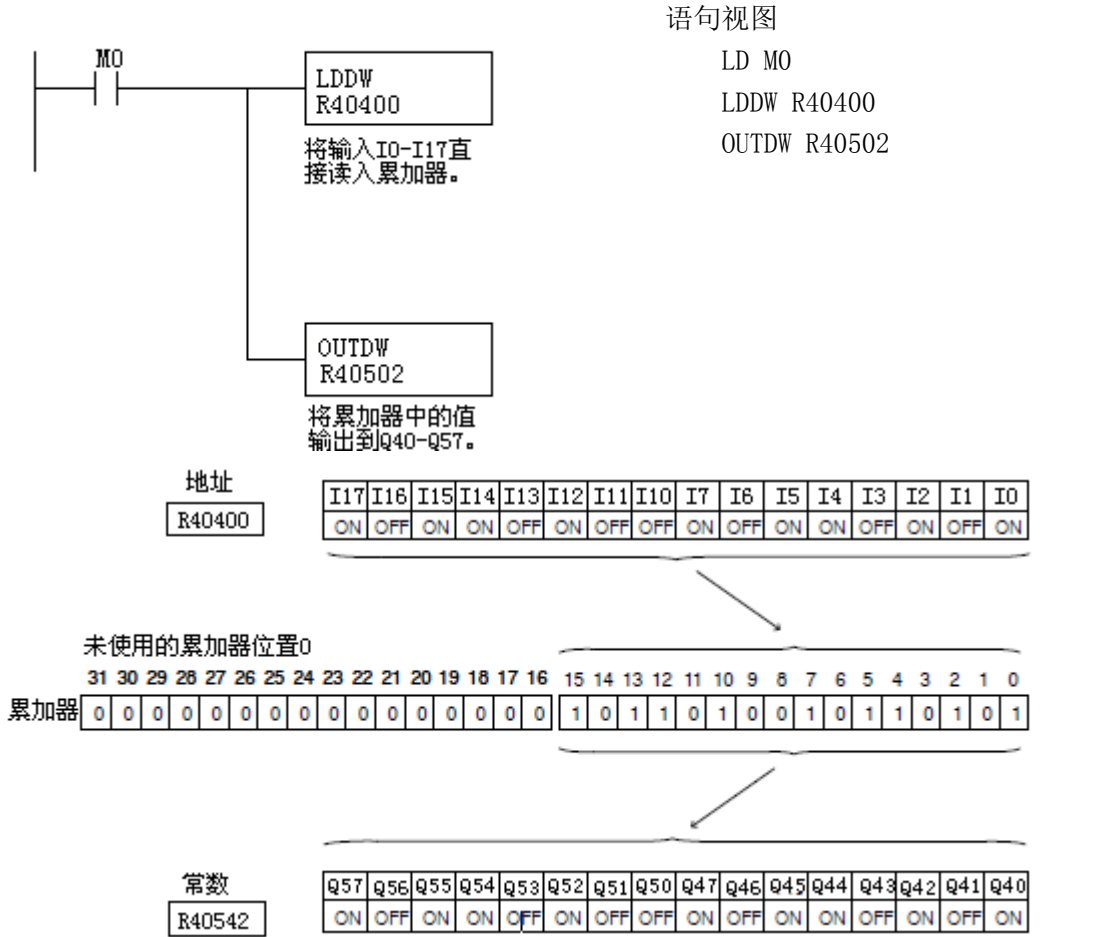
4.13 直接 16 位输出指令（OUTDW）

输入 OUTDW 指令，当其执行时，将一个 16 位二进制数据输出到 R 寄存器中。本体框架上的所有输出点地址都是有效地址。此指令可替代 OUTDF 指令。



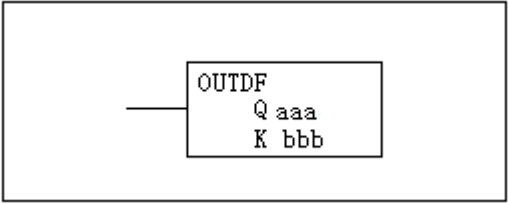
操作数类型	NK1 范围
	aaaaa
寄存器	R 40500-40577

下面的例子中，当 M0 为 ON 时，I0-I17 被 LDDW 指令读入累加器，OUTDW 指令直接将累加器的数据读输出到 Q40-Q57。这种方法用于将输入点快速输出到输出点（无需等待 CPU 扫描周期的完成）。



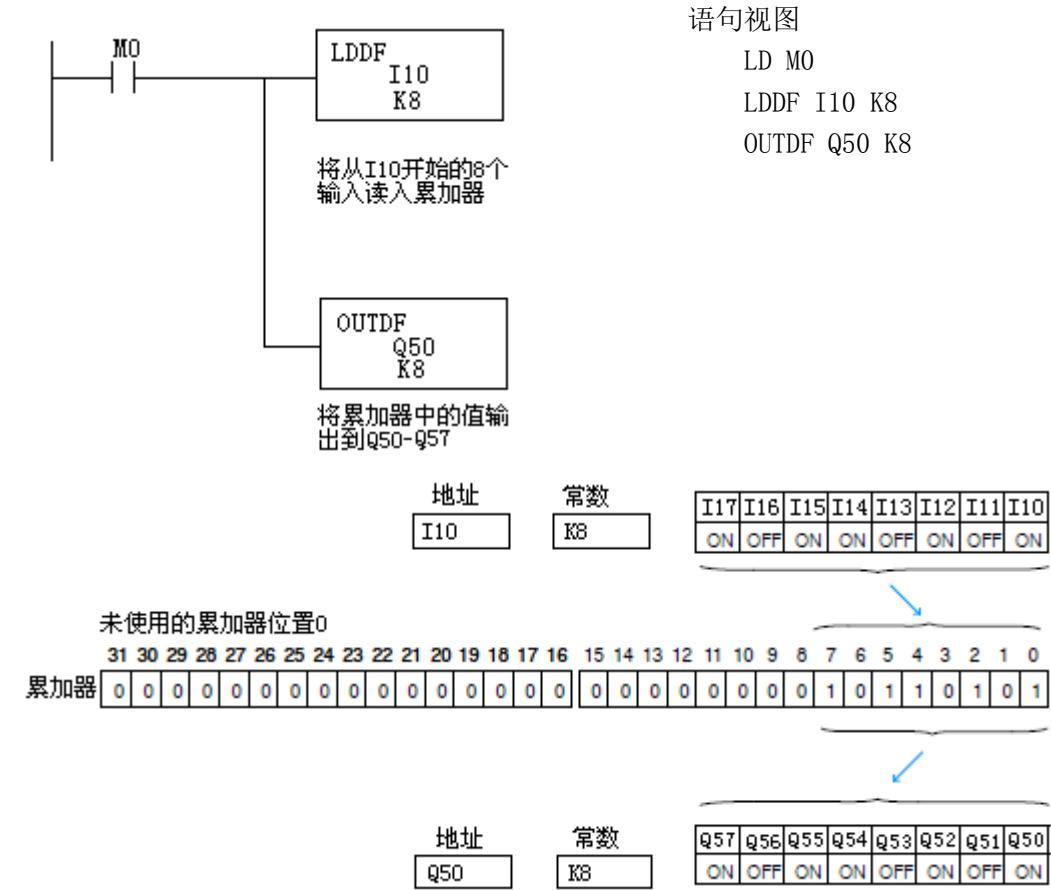
4.14 直接输出任意位指令（OUTDF）

输入 OUTDF 指令，当指令执行时，将 1-32 位二进制数据从累加器中输出到指定输出点中。未使用的累加器位被设置为 0。



操作数类型		NK1 范围	
		aaa	bbb
输出	Q	0-1777	-
常数	K	-	1-32

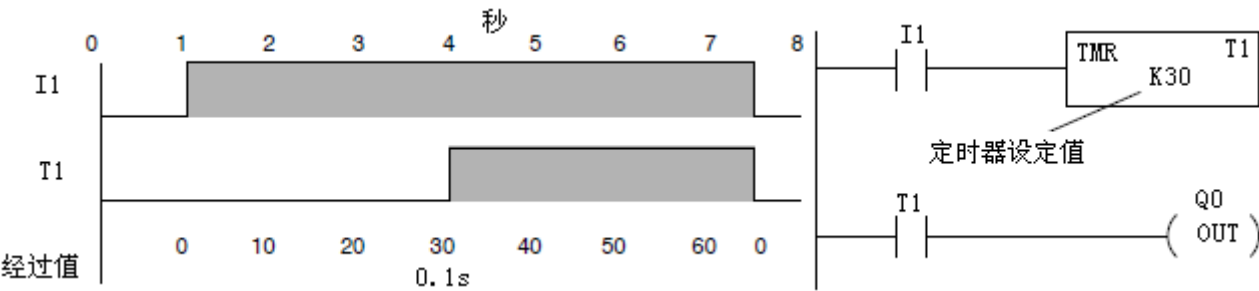
下面的例子中，当 M0 为 ON 时，输入 I10-I17 被 LDDF 指令读入累加器，OUTDF 指令将累加器中的数据输出到 Q50-Q57。这种方法用于将输入点快速输出到输出点（无需等待 CPU 扫描周期的完成）。



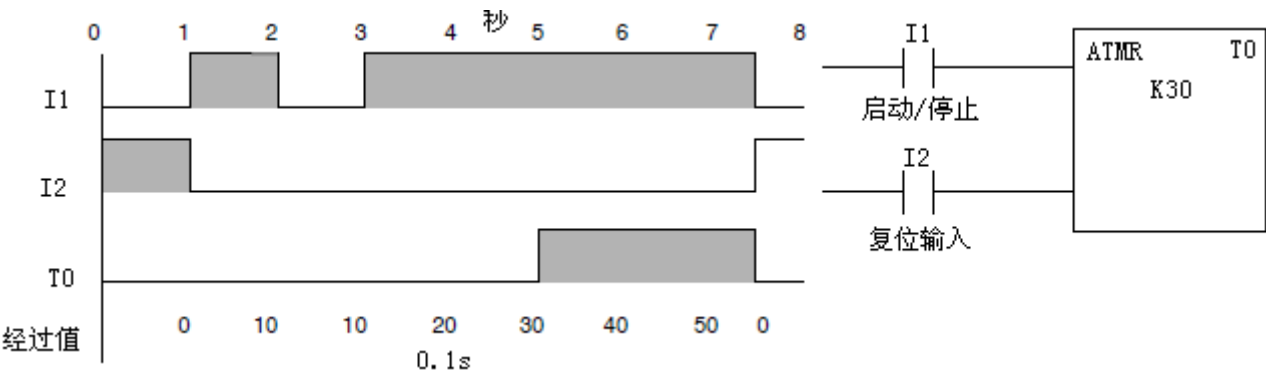
第 5 章 定时器、计数器、移位寄存器指令

5.1 定时器的使用方法

定时器用于给事件规定一个期望的时间。单输入定时器只要输入是 ON 就计时，当输入由 ON 变为 OFF，定时器经过值被复位为 0。有 0.1 秒和 0.01 秒定时器，定时最大值分别为 999.9 秒和 99.99 秒。当定时器经过值等于或大于设定值时，定时器会接通一个开关量。下图演示了定时器输入、经过值、设定值和定时器接通点之间的关系。



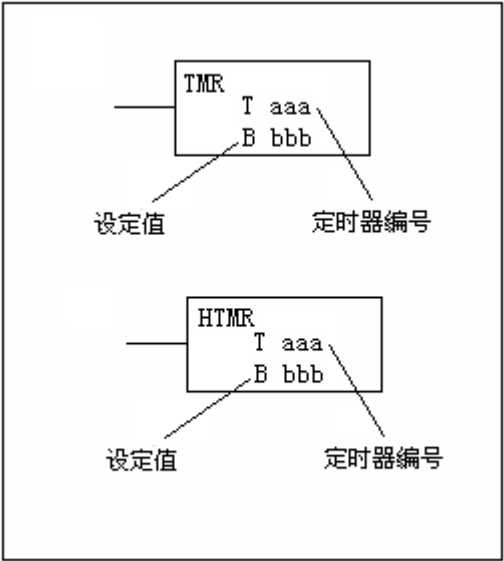
有些应用需要累积定时器，累积定时器可以计时、停止、然后再从先前停止的地方开始计时。累积定时器同普通定时器相似，但是需要两个输入。启动/停止输入用于启动和停止定时器。计时器停止时，计时经过值保持不变，当计时重新开始时，计时经过值从原来停止的地方累加。当复位输入变为 ON 时，计时经过值被复位为 0。有 0.1 秒和 0.01 秒累积定时器，定时最大值分别为 9999999.9 秒和 999999.99 秒。下图演示了定时器输入、复位条件、经过值、设定值和定时器接通点之间的关系。



5.2 0.1 秒定时器（TMR）和 0.01 秒定时器（HTMR）

输入 0.1 秒定时器 TMR 以 0.1 秒为单位增计时，定时范围 0-999.9s，单输入。0.01 秒定时器 HTMR 以 0.01 秒为单位增计时，定时范围 0-99.99s，单输入。当输入为 ON 时，这些定时器增计时，当输入为 OFF 时，定时器经过值复位。

指令详述：
定时器编号（T aaa）：定时器编号
设定值（B bbb）：常数（K）或寄存器地址。
经过值：计时器经过值存储在相对应的 R 寄存器或 T 存储地址中。例如，定时器 T3 的经过值存放在 R3 中。
定时器接通点：经过值等于或大于设定值时，定时器接通点接通，例如，定时器 2 接通时 T2 为 ON。



定时器指令上接通点和经过值没有标示



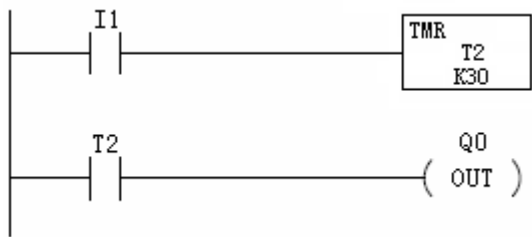
注意：仅当程序或操作单元需要改变设定值时，才将设定值放入寄存器中。

操作数类型		NK1 范围	
	B	aaa	bbb
定时器	T	0-777	-
设定值寄存器	R	-	所有（附录 1）
指针（仅设定值）	P	-	所有（附录 1）
常数（仅设定值）	K	-	0-9999

定时器有两种用法，一种是使用定时器接通点触发某一事件，一种是使用比较指令在同一个定时器的不同时间点进行比较来触发事件。下面有这两种方法的编程举例。

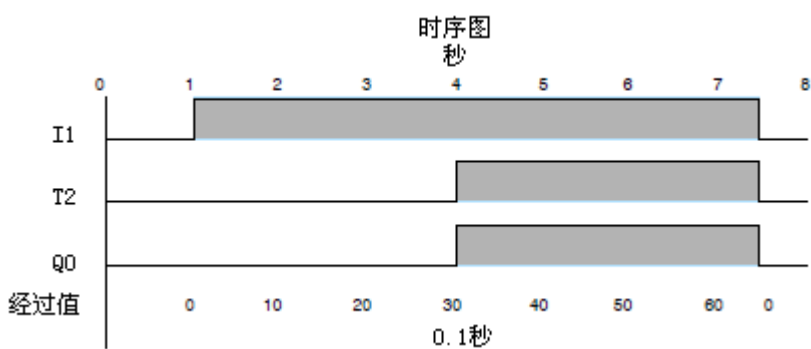
5.2.1 使用定时器接通点的程序举例

下面的例子中，定时器设定值是 3 秒，当定时器计时到 3 秒时，定时器接通点 T2 变为 ON。当 I1 变为 OFF 时，定时器经过值复位，接通点 T2 变为 OFF。



语句视图

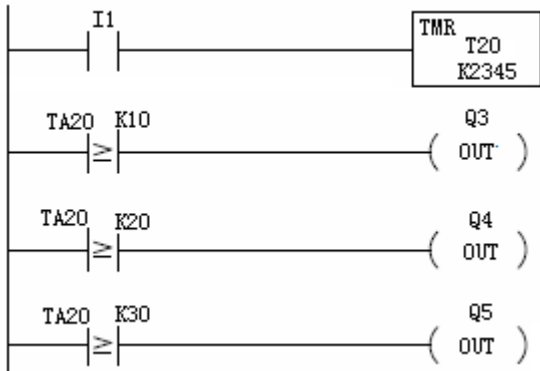
```
LD I1
TMR T2 K30
LD T2
OUT Q0
```



5.2.2 使用比较指令的程序举例

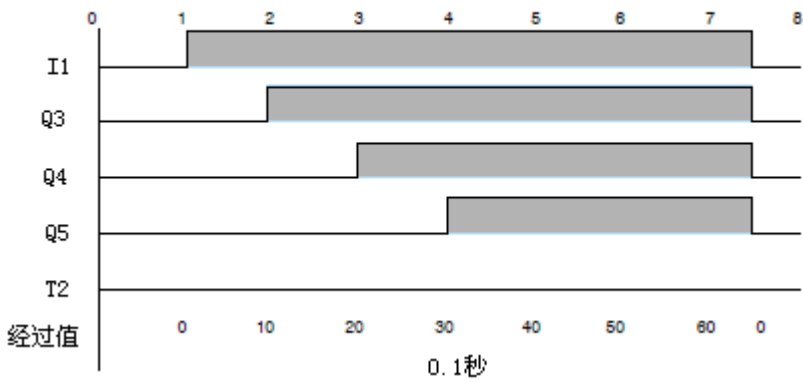
下面的例子中，定时器设定值是 234.5 秒，比较触点用来以 1 秒的间隔接通 Q3、Q4 和 Q5。当 I1 变为 OFF 时，定时器经过值被复位为 0，比较触点将 Q3、Q4 和 Q5 变为 OFF。

语句视图



```
LD I1
TMR T20 K2345
LDGE TA20 K10
OUT Q3
LDGE TA20 K20
OUT Q4
LDGE TA20 K30
OUT Q5
```

时序图



5.3 0.1 秒累积定时器（ATMR）和 0.01 秒累积定时器(AHTMR)

输入 0.1 秒累积定时器 ATMR 以 0.1 秒为单 位 进 行 增 计 时 ， 定 时 范 围 为 0-9999999.9。ATMR 占用两个 R 寄存器。

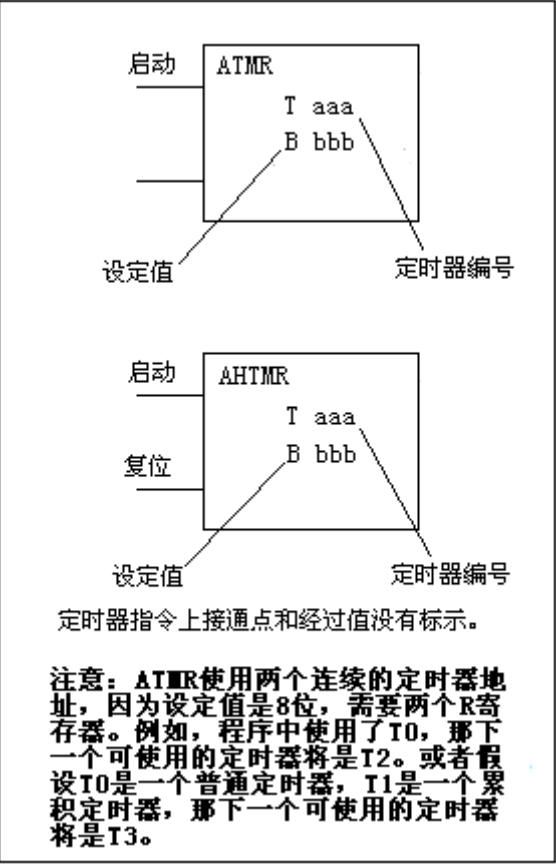
0.01 秒累积定时器 AHTMR 以 0.01 秒为单 位 进 行 增 计 时 ， 定 时 范 围 为 0-9999999.99。AHTMR 占用两个 R 寄存器。

这两种定时器有两个输入：启动和复位。当启动输入为 ON 时计时开始；当启动输入为 OFF 时计时停止,但是经过值保持而不复位。当复位输入为 ON 时经过值复位；当复位输入为 OFF 时计时允许。

指令详述：
定时器编号（T aaa）：定时器编号
设定值（B bbb）：常数（K）或两个连续的寄存器地址（R）。

经过值：计时器经过值是一个双字，存储在相对应的 R 寄存器或 T 存储地址中。例如，定时器 T3 的经过值存放在 R3 和 R4 中。

定时器接通点:经过值等于或大于设定值时，定时器接通点接通，例如，定时器 2 接通时 T2 为 ON。



注意：累积定时器占用两个连续的 R 寄存器，因此，累积定时器的编号不可连续，例如，使用了定时器 ATMR T1，那么下一个有效的定时器应该是 T3 而不是 T2。

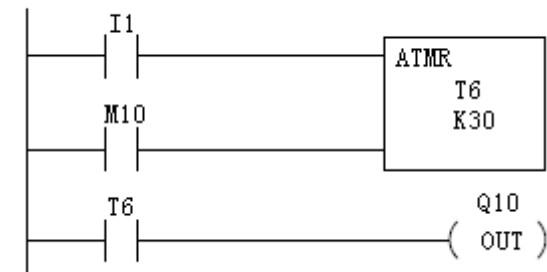
注意：仅当程序需要改变设定值时，才将设定值放入寄存器中。

操作数类型		NK1 范围	
	B	aaa	bbb
定时器	T	0-776	-
设定值寄存器	R	-	所有（附录 1）
指针（仅设定值）	P	-	所有（附录 1）
常数（仅设定值）	K	-	0-9999

定时器有两种用法，一种是使用定时器接通点触发某一事件，一种是使用比较指令在同一个定时器的不同时间点进行比较来触发事件。下面有这两种方法的编程举例。

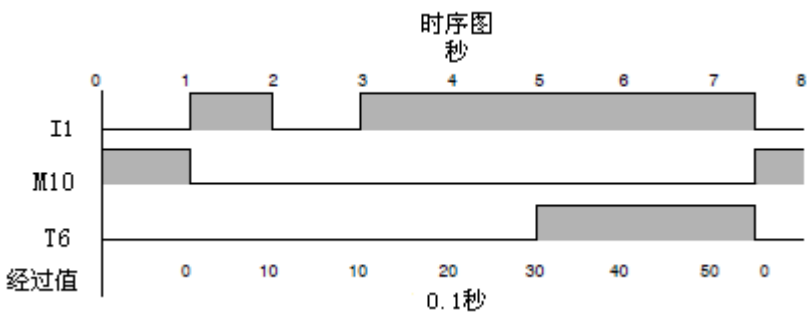
5.3.1 使用累积定时器接通点的程序举例

下面的例子中，累积定时器设定值是 3 秒，当定时器计时到 3 秒时，定时器接通点 T6 变为 ON。注意本例中，累积定时器先计时 1 秒，然后停 1 秒，然后再继续计时，当 M10 变为 ON 时，定时器经过值复位，接通点 T2 变为 OFF。



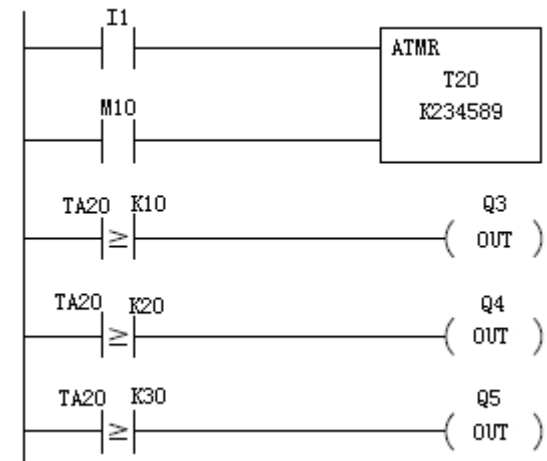
语句视图

```
LD I1
LD M10
ATMR T6 K30
LD T6
OUT Q10
```



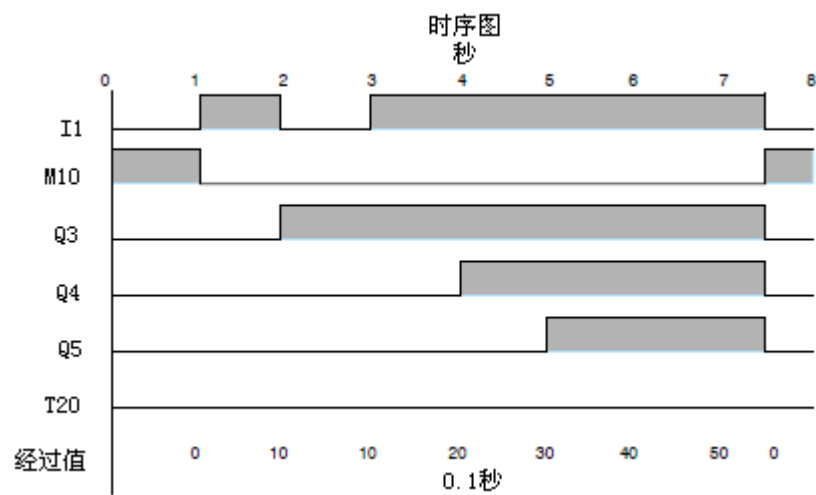
5.3.2 使用比较指令的程序举例

下面的例子中，累积定时器设定值是 23458.9 秒，比较触点以 1 秒的间隔来接通 Q3、Q4 和 Q5。当 M10 变为 ON 时，定时器经过值被复位为 0，比较触点将 Q3、Q4 和 Q5 变为 OFF。



语句视图

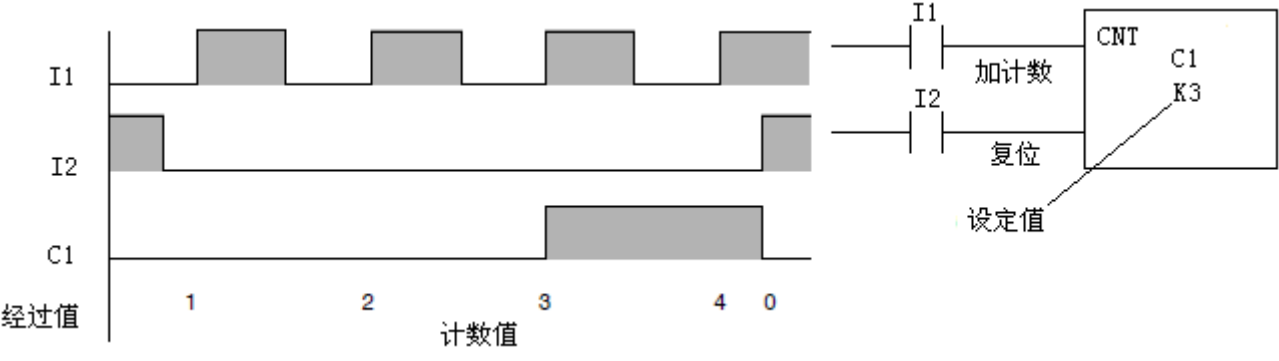
```
LD I1
LD M10
ATMR T20 K234589
LDGE TA20 K10
OUT Q3
LDGE TA20 K20
OUT Q4
LDGE TA20 K30
OUT Q5
```



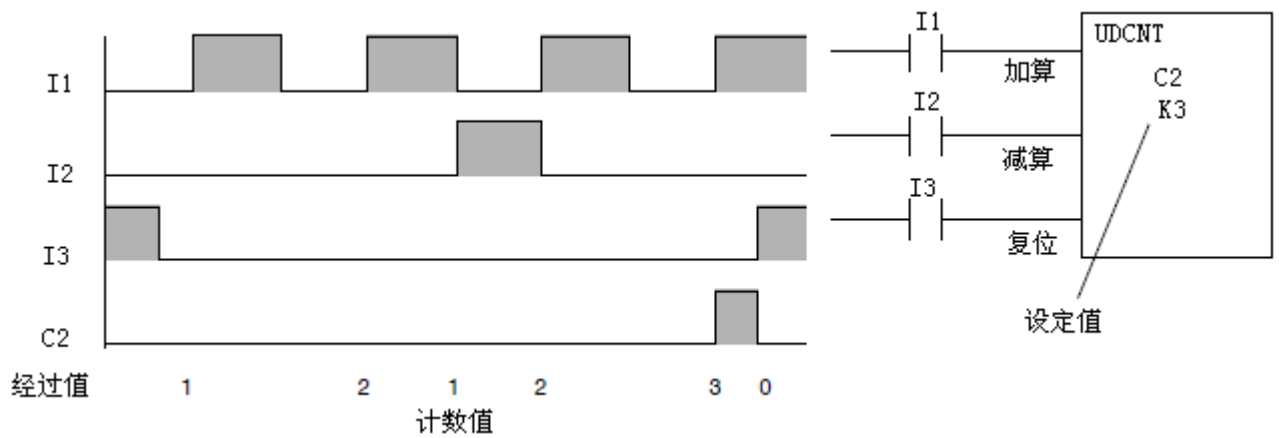
5.4 计数器的使用方法

计数器用于计数事件。计数器有加计数器、加减计数器和级式计数器（用于级式语言编程）。

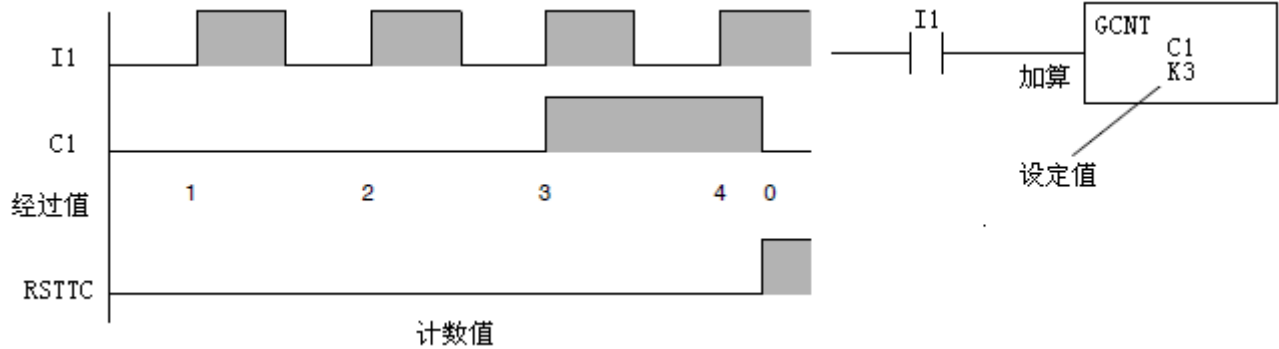
加算计数器（带复位端）有两个输入，一个计数输入一个复位输入。最大计数值是 9999。下面的时序图演示了计数输入、计数复位、对应开关量、经过值和设定值之间的关系。



加减计数器有三个输入：加算输入、减算输入和复位输入。最大计数值是 99999999。下面的时序图演示了计数输入、计数复位、对应开关量、经过值和设定值之间的关系。



加算计数器（不带复位端）有一个计数输入，由 RSTTC 指令复位。这个计数器用于级式语言编程，最大计数值是 9999，下面的时序图演示了计数输入、对应开关量、经过值、设定值和复位指令之间的关系。



5.5 加算计数器(带复位端) (CNT)

输入 CNT 指令有两个输入，当计数输入由 OFF 变为 ON 时开始加计数；当复位输入为 ON 时，计数值复位到 0。当经过值等于设定值时，对应开关量变为 ON，计数值继续增加到最大值 9999。计数最大值保持到计数器被复位。

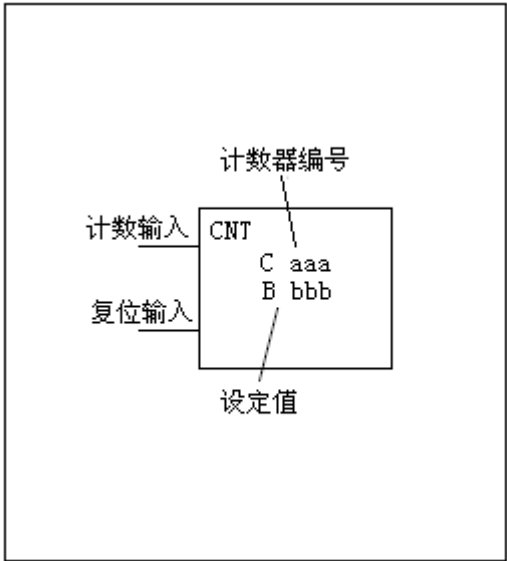
指令详述：

计数器编号 (C aaa)：计数器编号

设定值 (B bbb)：常数 (K) 或 R 寄存器地址。

经过值：计数器经过值存储在相对应的 R 寄存器或 C 存储地址中。R 寄存器地址是计数器编号+1000。例如，计数器 C3 的经过值存放在 R1003 中。

计数器接通点：经过值等于或大于设定值时，计数器接通点接通，例如，计数器 2 接通时 C2 为 ON。



计数器指令上接通点和经过值没有标示

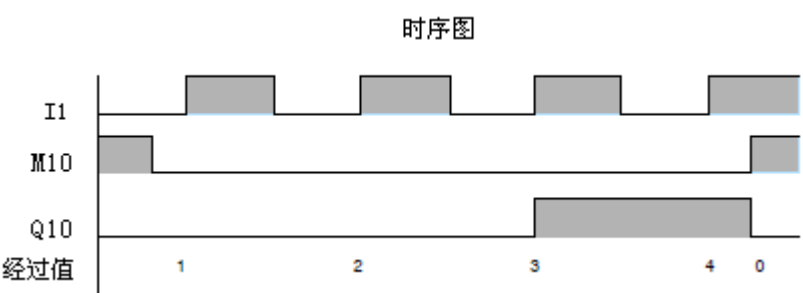
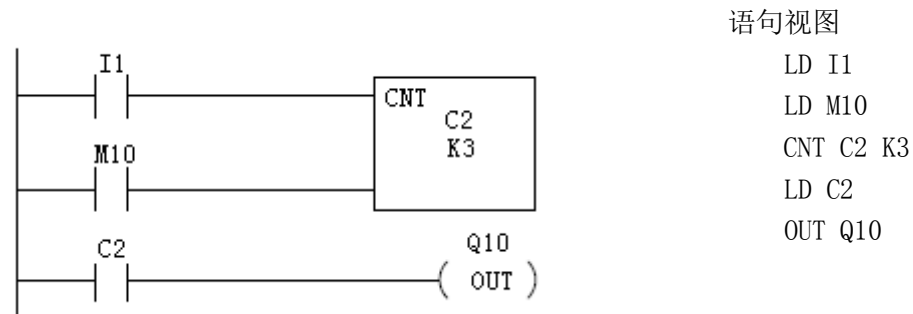


注意： 仅当程序需要改变设定值时，才将设定值放入寄存器中。

操作数类型		NK1 范围	
	B	aaa	bbb
计数器	C	0-777	-
设定值寄存器	R	-	所有（附录 1）
指针（仅设定值）	P	-	所有（附录 1）
常数（仅设定值）	K	-	0-9999

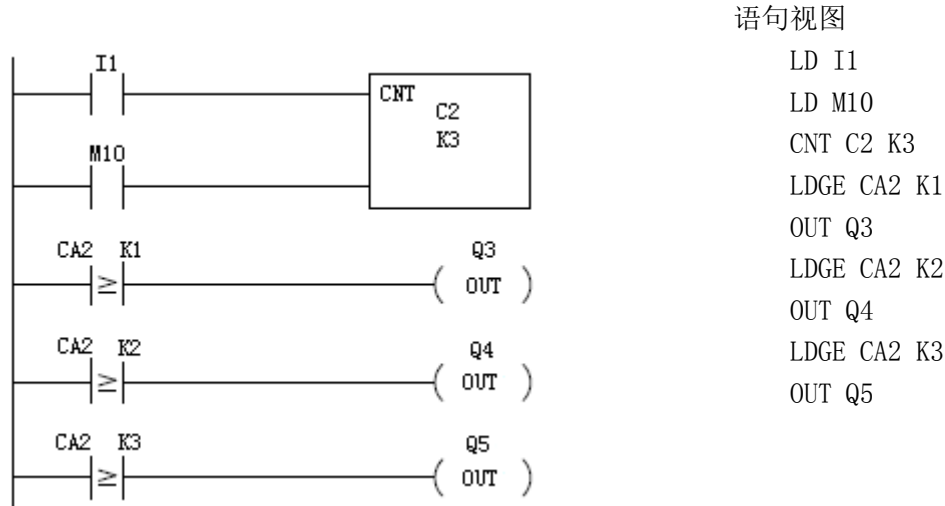
5.5.1 使用计数器接通点的程序举例

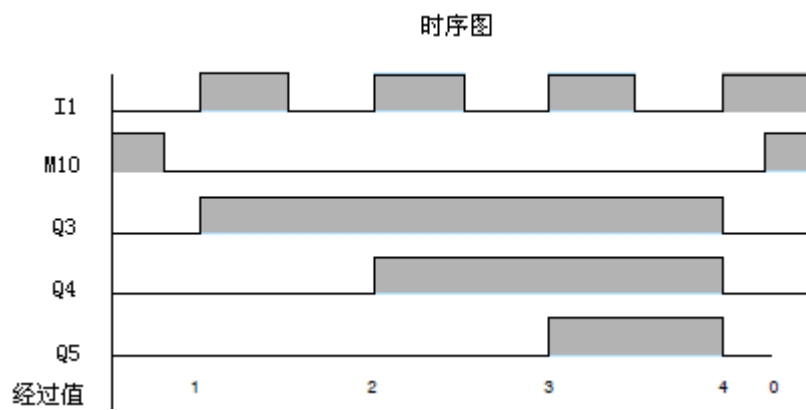
下面的例子中，每当 I1 由 OFF 变为 ON 时，计数器 C2 就加 1。当计数值等于设定值 3 时，计数器接通点 C2 变为 ON，Q10 被接通。当复位信号 M10 变为 ON 时，计数器接通点变为 OFF，经过值复位为 0。计数器 C2 的经过值保存在寄存器 R1002 中。



5.5.2 使用比较指令的程序举例

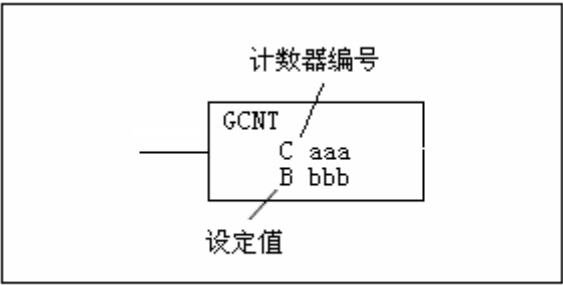
下面的例子中，每当 I1 由 OFF 转变为 ON，计数器 C2 将加 1。比较触点用于在不同的计数值将 Q3、Q4 和 Q5 接通。当计数器被复位时，比较触点将变为 OFF。当复位输入 M10 变为 ON 时，计数器接通点将变为 OFF，计数经过值被复位为 0。





5.6 加算计数器(不带复位端) (GCNT)

输入 GCNT 指令只有一个输入，当这个输入由 OFF 变为 ON 时开始加计数。这个计数器区别于其它计数器，因为它将保持经过值直到 RSTTC 指令将其复位。GCNT 指令设计用于级式语言编程，也可以用于继电器程序中。当经过值等于设定值时，对应开关量变为 ON，计数值继续增加到最大值 9999。计数最大值保持到计数器被复位。



计数器指令上接通点和经过值没有标示。

指令详述：

计数器编号 (C aaa)：计数器编号

设定值 (B bbb)：常数 (K) 或 R 寄存器地址。

经过值：计数器经过值存储在相对应的 R 寄存器或 C 存储地址中。R 寄存器地址是计数器编号+1000。例如，计数器 C3 的经过值存放在 R1003 中。

计数器接通点：经过值等于或大于设定值时，计数器接通点接通，例如，计数器 2 接通时 C2 为 ON。



注意：当将 GCNT 用于级中时，在 GCNT 指令的输入发生一个 0→1 的转换时，其所在的级至少被激活了一个扫描周期，否则，相当于输入没有发生 0→1 的转换，计数器将不计数。

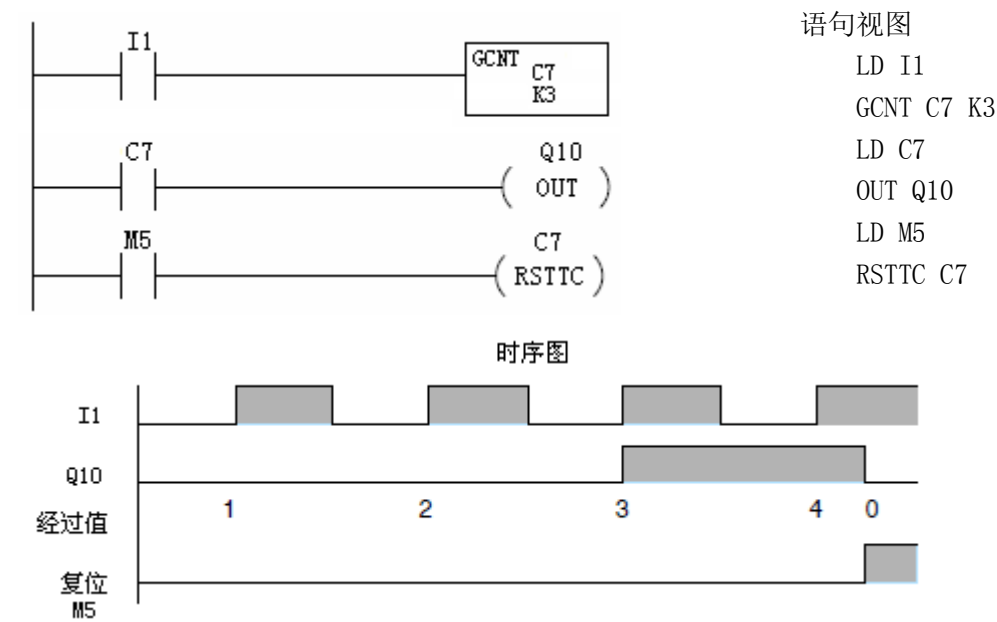


注意：仅当程序或操作单元需要改变设定值时，才将设定值放入寄存器中。

操作数类型		NK1 范围	
	B	aaa	bbb
计数器	C	0-777	-
设定值寄存器	R	-	所有（附录 1）
指针（仅设定值）	P	-	所有（附录 1）
常数（仅设定值）	K	-	0-9999

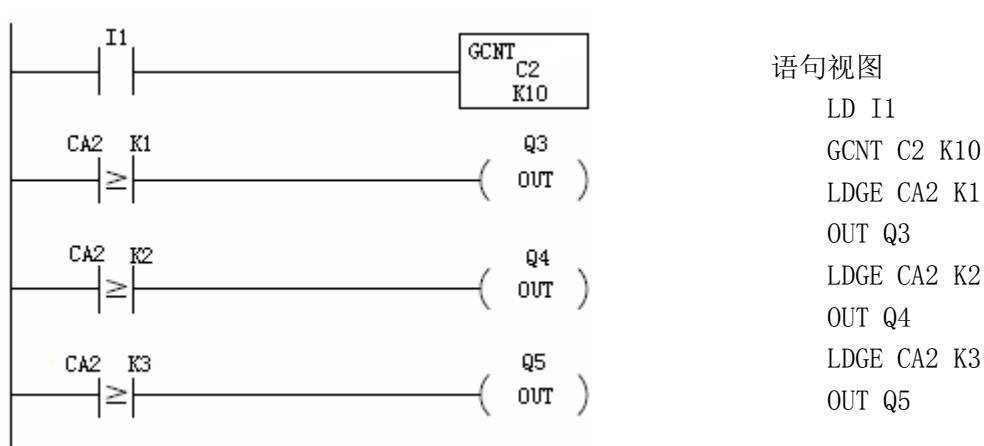
5.6.1 使用计数器接通点的程序举例

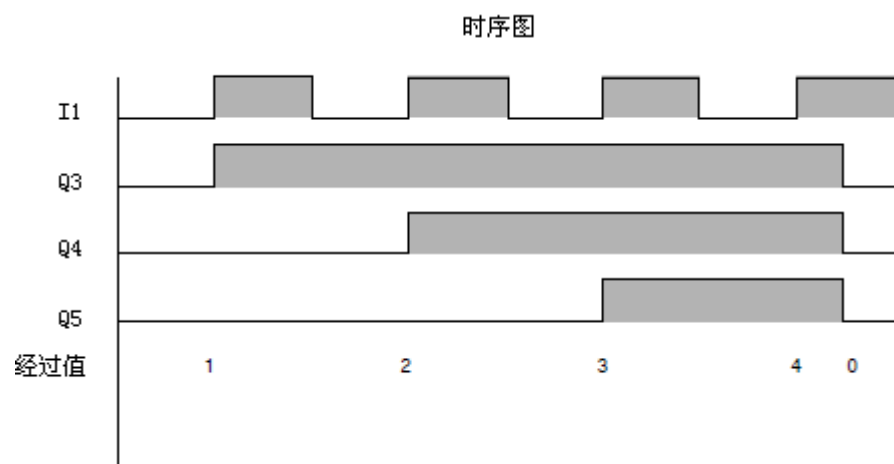
下面的例子中，每当 I1 由 OFF 变为 ON 时，计数器 C7 就加 1。当计数值等于设定值 3 时，计数器接通点 C7 变为 ON，Q10 被接通。计数器接通点一旦接通就保持 ON 状态直到使用 RSTTC 指令将其复位。计数器被复位后，经过值被复位为 0。计数器 C7 的经过值保存在寄存器 R1007 中。



5.6.2 使用比较指令的程序举例

下面的例子中，每当 I1 由 OFF 变为 ON，计数器 C2 将加 1。比较触点用于在不同的计数值将 Q3、Q4 和 Q5 接通。当计数器被复位时，比较触点将变为 OFF，计数经过值被复位为 0，计数经过值被存放在 R1002 中。





5.7 加减计数器（UDCNT）

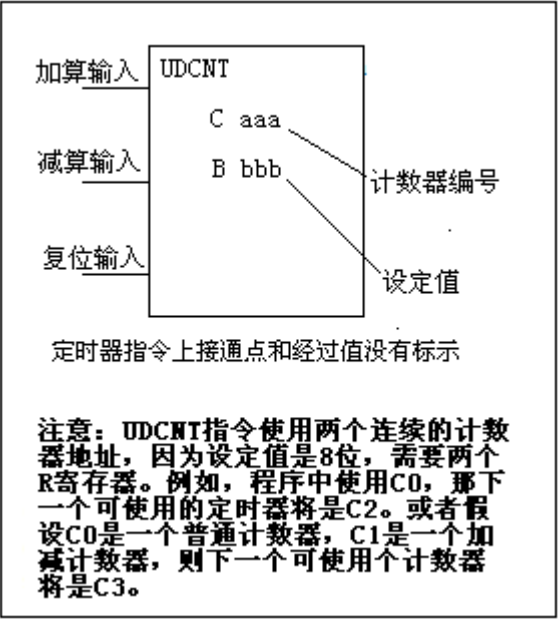
输入 UDCNT 指令，当加算输入由 OFF 变为 ON 时开始加计数；当减算输入由 OFF 变为 ON 时开始减计数；当复位输入为 ON 时，计数值复位为 0。计数范围是 0-99999999。不使用的计数输入要保持 OFF 状态以使计数功能有效。

指令详述：

计数器编号（C aaa）：计数器编号
设定值（B bbb）：常数（K）或两个连续的 R 寄存器地址。

经过值：计数器经过值是一个双字值，存储在相对应的 R 寄存器或 C 存储地址中。R 寄存器地址是计数器编号+1000。例如，计数器 C5 的经过值存放在 R1005 和 R1006 中。

计数器接通点：经过值等于或大于设定值时，计数器接通点接通，例如，计数器 C2 接通时 C2 为 ON。



注意：加减计数器 UDCNT 占用两个连续的 R 寄存器，因此，加减计数器的编号不可连续，例如，使用了计数器 UDCNT C1，那么下一个有效的计数器应该是 C3 而不是 C2。

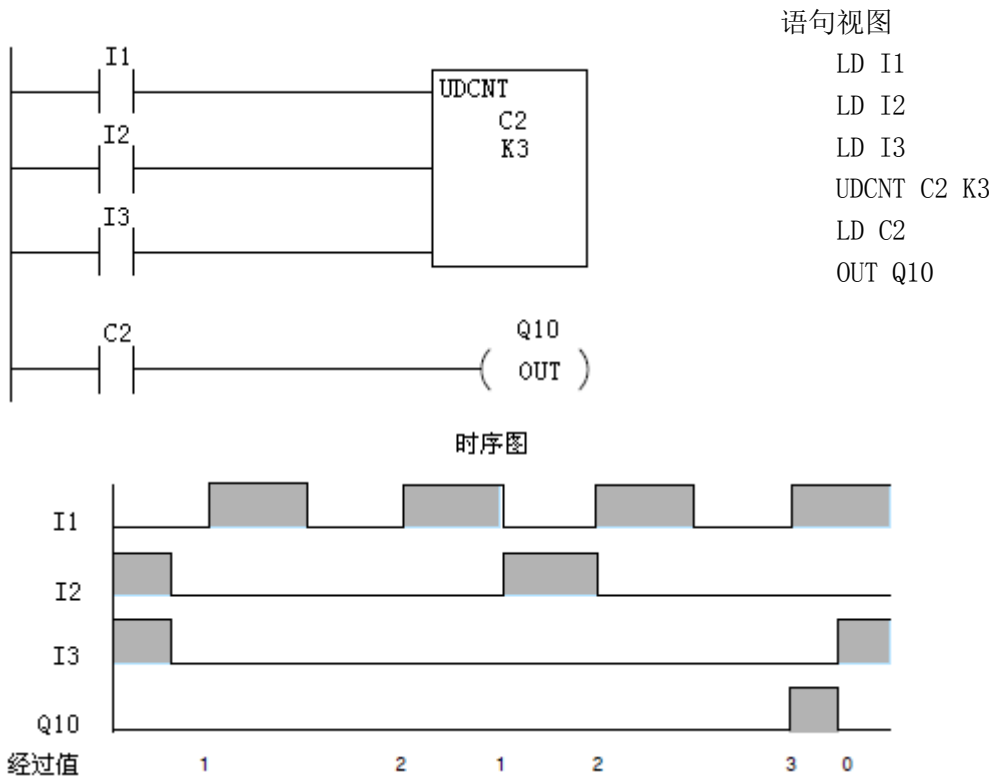


注意：仅当程序或操作单元需要改变设定值时，才将设定值放入寄存器中。

操作数类型		NK1 范围	
	B	aaa	bbb
计数器	C	0-776	-
设定值寄存器	R	-	所有（附录 1）
指针（仅设定值）	P	-	所有（附录 1）
常数（仅设定值）	K	-	0-99999999

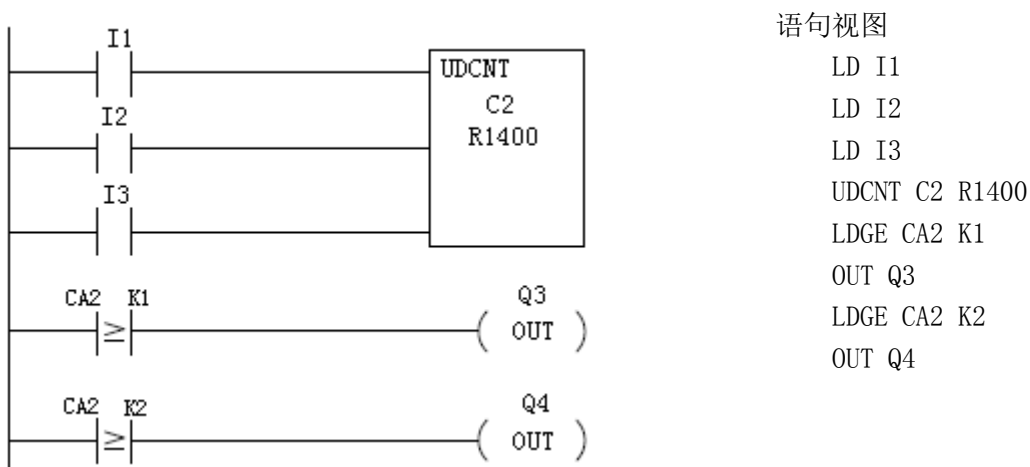
5.7.1 使用计数器接通点的程序举例

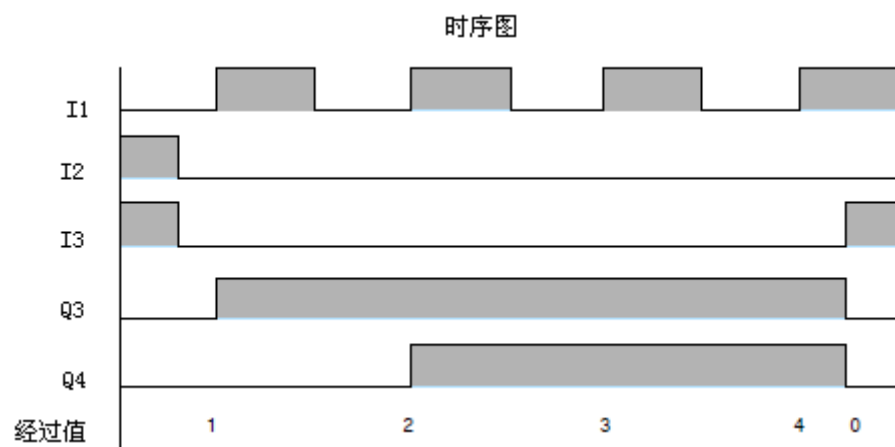
下面的例子中，如果 I2 和 I3 为 OFF，当 I1 由 OFF 变为 ON 时，计数器 C2 就加 1。如果 I1 和 I3 为 OFF，当 I2 由 OFF 变为 ON 时，计数器 C2 就减 1。当计数值等于设定值 3 时，计数器接通点 C2 变为 ON，Q10 被接通。当复位信号 I3 变为 ON 时，计数器接通点变为 OFF，经过值复位为 0。



5.7.2 使用比较指令的程序举例

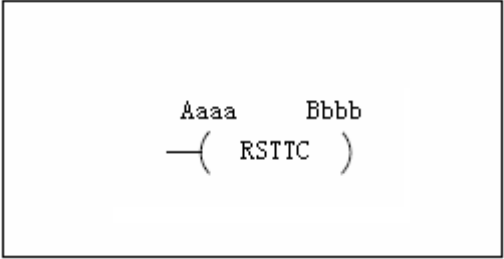
下面的例子中，每当 I1 由 OFF 变为 ON，计数器 C2 将加 1。比较触点用于在不同的计数值将 Q3 和 Q4 接通。当计数器被复位时，比较触点将变为 OFF。当复位输入 I3 变为 ON 时，计数器接通点将变为 OFF，计数经过值被复位为 0。





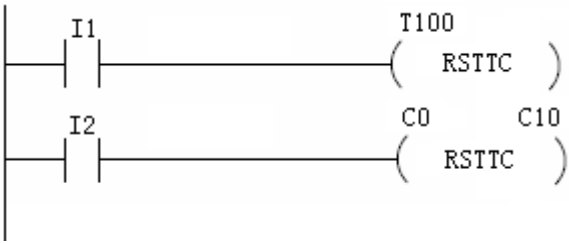
5.8 定时器、计数器复位指令（RSTTC）

输入 RSTTC 指令是使定时器/计数器的经过值复位的指令。其可以是单步指令，也可以是两步指令（对一个范围复位）。执行此指令后，定时器/计数器的经过值复位为“0”。



操作数类型		NK1 范围	
A/B		aaa	bbb
定时器	T	0-777	0-777
计数器	C	0-777	0-777

下面的例子中，当 I1 为 ON 时，定时器 T100 的经过值为复位为 0；当 I2 为 ON 时，计数器 C0-C10 的经过值被复位为 0. 。



语句视图

```
LD I1
RSTTC T100
LD I2
RSTTC C0 C10
```

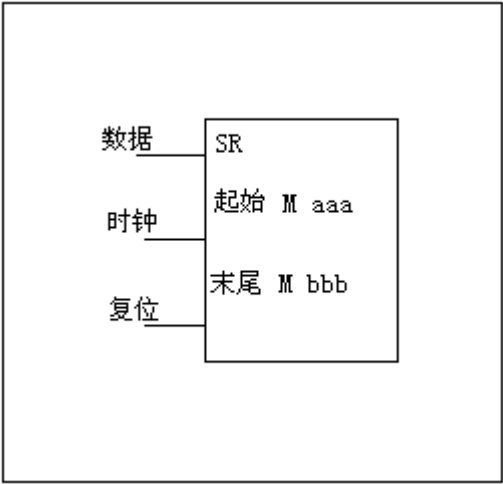
5.9 移位寄存器指令（SR）

输入 SR 指令将预先确定范围的内部线圈移位。内部线圈范围以 8 点为单位，起始点和末尾点必须是 8 点的两端，也就是说如果起始定位号是 $M\times\times\times 0$ ，则末尾定义号为 $M\times\times\times 7$ ，移位方向是 $M\times\times\times 0\rightarrow M\times\times\times 7$ ；如果起始定位号是 $M\times\times\times 7$ ，则末尾定义号为 $M\times\times\times 0$ ，移位方向是 $M\times\times\times 7\rightarrow M\times\times\times 0$ 。

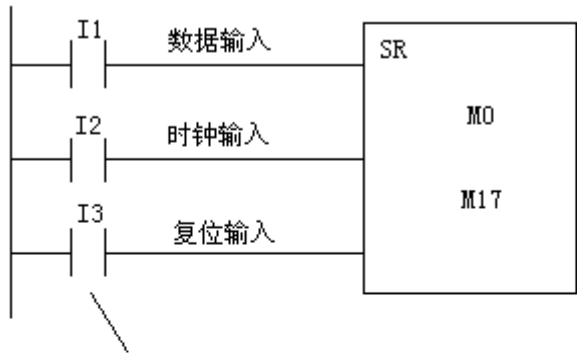
SR 指令的三个输入：

- 数据：决定了要进入存储器的值（是 1 或是 0）。
- 时钟：当时钟输入由 OFF→ON 时，将数据输入的状态送入起始位，同时移位寄存器内各位的状态均向下移 1 位。
- 复位：将移位寄存器复位。

当时钟输入由 OFF→ON 时，将数据输入的状态送入起始位，同时移位寄存器内各位的状态向下移 1 位，移位方向取决于起始和末尾地址。例如，假设起始定义号为 M0，末尾定义号为 M17，则这 16 位由低地址位向高地址位移动；假设起始定义号为 M17，末尾定义号为 M0，则这 16 位由高地址位向低地址位移动。最大移位范围取决于可用内部线圈地址范围，最小移位范围是 8 点内部线圈位。



操作数类型		NK1 范围	
		aaa	bbb
内部线圈	M	0-3777	0-3777



```
语句视图
LD I1
LD I2
LD I3
SR M0 M17
```



第 6 章 累加器/数据堆栈和输出指令

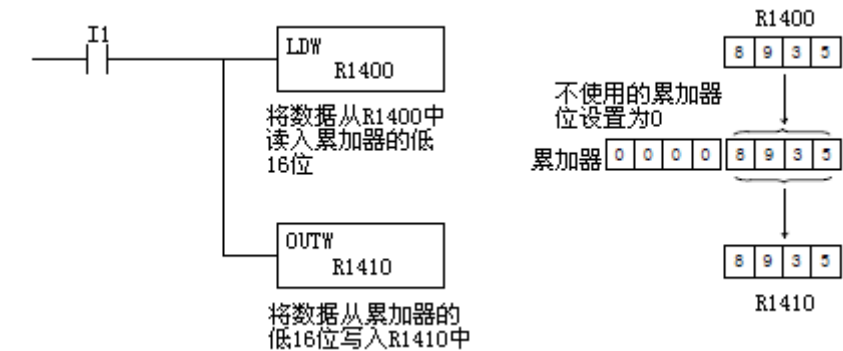
6.1 累加器、堆栈的使用方法

6.1.1 累加器的使用方法

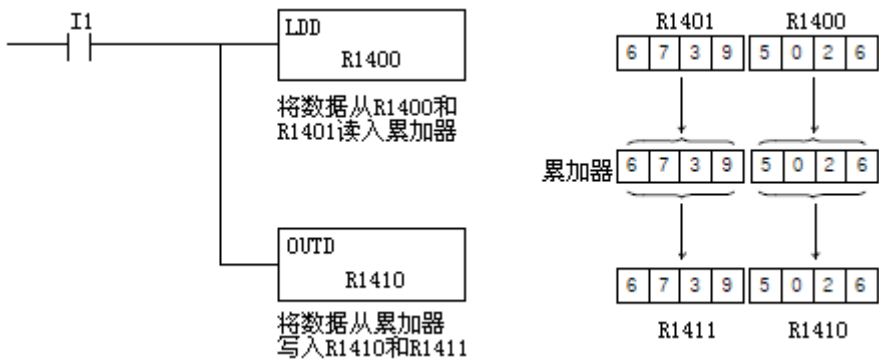
累加器是一种特殊寄存器，由 32 位组成，临时存储数据处理结果。例如，必须使用累加器来完成加、减、除等数学运算。因为累加器是 32 位，可以存放一个 8 位 BCD 数，或是一个 32 位二进制补码数。每个 CPU 扫描周期结束累加器都被复位为 0。

数据读入累加器

读入和写入指令用于将数据从 R 寄存器读入累加器或是将数据从累加器写入 R 寄存器。下面的例子将数据从 R1400 写入 R1410。

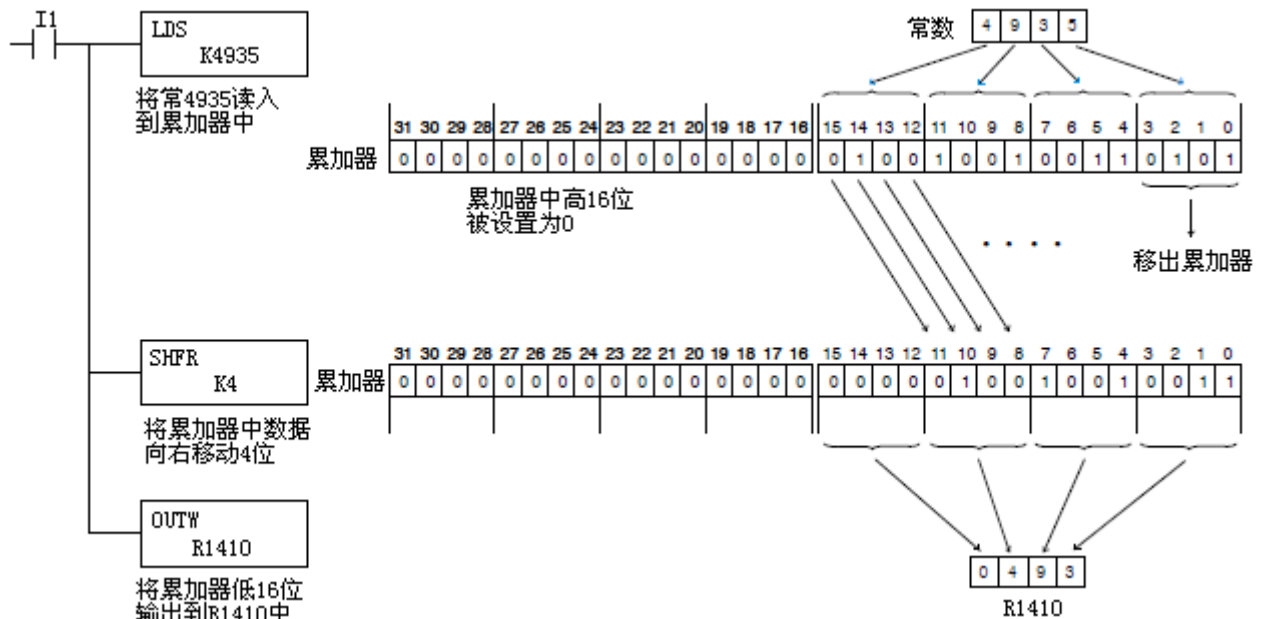


由于累加器是 32 位，R 寄存器是 16 位，因此 LDD 和 OUTD 指令（及其类似指令）可将两个连续的 R 寄存器数据或一个 8 位 BCD 数读入累加器或从累加器中写入到 R 寄存器中。例如，如果想把 R1400 和 R1401 中的数据读入到 R1410 和 R1411 中，最简单有效的方法如下：

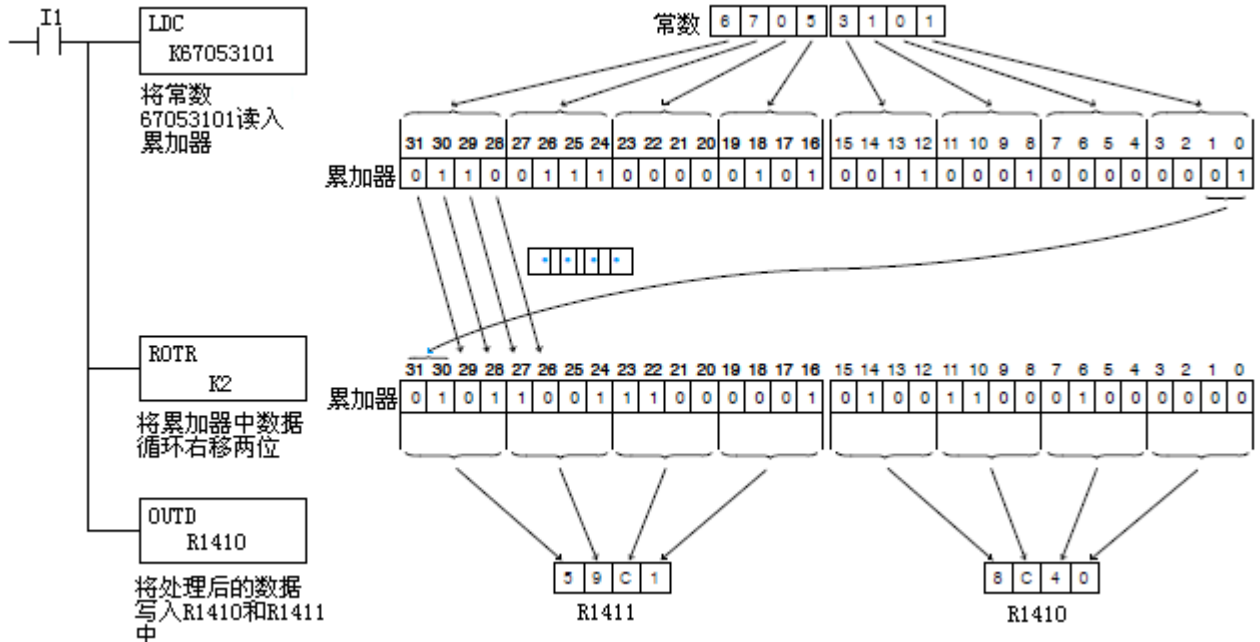


改变累加器数据

处理数据的指令也要用到累加器。数据在累加器中处理，处理过的数据暂时存放在累加器中，下面的例子中，将 BCD 常数 4935 读入累加器中，右移 4 位后将结果输出到 R1410 中。

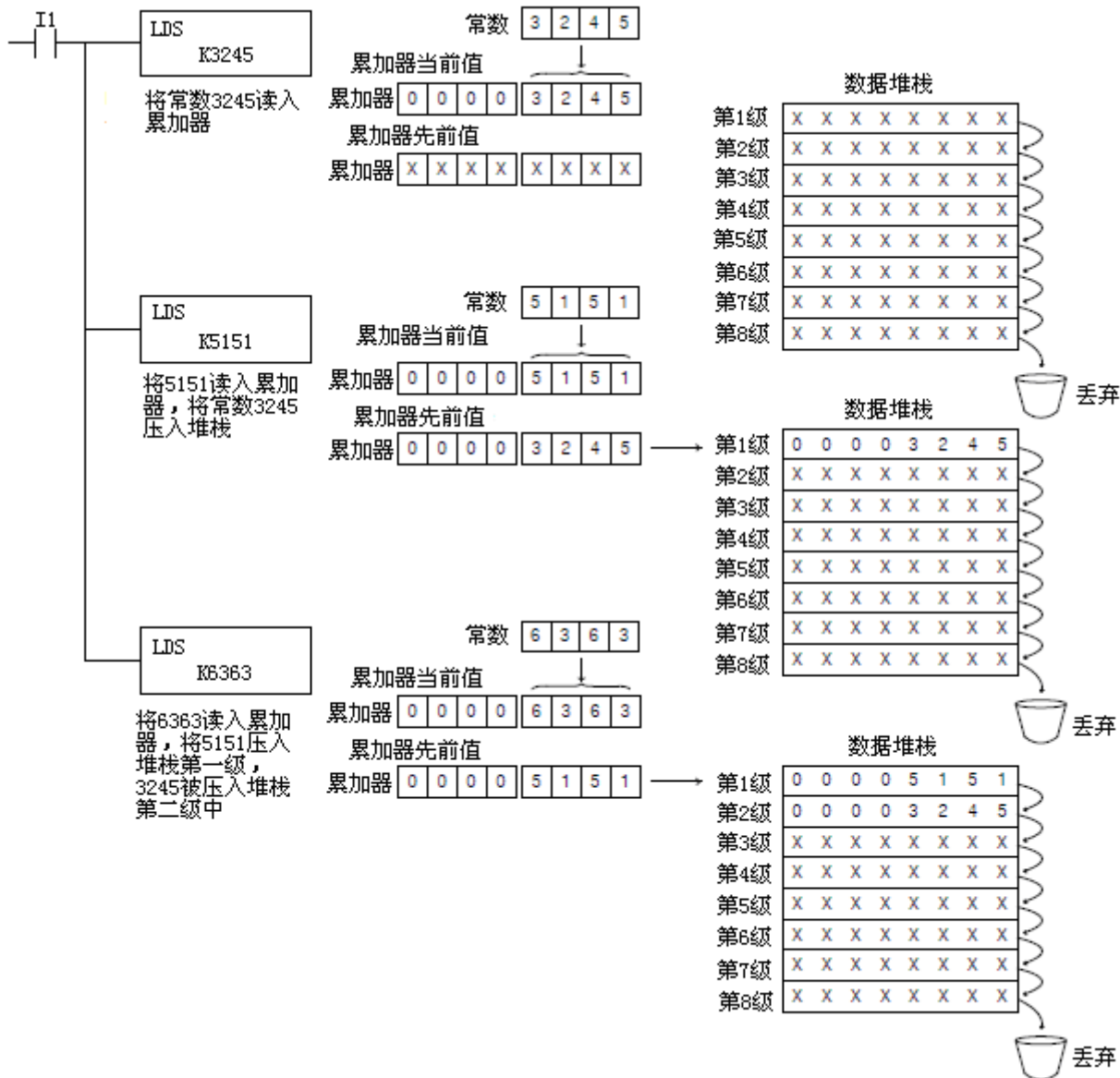


有些处理数据的指令会使用到累加器的 32 位。这些指令处理累加器中的两个连续的 R 寄存器数据或一个 8 位 BCD 数。下面的例子中，将 BCD 数 67053101 循环右移两位后输出到 R1410 和 R1411 中。

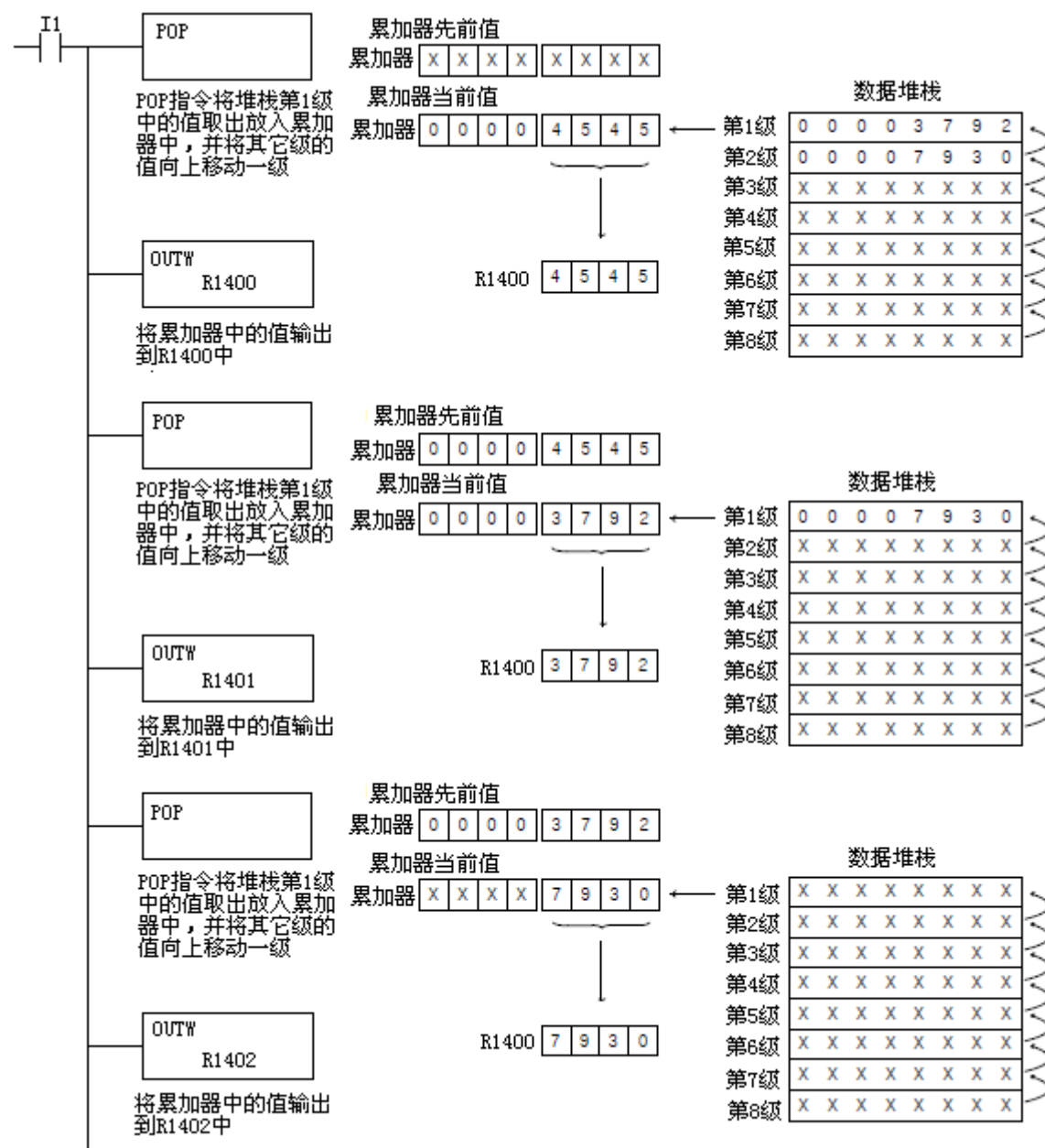


6.1.2 数据堆栈的使用方法

数据堆栈用于需要一个以上的参数来执行一个功能或用户自定义功能的指令。数据堆栈用于有多个读入指令执行，但是没有使用输出指令的场合。第一个读入指令将一个数据读入累加器，之后每个不使用输出指令的读入指令都将数据读入累加器，而累加器中的原有数据被压入数据堆栈。输出指令使其前面一个读入指令失效，当下一个读入指令执行时，不会将累加器中的数据压入数据堆栈中。每次将一个数据压入数据堆栈，堆栈中原有的数据就被压入下一级。数据堆栈有 8 级（8 个 32 位寄存器）。如果第 8 级中有数据，当一个新数据被压入堆栈后，第 8 级中的数据被压出堆栈，并且不能恢复。



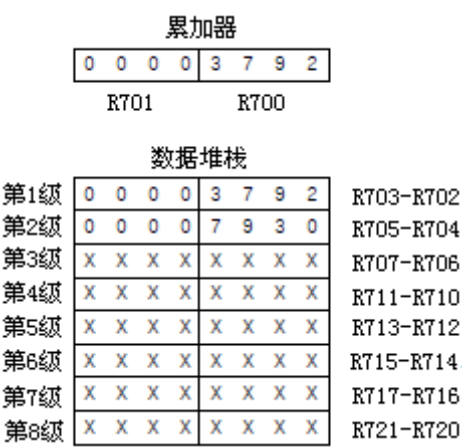
POP 指令将堆栈第 1 级的数据取出放入累加器中，累加器中记忆的内容在执行 POP 指令之后丢失，堆栈中 2-8 级的内容依次上移一级。



6.1.3 累加器、数据堆栈中的数据地址

有时需要读取数据堆栈的数据，但是尚未将其从堆栈中弹出。累加器和数据堆栈中有相应的累加器地址，可以通过程序访问。

不可向这些地址写入数据，但是可以读取它们的数据，或将它们用于比较指令等。



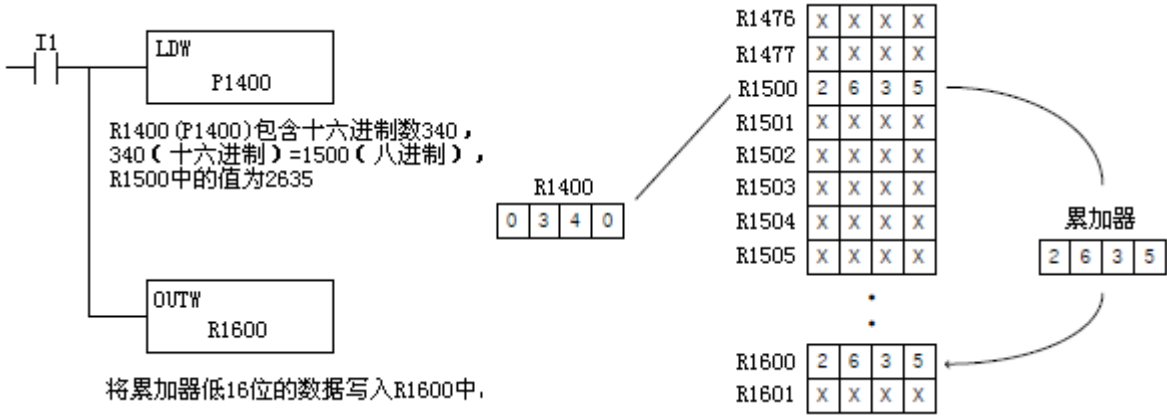
6.1.4 指针的使用方法

NK1 系列的许多指令允许将 R 寄存器指针作为操作数。指针在梯形图编程中很有用，但是如果如果没有指针（也被称为间接寻址）的使用经验的话可能比较难理解。指针允许指令通过指针值获取 R 寄存器地址中的数据。

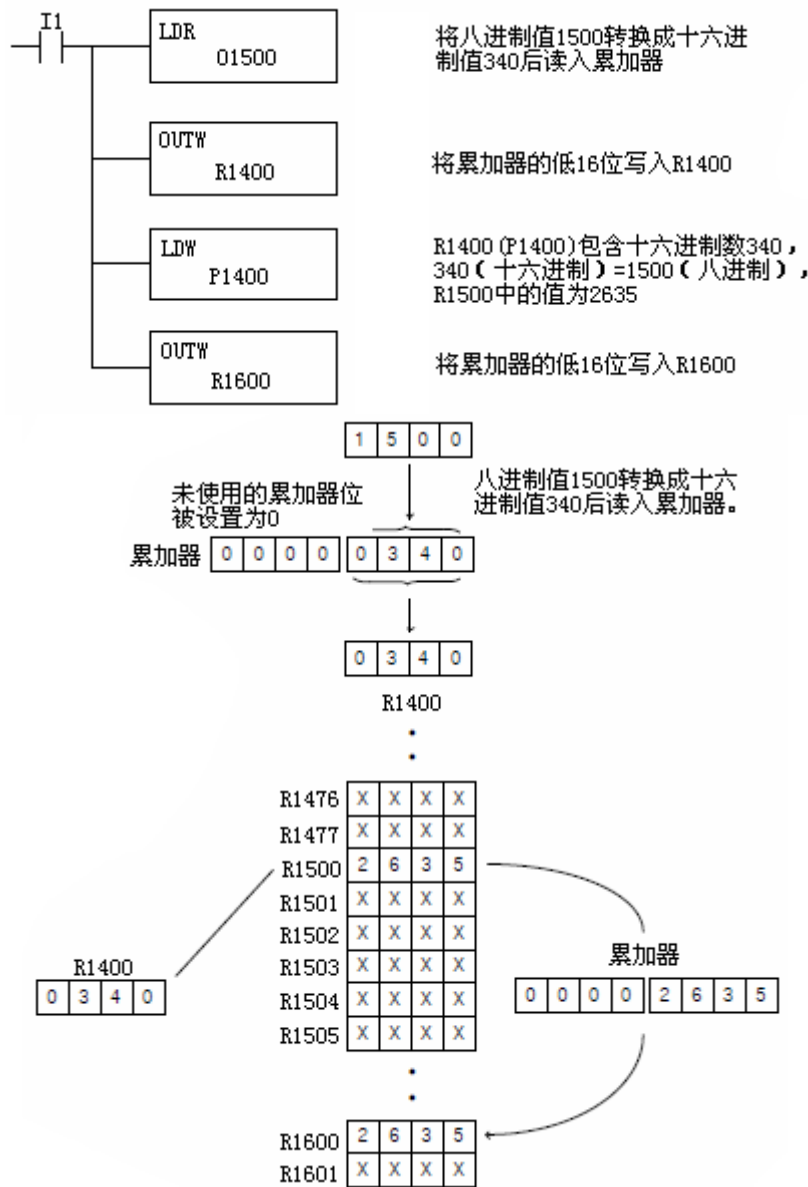


注意： NK1 中 R 寄存器地址是八进制的，但是 R 寄存器中的地址值是十六进制值。使用 LDR 读入指令将地址值读入指针，指令自动完成八进制到十六进制的转换。

下面的例子中，在读入指令中使用指针作为操作数。R 寄存器地址 1400 是指针地址，R1400 中的数据 340 是十六进制值，其对应的八进制值是 1500。CPU 将 R1500 中的数据(2635)读入累加器的低位中。

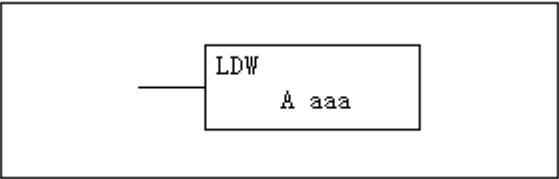


下面的例子同上面的例子类似，LDR 指令自动将八进制地址转换成十六进制数。



6.2 读入指令（LDW）

输入 LDW 指令是一个 16 位指令，将指定 R 寄存器中的数据读入累加器的低 16 位中，高 16 位被设置为 0。



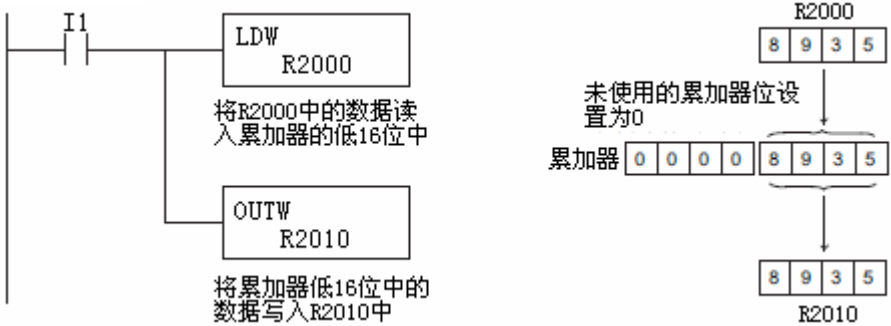
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP76	读入累加器的数据为 0 时 ON



注意：如果两个读入指令连续，则第一个读入指令的值将被压入数据堆栈的第 1 级中。

下面的例子中，当 I1 为 ON 时，R2000 中的数据被读入累加器中然后写入到 R2010 中。

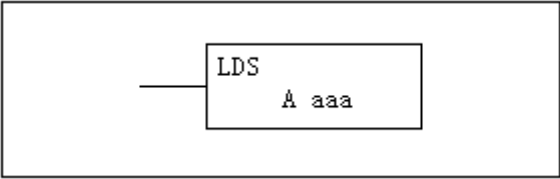


语句视图

```
LD I1
LDW R2000
OUTW R2010
```

6.3 读入指令（LDS）

 LDS 指令是一个 16 位指令，将一个 4 位常数读入累加器的低 16 位中，高 16 位被设置为 0。



操作数类型		NK1 范围
	A	aaa
常数	K	0-FFFF

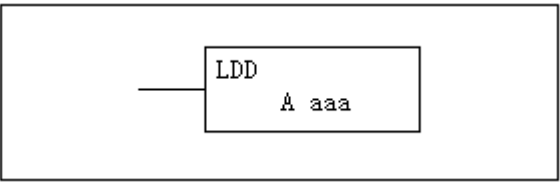
受影响的标志线圈	描述
SP76	读入累加器中的数据为 0 时 ON



注意：如果两个读入指令连续，则第一个读入指令的值将被压入数据堆栈的第 1 级中。

6.4 读入指令（LDD）

输入 LDD 指令是一个 32 位指令，将指定的两个连续的寄存器中的数据读入累加器。



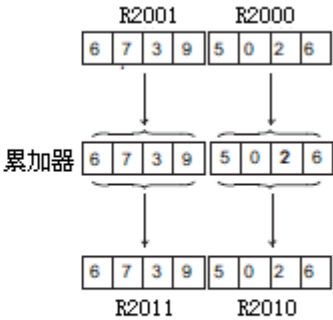
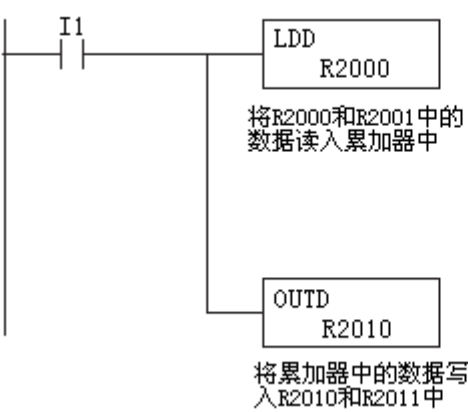
操作数类型		NK1 范围
A		aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP76	读入累加器中的数据为 0 时 ON



注意：如果有两个连续的读入指令，第一个读入指令的数据将被压入数据堆栈中。

下面的例子中，当 I1 为 ON 时，R2000 和 R2001 中的值将被装入累加器中然后输出到 R2010 和 R2011 中。

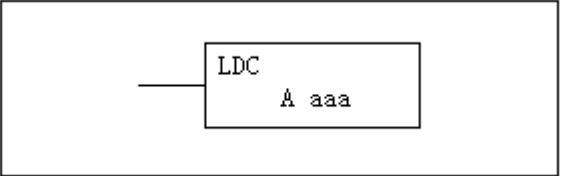


语句视图

```
LD I1
LDD R2000
OUTD R2010
```

6.5 读入指令（LDC）

 LDC 指令是一个 32 位指令，将 8 位常数读入累加器。



操作数类型	NK1 范围
A	aaa
常数	K 0-FFFFFFFF

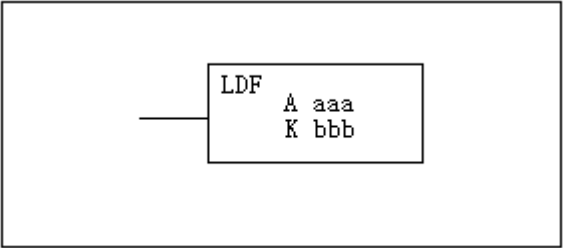
受影响的标志线圈	描述
SP76	读入累加器中的数据为 0 时 ON



注意：如果有两个连续的读入指令，第一个读入指令的数据将被压入数据堆栈中。

6.6 任意位读入指令（LDF）

输入 LDF 指令用于将 1-32 个连续位读入累加器，指令需要一个起始地址（A aaa）和一个位数（K bbb），不使用的累加器位被设置为 0。



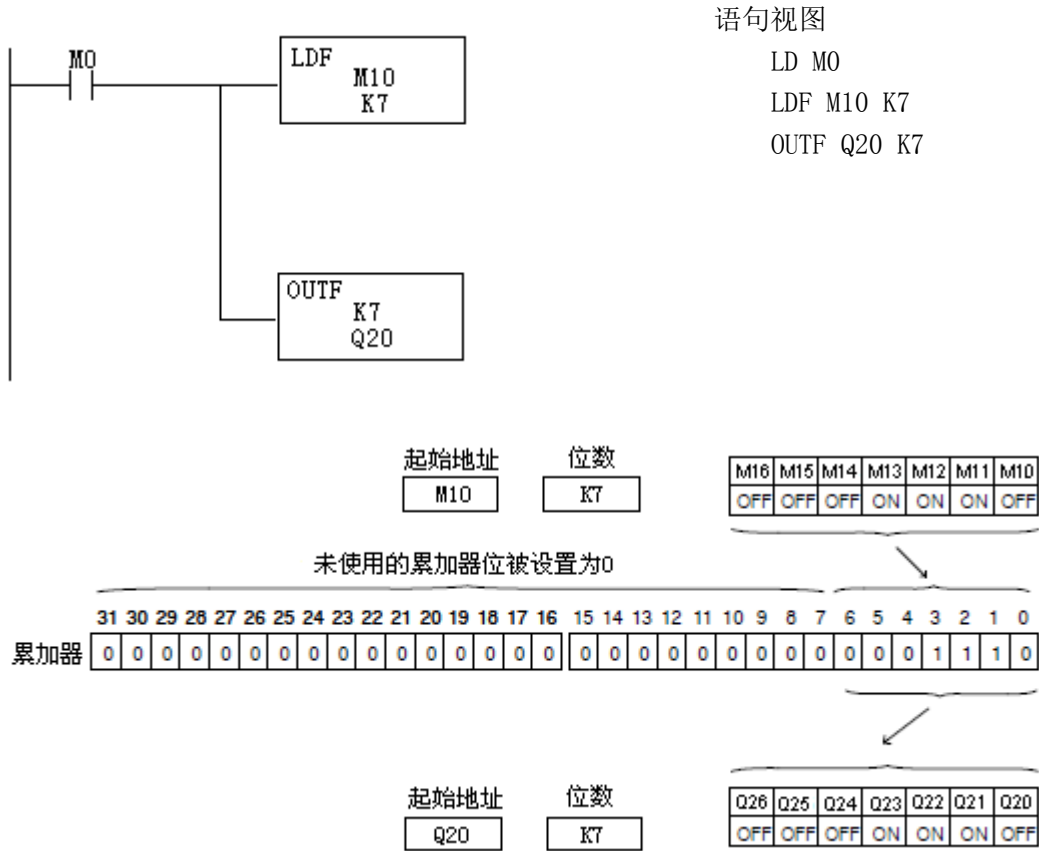
操作数类型		NK1 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777
级	S	0-1777
定时器	T	0-777
计数器	C	0-777
特殊线圈	SP	0-1777
通讯 I/O	GI/GQ	0-3777
常数	K	1-32

受影响的标志线圈	描述
SP76	读入指令读入的数据为 0 时 ON



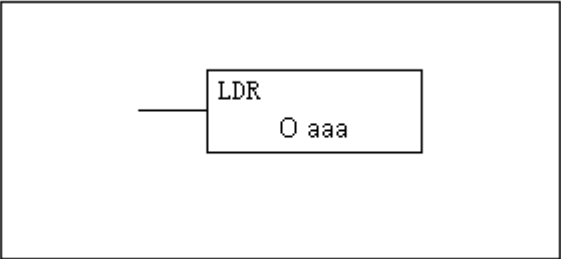
注意：如果有两个连续的读入指令，第一个读入指令的数据将被压入数据堆栈中。

下面的例子中，当 M0 为 ON 时，LDF 指令将 M10-M16（7 位）读入累加器中，然后 OUTF 指令将累加器的低 7 位写入到 Q20-Q26 中。



6.7 寄存器地址读入指令（LDR）

输入 LDR 指令是一个 16 位指令，它将八进制值或地址转换为十六进制值或地址并将其写入累加器。



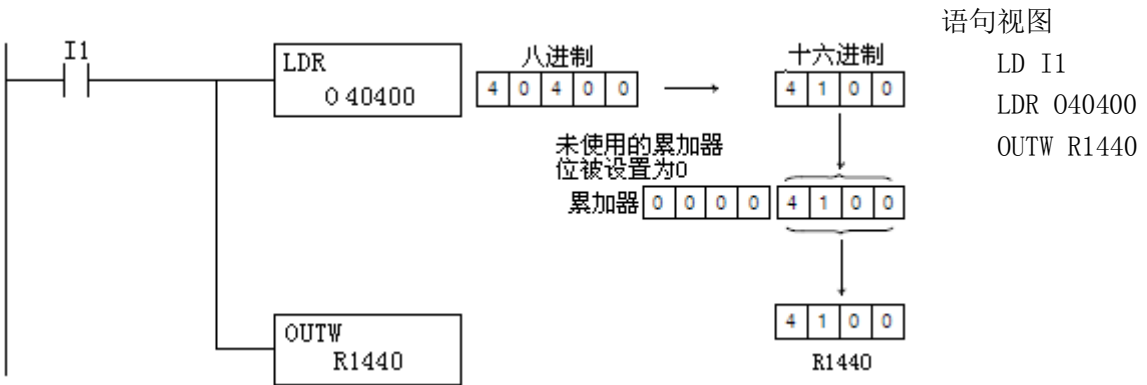
操作数类型	NK1 范围
	aaa
八进制地址	0 所有（附录 1）

受影响的标志线圈	描述
SP76	读入累加器中的数据为 0 时 ON



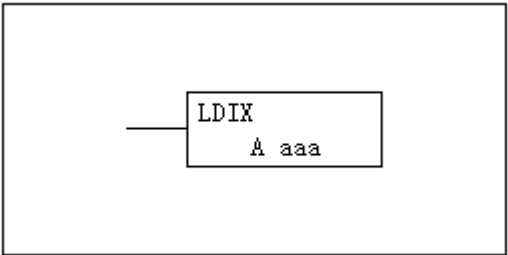
注意：如果有两个连续的读入指令，第一个读入指令的数据将被压入数据堆栈中。

下面的例子中，当 I1 为 ON 时，八进制数据 40400 将被转换为十六进制数据 4100 并被读入累加器中。累加器的低 16 位被写入 R1440 中。



6.8 索引 16 位读入指令（LDIX）

输入 LDIX 指令是一个 16 位指令，它指定一个源地址，这个源地址加上数据堆栈第 1 级中的偏移量得到一个新的寄存器地址，将这个新寄存器地址中的数据读入累加器的低 16 位中，累加器的高 16 位被设置为 0。堆栈第 1 级中的数据需是十六进制数。



小技巧：LDR 指令可将八进制数转换为十六进制数并将其读入累加器。

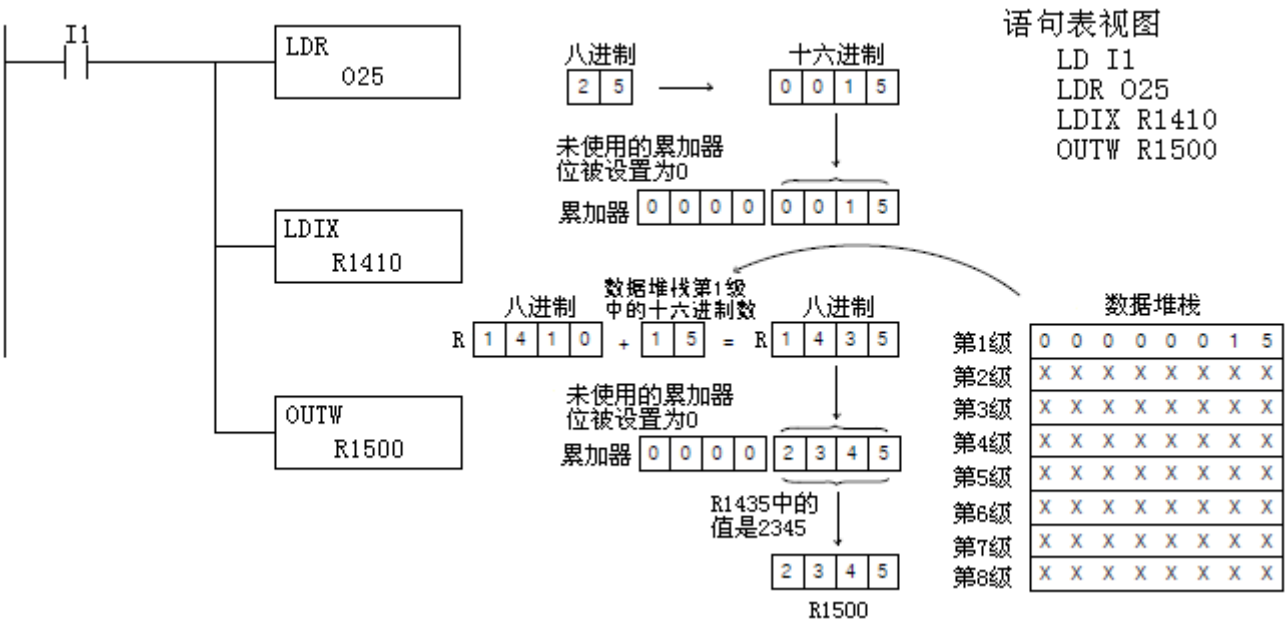
操作数类型	NK1 范围	
A	aaa	
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP76	读入累加器中的数据为 0 时 ON



注意：如果有两个连续的读入指令，第一个读入指令的数据将被压入数据堆栈中。

下面的例子中，当 I1 为 ON 时，八进制数据 25 的十六进制数据 15 被装入数据堆栈的第 1 级中（LDIX 指令执行时）。R 寄存器地址 1410 加上第 1 级中的数据八进制值 25 得到寄存器地址值 1435，将 R1435 中的数据读入累加器的低 16 位中，R1435 中的数据为 2345。OUTW 指令将累加器低 16 位的数据写入到 R1500 中。

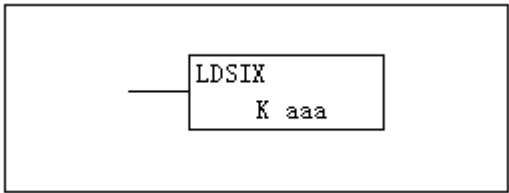


6.9 程序块索引读入指令（LDSIX）

输入 LDSIX 指令是一个 16 位指令，它指定了存放数字或 ASCII 常数的数据标记区的标号。

LDSIX 指令将数据堆栈第 1 级中的数据作为偏移量，这个偏移量指定了数据标记区中哪个数据将被写入累加器中。堆栈第 1 级中的数据需是十六进制数。

小技巧：LDR 指令可将八进制数转换为十六进制数并将其读入累加器。

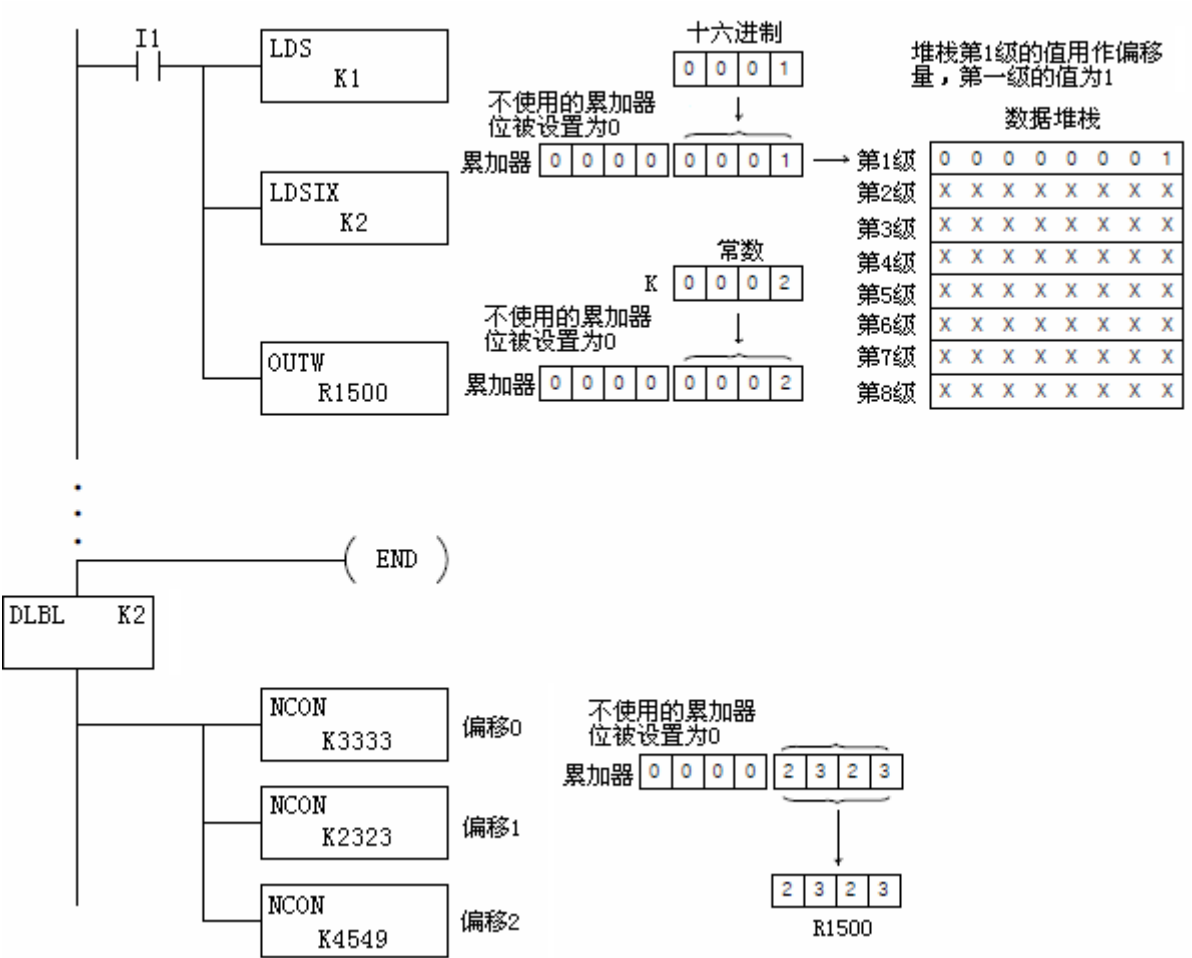


操作数类型	NK1 范围
	aaa
常数 K	1-FFFF



注意：如果有两个连续的读入指令，第一个读入指令的数据将被压入数据堆栈中。

下面的例子中，当 I1 为 0N 时，LDS 指令将偏移量常数 1 读入累加器，当 LDSIX 指令执行时，偏移量被压入数据堆栈的第 1 级中。LDSIX 指令指定了数据标记区的标号（DLBL K2），并将相应偏移值的数据读入累加器中。

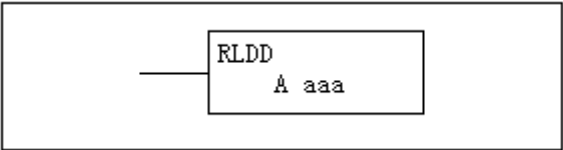


语句视图

```
LD I1
LDS K1
LDSIX K2
OUTW R1500
...
END
DLBL K2
NCON K3333
NCON K2323
NCON K4549
```

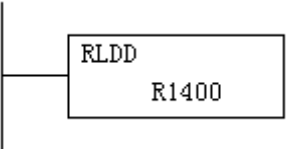
6.10 实数读入指令（RLDD）

输入 RLDD 指令将存放在两个连续寄存器中的实数读入累加器。



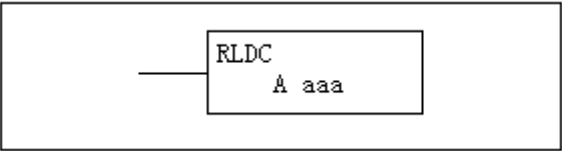
操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

使用 RLDD 指令可将一个实数存入 R1400 和 R1401 中，如右图所示，然后可将其进行实数运算，或将其转化成一个二进制数。



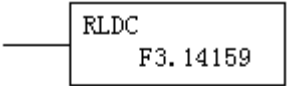
6.11 实数读入指令（RLDC）

输入 RLDC 指令将一个浮点常数读入累加器。

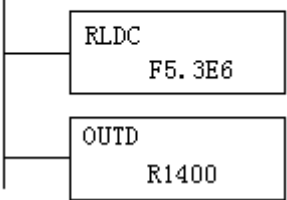


操作数类型	NK1 范围	
	A	aaa
浮点常数	F	-3.402823E+038~+3.402823E+038

KPPsoft 允许直接输入实数，如右图所示，以“F”开头，后面的数就是实数，例如右图中的“F11”。要想输入负数，在“F”后加上“-”号即可。



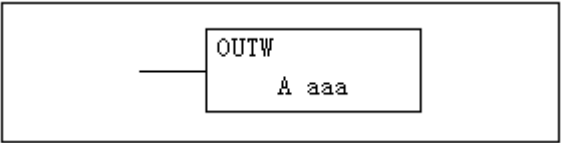
对于特别大和特别小的数，可以使用指数计数法，如右图中的 5.3E6 就是 5300000，OUTD 指令将其存放在 R1400 和 R1401 中。



这些数据都是 IEEE 32 位实数格式，因此不管这些数很大或是很小，都占用两个 R 寄存器。如果用十六进制、二进制或是 BCD 格式查看储存的实数，显示的数据会很难理解。跟其它类型的数据一样，必须清楚实数的存储位置，以便使用正确的指令将其读出。

6.12 16 位写入指令（OUTW）

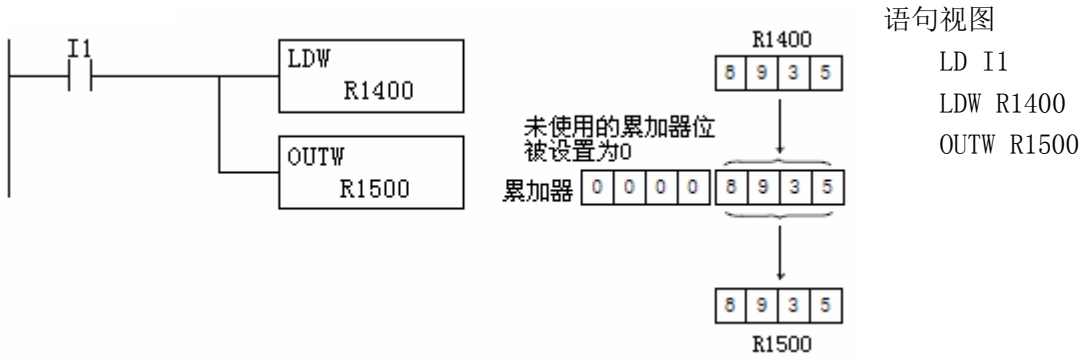
输入 OUTW 指令将累加器的低 16 位写入到 R 寄存器中。



操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

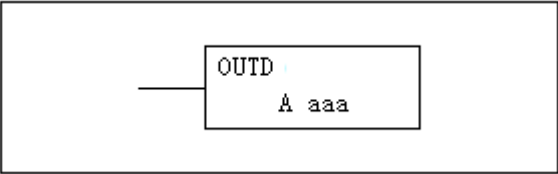
受影响的标志线圈	描述
SP76	读入累加器中的数据为 0 时 ON

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器的低 16 位，OUTW 指令将累加器的低 16 位写入 R1500 中。



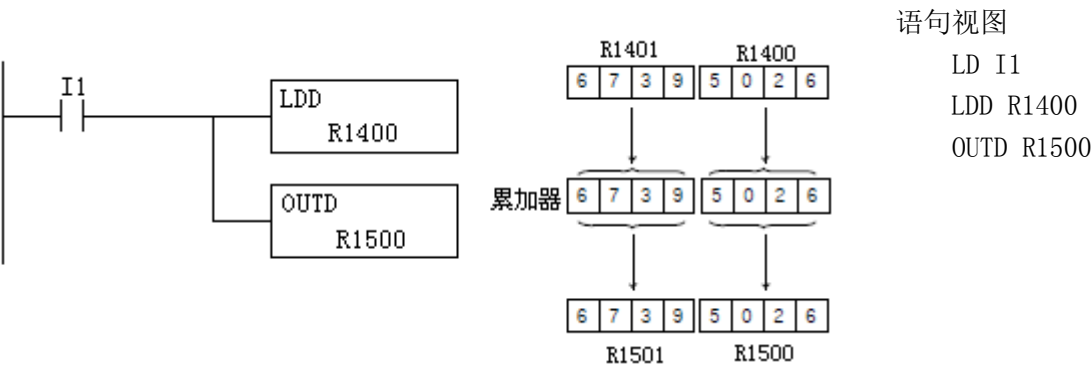
6.13 32 位写入指令（OUTD）

输入 OUTD 指令是一个 32 位指令，用于将累加器的数据写入到两个连续的 R 寄存器中，第一个 R 寄存器的地址为 A aaa。



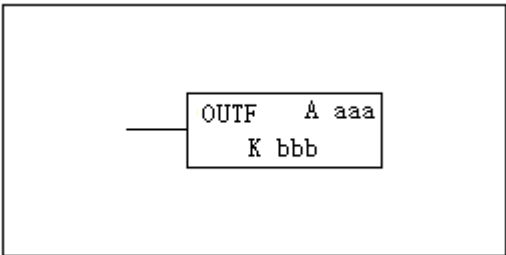
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的 32 位数据读入累加器，OUTD 指令将累加器的 32 位数据写入到 R1500 和 R1501 中。



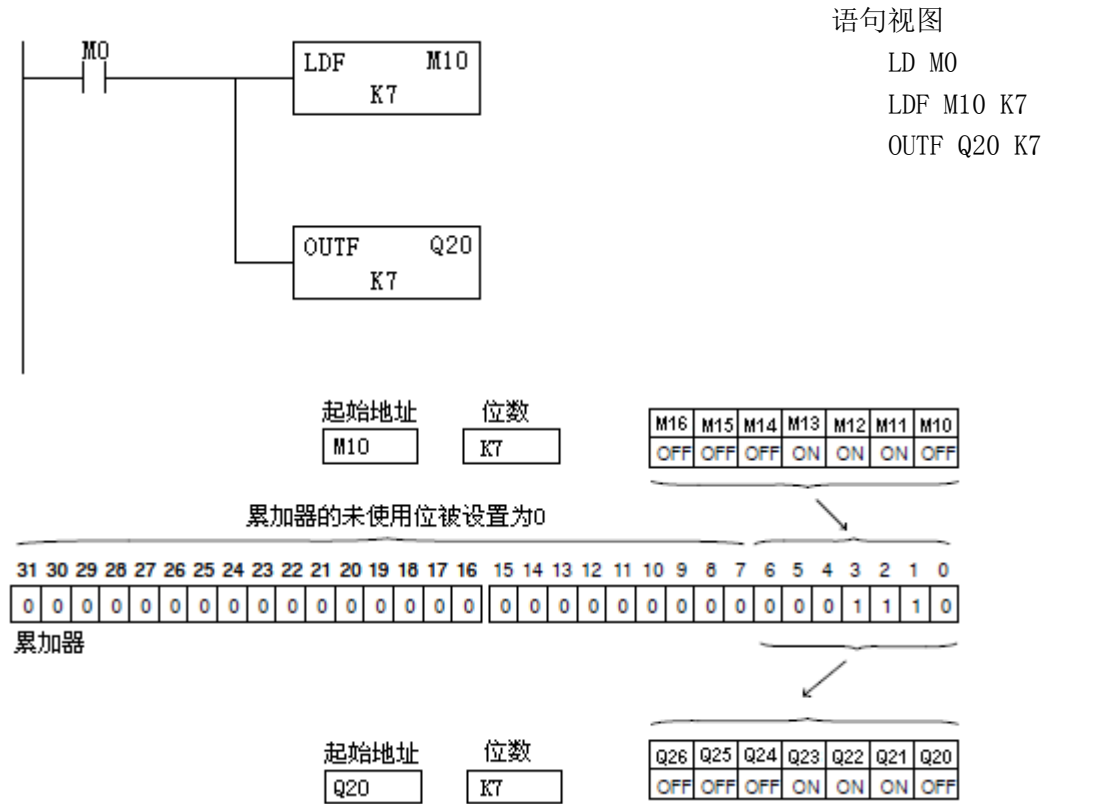
6.14 任意位写入指令（OUTF）

输入 OUTF 指令将累加器的 1-32 位写入到指定的地址中。指令需要目标地址的起始地址和输入的位数。



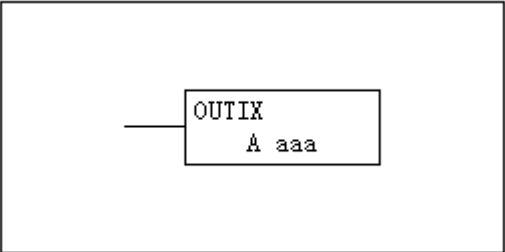
操作数类型		NK1 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777
通讯 I/O	GI/GQ	0-3777
常数	K	1-32

下面的例子中，当 M0 为 ON 时，LDF 指令将 M10-M16（7 位）读入累加器中，OUTF 指令将累加器的低 7 位写入到 Q20-Q26 中。



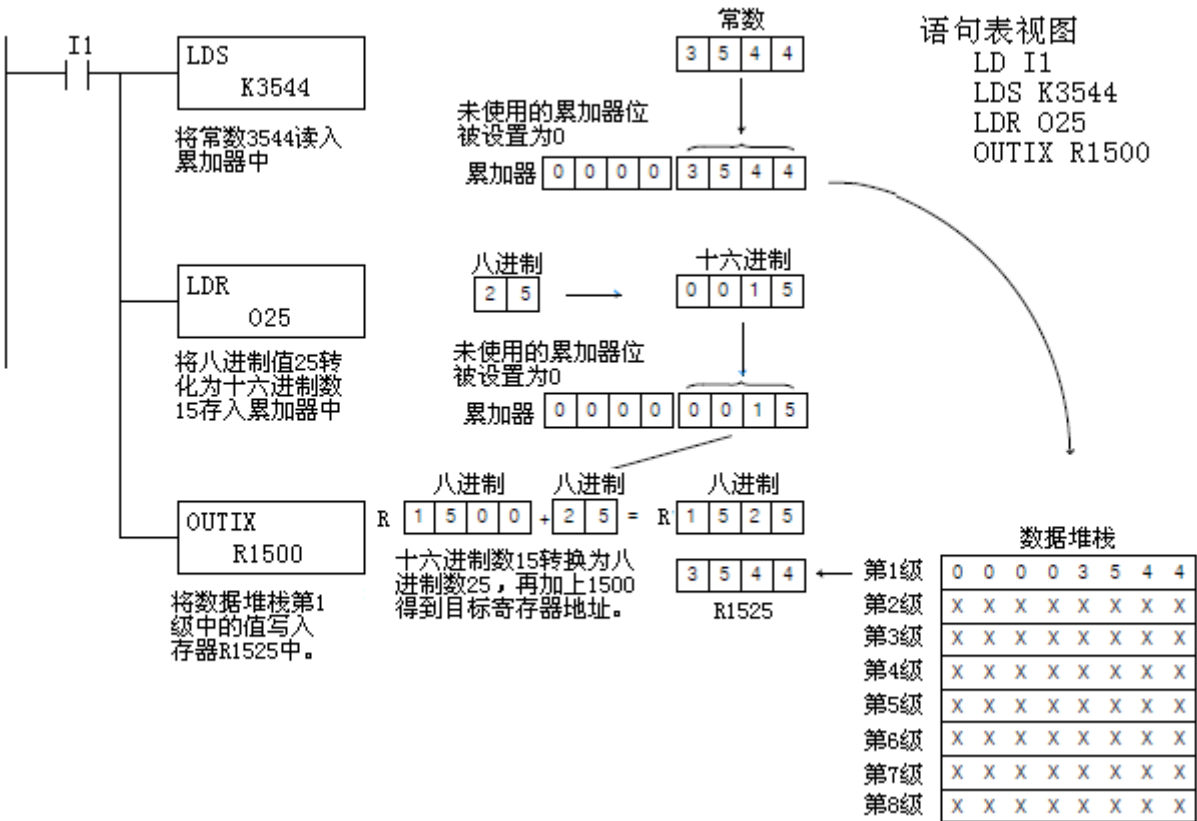
6.15 索引 16 位写入指令（OUTIX）

输入 OUTIX 指令是一个 16 位指令，它将数据堆栈第 1 级中的数据写入到目标地址中，目标地址由指令中指定的地址加上累加器中的偏移量得到。累加器的高 16 位被设置为 0。



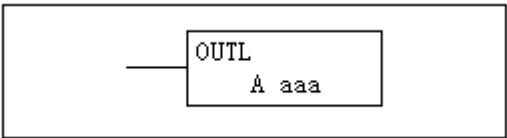
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

下面的例子中，当 I1 为 ON 时，LDS 指令将常数 3544 读入累加器中。LDR 指令将八进制数 25 的十六进制数 15 读入累加器，常数 3544 被压入数据堆栈的第 1 级中。R 寄存器地址 1500 加上累加器中的数 25（八进制）得到目标寄存器地址值 1525，将数据堆栈第 1 级中的数据 3544 读入寄存器 R1525 中。



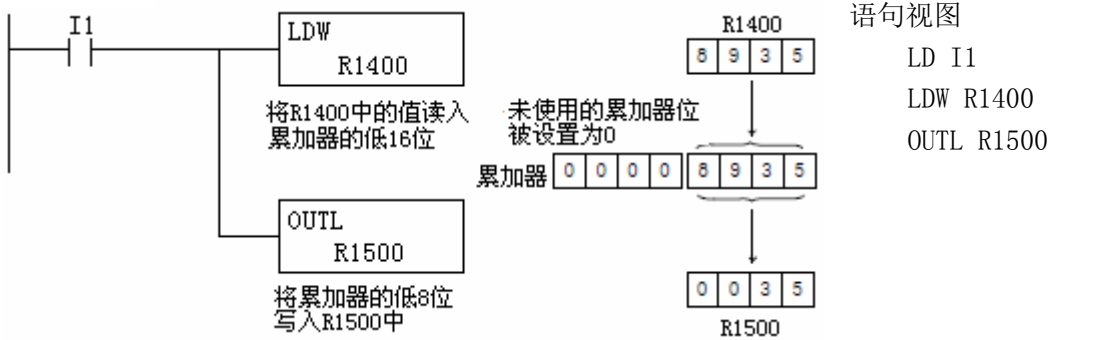
6.16 低 8 位写入指令（OUTL）

输入 OUTL 指令将累加器的低 8 位写入到指定寄存器的低 8 位中。



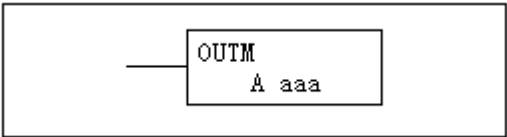
操作数类型	NK1 范围
A	aaa
R 寄存器	R 所有（附录 1）

下面的例子中，当 I1 为 ON 时，R1400 中的数据被读入累加器的低 16 位中。累加器的低 8 位被写入 R1500 中。



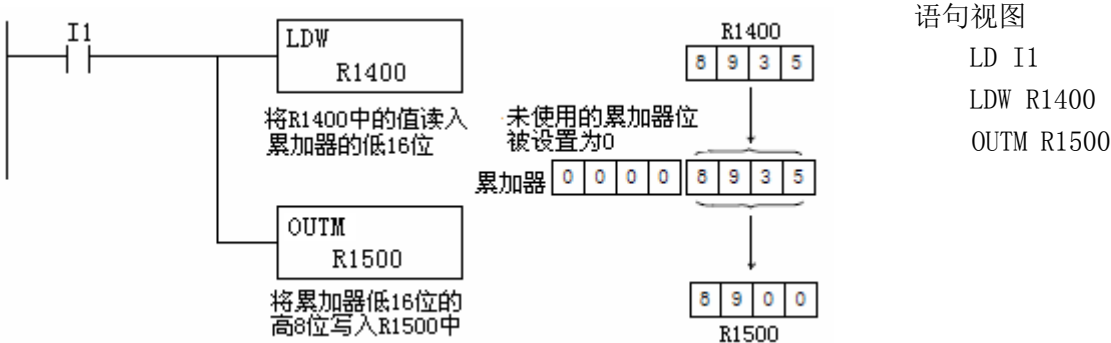
6.17 高 8 位写入指令（OUTM）

输入 OUTM 指令将累加器的低 16 位的高 8 位写入到指定寄存器的高 8 位中。



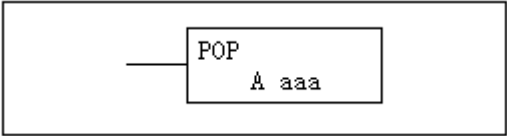
操作数类型	NK1 范围
A	aaa
R 寄存器	R 所有（附录 1）

下面的例子中，当 I1 为 ON 时，R1400 中的数据被读入累加器的低 16 位中。累加器低 16 位的高 8 位被写入 R1500 中。



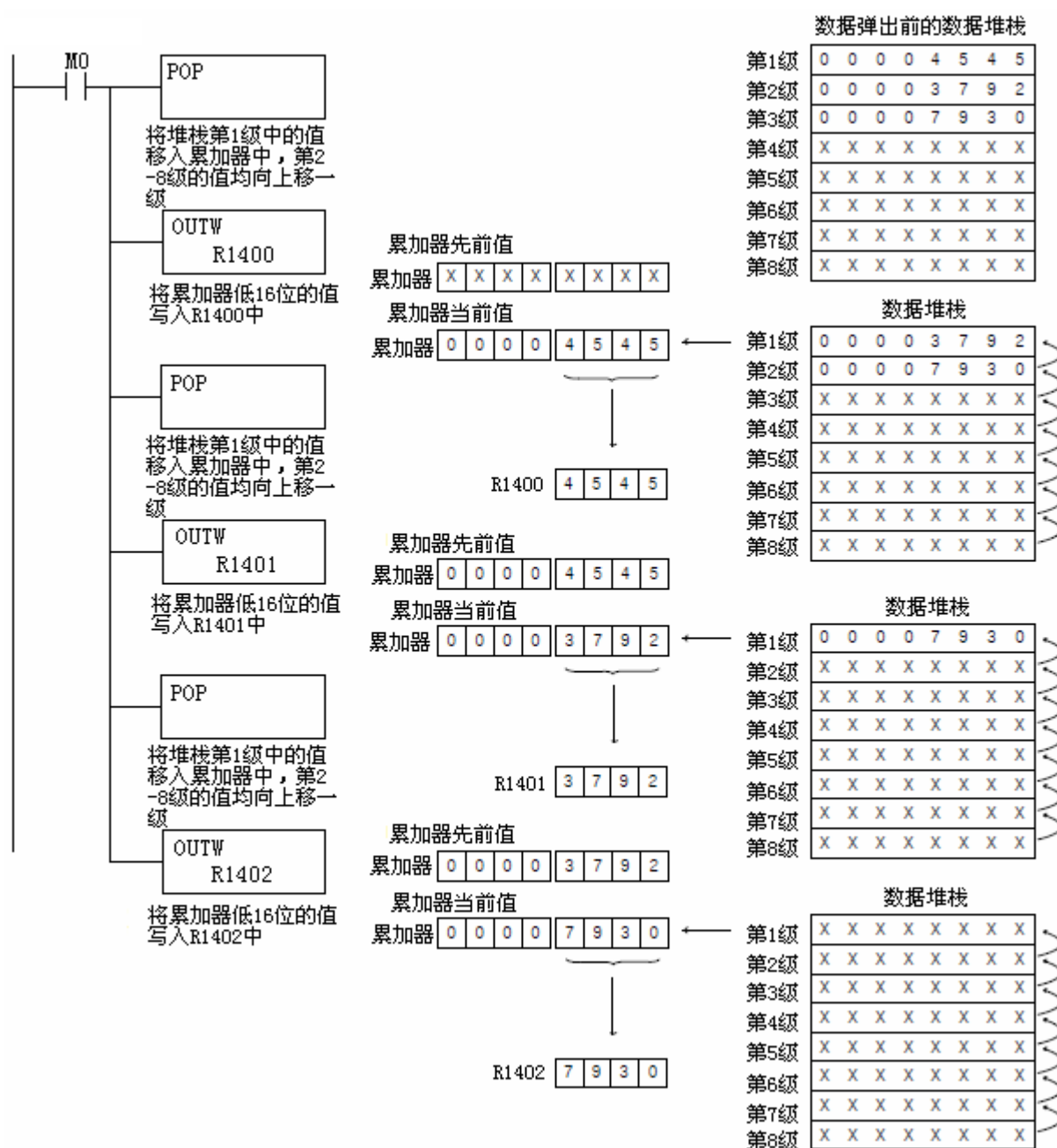
6.18 数据堆栈弹出指令（POP）

输入 POP 指令将数据堆栈第 1 级中的数据移入累加器中,并将堆栈 2-8 级中的数据都向上移一级。



下面的例子中，当 M0 为 ON 时，第一个 POP 指令将堆栈第 1 级中的数据 4545 移入累加器中,OUTW 指令将累加器的数据写入 R1400 中;第二个 POP 指令将堆栈第 1 级中的数据 3792 移入累加器，OUTW 指令将其写入 R1401 中；第三个 POP 指令将堆栈第 1 级中的数据 7930 移入累加器中，OUTW 指令将其写入 R1402 中。如果堆栈中的数据多于 16 位（4 字），请使用 OUTD 指令，每个数据占用两个连续的寄存器。

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON



语句视图

LD M0

POP

OUTW R1400

POP

OUTW R1401

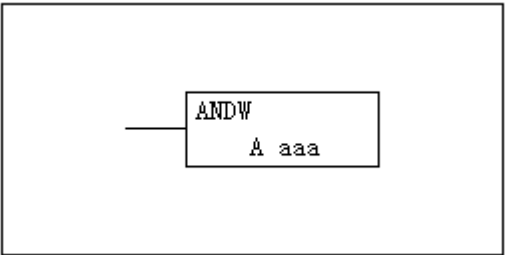
POP

OUTW R1402

第 7 章 累加器逻辑指令

7.1 16 位逻辑与指令（ANDW）

输入 ANDW 指令是一个 16 位指令，将累加器低 16 位的数据同指定寄存器（A aaa）中的数据逻辑与，结果存放在累加器中。如果累加器中的结果是 0，则零标志线圈 ON。



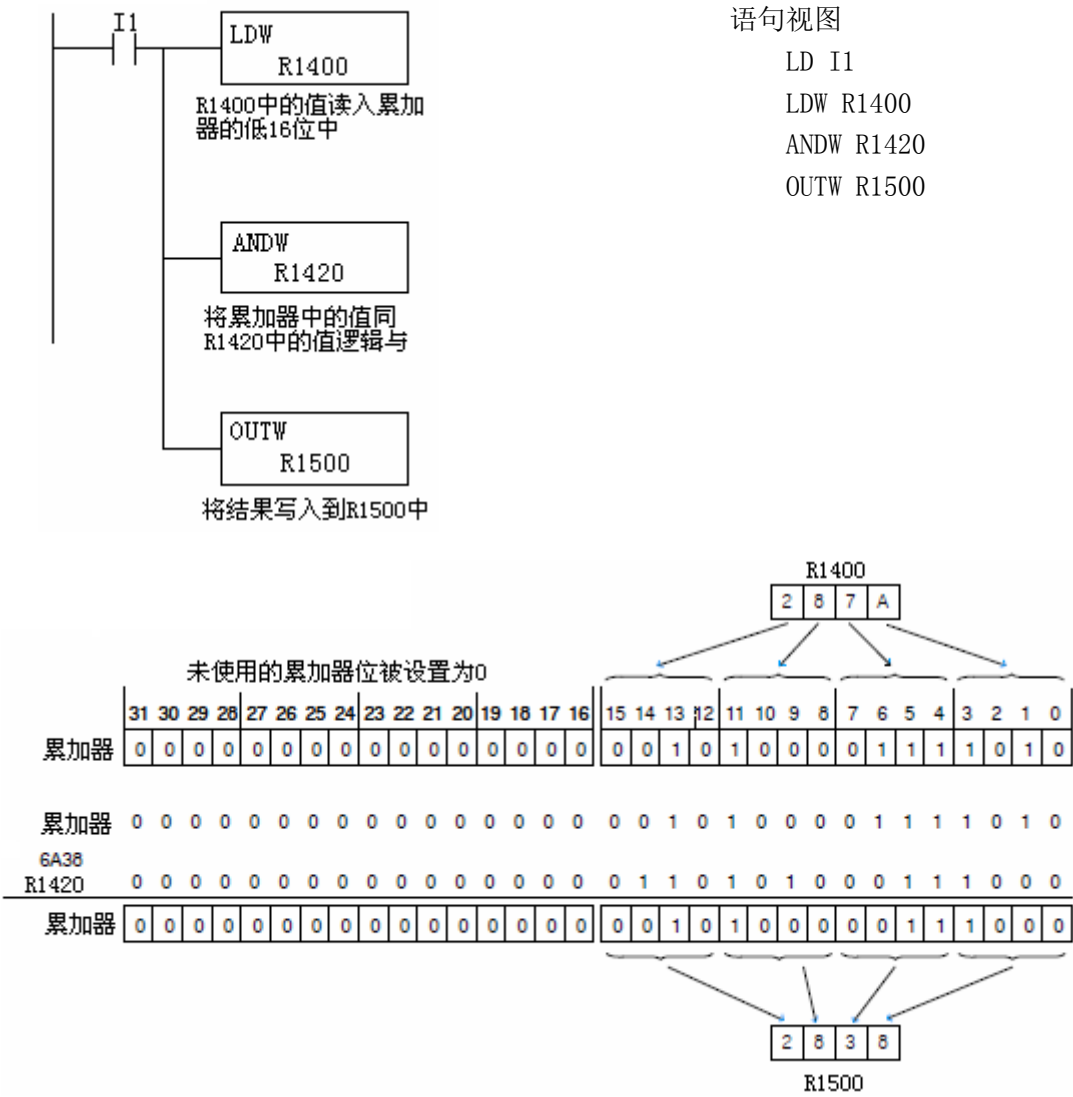
操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON



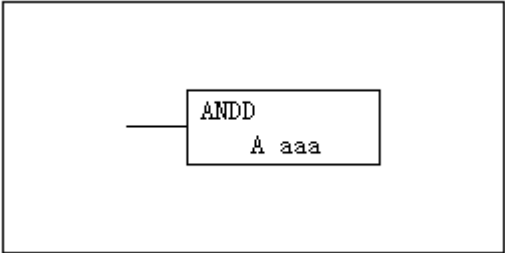
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，ANDW 指令将累加器中的数据同 R1420 中的数据逻辑与，OUTW 指令将结果写入到 R1500 中。



7.2 32 位逻辑与指令（ANDD）

输入 ANDD 指令是一个 32 位指令，将累加器的数据同指定的两个连续寄存器中的数据逻辑与，结果存放在累加器中。如果累加器中的结果是 0 或负数（最高位为 1），则零标志线圈 ON。



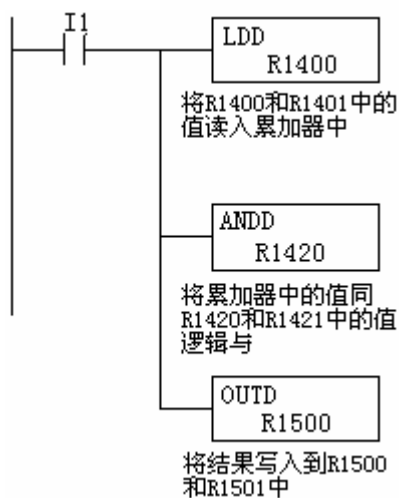
操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON



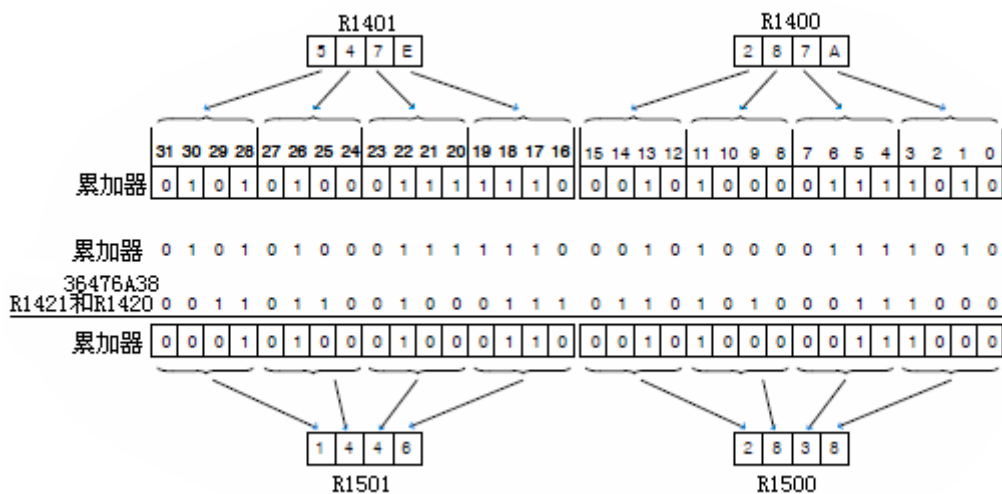
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器，ANDD 指令将累加器中的数据同 R1420 和 R1421 中的数据逻辑与，OUTD 指令将结果写入到 R1500 和 R1501 中。



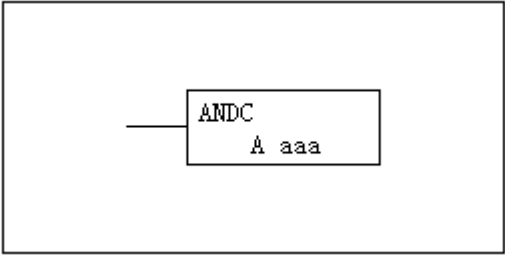
语句视图

```
LD I1
LDD R1400
ANDD R1420
OUTD R1500
```



7.3 32 位常数逻辑与指令（ANDC）

输入 ANDC 指令是一个 32 位常数指令，将累加器的数据同一个 8 位常数（最大）逻辑与，结果存放在累加器中。如果累加器中的结果是 0 或负数（最高位为 1），则零标志线圈 ON。



操作数类型	NK1 范围
A	aaa
常数	K 0-FFFFFFFF

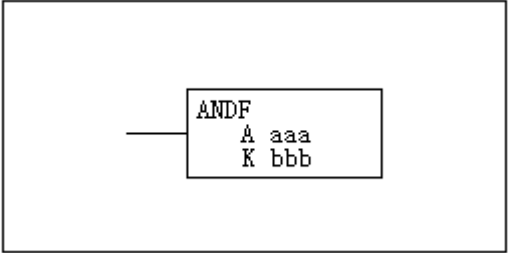
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

7.4 任意位长逻辑与指令（ANDF）

输入 ANDF 指令将累加器中的二进制数据同指定位数的二进制数据进行逻辑与。指令需要指定起始地址（A aaa）和位数（K bbb）。如果累加器中的结果是 0 或负数（最高位为 1），则零标志线圈 ON。



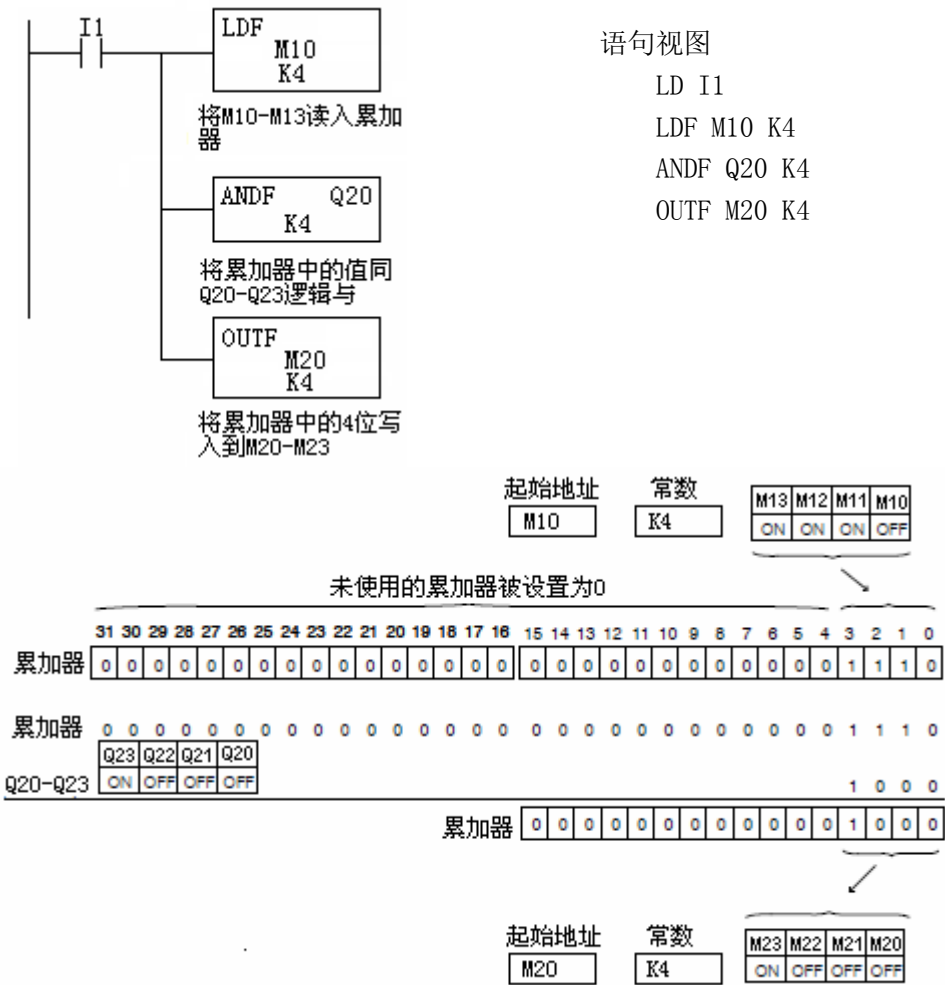
操作数类型		NK1 范围	
	A	aaa	bbb
输入	I	0-1777	-
输出	Q	0-1777	-
内部线圈	M	0-3777	-
级	S	0-1777	-
定时器	T	0-777	-
计数器	C	0-777	-
特殊线圈	SP	0-1777	-
通讯 I/O	GI/GQ	0-3777	-
常数	K	-	1-32

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON



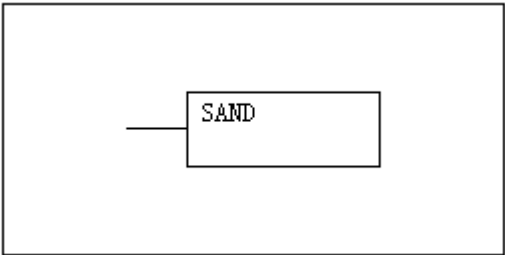
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDF 指令将 M10-M13（4 位）读入累加器，ANDF 指令将累加器中的数据同 Q20-Q23 逻辑与，OUTF 指令将结果写入到 M20-M23。



7.5 堆栈逻辑与指令（SAND）

输入 SAND 指令是一个 32 位指令，将累加器中的数据同数据堆栈第 1 级中的数据逻辑与，结果存放在累加器中。数据堆栈第 1 级中的数据移出，其它级均向上移一级。如果累加器中的结果是 0 或负数（最高位为 1），则零标志线圈 ON。

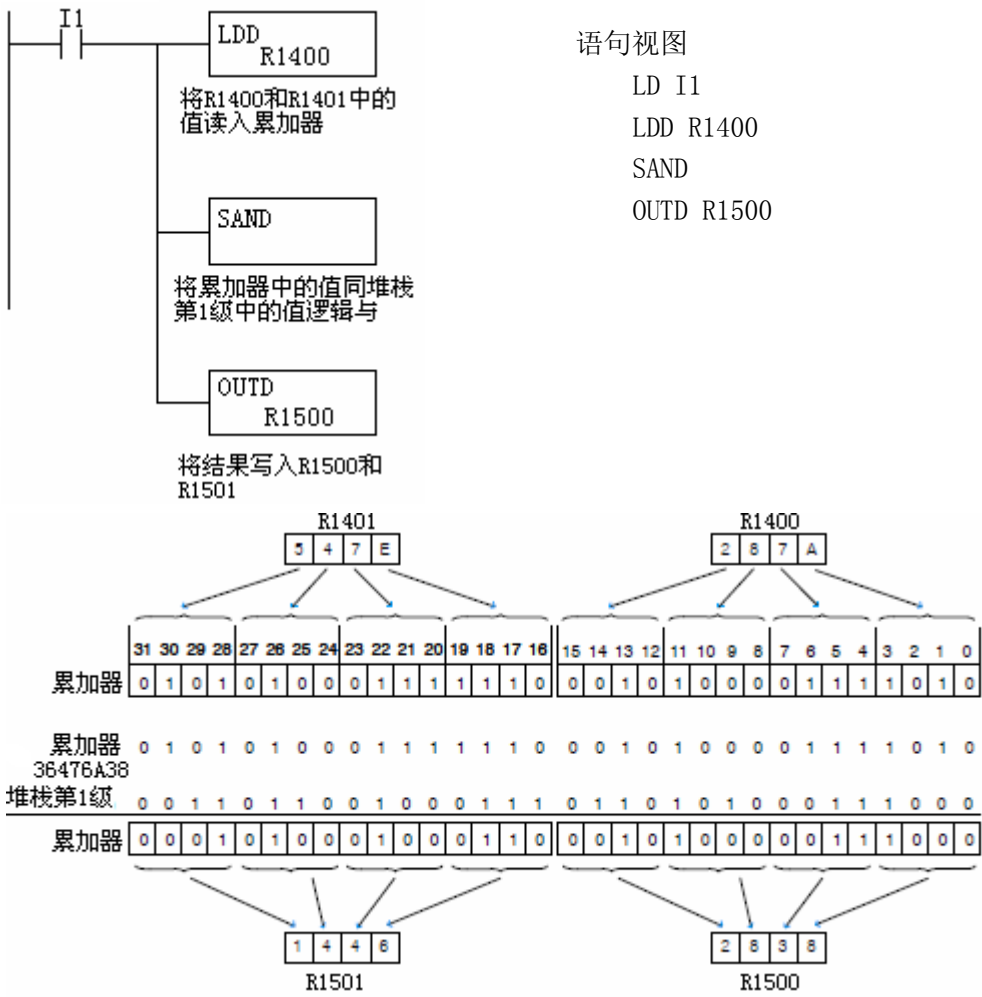


受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON



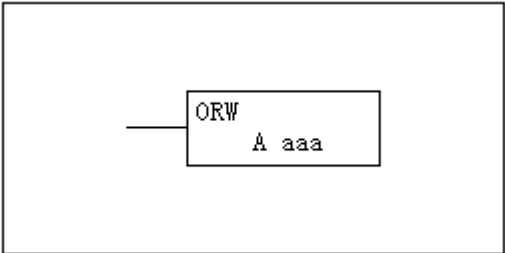
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，累加器中二进制数据同数据堆栈第 1 级中的二进制数据逻辑与，结果存放在累加器中，OUTD 指令将结果写入到 R1500 和 R1501 中。



7.6 16 位逻辑或指令（ORW）

输入 ORW 指令是一个 16 位指令，将累加器低 16 位的数据同指定寄存器（A aaa）中的数据逻辑或，结果存放在累加器中。如果累加器中的结果是 0，则零标志线圈 ON。

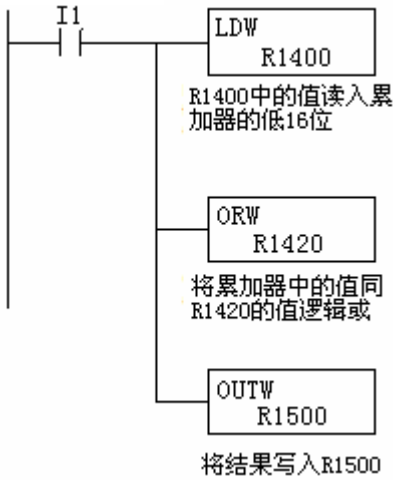


操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON

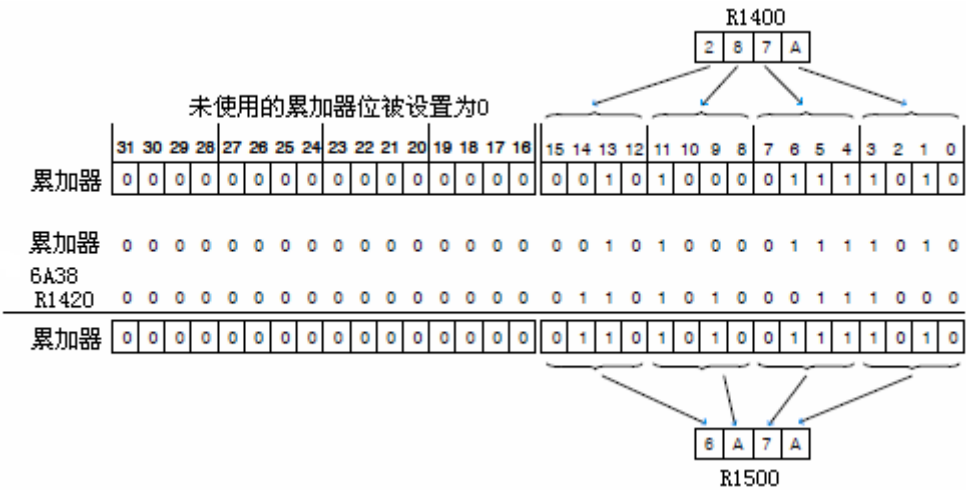
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，ORW 指令将累加器中的数据同 R1420 中的数据逻辑或，OUTW 指令将结果写入到 R1500 中。



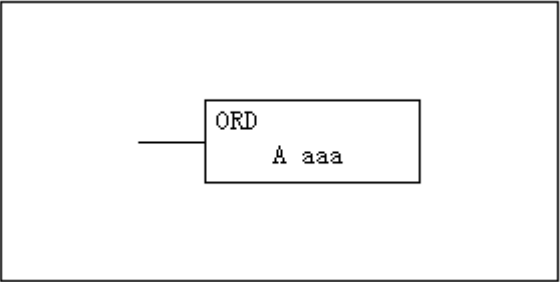
语句视图

```
LD I1
LDW R1400
ORW R1420
OUTW R1500
```



7.7 32 位逻辑或指令（ORD）

输入 ORD 指令是一个 32 位指令，将累加器的数据同指定的两个连续寄存器中的数据逻辑或，结果存放在累加器中。如果累加器中的结果是 0 或负数(最高位为 1)，则零标志线圈 ON。



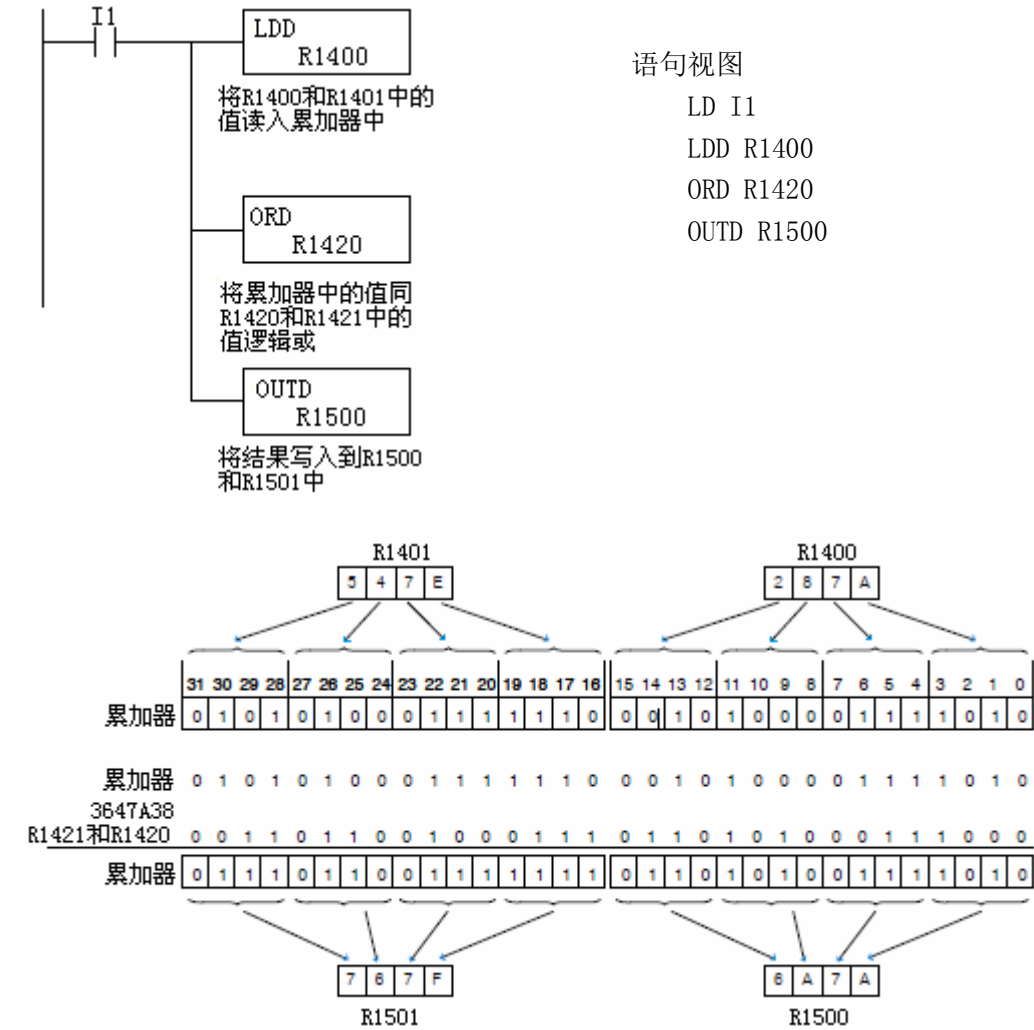
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON



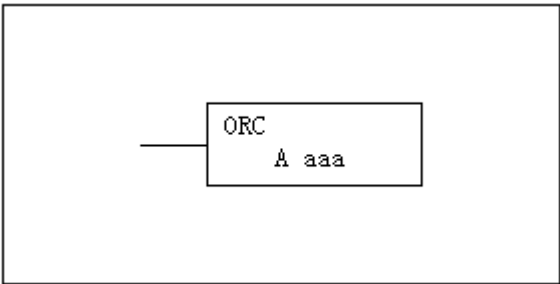
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器，ORD 指令将累加器中的数据同 R1420 和 R1421 中的数据逻辑或，OUTD 指令将结果写入到 R1500 和 R1501 中。



7.8 32 位常数逻辑或指令（ORC）

输入 ORC 指令是一个 32 位常数指令，将累加器的数据同一个 8 位常数（最大）逻辑或，结果存放在累加器中。如果累加器中的结果是 0 或负数（最高位为 1），则零标志线圈 ON。



操作数类型	NK1 范围
A	aaa
常数	K 0-FFFFFFFF

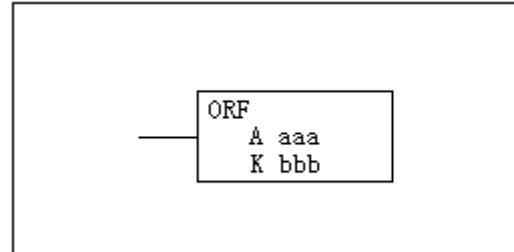
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

7.9 任意位长逻辑与指令（ORF）

输入 ORF 指令将累加器中的二进制数据同指定位数的二进制数据进行逻辑或。指令需要指定起始地址（A aaa）和位数（K bbb）。如果累加器中的结果是 0 或负数（最高位为 1），则零标志线圈 ON。



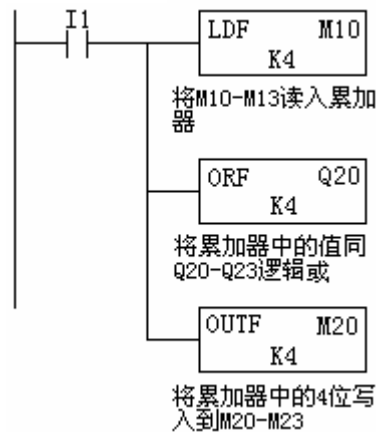
操作数类型		NK1 范围	
	A	aaa	bbb
输入	I	0-1777	-
输出	Q	0-1777	-
内部线圈	M	0-3777	-
级	S	0-1777	-
定时器	T	0-777	-
计数器	C	0-777	-
特殊线圈	SP	0-1777	-
通讯 I/O	GI/GQ	0-3777	-
常数	K	-	1-32

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDF 指令将 M10-M13（4 位）读入累加器，ORF 指令将累加器中的数据同 Q20-Q23 逻辑或，OUTF 指令将结果写入 M20-M23。



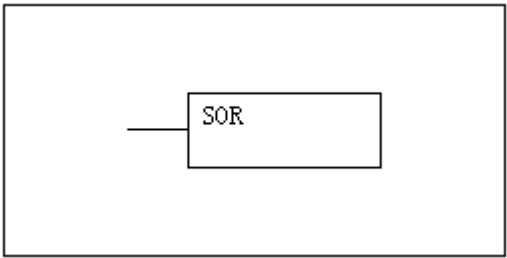
语句视图

```
LD I1
LDF M10 K4
ORF Q20 K4
OUTF M20 K4
```



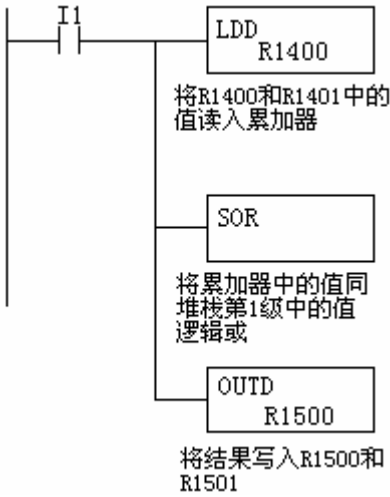
7.10 堆栈逻辑或指令（SOR）

输入 SOR 指令是一个 32 位指令，将累加器中的数据同数据堆栈第 1 级中的数据逻辑或，结果存放在累加器中。数据堆栈第 1 级中的数据移出，其它级均向上移一级。如果累加器中的结果是 0 或负数（最高位为 1），则零标志线圈 ON。



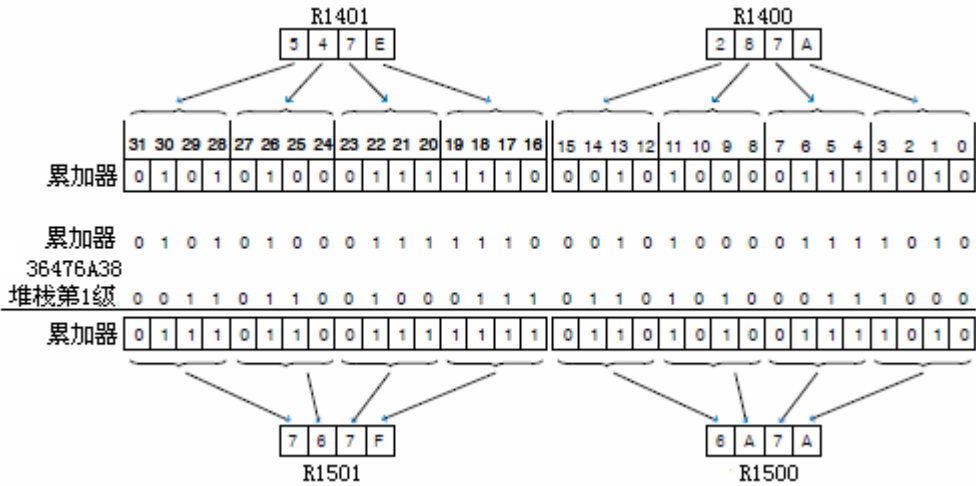
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

下面的例子中，当 I1 为 ON 时，累加器中的数据同数据堆栈第 1 级中的数据逻辑或，结果存放在累加器中，OUTD 指令将其写入到 R1500 和 R1501 中。



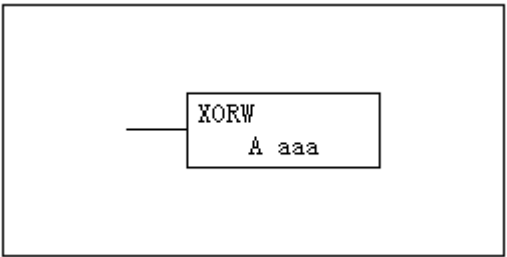
语句视图

```
LD I1
LDD R1400
SOR
OUTD R1500
```



7.11 16 位逻辑异或指令（XORW）

输入 XORW 指令是一个 16 位指令，将累加器低 16 位的数据同指定寄存器（A aaa）中的数据逻辑异或，结果存放在累加器中。如果累加器中的结果是 0，则零标志线圈 ON。



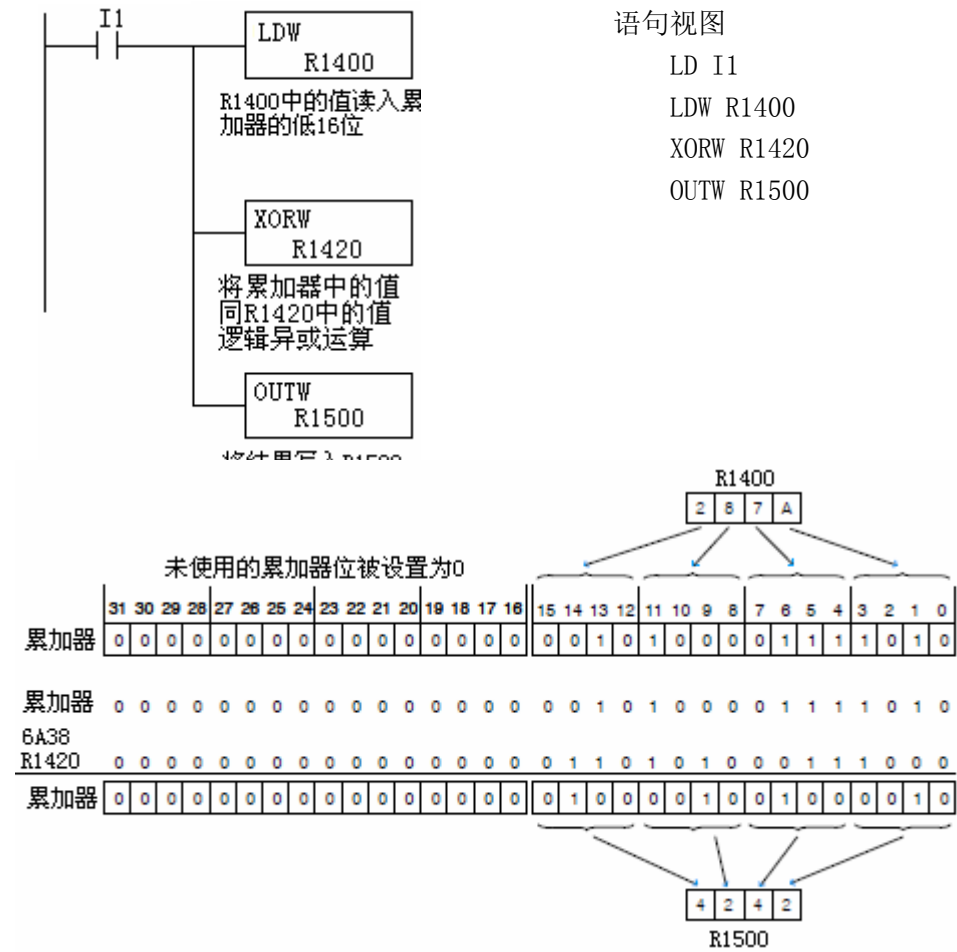
操作数类型	NK1 范围	
A	aaa	
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON



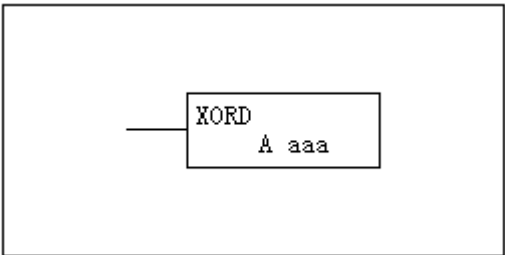
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，XORW 指令将累加器中的数据同 R1420 中的数据进行逻辑异或运算，OUTW 指令将结果写入到 R1500 中。



7.12 32 位逻辑异或指令（XORD）

输入 XORD 指令是一个 32 位指令，将累加器的数据同指定的两个连续寄存器中的数据进行逻辑异或运算，结果存放在累加器中。如果累加器中的结果是 0 或负数（最高位为 1），则零标志线圈 ON。



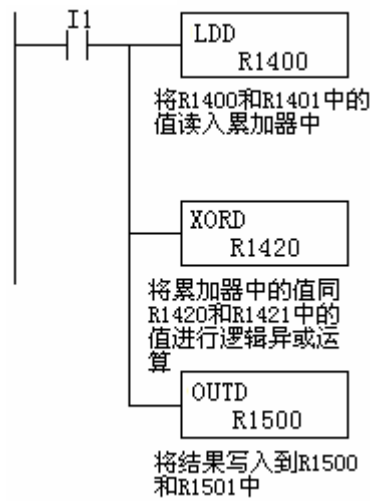
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON



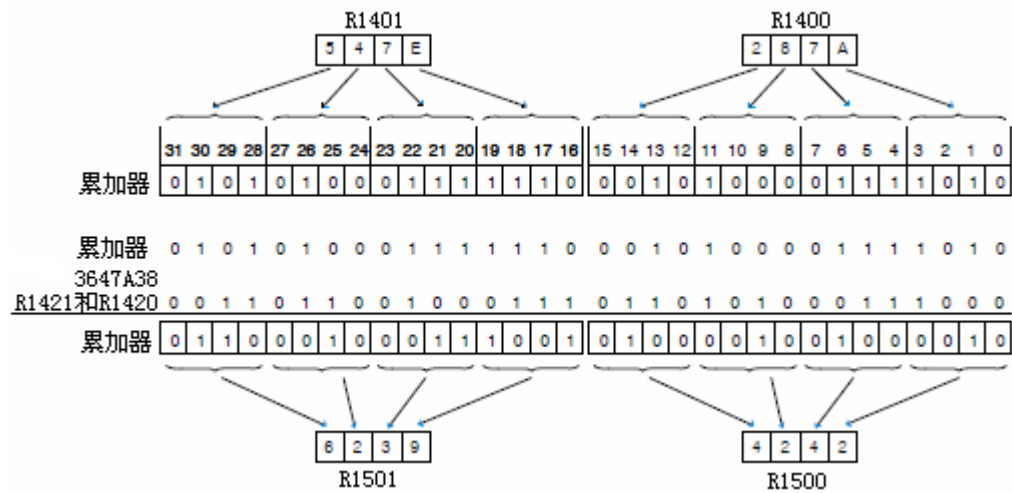
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器，XORD 指令将累加器中的数据同 R1420 和 R1421 中的数据进行逻辑异或运算，OUTD 指令将结果写入到 R1500 和 R1501 中。



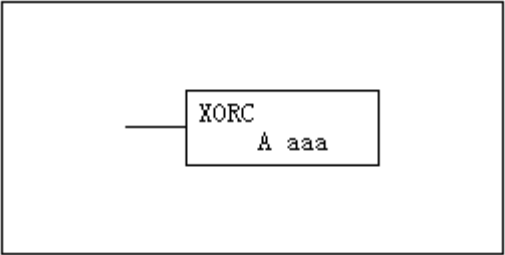
语句视图

```
LD I1
LDD R1400
XORD R1420
OUTD R1500
```



7.13 32 位常数逻辑异或指令（XORC）

 XORC 指令是一个 32 位常数指令，将累加器的数据同一个 8 位常数（最大）进行逻辑异或运算，结果存放在累加器中。如果累加器中的结果是 0 或负数（最高位为 1），则零标志线圈 ON。



操作数类型	NK1 范围	
	A	aaa
常数	K	0-FFFFFFFF

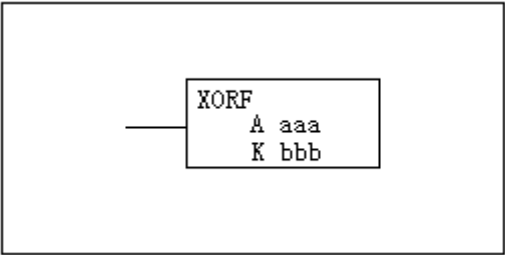
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

7.14 任意位长逻辑异或指令（XORF）

输入 XORF 指令将累加器中的二进制数同指定位数（1-32 位）的二进制数进行逻辑异或运算。指令需要指定起始地址（A aaa）和位数（K bbb）。如果累加器中的结果是 0 或负数（最高位为 1），则零标志线圈 ON。



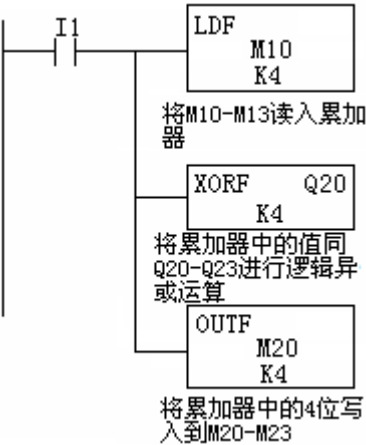
操作数类型		NK1 范围	
	A	aaa	bbb
输入	I	0-1777	-
输出	Q	0-1777	-
内部线圈	M	0-3777	-
级	S	0-1777	-
定时器	T	0-777	-
计数器	C	0-777	-
特殊线圈	SP	0-1777	-
通讯 I/O	GI/GQ	0-3777	-
常数	K	-	1-32

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON



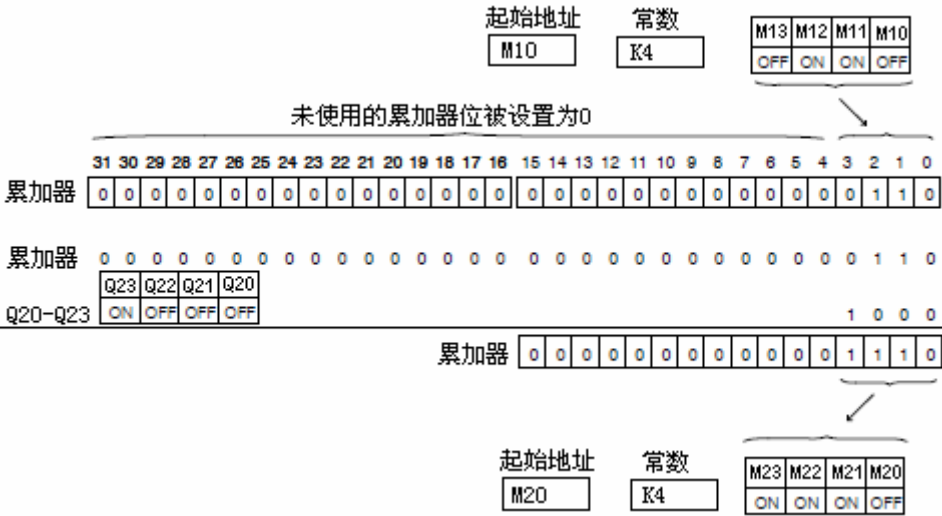
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDF 指令将 M10-M13（4 位）读入累加器，XORF 指令将累加器中的数据同 Q20-Q23 进行逻辑异或运算，OUTF 指令将结果写入 M20-M23。



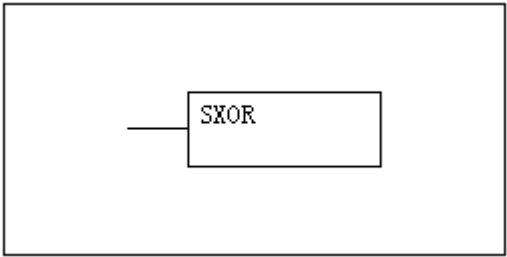
语句视图

```
LD I1
LDF M10 K4
XORF Q20 K4
OUTF M20 K4
```



7.15 堆栈逻辑异或指令（SXOR）

输入 SXOR 指令是一个 32 位指令，将累加器中的数据同数据堆栈第 1 级中的数据进行逻辑异或运算，结果存放在累加器中。数据堆栈第 1 级中的数据移出，其它级均向上移一级。如果累加器中的结果是 0 或负数（最高位为 1），则零标志线圈 ON。

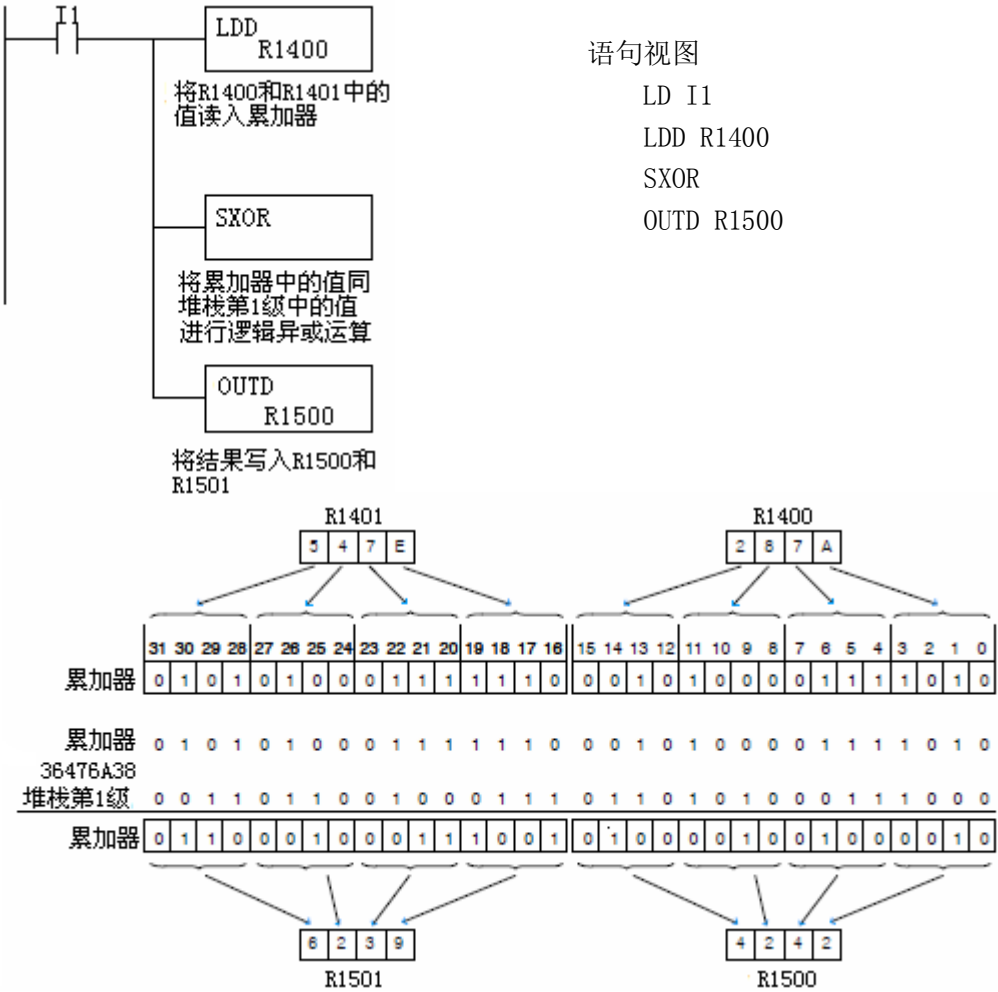


受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON



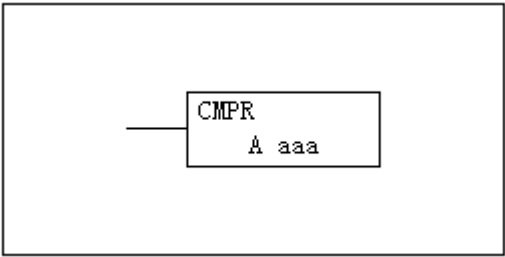
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，累加器中的二进制数同数据堆栈第 1 级中的二进制数进行逻辑异或运算，结果存放在累加器中，OUTD 指令将其写入到 R1500 和 R1501 中。



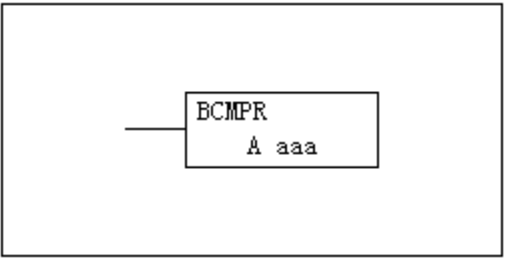
7.16 16 位比较指令（CMPR）

输入 CMPR 指令是一个 16 位指令，将累加器低 16 位的数据同指定寄存器（A aaa）中的数据进行比较运算，相应的特殊线圈变为 ON 来显示比较的结果。进行比较的数据格式可以是二进制或是 BCD，两个数据的格式要相同。



7.17 16 位二进制比较指令（BCMPR）

输入 BCMPR 指令是一个 16 位二进制比较指令。根据 R7633.15 位的数值决定是进行无符号还是有符号比较。相应的特殊线圈（与 CMPR 指令的特殊线圈相同）变为 ON 来显示比较的结果。
R7633.15=1，进行有符号比较；
R7633.15=0，进行无符号比较。



注：仅 V1.0.1.10 及以上的 KPPSoft 版本支持 BCMPR 指令。

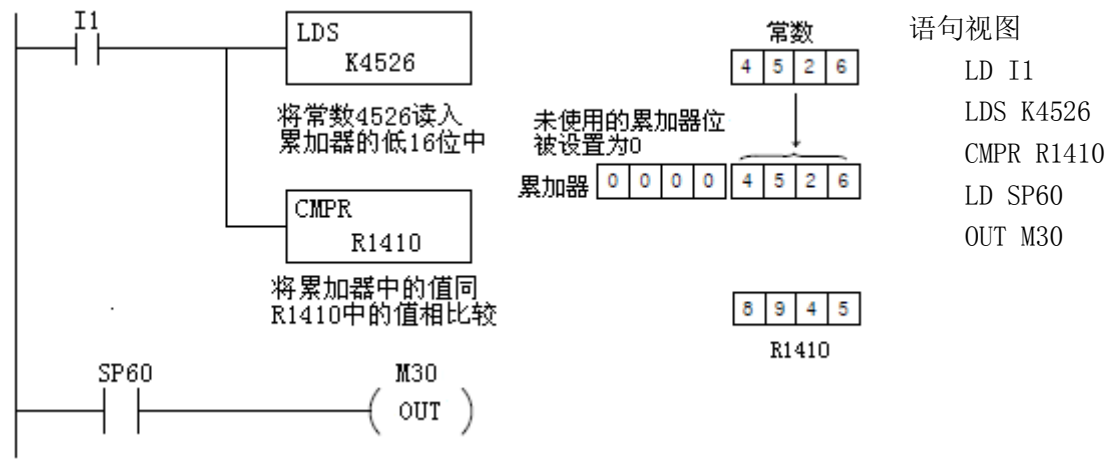
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON



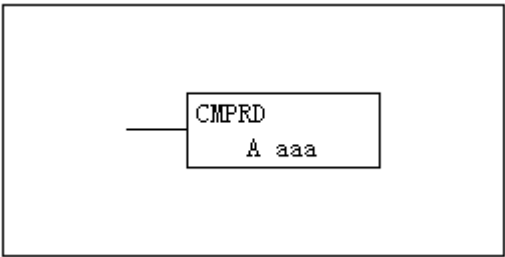
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDS 指令将常数 4526 读入累加器，CMPR 指令将累加器中的数据同 R1410 中的数据进行比较，相应的标志线圈将变为 ON 来显示比较结果。本例中，当累加器中的数据小于 R1410 中的数据时，SP60 就被置为 ON。



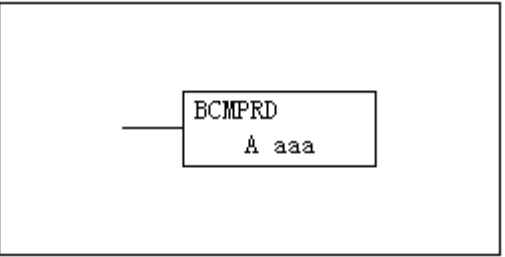
7.18 32 位比较指令（CMPRD）

输入 CMPRD 指令是一个 32 位指令，将累加器的数据同指定的两个连续寄存器中的数据进行比较运算，相应的特殊线圈变为 ON 来显示比较的结果。进行比较的数据格式可以是二进制或是 BCD，两个数据的格式要相同。



7.19 32 位二进制比较指令（BCMPRD）

输入 BCMPRD 指令是一个 32 位二进制比较指令。根据 R7633.15 位的数值决定是进行无符号还是有符号比较。相应的特殊线圈（与 CMPRD 指令的特殊线圈相同）变为 ON 来显示比较的结果。



R7633.15=1，进行有符号比较；

R7633.15=0，进行无符号比较。

注：仅 V1.0.1.10 及以上的 KPPSoft 版本支持 BCMPRD 指令。

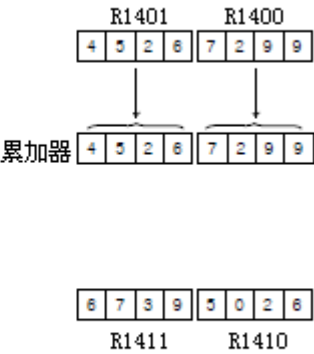
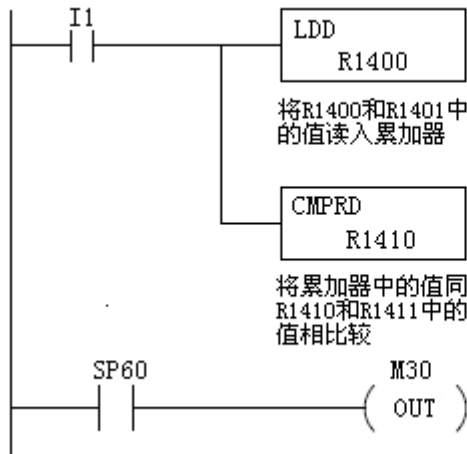
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器，CMPRD 指令将累加器中的数据同 R1410 和 R1411 中的数据进行比较运算，相应的标志线圈将变为 ON 来显示比较结果。本例中，当累加器中的数据小于 R1410 和 R1411 中的数据时，SP60 就被置为 ON。

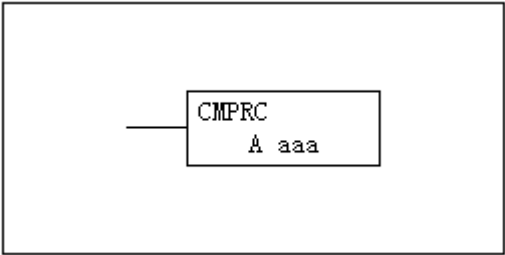


语句视图

```
LD I1
LDD R1400
CMPRD R1410
LD SP60
OUT M30
```

7.20 8 位常数比较指令（CMPRC）

输入 CMPRC 指令是一个 8 位常数指令，将累加器的数据同一个 8 位常数（最大）进行比较运算，相应的特殊线圈被置为 ON 来显示比较的结果。

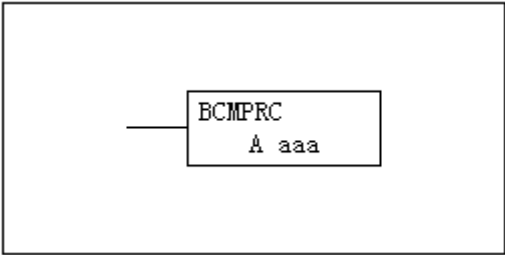


7.21 8 位常数二进制比较指令（BCMPRC）

输入 BCMPRC 指令是一个 8 位常数二进制比较指令。根据 R7633.15 位的数值决定是进行无符号还是有符号比较。相应的特殊线圈（与 CMPRC 指令的特殊线圈相同）变为 ON 来显示比较的结果。

R7633.15=1，进行有符号比较；

R7633.15=0，进行无符号比较。



注：仅 V1.0.1.10 及以上的 KPPSoft 版本支持 BCMPRC 指令。

操作数类型		NK1 范围
	A	aaa
常数	K	0-FFFFFFFF

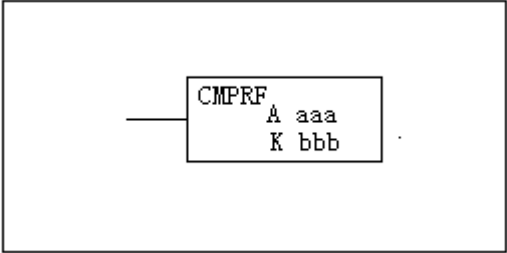
受影响的标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

7.22 任意位长比较指令（CMPRF）

输入 CMPRF 指令将累加器中的数据同指定位数的数据进行比较运算。指令需要指定起始地址（A aaa）和位数（K bbb）。相应的特殊线圈被置为 ON 来显示比较的结果。



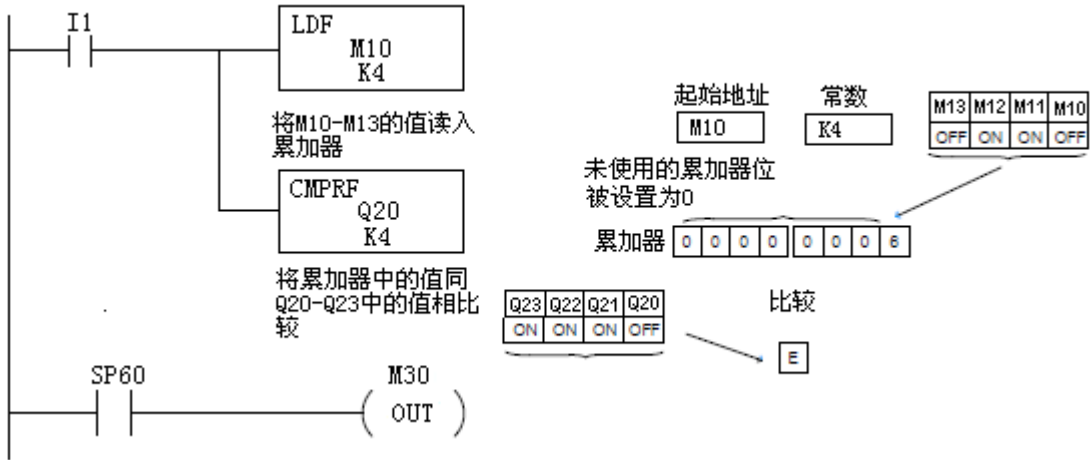
操作数类型		NK1 范围	
	A	aaa	bbb
输入	I	0-1777	-
输出	Q	0-1777	-
内部线圈	M	0-3777	-
级	S	0-1777	-
定时器	T	0-777	-
计数器	C	0-777	-
特殊线圈	SP	0-1777	-
通讯 I/O	GI/GQ	0-3777	-
常数	K	-	1-32

受影响的标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDF 指令将 M10-M13（4 位）读入累加器，CMPRF 指令将累加器中的数据同 Q20-Q23（十六进制数 E）进行比较运算，相应的标志线圈将被置为 ON 来显示比较的结果。本例中，当累加器中的数据小于 Q20-Q23 中的数据时，SP60 就变为 ON。

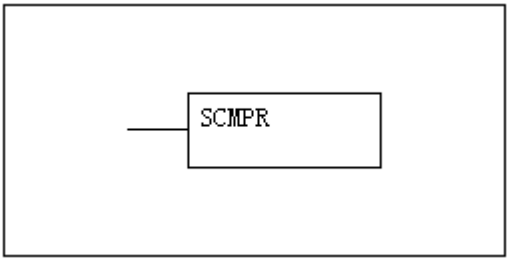


语句视图

```
LD I1
LDF M10 K4
CMPRF Q20 K4
LD SP60
OUT M30
```

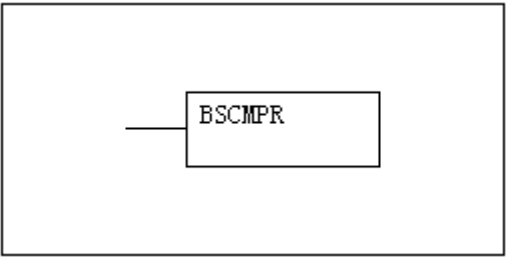
7.23 堆栈比较指令（SCMPR）

输入 SCMPR 指令是一个 32 位指令，将累加器中的数据同数据堆栈第 1 级中的数据进行比较运算，相应的特殊线圈被置为 ON 来显示比较的结果。



7.24 堆栈二进制比较指令（BSCMPR）

输入 BSCMPR 指令是一个 32 位二进制比较指令。将累加器中的数据同数据堆栈第 1 级中的数据进行比较运算。根据 R7633.15 位的数值决定是进行无符号还是有符号比较。相应的特殊线圈（与 SCMPR 指令的特殊线圈相同）变为 ON 来显示比较的结果。



R7633.15=1，进行有符号比较；

R7633.15=0，进行无符号比较。

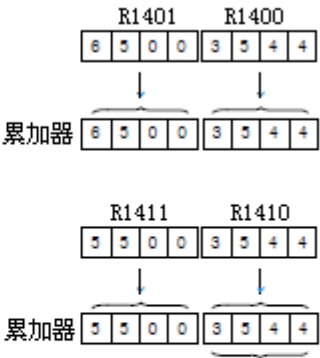
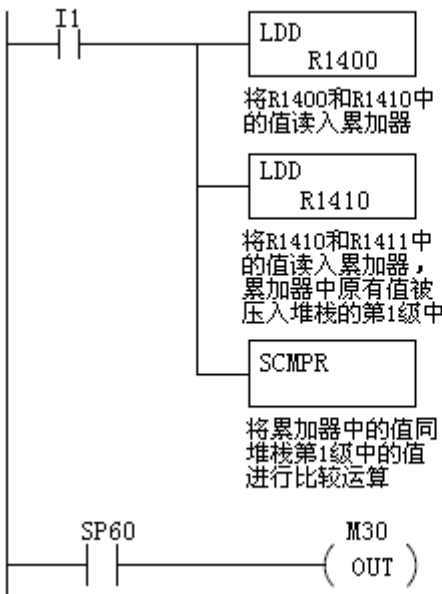
注：仅 V1.0.1.10 及以上的 KPPSoft 版本支持 BSCMPR 指令。

受影响的标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据装入累加器，下一个 LDD 指令将 R1410 和 R1411 中的数据装入累加器，累加器中原有的数据被压入数据堆栈的第 1 级中。SCMPR 指令将累加器中数据同数据堆栈第 1 级中的数据进行比较运算，当累加器中的数据小于堆栈第 1 级中的数据时，SP60 被置为 ON，M30 被接通。

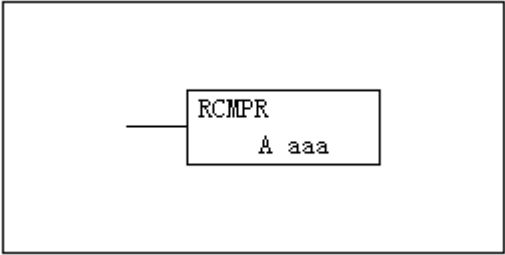


语句视图

LD I1
LDD R1400
LDD R1410
SCMPR
LD SP60
OUT M30

7.25 实数比较指令（RCMPR）

输入 RCMPR 指令是一个 32 位指令，将累加器的数据同指定的两个连续寄存器中的实数进行比较运算，相应的特殊线圈被置为 ON 来显示比较的结果。进行比较的数据长度都是 32 位。



操作数类型	NK1 范围	
A	aaa	
R 寄存器	R	所有(附录 1)
指针	P	所有(附录 1)

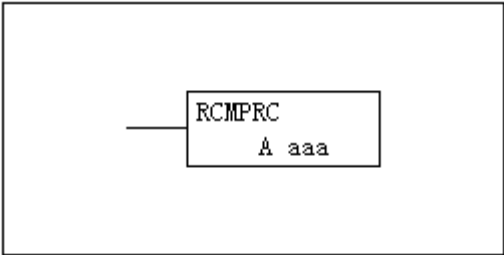
受影响的标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON
SP71	指定了不存在间接寄存器的区域时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

7.26 实数常数比较指令（RCMPRC）

 RCMPRC 指令是一个实数常数指令，将累加器的数据同一个实数常数进行比较运算，相应的特殊线圈被置为 ON 来显示比较的结果。



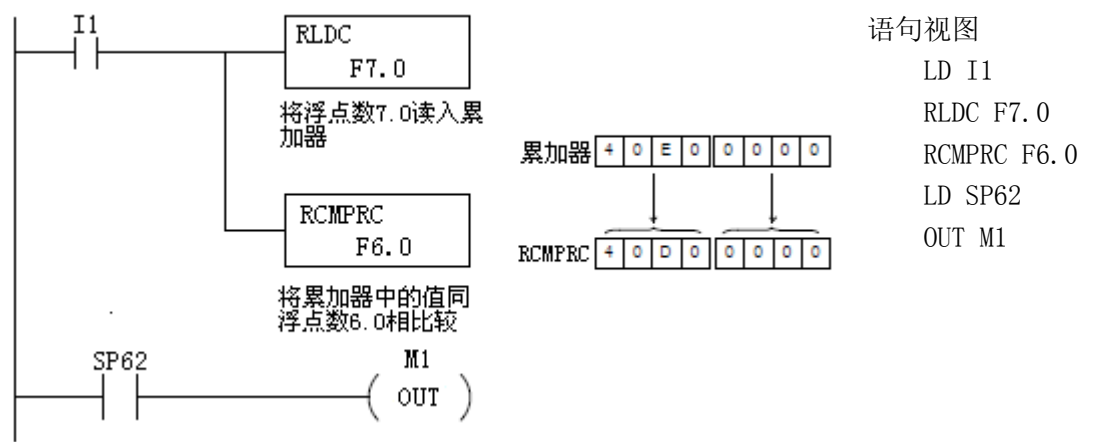
操作数类型	NK1 范围	
A	aaa	
常数	F	-3. 402823E+038~+3. 402823E+038

受影响的标志线	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON
SP71	指定了不存在间接寄存器的区域时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，RLDC 指令将实数 7.0 读入累加器，RCMPRC 指令将累加器中的数据同实数 6.0 进行比较运算，因为 $7 > 6$ ，因此特殊线圈 SP62 被置为 ON。

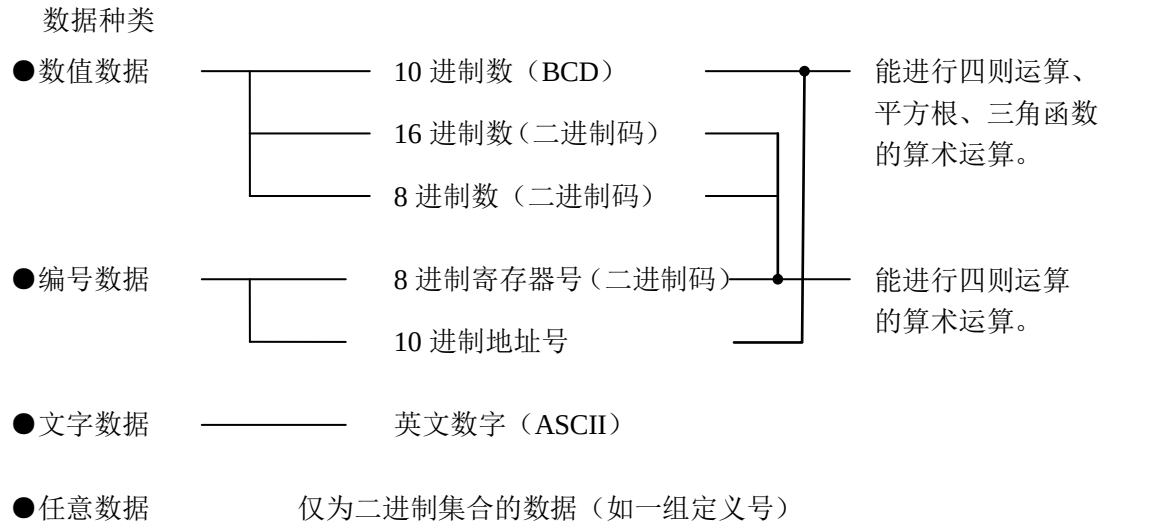


第 8 章 数据处理指令的解说

8.1 数据的形态

(1) PLC 使用的数据

PLC 可以处理各种不同用途的数据。作为数据处理的对象，是多位数集合起来构成的数据，应该是什么形式都可以的。但当进行算术运算时，因其用途上的特点，必须是数值数据。



(2) 10 进制数的形态

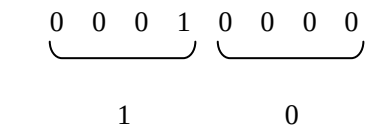
10 进制数是用得最多的数值，输入 PLC、PLC 输出的数据、常数数据等，大多用 10 进制形式。但是 PLC 的 I/O 模块及内部处理所表示的数据，必须是二进制的 ON/OFF (“1”，“0”) 状态的集合。因此，在 PLC 内部表示 10 进制数的是 BCD 码 (即二进制码表示的十进制数)，即 1 位 10 进制数用 4 位二进制数来表示。

10 进制数							
第 8 位	第 7 位	第 6 位	第 5 位	第 4 位	第 3 位	第 2 位	第 1 位
9~0	9~0	9~0	9~0	9~0	9~0	9~0	9~0

BCD 码 (2 进制码表示的 10 进制数)							
8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1
×10 ⁷	×10 ⁶	×10 ⁵	×10 ⁴	×10 ³	×10 ²	×10	×1

BCD 每位与 10 进制数的关系

BCD 运算的场合，不能使用右表以外的代码
例:BCD 码不能将十进制数 10 表示为 1010。
十进制数的 10 以 BCD 码表示为下述数码。



10 进数		0	1	2	3	4	5	6	7	8	9
BCD 码	8	0	0	0	0	0	0	0	0	1	1
	4	0	0	0	0	1	1	1	1	0	0
	2	0	0	1	1	0	0	1	1	0	0
	1	0	1	0	1	0	1	0	1	0	1

(3) 16 进制数的形态

16 进制数是 2 进制码以 4 位为单位数字化的数值。(1 位数值由 0~F 来表示)。

16 进制数 (下位)

第 4 位 F ⁰	第 3 位 F ⁰	第 2 位 F ⁰	第 1 位 F ⁰

二进制码 (下位)

215	214	213	212	211	210	29	28	27	26	25	24	23	22	21	20
(8, 4, 2, 1) × 2 ¹²				(8, 4, 2, 1) × 2 ⁸				(8, 4, 2, 1) × 2 ⁴				(8, 4, 2, 1) × 2 ⁰			

16 进制数 (上位)

第 8 位 F ⁰	第 7 位 F ⁰	第 6 位 F ⁰	第 5 位 F ⁰

二进制码 (上位)

231	230	229	228	227	226	225	224	223	222	221	220	219	218	217	216
(8, 4, 2, 1) × 2 ²⁸				(8, 4, 2, 1) × 2 ²⁴				(8, 4, 2, 1) × 2 ²⁰				(8, 4, 2, 1) × 2 ¹⁶			

1 位 16 进制数的二进制码

(10) (11) (12)

(13) (14) (15)

16 进制数	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
二 进 制 码	8	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	4	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
	2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

10 进制数与 16 进制数

10 进制数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		32
16 进制数	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	20

(4) 8 进制数的形态

8 进制数是将二进制码以 3 位为单位数字化的数值, 用于功能存储器编号等。

※I/O 内部继电器、级、定时器、计数器等均用 8 进制数表示。

8 进制数

	第 5 位 7 ⁰	第 4 位 7 ⁰	第 3 位 7 ⁰	第 2 位 7 ⁰	第 1 位 7 ⁰

二进制码 (下位)

215	214	213	212	211	210	29	28	27	26	25	24	23	22	21	20
(4, 2, 1) × 2 ¹²			(4, 2, 1) × 2 ⁹			(4, 2, 1) × 2 ⁶			(4, 2, 1) × 2 ³			(4, 2, 1) × 2 ⁰			

1 位 8 进制数的二进制码

8 进制数	0	1	2	3	4	5	6	7
二 进 制 码	4	0	0	0	0	1	1	1
	2	0	0	1	1	0	0	1
	1	0	1	0	1	0	1	0

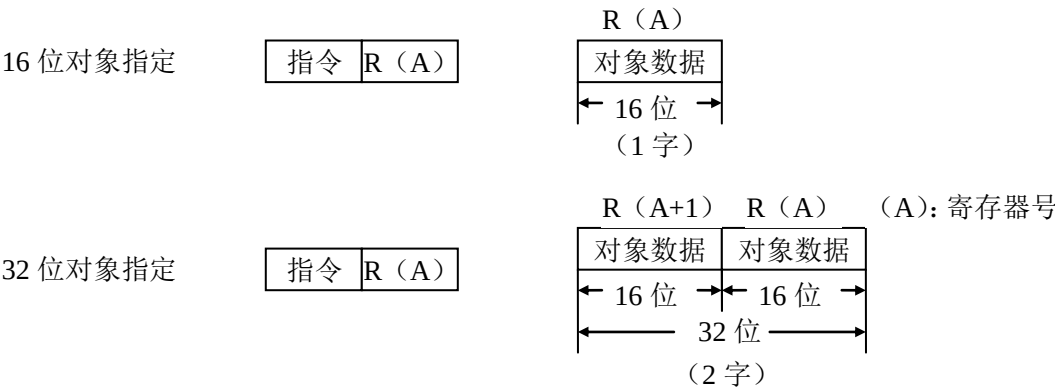
10 进制数与 8 进制数

10 进制数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
8 进制数	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20	21	22	23	24

8.2 数据的指定

(1) 指定寄存器号 R

指令中的操作数指定存储对象数据的寄存器号，另外，以 R 指定的指令中有以 16 位为对象的指令和以 32 位为对象的指令。

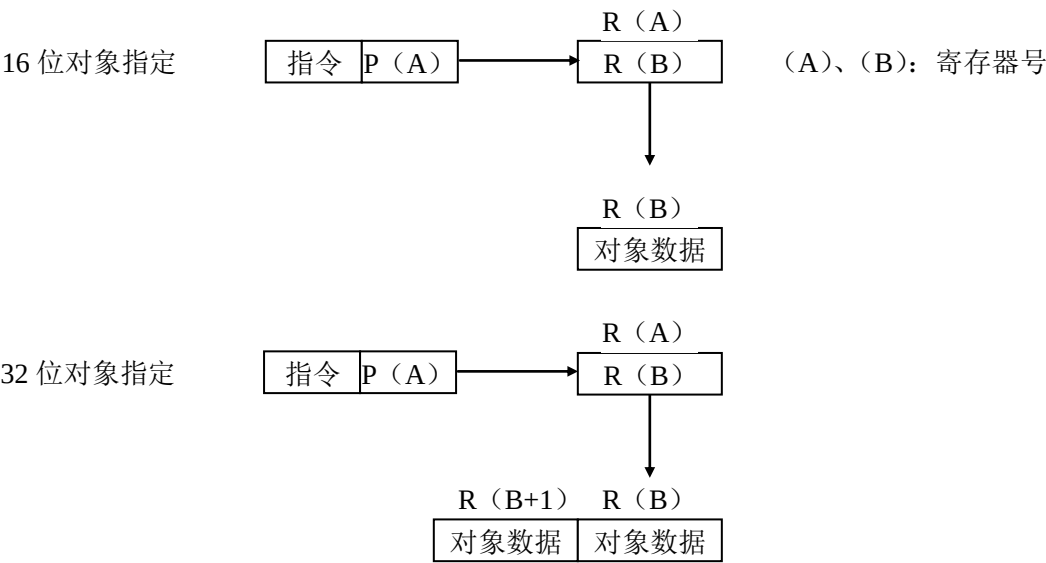


例：将 8 位 10 进制数存储在 2 个寄存器里的场合。

10 进制数		7		3		4		1		5		8		2	
		4													
BCD 码	位数	第 8 位	第 7 位	第 6 位	第 5 位	第 4 位	第 3 位	第 2 位	第 1 位						
	加权	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1					
	数据	0 1 1 1	0 0 1 1	0 1 0 0	0 0 0 1	0 1 0 1	1 0 0 0	0 0 10	0 1 0 0						
寄存器号		R2001						R2000							

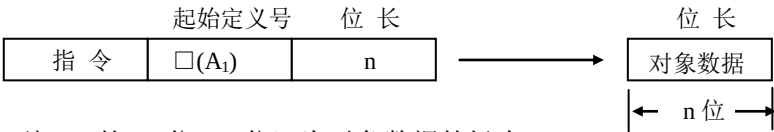
(2) 寄存器号的间接指定 P

间接指定 P 所指定的 R (A) 寄存器中存储的是 R (B) 寄存器号，R (B) 寄存器中存储所需处理的对象数据。

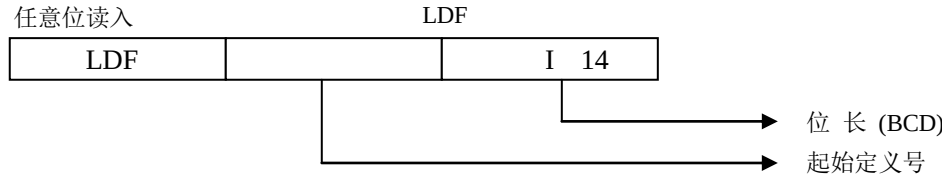
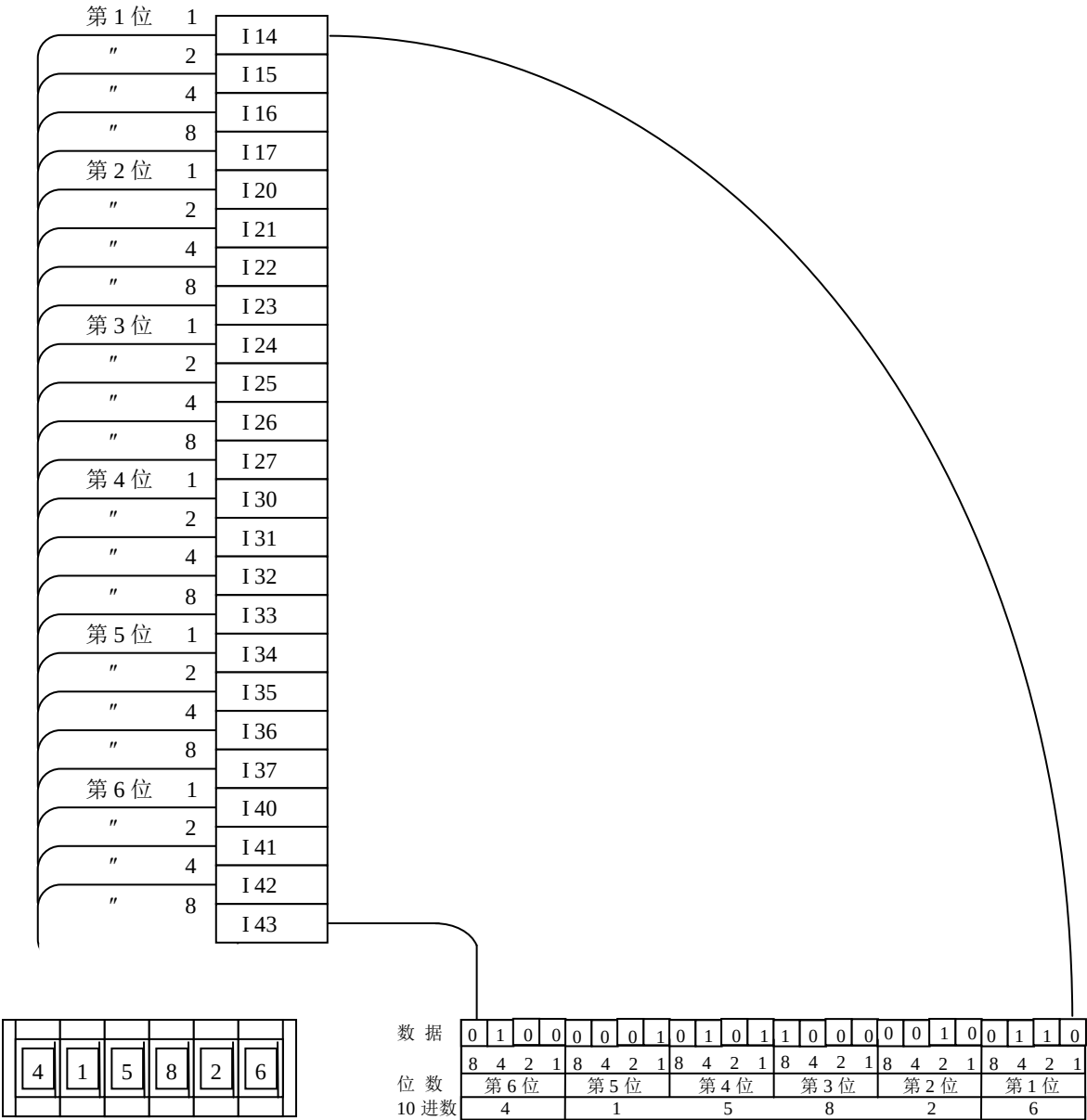


(3) 任意位长指定 I, Q, GI, GQ, M, S, T, C, SP

对象数据不为 16 位或 32 位时，或从位号中途开始（即不是从 $\times\times\times 0$ 定义号开始）的数据时，用该方法。

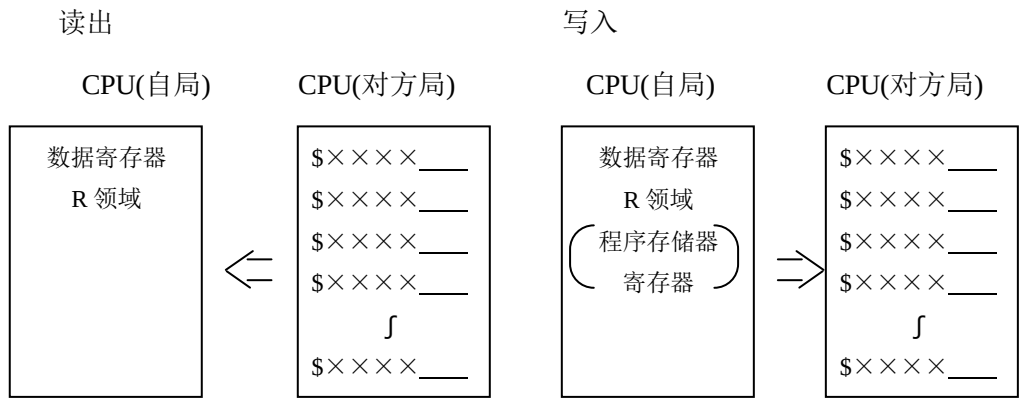


例：以从输入 I14 到 I43 的 24 位（6 位）为对象数据的场合。



(4) 程序存储器地址指定\$

用于模块对象的特殊指令，通过 PLC 通讯模块从其他 PLC 读出程序或者向其他 PLC 写入程序时使用。

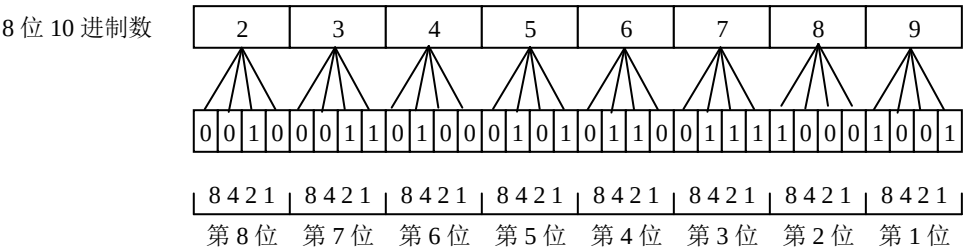


(5) 10 进制或 16 进制常数指定 K

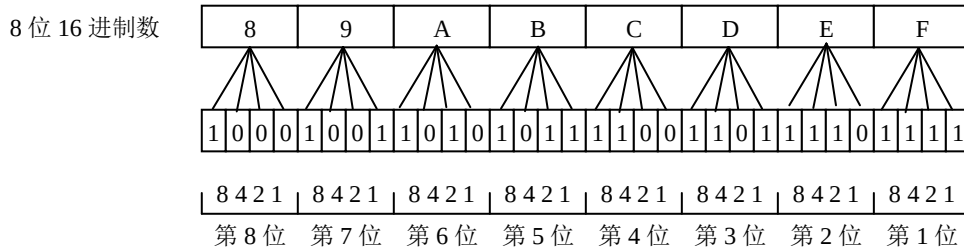
数据处理上所需要的固定数值由程序指令直接给出。

K 用来指定 10 进制常数（BCD 码）或者 16 进制常数（二进制码）。10 进制常数时由 0~9 的键来指定，16 进常数时，由 0~F 键来指令。以 K 指定的数据，将 1 位分解为 4 位二进制来记忆。

例：8 位 10 进制数的表示



例：8 位 16 进制数的表示



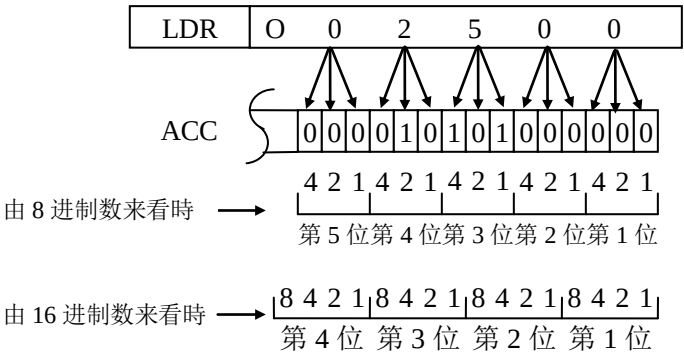
(6) 8 进制常数指定（寄存器号）0（octal）

寄存器号以 8 进制数来分配，将该寄存器号作为数据在程序上进行处理时，以 0（octal）来指定。

具体用 LDR 指令

指令	操作数
LDR	0 (A)

例：(A) 为 2500



由操作常数指定 8 进制数，为二进制码，因此可进行二进制运算。

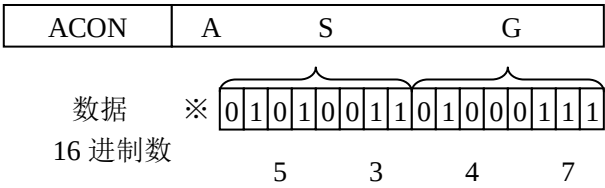
(7) ASCII 码指定 A（ASCII）

登记 ASCII 码数据时用。

指令	操作数
ACON	A (B)

在 (B) 处，可设定两个 0~9 及 A~Z 的英文字母、数字。

例：登记 ASCII 文字“SG”

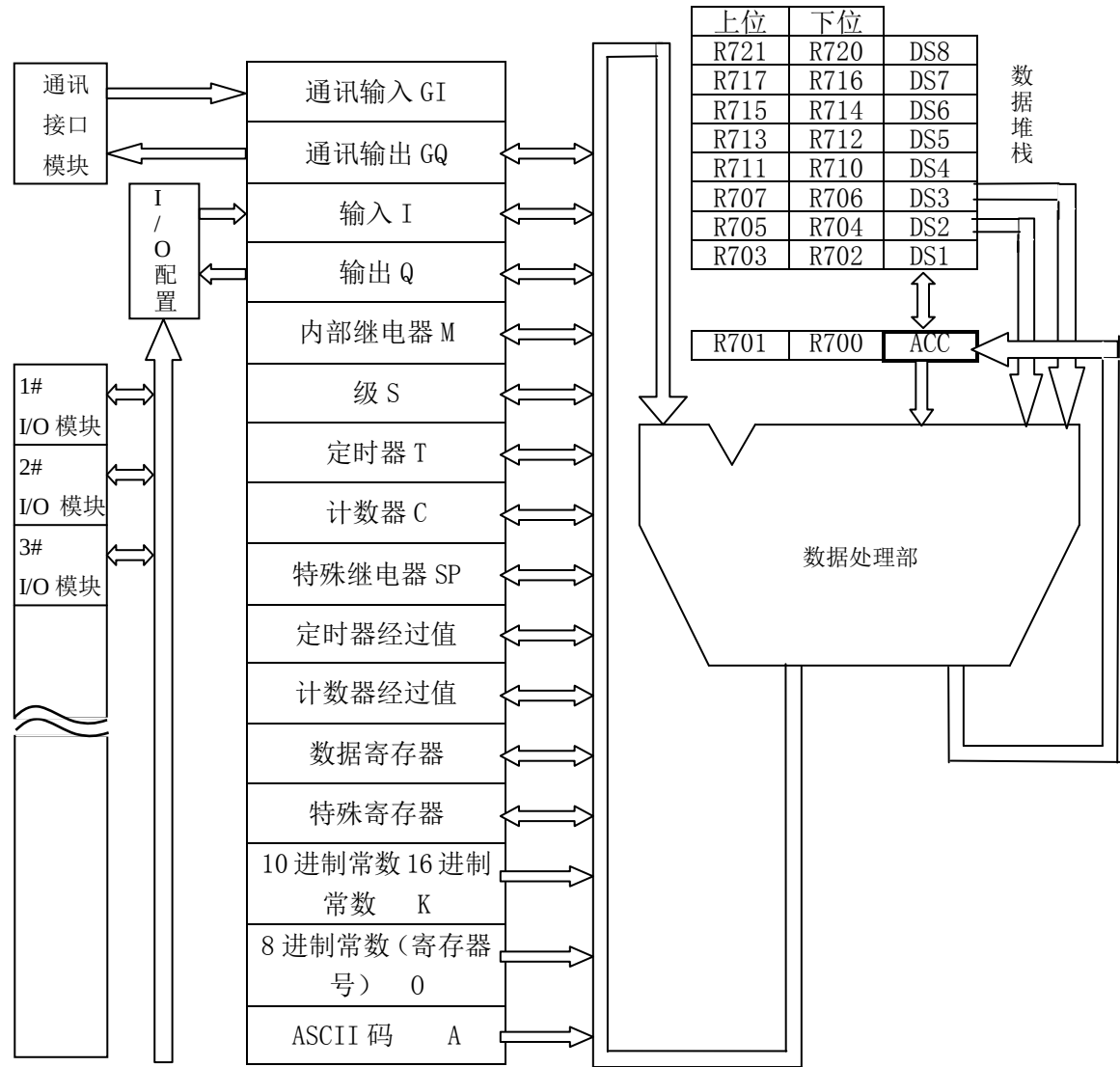


※请参见附录“ASCII 字码表”。

8.3 数据处理的基本形式

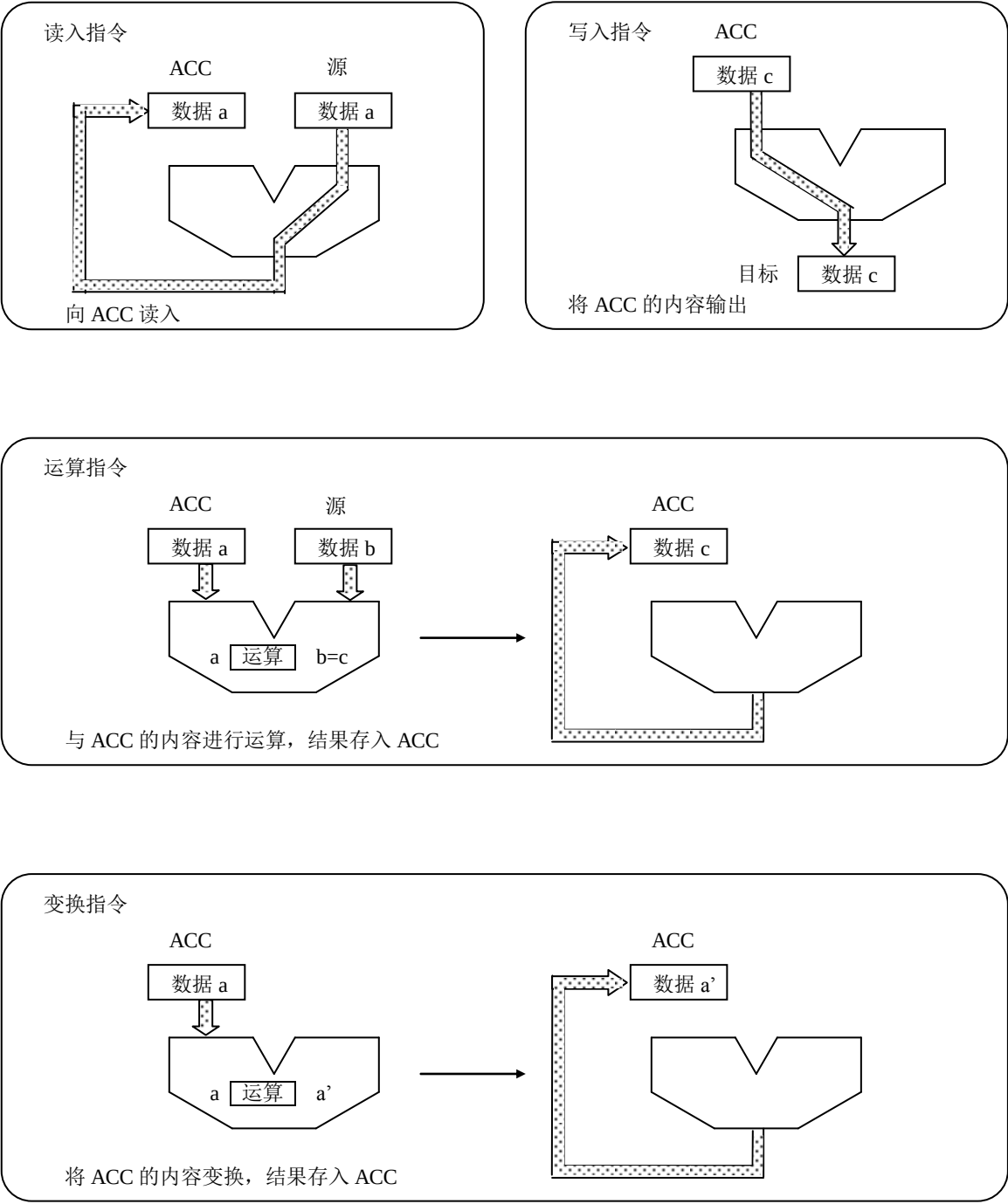
8.3.1 数据的流向

各机种 PLC 可处理各种存储器内的数据和常数，而这些数据处理基本上以 ACC（累加器）为中心进行处理。



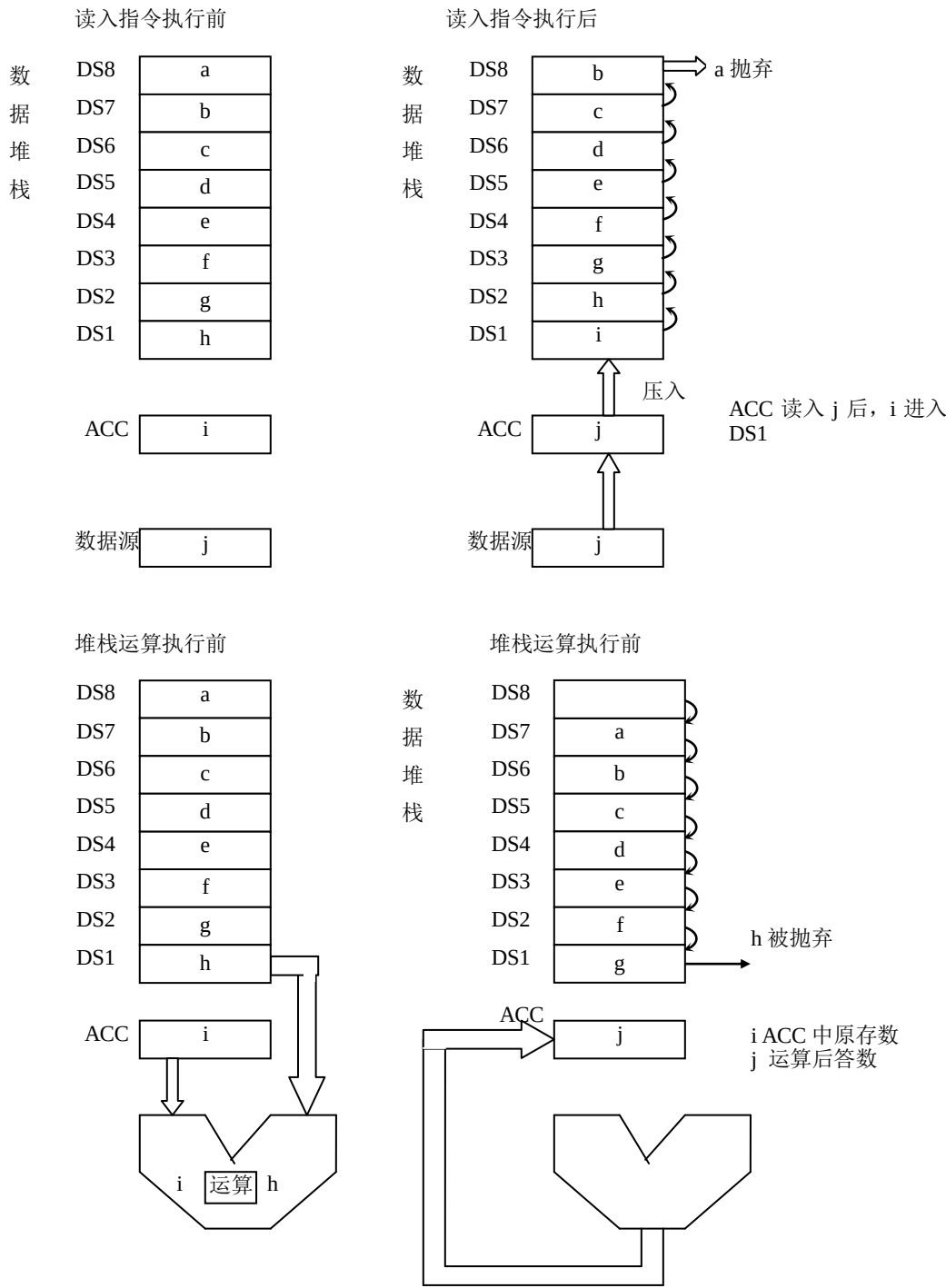
8.3.2 累加器的性质

累加器是一种特殊寄存器，它由 32 位构成，暂时存储数据处理结果。
基本的数据处理指令分为如下 4 种。



8.3.3 数据堆栈的性质

数据堆栈是累加器内的数据暂存用的特殊寄存器，执行读入指令时，累加器的内容被压入数据堆栈，由堆栈运算指令或 POP 指令取出。



另外，在许多有参照数据的指令中，数据堆栈与 ACC 一起，用于存储参照数据。

8.3.4 标志的性质

表示数据处理的结果数据以外的信息，有 13 种标志。

某些指令执行后，通过有关的标志继电器的 ON 或 OFF 状态来表示指令执行的部分结果。与该指令无关的标志继电器状态不变。

运算出 错	小于	等于	大于	零	半借 位	借位	半进 位	进位	符号	溢出	数据出 错	读零
SP053	SP060	SP061	SP062	SP063	SP064	SP065	SP066	SP067	SP070	SP073	SP075	SP076

1. 运算出错标志（SP053）（ER1）

某些指令中，如操作数超出规定范围，而使运算结果出错，则为 ON

2. 小于（SP060）、等于（SP061）、大于（SP062）标志。

执行比较指令时 ACC 的值与比较值相比，根据比较结果使对应的一个继电器为 ON 状态，另两个为 OFF 状态。

3. 零标志（SP063）（Z）

处理结果为“0”时 ON，不为“0”时 OFF。

4. 半借位标志（SP064）（HB）

相减的结果，由第 4 位向第 5 位借位时 ON，没有借位时 OFF。

5. 借位标志（SP065）（B）

相减的结果，差为负值（数据为 100000000 补数）而向第 9 位借位时 ON，没有借位时 OFF。

6. 半进位标志（SP066）（HC）

加法运算的结果从第 4 位向第 5 位进位时 ON，没有进位 OFF。

7. 进位标志（SP067）（C）

加法运算的结果，超过 8 位向第 9 位进位时 ON，没有进位时 OFF。

8. 符号（SP070）（S）

实行装入指令或运算指令时，指令实行后，把 ACC 的最上位的状态作为该标志继电器的状态。

9. 溢出（SP073）（OV）

处理的结果数据超过 32 位时 ON，正常时 OFF。

10. 数据出错（SP075）（DE）

数据运算时，运算数据不是 BCD 数据时 ON，是 BCD 数据时 OFF。

11. 读零标志（SP076）（RZ）

指令指定的源数据为“0”时 ON，不为“0”时 OFF。

注：在本手册中，提到 4 位，8 位数据，这时，位指的是由四位二进制数（bit）组成的一位 BCD 数或十六进制数。

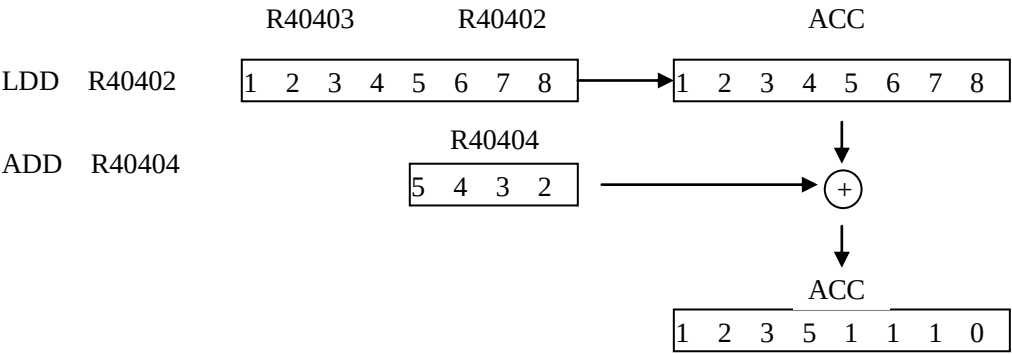
提到 16 位，32 位，此处的位，指二进制数的一位（bit）。

8.3.5 算术运算的思考方法

(1) 运算位数

在算术运算中，除特例外，运算数据是指定的位数，被运算数据是 8 位数，运算结果除特例外也为 8 位数。

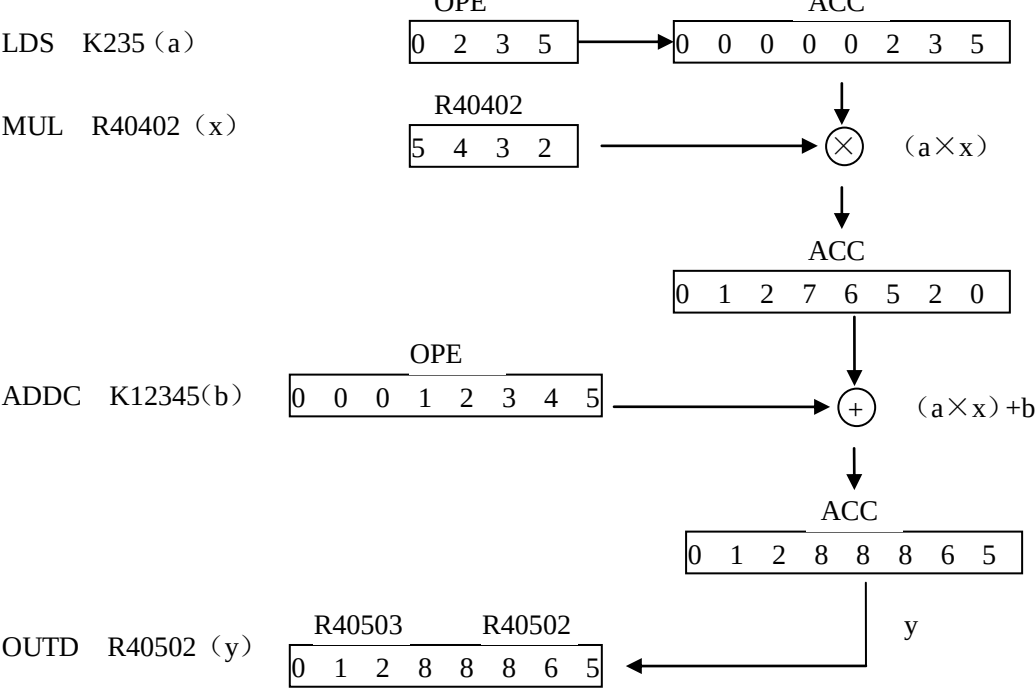
例：8 位数+4 位数=8 位数



(2) 算式的指令化

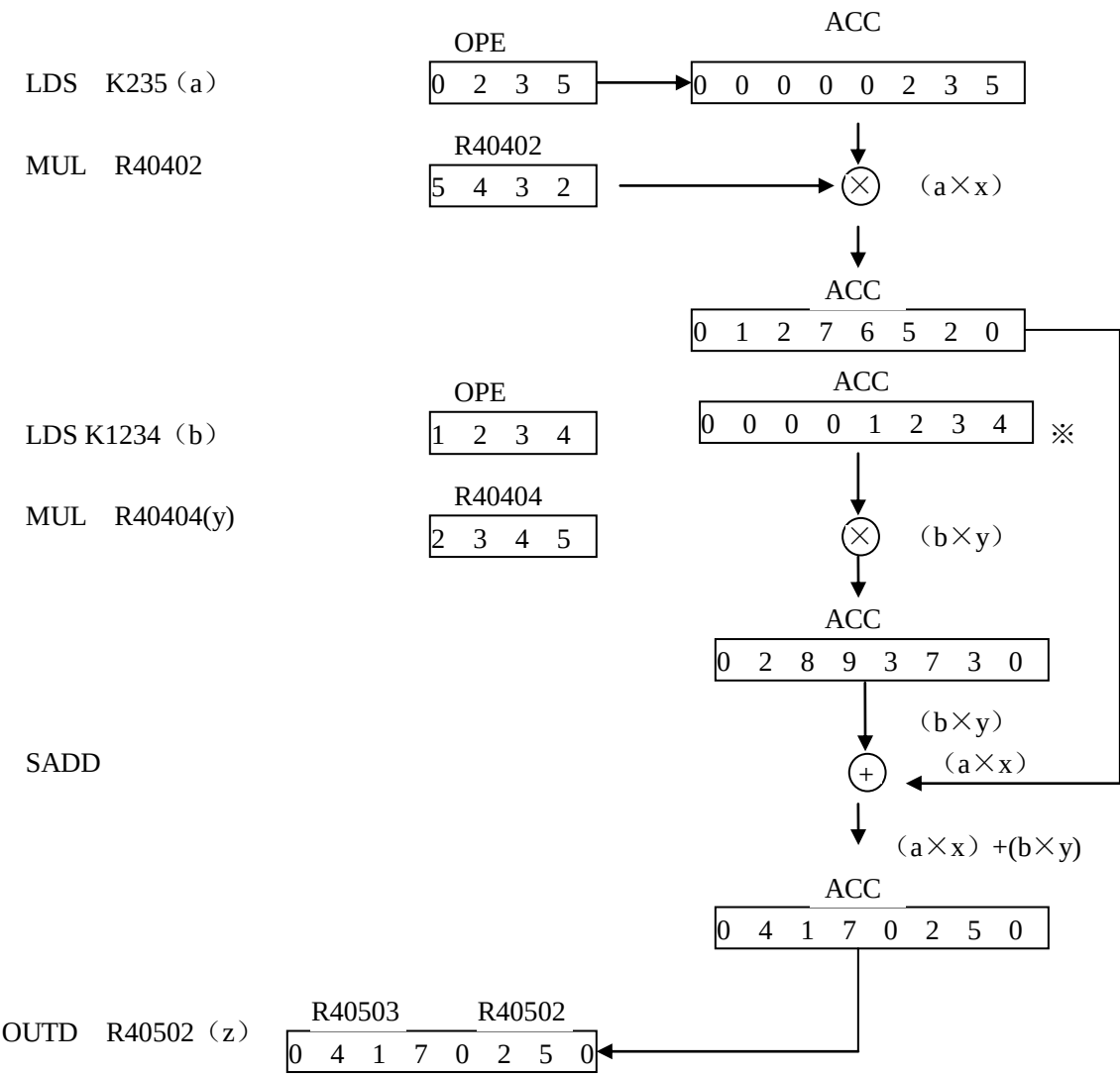
运算结果总是存在 ACC 里，对结果再进行别的运算时，可继续进行。

例：y=ax+b



注：“OPE”表示“操作数”。

对各分别算出的答案再进行运算时，用堆栈运算。
例： $z=ax+by$



※执行 LDS K1234(b) 指令时，把 ACC 中的原数值 (01276520) 压入数据堆栈。

(3) 负值范围的数值。

本来 CPU 上基本只处理 8 位正整数。但如在使用方法上想想办法，负值范围的数据也可处理。

● 只在正值范围使用时的思考方法

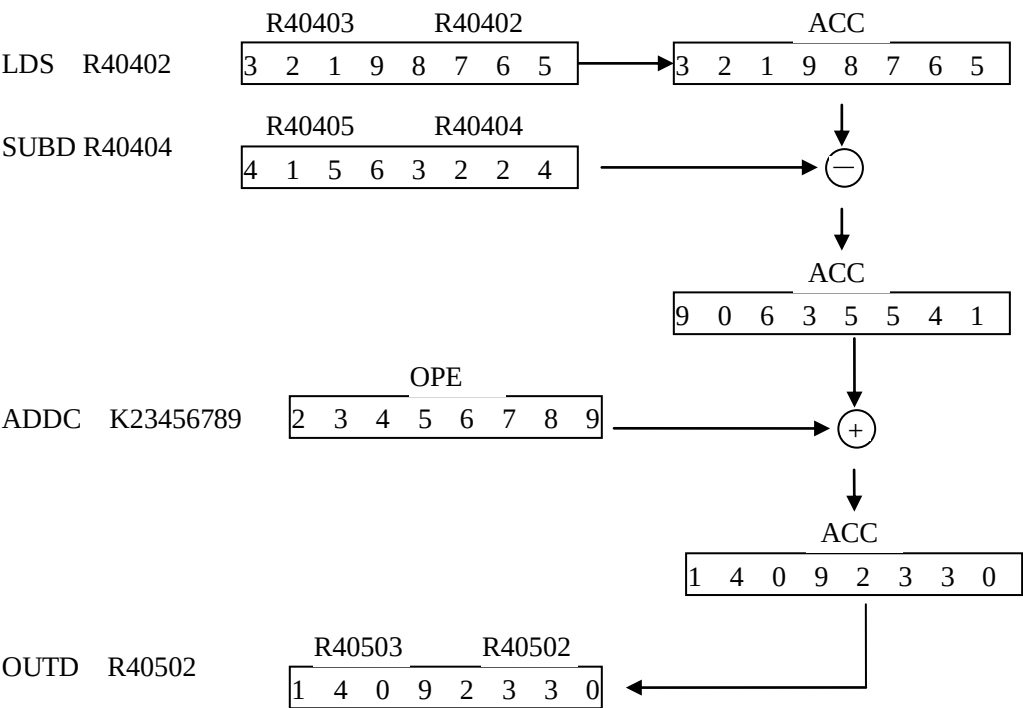
算术运算进行 8 位正整数运算，因此可将 8 位全部作为数据进行处理。此时 BCD 数据的范围为 0~99999999，二进制数据的范围为 0~FFFFFFF，另外执行算术运算指令时的进位，成为向第 9 位进位的信号，而借位则为向第 9 位借位的信号，因此超过 8 位的数据也可管理。



● 计算途中取负值而最终结果为正值时

进行算术运算时，中途数据一时为负值，而最终取正值时，以通常的处理即可达到目的。

例：



将该计算与带符号的数值相比如下:

带符号计算		数据处理
32198765	借位	32198765
<u>-41563224</u>		<u>-41563224</u>
-09364459		90635541
<u>+23456789</u>	进位	<u>+23456789</u>
+14092330		14092330

该计算过程中，途中经过情况不同，但最终答案一致，是因为数据处理中借位的数与进位的数一致。

(因 $-1000000000+1000000000=0$)

最终的计算结果肯定是正值时，因借位与进位一致，用通常运算处理没问题。

●数据在-19999999~79999999 范围内时

设定 32 位数据的最上位为符号位，“0”时为正值，“1”时为负值，数据看作 100000000 的补数。

正范围

负范围

最上位为“1”时，该数据为负值，存入的是该数据的 100000000 的补数。

以这样的方法管理的数据，外部输出数据等需要绝对值时，用 BDCPL 指令求出绝对值再输出。

例：

$$\begin{array}{r}
 \text{带符号计算} \\
 00000001 \\
 -00000002 \\
 \hline
 99999999
 \end{array}
 \quad \leftarrow \text{最上位为 1，因此为负值}$$

$$\begin{array}{r}
 \text{补数变换} \\
 00000001
 \end{array}
 \quad \leftarrow \text{该值为绝对值}$$

(4) 二进制运算时的符号处理

● 有符号、无符号的演算例

下面是有符号运算及无符号运算时的例子程序。

例： LDS K 8080
 BMULS K 20
 OUTD R 2000

◆ 有符号

 F F F F 8 0 8 0 (符号扩展)
× 2 0

 F F F 0 1 0 0 0

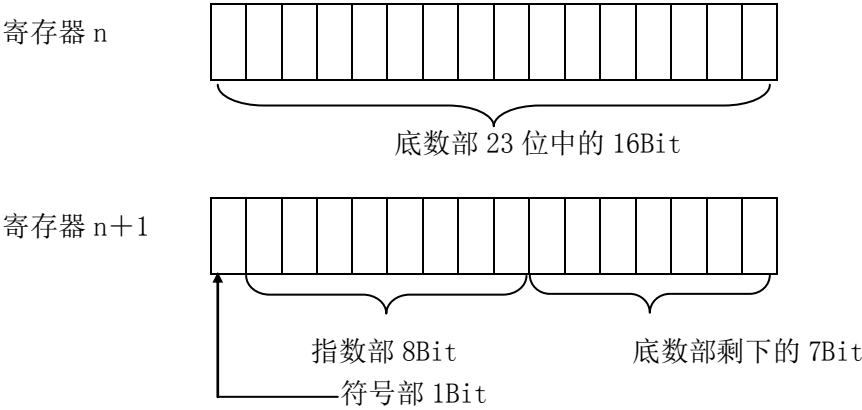
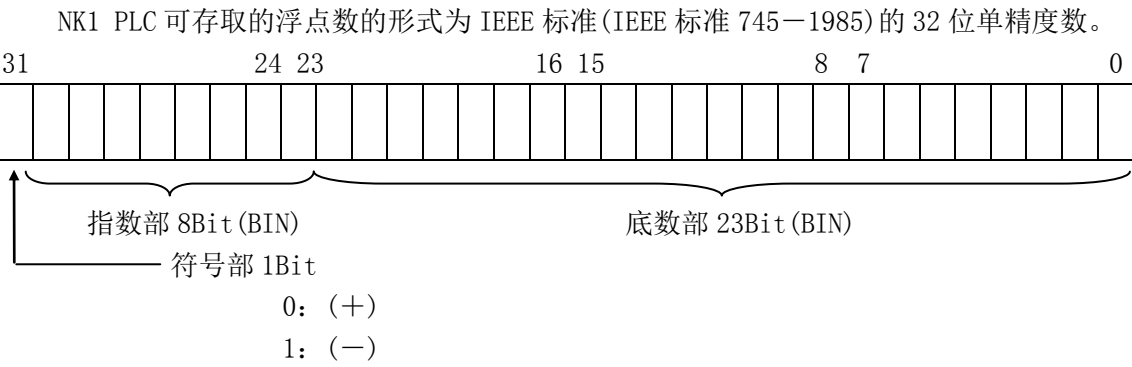
◆ 无符号

 0 0 0 0 8 0 8 0 (无符号扩展)
× 2 0

 0 0 1 0 1 0 0 0

注意：有符号和无符号的运算结果可能会不相同。

8.4 浮点数的形式



※ 浮点数的表现形式如下表

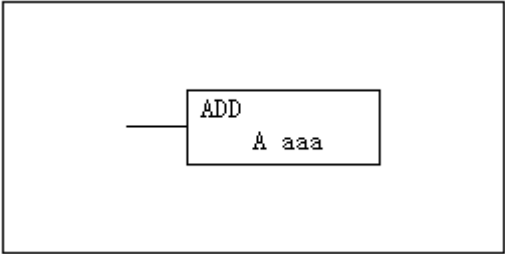
指数部 底数部	0	非全 0 或全 1	全 Bit 位为 1
0	0	正规化数	无穷大
0 以外	非正规化数	正规化数	非数

- 1) 正规化数
正规化数值表示为: (符号部±) $2^{(\text{指数部}-127)} \times (1+\text{底数部} \times 2^{-23})$
- 2) 非正规化数
非正规化数值表示为: (符号部±) $2^{(\text{指数部}-127)} \times (\text{底数部} \times 2^{-23})$
- 3) 无穷大
其值表示为: $-\infty$ 、 $+\infty$
- 4) 非数
没有对应的数值或对应的无穷大数。

第 9 章 算术运算指令

9.1 4 位 BCD 加法指令（ADD）

输入 ADD 指令是一个 16 位指令，将累加器中的 BCD 数同指定寄存器（A aaa）中的 BCD 数相加。



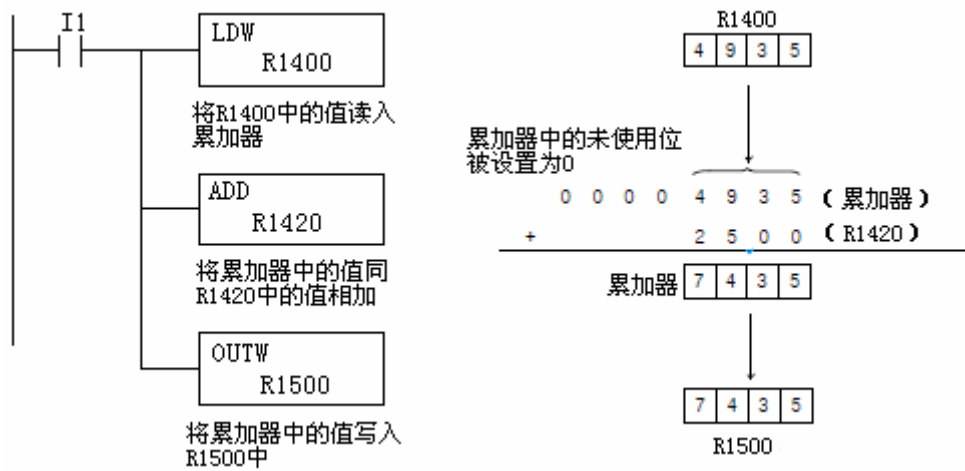
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，ADD 指令将累加器中低 16 位的数据同 R1420 中的数据相加，OUTW 指令将累加器中的数据写入 R1500 中。

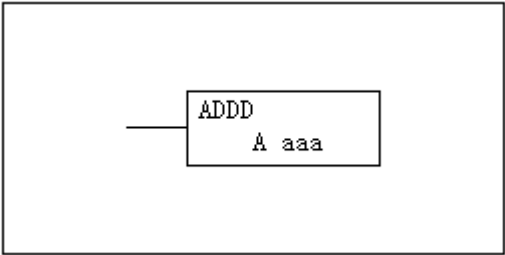


语句视图

```
LD I1
LDW R1400
ADD R1420
OUTW R1500
```

9.2 8 位 BCD 加法指令（ADDD）

 **输入** ADDD 指令是一个 32 位指令，将累加器中的 BCD 数同指定寄存器（A aaa）中的 BCD 数相加，结果存放在累加器中。



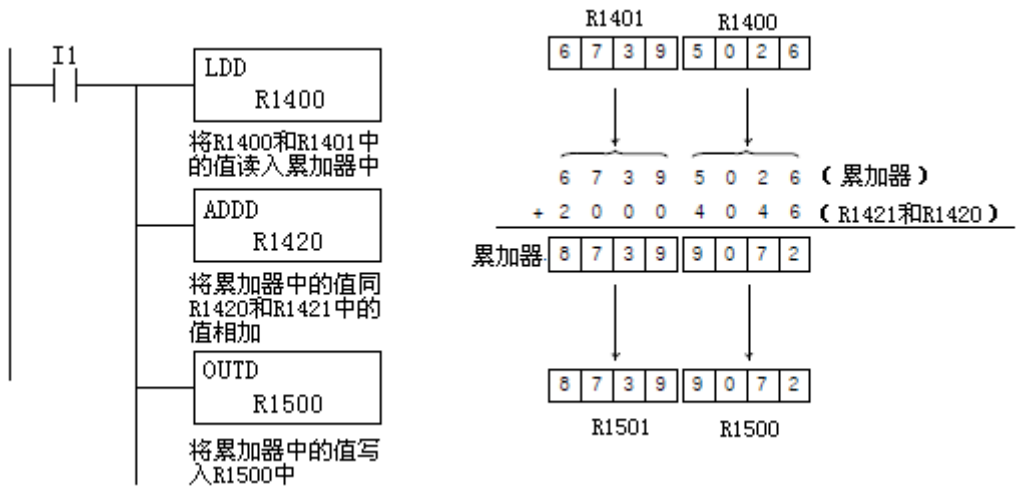
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据装入累加器中，ADDD 指令将累加器中的数据同 R1420 和 R1421 中的数据相加，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。

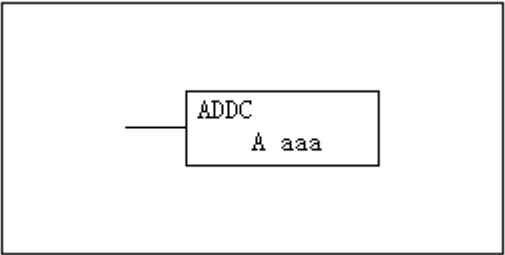


语句视图

```
LD I1
LDD R1400
ADDD R1420
OUTD R1500
```

9.3 8 位 BCD 常数加法指令（ADDC）

 **输入** ADDC 指令是一个 32 位指令，将累加器中的 BCD 数同 8 位 BCD 常数相加，结果存放在累加器中。



操作数类型	NK1 范围	
	A	aaa
常数	K	0-99999999

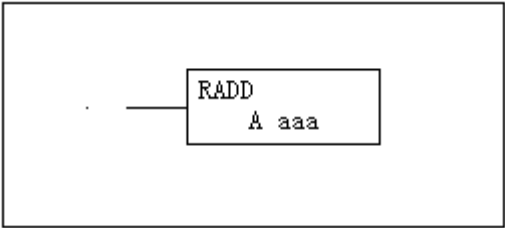
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.4 实数加法指令（RADD）

输入 RADD 指令是一个 32 位指令，将累加器中的实数同两个连续寄存器中的实数相加。相加的两个实数必须是 IEEE 实数格式，结果存放在累加器中。



操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

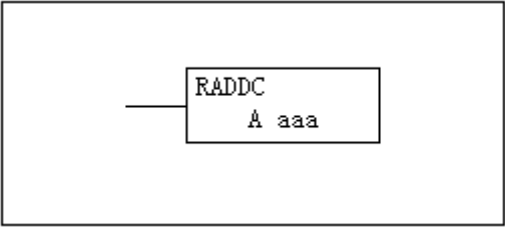
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.5 实数常数加法指令（RADD C）

 RADD C 指令是一个 32 位指令，将累加器中的实数同一个实数常数相加。相加的两个实数必须是 IEEE 实数格式，结果存放在累加器中。

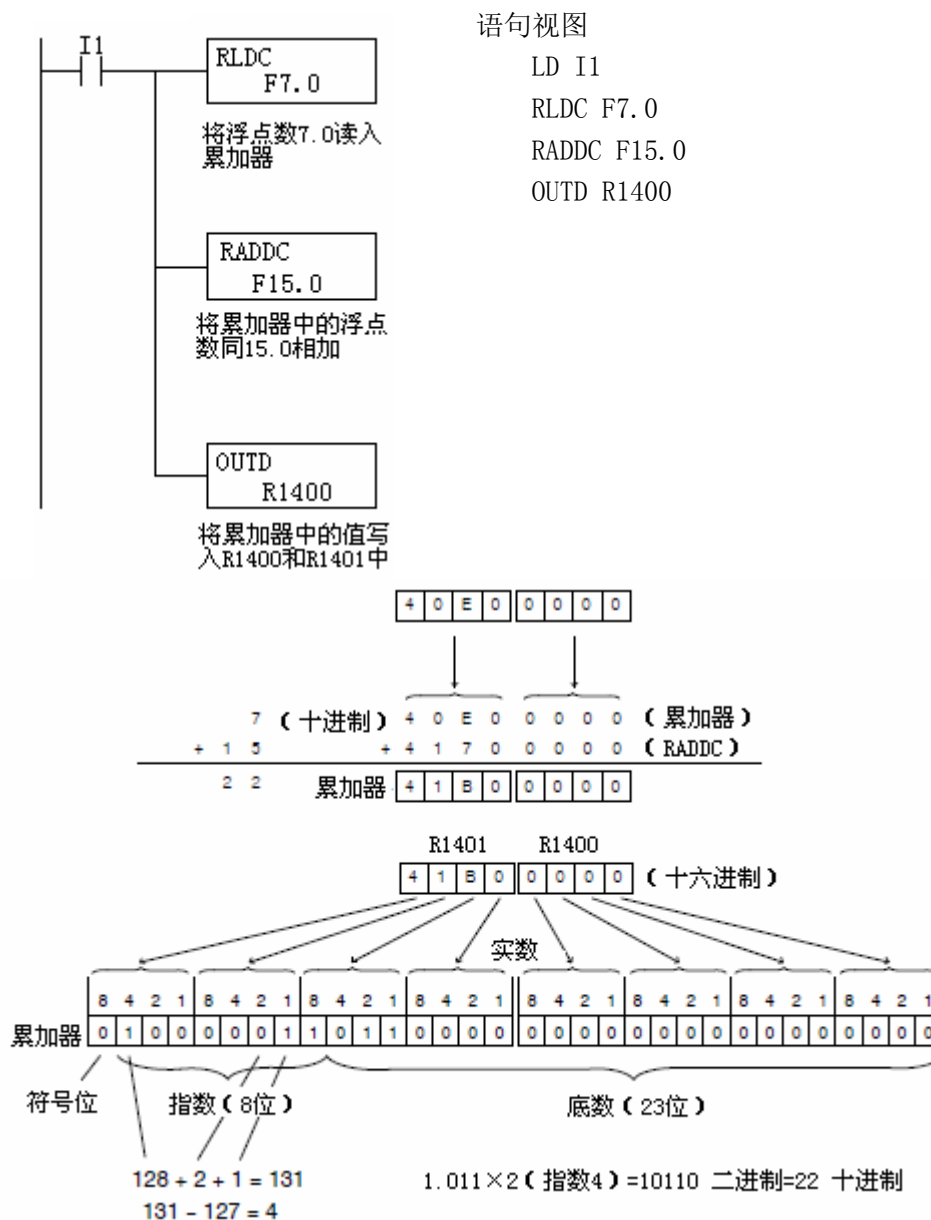


操作数类型	NK1 范围	
A	aaa	
常数	F	-3. 402823E+038 [~] +3. 402823E+038

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON



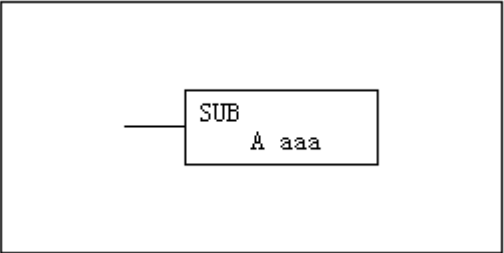
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。



注意：如果指令中的实数比累加器中的实数小 16,777,216 倍，指令将不执行。

9.6 4 位 BCD 减法指令（SUB）

 SUB 指令是一个 16 位指令，将累加器中的 BCD 数减去指定寄存器（A aaa）中的 BCD 数。



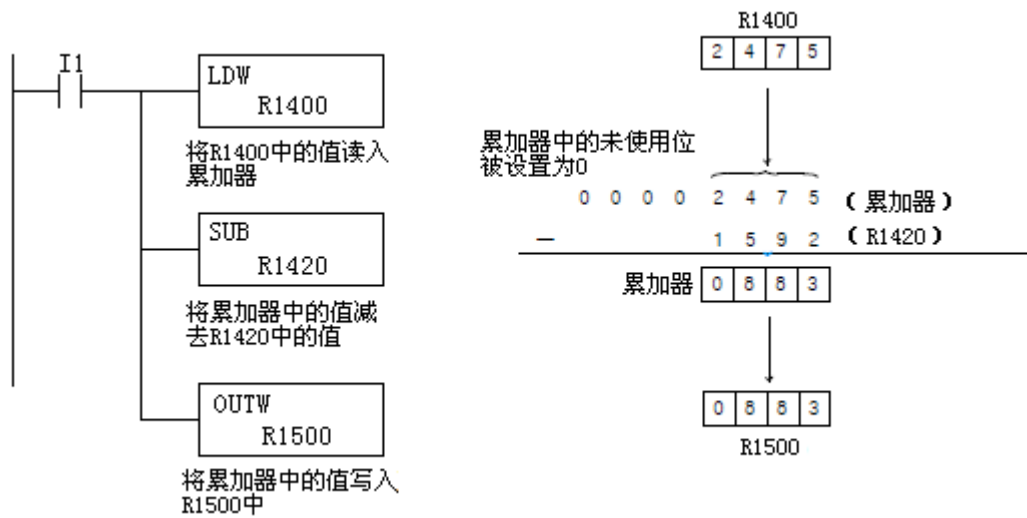
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，SUB 指令将累加器中低 16 位的数据减去 R1420 中的数据，OUTW 指令将累加器中的数据写入 R1500 中。



语句视图

LD I1

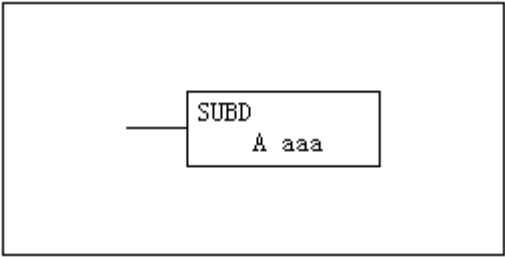
LDW R1400

SUB R1420

OUTW R1500

9.7 8 位 BCD 减法指令（SUBD）

 SUBD 指令是一个 32 位指令，将累加器中的 BCD 数减去指定寄存器（A aaa）中的 BCD 数，结果存放在累加器中。



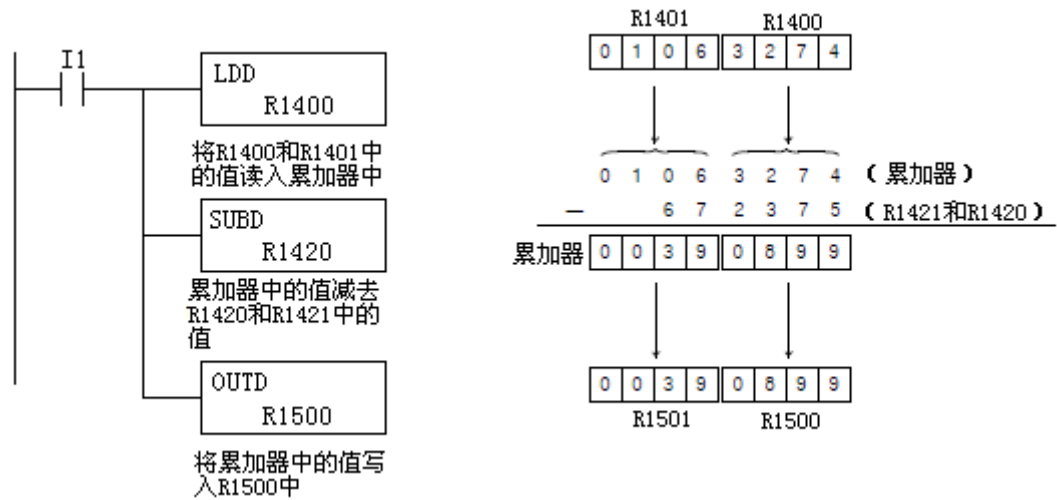
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据装入累加器中，SUBD 指令将累加器中的数据减去 R1420 和 R1421 中的数据，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。

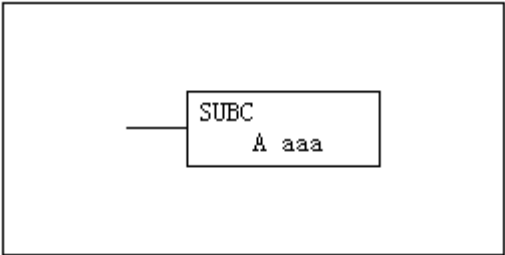


语句视图

```
LD I1
LDD R1400
SUBD R1420
OUTD R1500
```

9.8 8 位 BCD 常数减法指令（SUBC）

 SUBC 指令是一个 32 位指令，将累加器中的 BCD 数减去一个 8 位 BCD 常数，结果存放在累加器中。



操作数类型	NK1 范围	
	A	aaa
常数	K	0-99999999

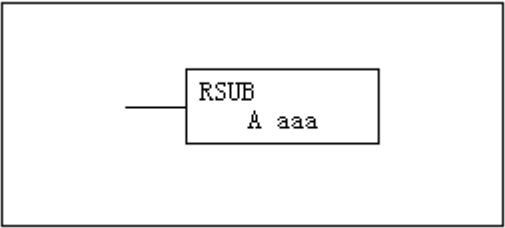
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.9 实数减法指令（RSUB）

输入 RSUB 指令是一个 32 位指令，将累加器中的实数减去两个连续寄存器中的实数。两个实数必须都是 IEEE 实数格式。运算结果是 32 位实数，存放在累加器中。



操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

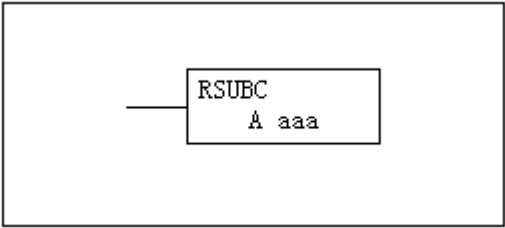
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.10 实数常数减法指令（RSUBC）

输入 RSUBC 指令是一个 32 位指令，将累加器中的实数减去一个实数常数。两个实数必须是 IEEE 实数格式，结果存放在累加器中。

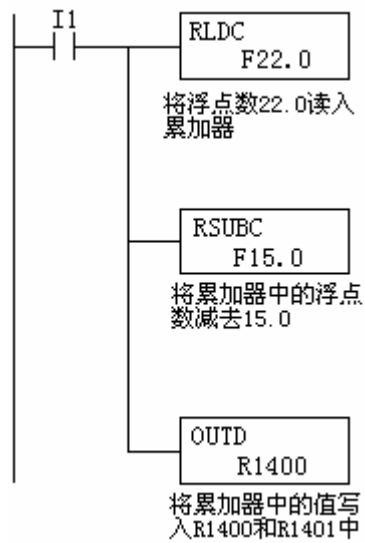


操作数类型	NK1 范围
A	aaa
常数	F -3. 402823E+038~+3. 402823E+038

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON

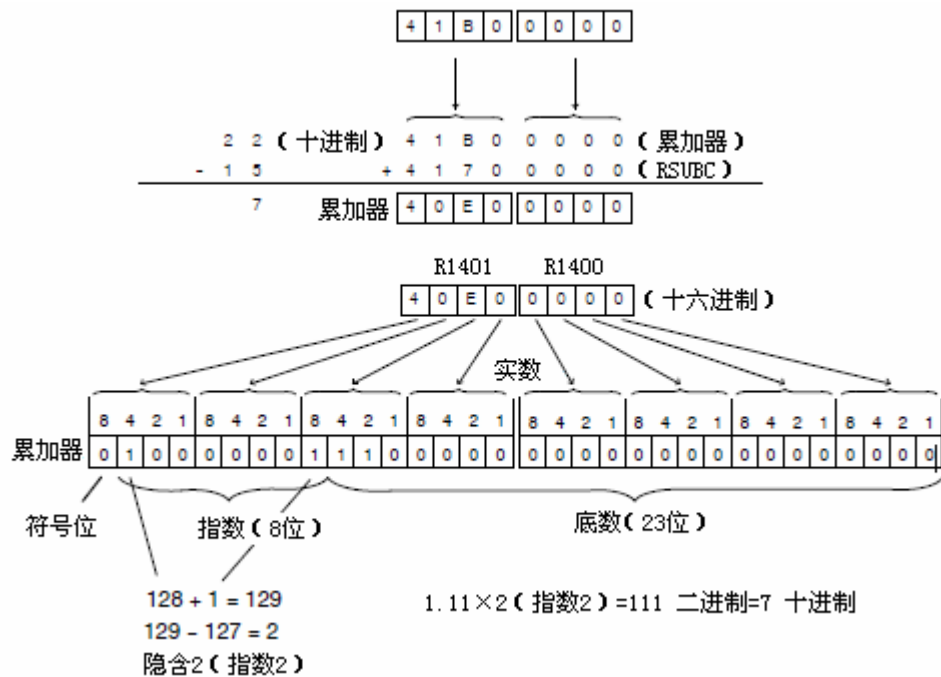


注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。



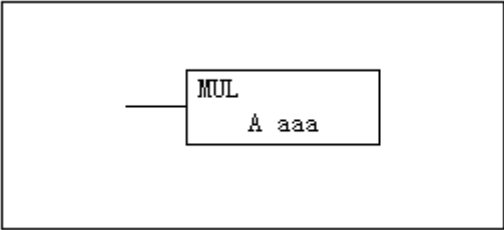
语句视图

```
LD I1
RLDC F22.0
RSUBC F15.0
OUTD R1400
```



9.11 4 位 BCD 乘法指令（MUL）

输入 MUL 指令是一个 16 位指令，将累加器中低 16 位中的 BCD 数乘以指定寄存器（A aaa）中的 BCD 数，结果有可能是 8 位 BCD 数，存放在累加器中。



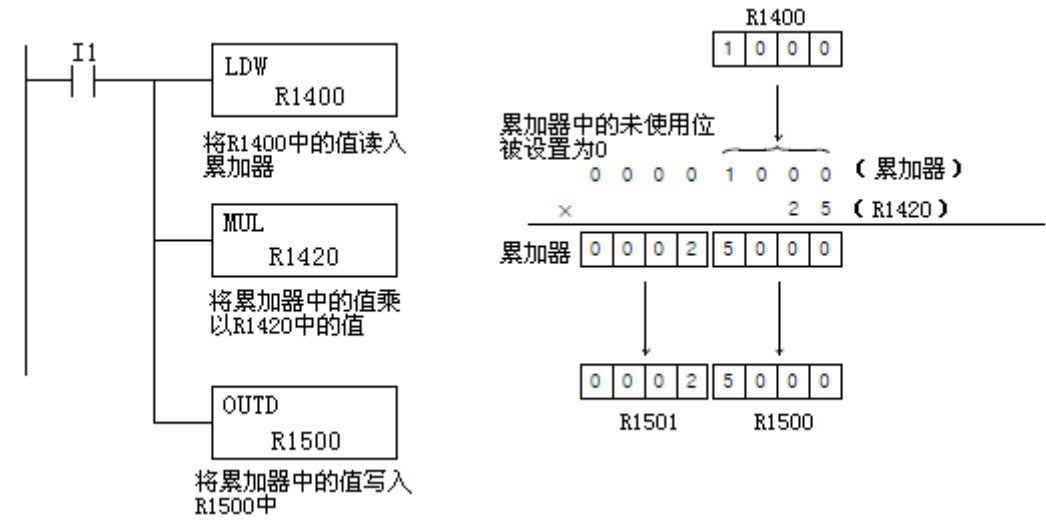
操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，MUL 指令将累加器中低 16 位的数据乘以 R1420 中的数据，OUTW 指令将累加器中的数据写入 R1500 和 R1501 中。

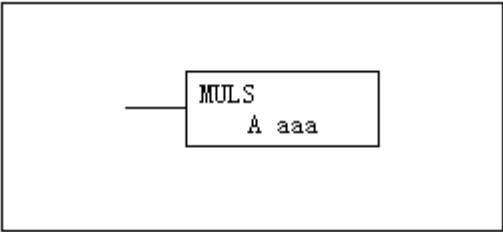


语句视图

```
LD I1
LDW R1400
MUL R1420
OUTD R1500
```

9.12 4 位 BCD 常数乘法指令（MULS）

输入 MULS 指令是一个 16 位指令，将累加器中低 16 位中的 BCD 数乘以一个 4 位（最大）BCD 常数，结果有可能是 8 位 BCD 数，存放在累加器中。



操作数类型	NK1 范围	
	A	aaa
常数	K	0-9999

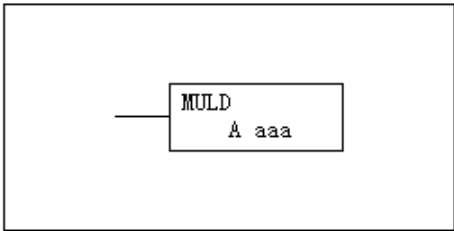
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.13 8 位 BCD 乘法指令（MULD）

输入 MULD 指令是一个 32 位指令，将累加器中的 8 位 BCD 数乘以两个连续寄存器（A aaa）中的 BCD 数，结果的低 8 位存放在累加器中，高位存放在数据堆栈中。



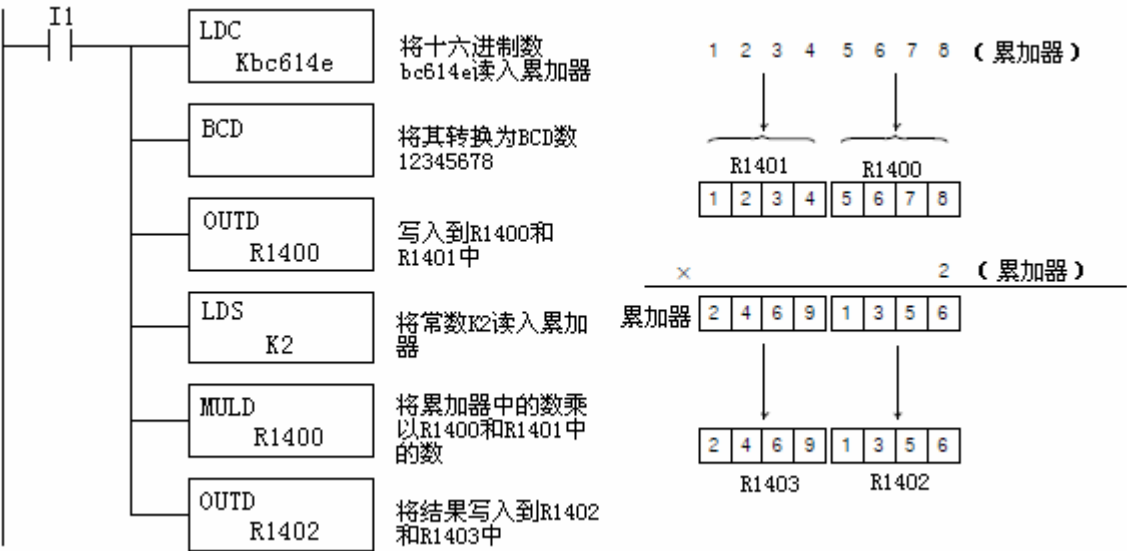
操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDC 指令将十六进制常数 Kbc614e 读入累加器中，BCD 指令将其转换为 BCD 数 12345678，写入到 R1400 和 R1401 中。LDS 指令将常数 K2 读入累加器中，MULD 指令将累加器中的数据乘以 R1400 和 R1401 中的数据 12345678，OUTD 指令将累加器中的数据写入 R1402 和 R1403 中。



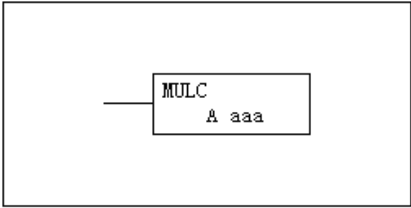
语句视图

```
LD I1
LDC Kbc614e
BCD
OUTD R1400
LDS K2
MULD R1400
OUTD R1402
```

9.14 8 位 BCD 常数乘法指令（MULC）

输入 MULC 指令是一个 32 位数据运算指令，将累加器中的 BCD 数乘上一个 8 位 BCD 常数，结果存放在累加器中。

注意，本运算指令只能产生 32 位数运算结果，当运算结果超出 32 位数时，报运算溢出错误（SP73=ON）



操作数类型	NK1 范围	
	A	aaa
常数	K	0-99999999

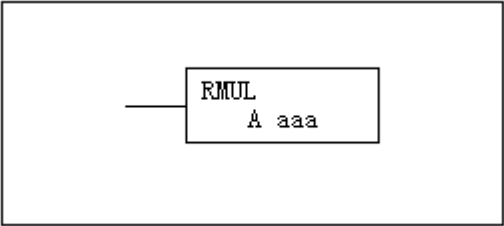
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据的 bit31=1 时 ON, bit31=0 时 OFF。
SP73	运算结果超出 32 位数时，为 ON (Overflow)。
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.15 实数乘法指令（RMUL）

 RMUL 指令将累加器中的实数乘以两个连续寄存器中的实数。两个实数必须都是 IEEE 实数格式，结果存放在累加器中。



操作数类型	NK1 范围	
A	aaa	
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

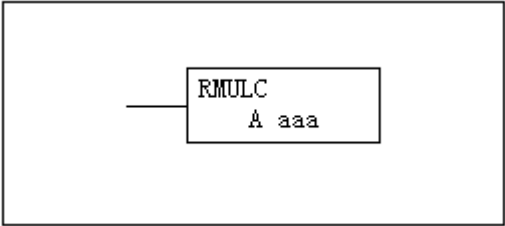
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.16 实数常数乘法指令（RMULC）

 **输入** RMULC 指令将累加器中的实数乘以一个实数常数。两个实数必须是 IEEE 实数格式。结果存放在累加器中。



操作数类型	NK1 范围	
	A	aaa
常数	F	-3. 402823E+038~+3. 402823E+038

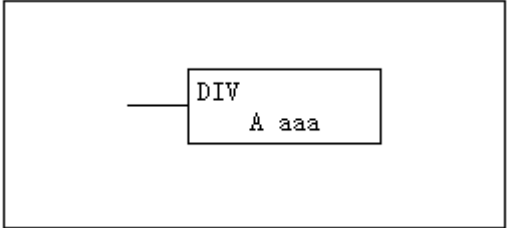
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.17 4 位 BCD 除法指令（DIV）

输入 DIV 指令是一个 16 位指令，将累加器中低 16 位中的 BCD 数除以指定寄存器（A aaa）中的 BCD 数，商存放在累加器中，余数存放在数据堆栈的第 1 级中。



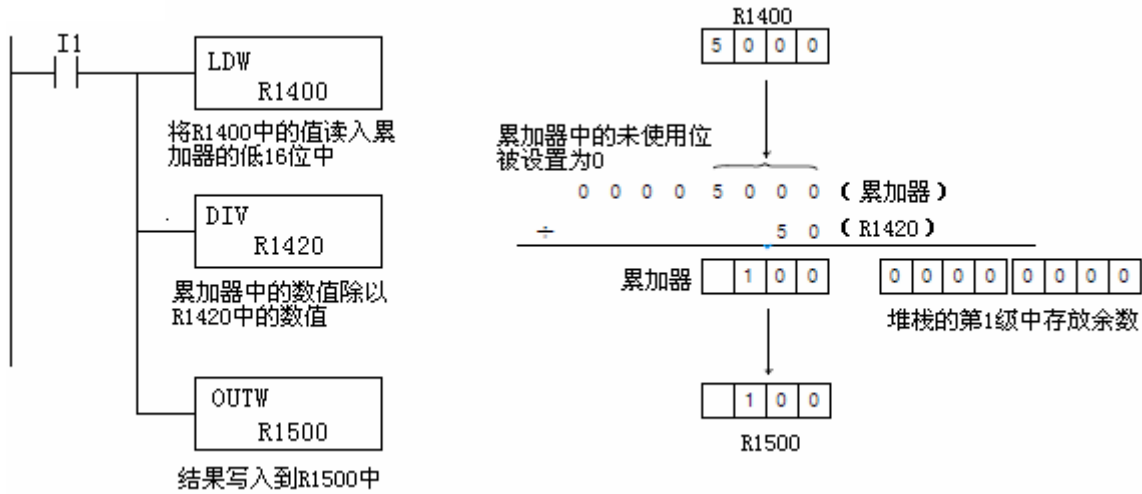
操作数类型	NK1 范围	
A	aaa	
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP53	不可进行运算处理时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，DIV 指令将累加器中低 16 位的数据除以 R1420 中的数据，OUTW 指令将累加器中的数据写入 R1500 中。



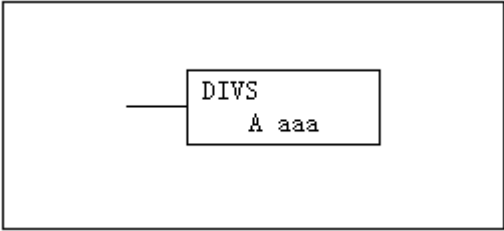
语句视图

```
LD I1
LDW R1400
DIV R1420
OUTW R1500
```

9.18 4 位 BCD 常数除法指令（DIVS）

输入 DIVS 指令是一个 16 位指令，将累加器中低 16 位中的 BCD 数除以一个 4 位（最大）BCD 常数，商存放在累加器中，余数存放在数据堆栈第 1 级中。

操作数类型	NK1 范围	
	A	aaa
常数	K	1-9999



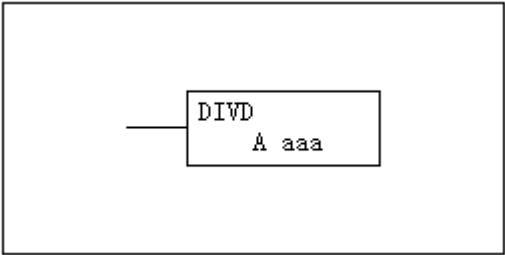
受影响的标志线圈	描述
SP53	操作数里的数据大于累加器中的数据以致不能运算时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.19 8 位 BCD 除法指令（DIVD）

输入 DIVD 指令是一个 32 位指令，将累加器中的 BCD 数除以指定的两个连续寄存器 (A aaa) 中的 BCD 数，商存放在累加器中，余数存放在数据堆栈的第 1 级中。



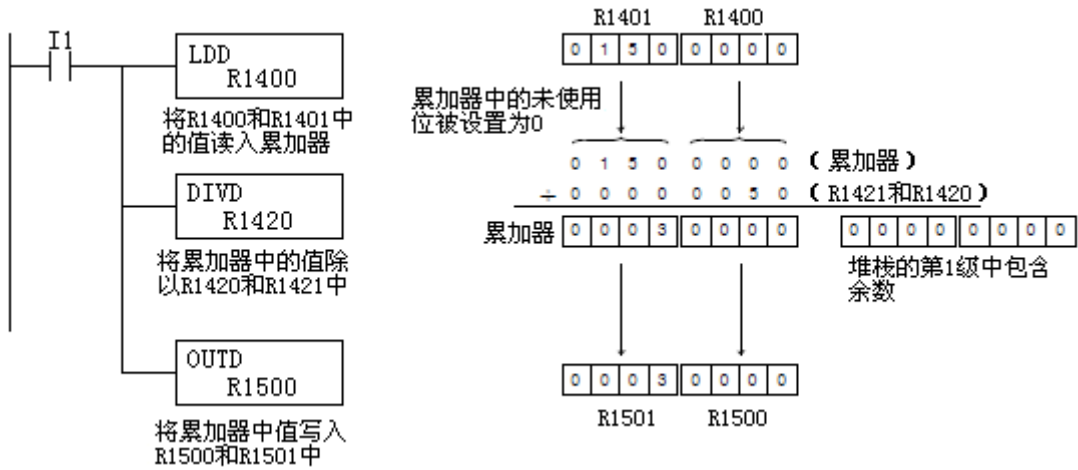
操作数类型		NK1 范围
A		aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP53	操作数里的数据大于累加器中的数据以致不能运算时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器中，DIVD 指令将累加器中的数据除以 R1420 和 R1421 中的数据，商存放在累加器中，余数存放在数据堆栈的第 1 级中，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。

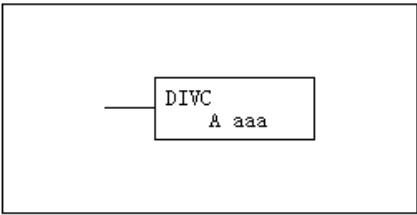


语句视图

```
LD I1
LDD R1400
DIVD R1420
OUTD R1500
```

9.20 8 位 BCD 常数除法指令 (DIVC)

输入 DIVC 指令是一个 32 位数据运算指令，将累加器中的 BCD 数除以一个 8 位 BCD 常数，结果的商部分存放在累加器中；余数部分存放在数据堆栈 (DS1) 中。



操作数类型	NK1 范围	
	A	aaa
常数	K	0-99999999

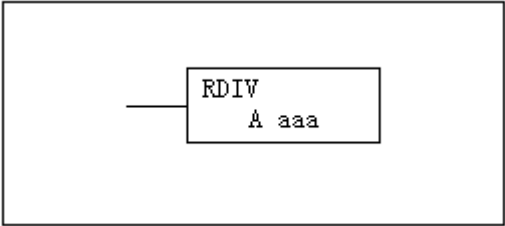
受影响的标志线圈	描述
SP53	除数为 0 时，为 ON。
SP63	指令运算导致累加器中的数据为 0 时 ON。
SP70	累加器中的数据的 bit31=1 时 ON, bit31=0 时 OFF。
SP75	BCD 运算时，运算的结果不是 BCD 时 ON。



注意： 只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.21 实数除法指令（RDIV）

输入 RDIV 指令将累加器中的实数除以两个连续寄存器中的实数。两个实数必须都是 IEEE 实数格式，结果存放在累加器中。



操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

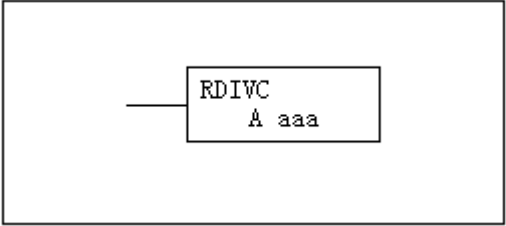
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.22 实数常数除法指令（RDIVC）

 RDIVC 指令将累加器中的实数除以一个实数常数。两个实数必须是 IEEE 实数格式。结果存放在累加器中。



操作数类型		NK1 范围
	A	aaa
常数	F	-3. 402823E+038~+3. 402823E+038

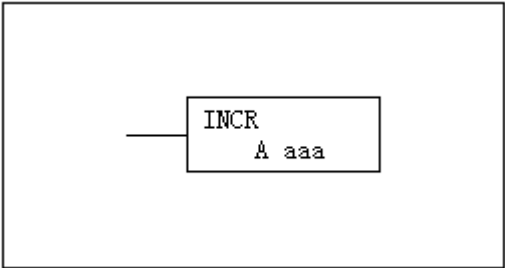
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

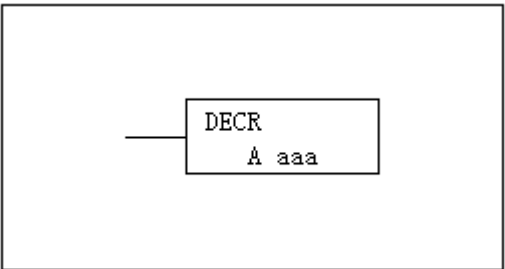
9.23 BCD 加 1 指令（INCR）

输入 INCR 指令对由操作数指定的寄存器中存储的数据进行加“1”运算，指令执行一次，就进行一次加“1”运算。



9.24 BCD 减 1 指令（DECR）

输入 DECR 指令对由操作数指定的寄存器中存储的数据进行减“1”运算，指令执行一次，就进行一次减“1”运算。



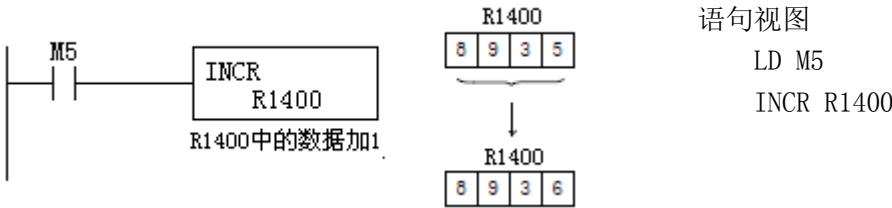
操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

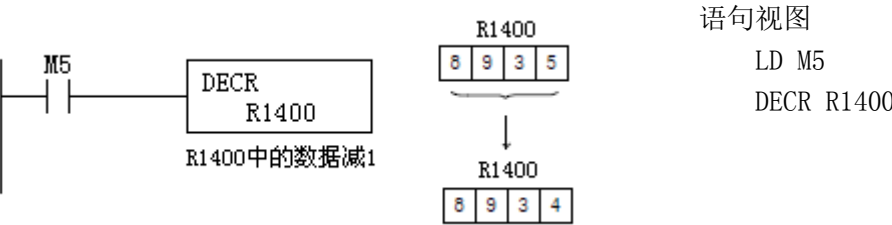


注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，每当 M5 为 ON 时，R1400 中的数据就增加 1。

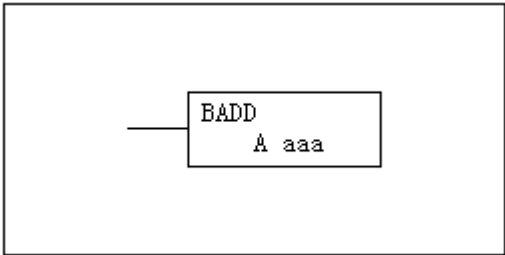


下面的例子中，每当 M5 为 ON 时，R1400 中的数据就减 1。



9.25 16 位（BIN）加法指令（BADD）

 BADD 指令是一个 16 位指令，将累加器低 16 位中的数据同指定寄存器（A aaa）中的二进制数相加，结果有可能为 32 位，存放在累加器中。



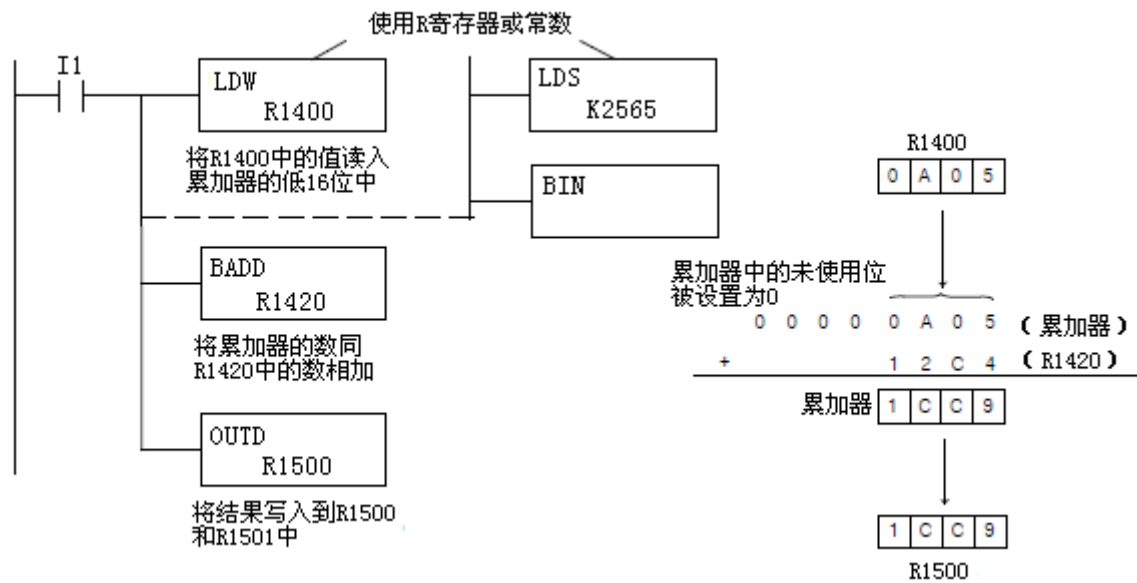
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，BADD 指令将累加器中低 16 位的数据同 R1420 中的数据相加，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。

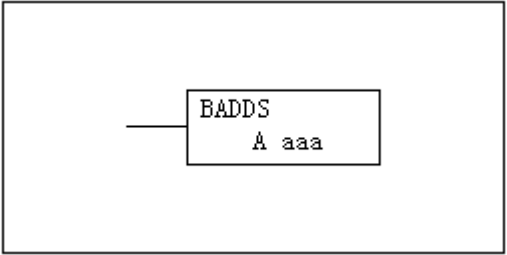


语句视图

```
LD I1
LDW R1400
BADD R1420
OUTD R1500
```

9.26 16 位常数（BIN）加法指令（BADDs）

输入 BADDs 指令是一个 16 位指令，将累加器低 16 位中的数据同 4 位常数相加，结果有可能为 32 位，存放在累加器中。



操作数类型	NK1 范围	
	A	aaa
常数	K	0~FFFF

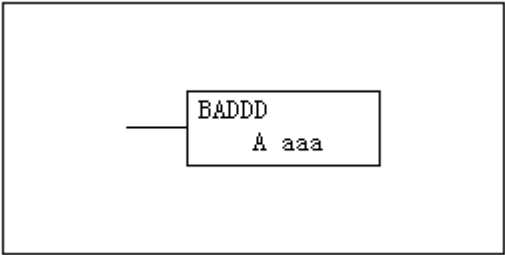
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.27 32 位（BIN）加法指令（BADDD）

输入 BADDD 指令是一个 32 位指令，将累加器中的二进制数同指定的两个连续寄存器（A aaa）中的二进制数相加，结果存放在累加器中。



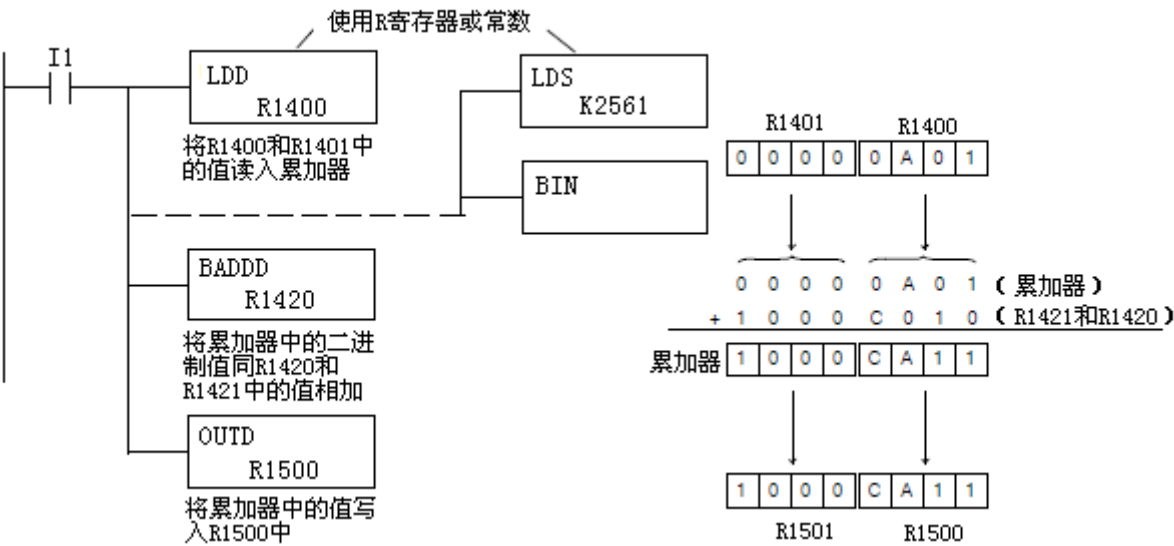
操作数类型	NK1 范围	
A	aaa	
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据装入累加器中，BADDD 指令将累加器中的二进制数据同 R1420 和 R1421 中的二进制数据相加，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。

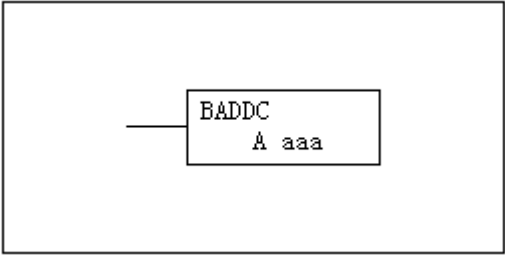


语句视图

```
LD I1
LDD R1400
BADDD R1420
OUTD R1500
```

9.28 32 位常数（BIN）加法指令（BADDc）

输入 BADDc 指令是一个 32 位指令，将累加器中的二进制数同一个 8 位（最大）常数相加，结果存放在累加器中。



操作数类型	NK1 范围	
	A	aaa
常数	K	0-FFFFFFFF

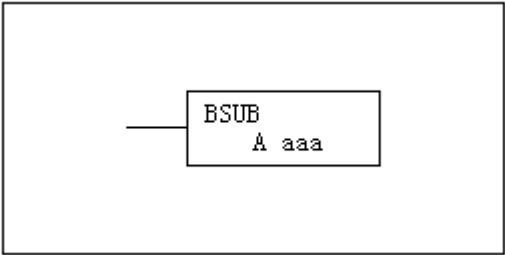
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.29 16 位（BIN）减法指令（BSUB）

输入 BSUB 指令是一个 16 位指令，将累加器低 16 位中的数据减去指定寄存器（A aaa）中的二进制数，结果存放在累加器中。



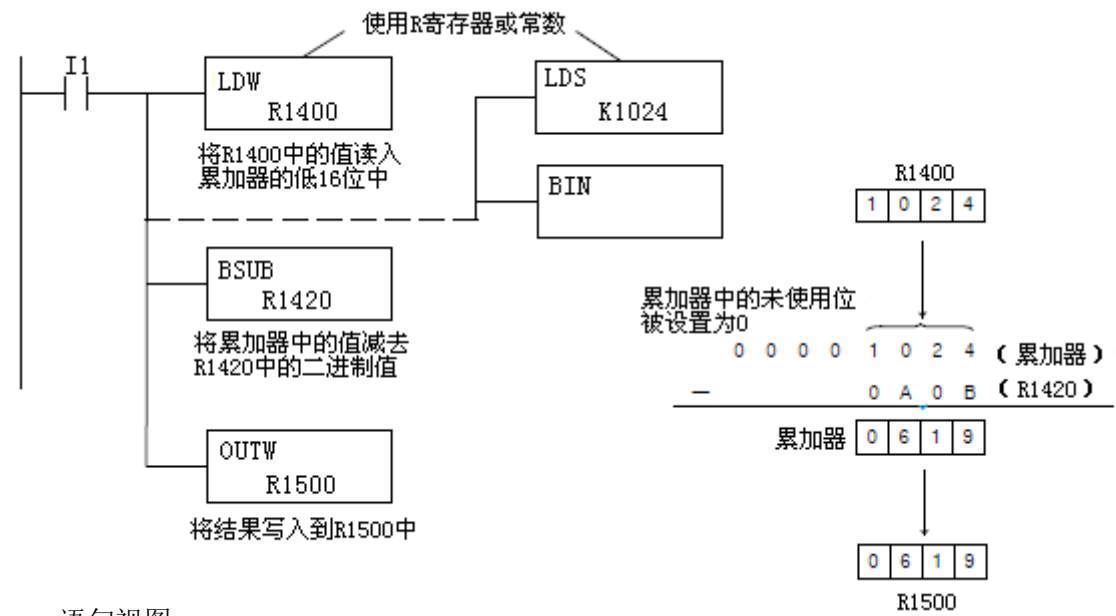
操作数类型	NK1 范围
A	aaa
R 寄存器	R 所有（附录 1）
指针	P 所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，BSUB 指令将累加器中低 16 位的数据减去 R1420 中的二进制数据，OUTW 指令将累加器中的数据写入 R1500 中。

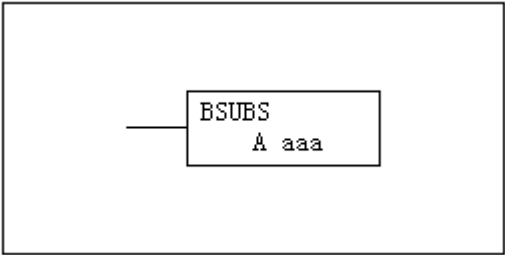


语句视图

```
LD I1
LDW R1400
BSUB R1420
OUTW R1500
```


9.30 16 位常数（BIN）减法指令（BSUBS）

输入 BSUBS 指令是一个 16 位指令，将累加器低 16 位中的数据减去一个 4 位（最大）常数，结果存放在累加器中。



操作数类型	NK1 范围	
	A	aaa
常数	K	0-FFFF

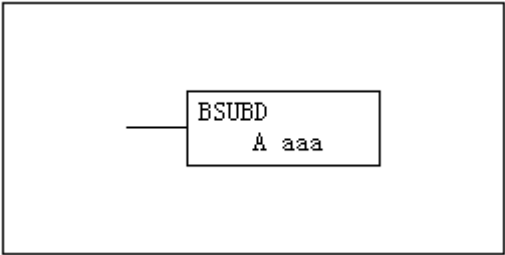
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.31 32 位（BIN）减法指令（BSUBD）

输入 BSUBD 指令是一个 32 位指令，将累加器中的二进制数减去指定的两个连续寄存器（A aaa）中的二进制数，结果存放在累加器中。



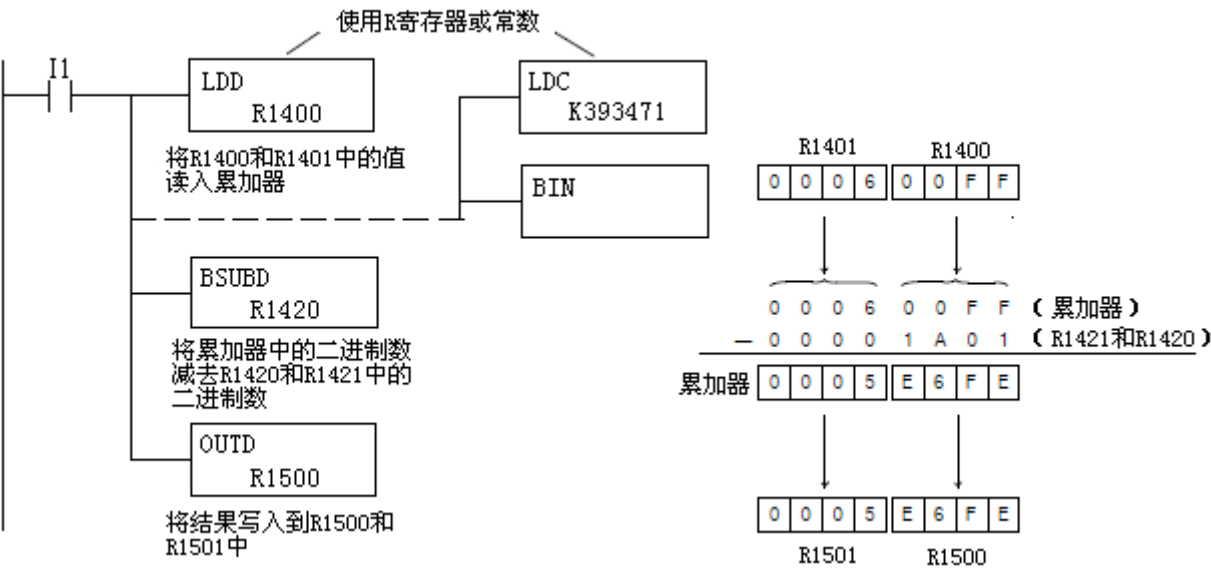
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据装入累加器中，BSUBD 指令将累加器中的二进制数减去 R1420 和 R1421 中的二进制数，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。

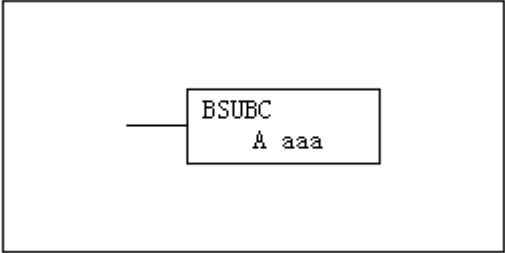


语句视图

```
LD I1
LDD R1400
BSUBD R1420
OUTD R1500
```

9.32 32 位常数（BIN）减法指令（BSUBC）

输入 BSUBC 指令是一个 32 位指令，将累加器中的二进制数减去一个 8 位（最大）常数，结果存放在累加器中。



操作数类型	NK1 范围	
	A	aaa
常数	K	0-FFFFFFFF

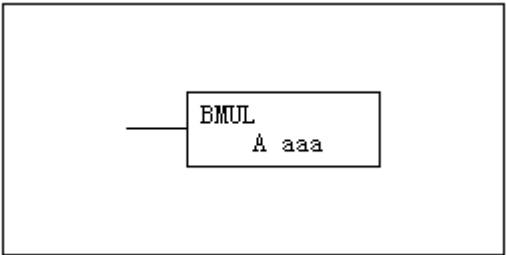
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.33 16 位（BIN）乘法指令（BMUL）

输入 BMUL 指令是一个 16 位指令，将累加器低 16 位中的数据乘以指定寄存器（A aaa）中的二进制数，结果可能会达到 32 位，存放在累加器中。



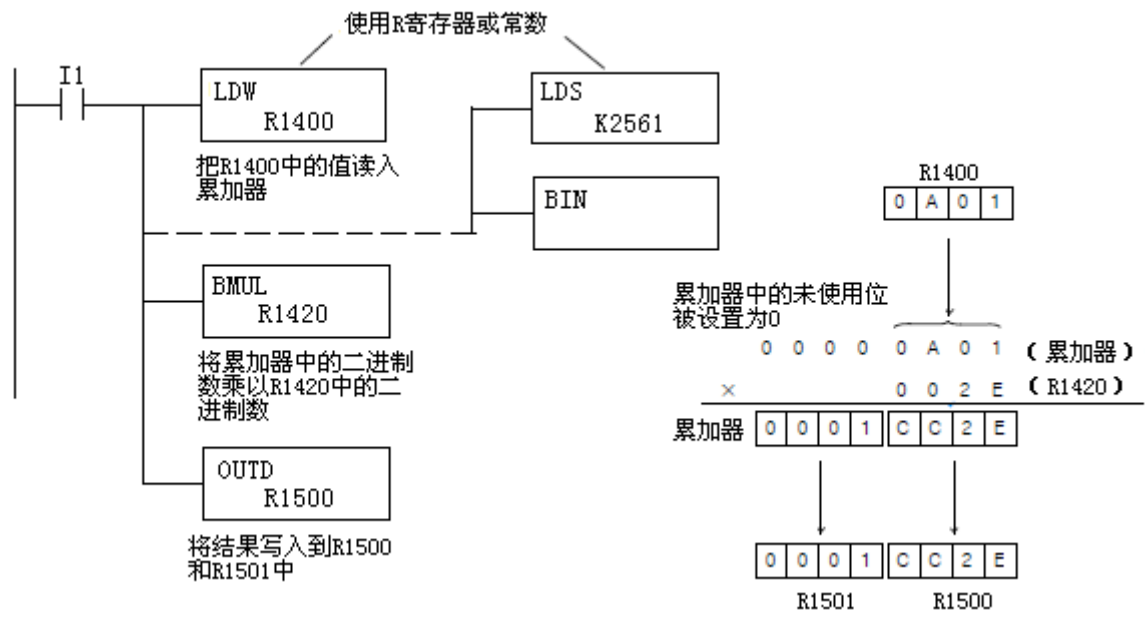
操作数类型	NK1 范围	
A	aaa	
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，BMUL 指令将累加器中低 16 位的数据乘以 R1420 中的二进制数，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。

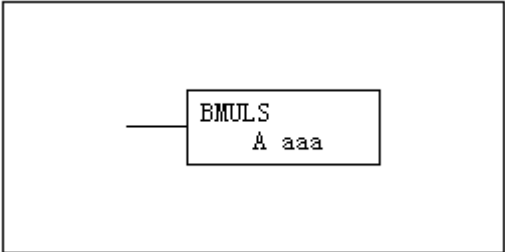


语句视图

```
LD I1
LDW R1400
BMUL R1420
OUTD R1500
```

9.34 16 位（BIN）常数乘法指令（BMULS）

输入 BMULS 指令是一个 16 位指令，将累加器低 16 位中的数据乘以一个 4 位（最大）常数，结果可能达到 32 位，存放在累加器中。



操作数类型	NK1 范围	
	A	aaa
常数	K	0~FFFF

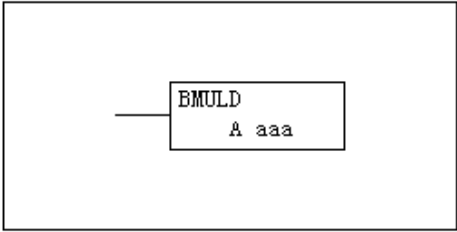
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.35 32 位（BIN）乘法指令（BMULD）

输入 BMULD R/P/K 指令是一个 32 位指令，将累加器中的二进制数乘上一个 8 位二进制数，结果存放在累加器中。当运算结果超出 32 位时，SP73=ON, 报数据溢出错误。



操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）
常数	K	0-FFFFFFFF

注：使用 R 数据类型时，不能使用每段数据寄存器的最后一个寄存器。

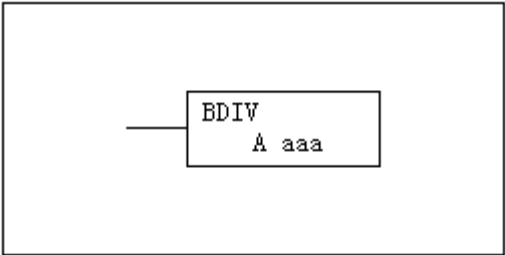
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	运算结果 ACC 的 BIT31 = 1 时 ON，BIT31 = 0 时 OFF
SP71	P 指针指定时，超出寄存器范围时为 ON
SP73	运算结果超出 32 位时 ON（Overflow）。



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.36 16 位（BIN）除法指令（BDIV）

输入 BDIV 指令是一个 16 位指令，将累加器低 16 位中的数据除以指定寄存器（A aaa）中的二进制数，商存放在累加器中，余数存放在堆栈的第 1 级中。



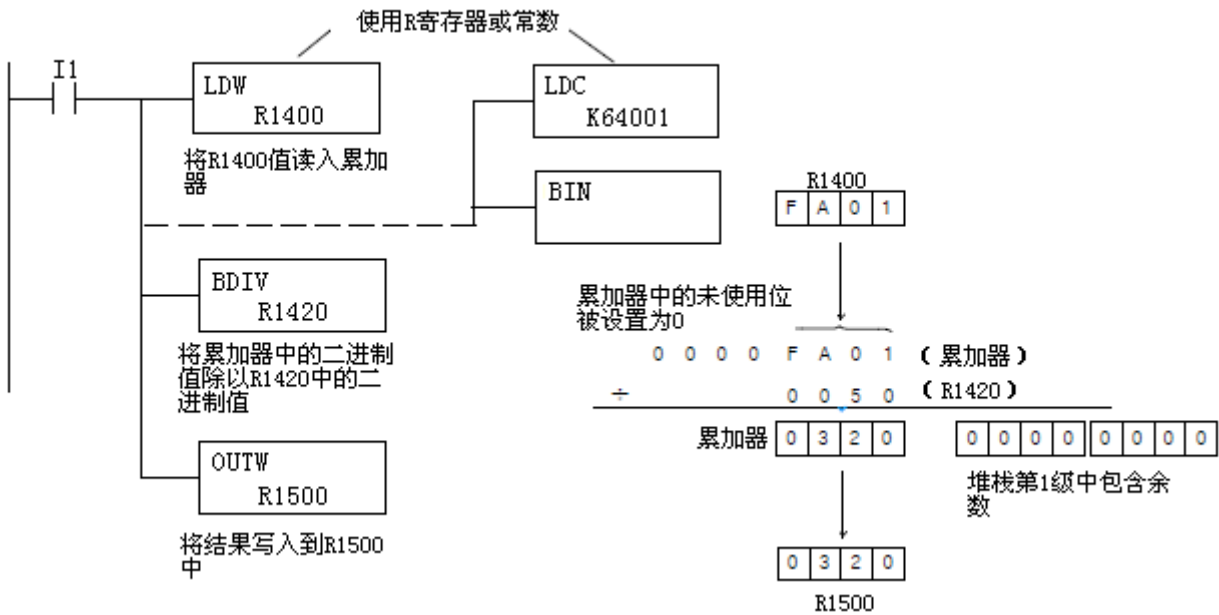
操作数类型	NK1 范围	
A	aaa	
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP53	不可进行运算处理时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，BDIV 指令将累加器中的二进制数除以 R1420 中的二进制数，OUTD 指令将累加器中的数据写入 R1500 中。

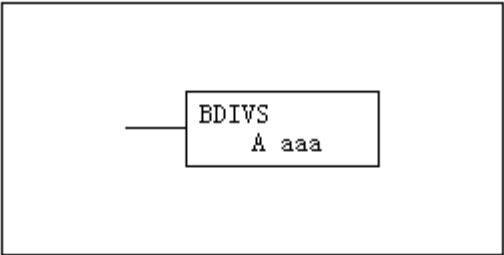


语句视图

```
LD I1
LDW R1400
BDIV R1420
OUTW R1500
```

9.37 16 位（BIN）常数除法指令（BDIVS）

 BDIVS 指令是一个 16 位指令，将累加器低 16 位中的数据除以一个 4 位（最大）二进制常数，结果存放在累加器中。



操作数类型	NK1 范围	
	A	aaa
常数	K	0-FFFF

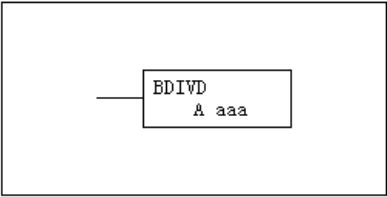
受影响的标志线圈	描述
SP53	不可进行运算处理时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.38 32 位（BIN）除法指令（BDIVD）

输入 BDIVD R/P/K 指令是一个 32 位指令，将累加器中的二进制数除以一个 8 位二进制数，结果的商部分存放在累加器中，余数部分存放在数据堆栈（DS1）中。



操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）
常数	K	0-FFFFFFFF

注：使用 R 数据类型时，不能使用每段数据寄存器的最后一个寄存器。

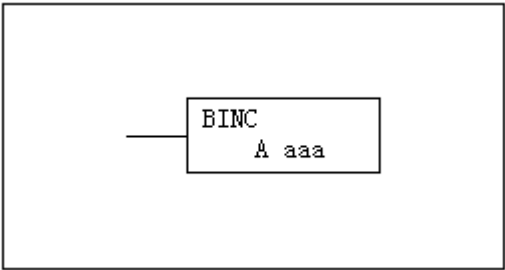
受影响的标志线圈	描述
SP53	除数为 0 时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	运算结果 ACC 的 BIT31 = 1 时 ON，BIT31 = 0 时 OFF
SP71	P 指针指定时，超出寄存器范围时为 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

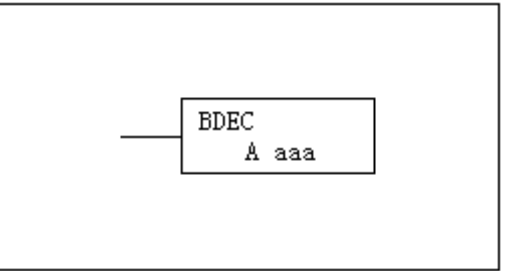
9.39 BIN 加 1 指令（BINC）

输入 BINC 指令对由操作数指定的寄存器中存储的二进制数据进行加“1”运算，指令每执行一次，就进行一次加“1”运算。



9.40 BIN 减 1 指令（BDEC）

输入 BDEC 指令对由操作数指定的寄存器中存储的二进制数据进行减“1”运算，指令每执行一次，就进行一次减“1”运算。



操作数类型	NK1 范围	
A	aaa	
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON



注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，每当 M5 为 ON 时，R1400 中的数据就增加 1。



语句视图

LD M5
BINC R1400

下面的例子中，每当 M5 为 ON 时，R1400 中的数据就减少 1。

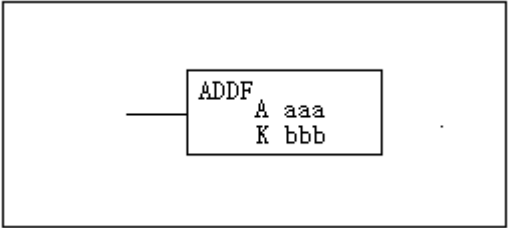


语句视图

LD M5
BDEC R1400

9.41 任意位长（BCD）加法指令（ADDF）

 ADDF 指令是一个 32 位指令，将累加器中的 BCD 数同一个指定位数的 BCD 数相加。指定的位数范围是 1-32 个连续的位，结果存放在累加器中。



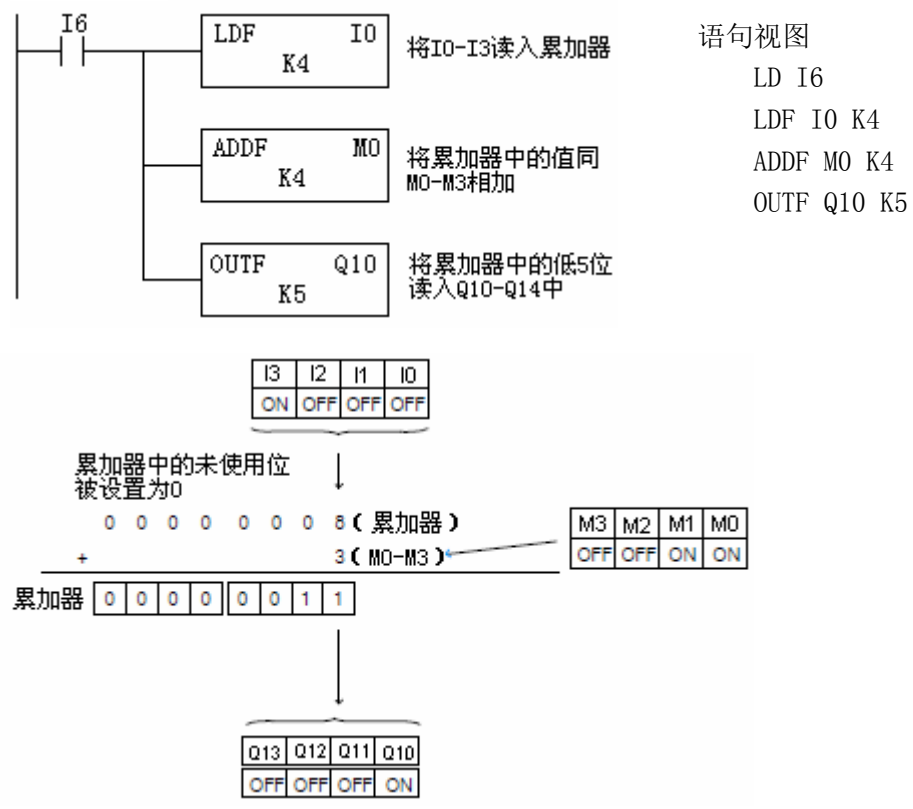
操作数类型		NK1 范围	
	A	aaa	bbb
输入	I	0-1777	-
输出	Q	0-1777	-
内部线圈	M	0-3777	-
级	S	0-1777	-
定时器	T	0-777	-
计数器	C	0-777	-
特殊线圈	SP	0-777	-
通讯 I/O	GI/GQ	0-3777	-
常数	K	-	1-32

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



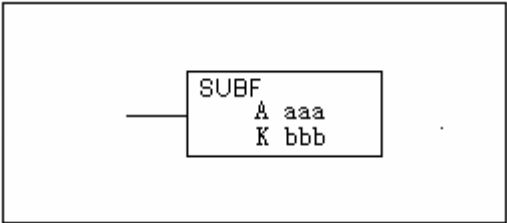
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I6 为 ON 时，LDF 指令将 I0-I3 读入累加器，ADDF 指令将累加器中的数据同 M0-M3 相加，结果存放在累加器中，OUTF 指令将累加器的低 5 位写入 Q10-Q14 中。



9.42 任意位长（BCD）减法指令（SUBF）

输入 SUBF 指令是一个 32 位指令，将累加器中的 BCD 数减去一个指定位数的 BCD 数。指定的位数范围是 1-32 个连续的位，结果存放在累加器中。



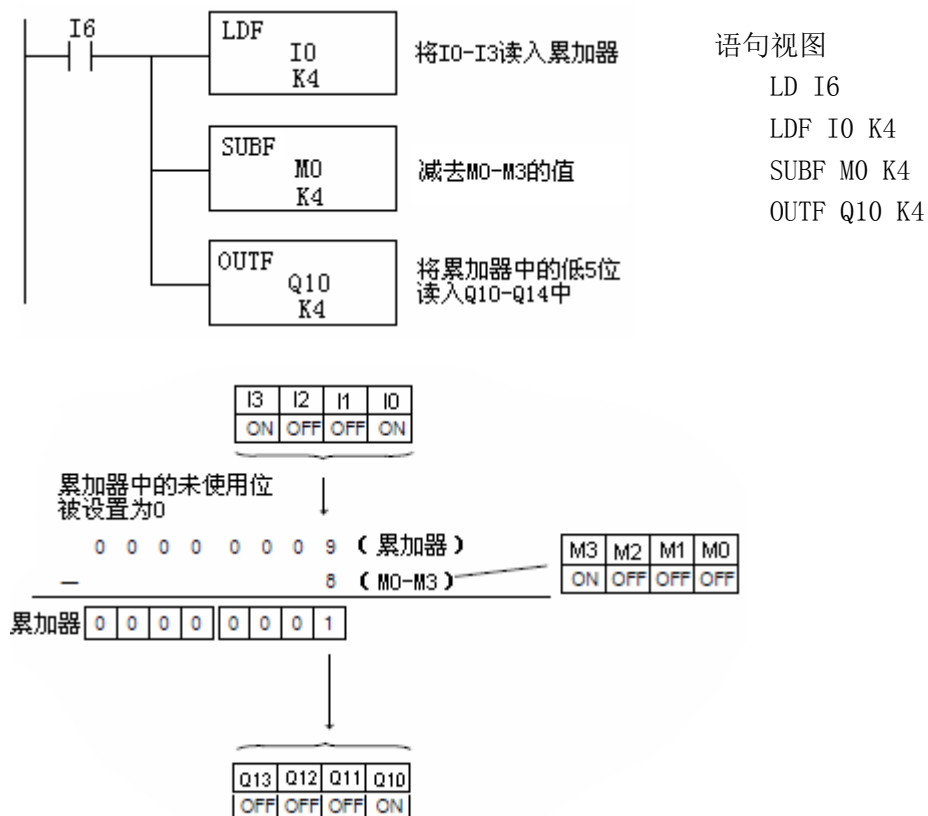
操作数类型		NK1 范围	
	A	aaa	bbb
输入	I	0-1777	-
输出	Q	0-1777	-
内部线圈	M	0-3777	-
级	S	0-1777	-
定时器	T	0-777	-
计数器	C	0-777	-
特殊线圈	SP	0-777	-
通讯 I/O	GI/GQ	0-3777	-
常数	K	-	1-32

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



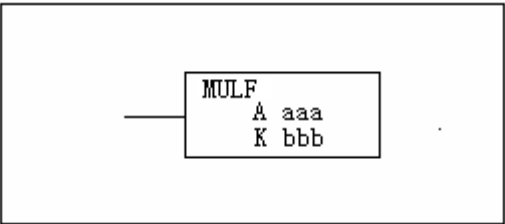
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I6 为 ON 时，LDF 指令将 I0-I3 读入累加器，SUBF 指令将累加器中的数据减去 M0-M3 的值，结果存放在累加器中，OUTF 指令将累加器的低 4 位写入 Q10-Q13 中。



9.43 任意位长（BCD）乘法指令（MULF）

输入 MULF 指令是一个 16 位指令，将累加器中的 BCD 数乘以一个指定位数的 BCD 数。指定的位数范围是 1-16 个连续的位，结果存放在累加器中。



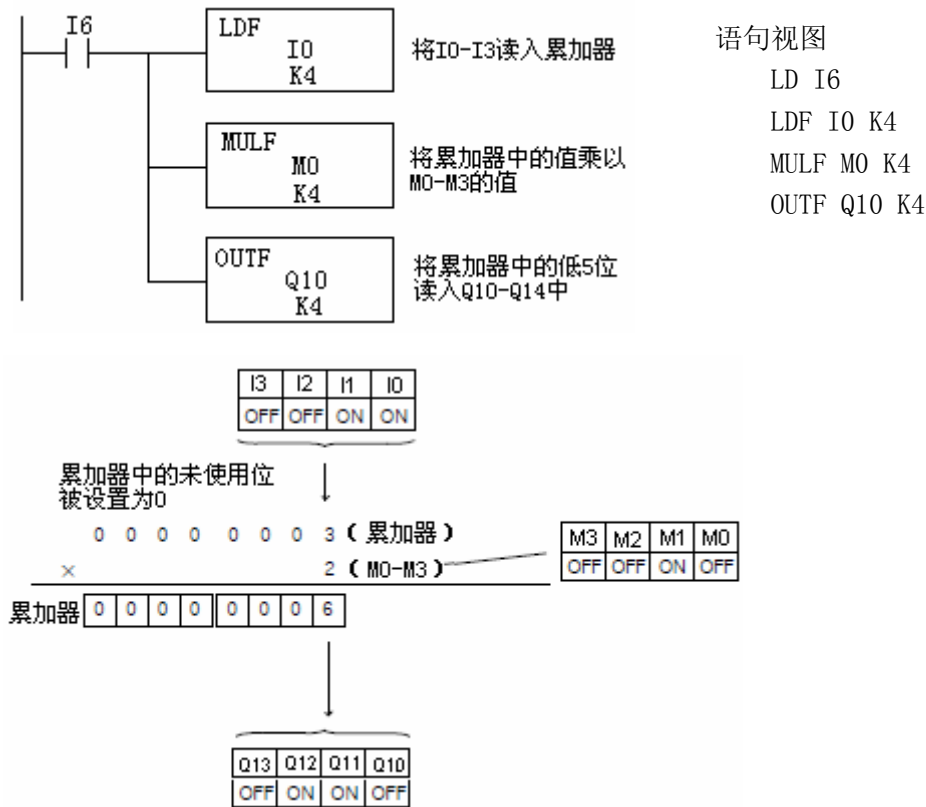
操作数类型		NK1 范围	
	A	aaa	bbb
输入	I	0-1777	-
输出	Q	0-1777	-
内部线圈	M	0-3777	-
级	S	0-1777	-
定时器	T	0-777	-
计数器	C	0-777	-
特殊线圈	SP	0-777	-
通讯 I/O	GI/GQ	0-3777	-
常数	K	-	1-16

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



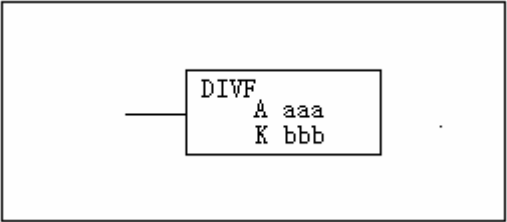
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I6 为 ON 时，LDF 指令将 I0-I3 读入累加器，MULF 指令将累加器中的数据乘以 M0-M3 的值，结果存放在累加器中，OUTF 指令将累加器的低 4 位写入 Q10-Q13 中。



9.44 任意位长（BCD）除法指令（DIVF）

输入 DIVF 指令是一个 16 位指令，将累加器中的 BCD 数除以一个指定位数的 BCD 数。指定的位数范围是 1-16 个连续的位，商存放在累加器中，余数存放在数据堆栈的第 1 级中。



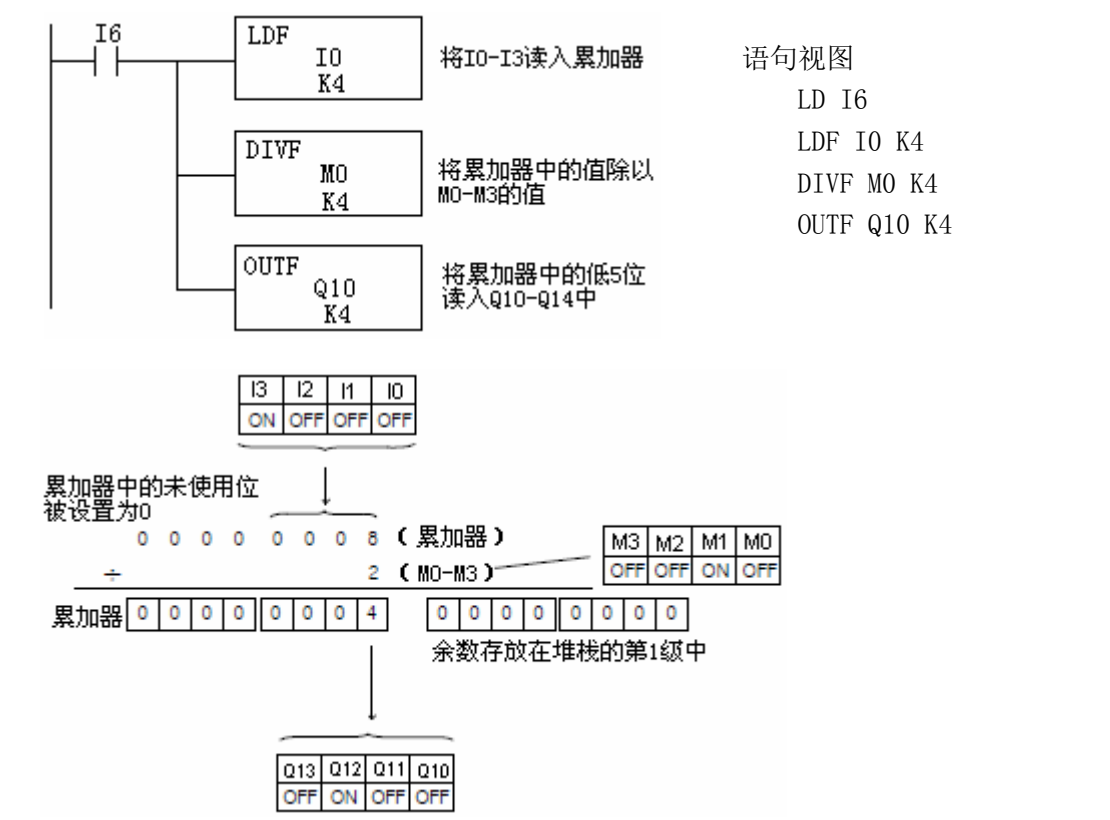
操作数类型		NK1 范围	
	A	aaa	bbb
输入	I	0-1777	-
输出	Q	0-1777	-
内部线圈	M	0-3777	-
级	S	0-1777	-
定时器	T	0-777	-
计数器	C	0-777	-
特殊线圈	SP	0-777	-
通讯 I/O	GI/GQ	0-3777	-
常数	K	-	1-16

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



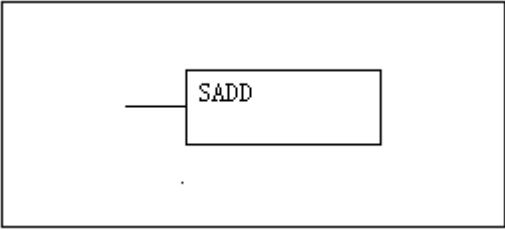
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I6 为 ON 时，LDF 指令将 I0-I3 读入累加器，DIVF 指令将累加器中的数据除以 M0-M3 的值，结果存放在累加器中，OUTF 指令将累加器的低 4 位写入 Q10-Q13 中。



9.45 堆栈加法指令（SADD）

 SADD 指令是一个 32 位指令，将累加器中的 BCD 数加上数据堆栈第 1 级中的 BCD 数，结果存放在累加器中。堆栈第 1 级中的数据移出，其他级均向上移动一级。

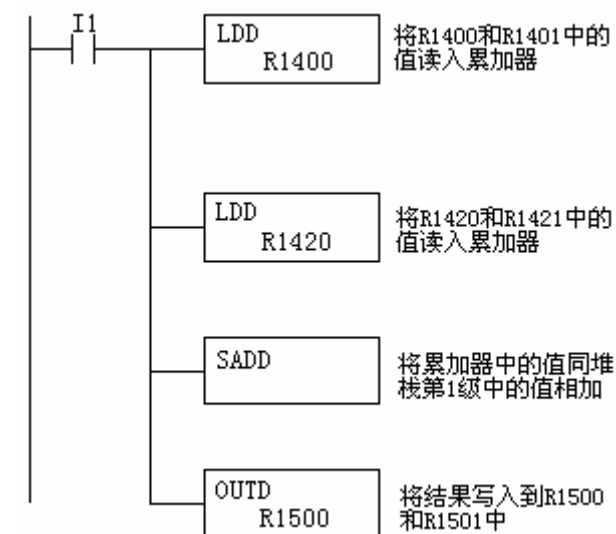


受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



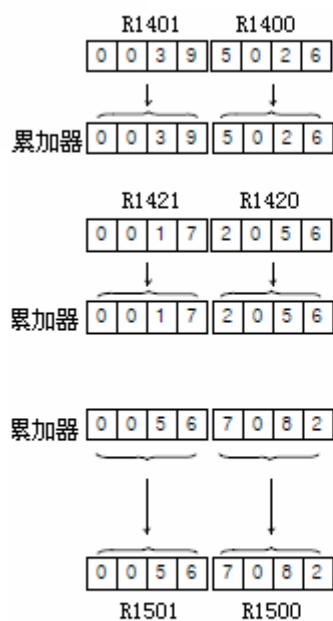
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，第一个 LDD 指令将 R1400 和 R1401 中的数据读入累加器，第二个 LDD 指令将 R1420 和 R1421 中的数据读入累加器，累加器中的原有数据被压入数据堆栈的第 1 级中。SADD 指令将累加器中的数据加上堆栈第 1 级中的数据，结果写入到 R1500 和 R1501 中。



语句视图

```
LD I1
LDD R1400
LDD R1420
SADD
OUTD R1500
```

数据堆栈
第一个LDD指令之后

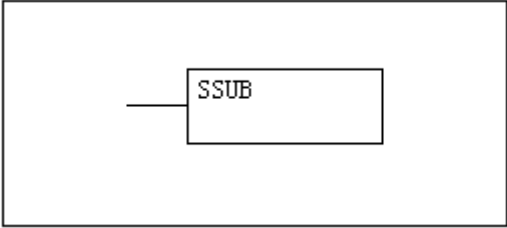
第1级	X	X	X	X	X	X	X	X
第2级	X	X	X	X	X	X	X	X
第3级	X	X	X	X	X	X	X	X
第4级	X	X	X	X	X	X	X	X
第5级	X	X	X	X	X	X	X	X
第6级	X	X	X	X	X	X	X	X
第7级	X	X	X	X	X	X	X	X
第8级	X	X	X	X	X	X	X	X

数据堆栈
第二个LDD指令之后

第1级	0	0	3	9	5	0	2	6
第2级	X	X	X	X	X	X	X	X
第3级	X	X	X	X	X	X	X	X
第4级	X	X	X	X	X	X	X	X
第5级	X	X	X	X	X	X	X	X
第6级	X	X	X	X	X	X	X	X
第7级	X	X	X	X	X	X	X	X
第8级	X	X	X	X	X	X	X	X

9.46 堆栈减法指令（SSUB）

输入 SSUB 指令是一个 32 位指令，将累加器中的 BCD 数减去数据堆栈第 1 级中的 BCD 数，结果存放在累加器中。堆栈第 1 级中的数据移出，其他级均向上移动一级。

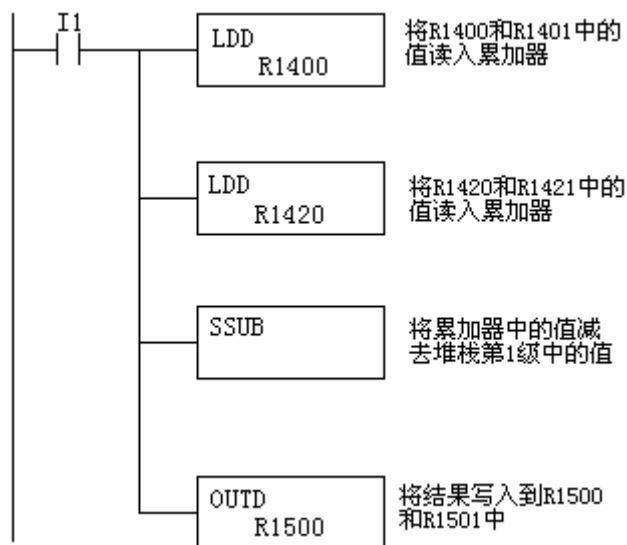


受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



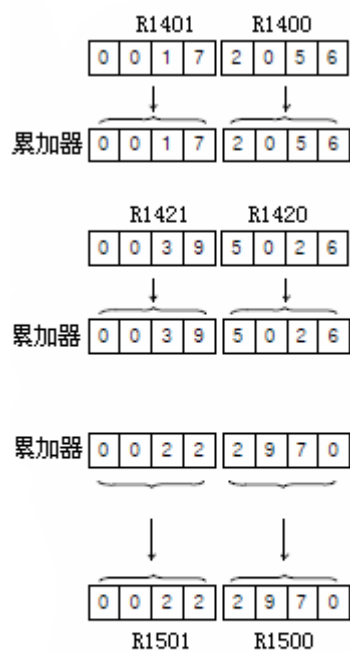
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，第一个 LDD 指令将 R1400 和 R1401 中的数据读入累加器，第二个 LDD 指令将 R1420 和 R1421 中的数据读入累加器，累加器中的原有数据被压入数据堆栈的第 1 级中。SSUB 指令将累加器中的数据减去堆栈第 1 级中的数据，结果写入到 R1500 和 R1501 中。



语句视图

```
LD I1
LDD R1400
LDD R1420
SSUB
OUTD R1500
```

数据堆栈
第一个LDD指令之后

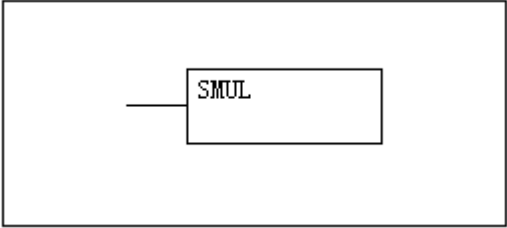
第1级	X	X	X	X	X	X	X
第2级	X	X	X	X	X	X	X
第3级	X	X	X	X	X	X	X
第4级	X	X	X	X	X	X	X
第5级	X	X	X	X	X	X	X
第6级	X	X	X	X	X	X	X
第7级	X	X	X	X	X	X	X
第8级	X	X	X	X	X	X	X

数据堆栈
第二个LDD指令之后

第1级	0	0	1	7	2	0	5	6
第2级	X	X	X	X	X	X	X	X
第3级	X	X	X	X	X	X	X	X
第4级	X	X	X	X	X	X	X	X
第5级	X	X	X	X	X	X	X	X
第6级	X	X	X	X	X	X	X	X
第7级	X	X	X	X	X	X	X	X
第8级	X	X	X	X	X	X	X	X

9.47 堆栈乘法指令（SMUL）

输入 SMUL 指令是一个 16 位指令，将累加器中的 4 位 BCD 数乘以数据堆栈第 1 级中的 4 位 BCD 数，结果存放在累加器中。堆栈第 1 级中的数据移出，其他级均向上移动一级。

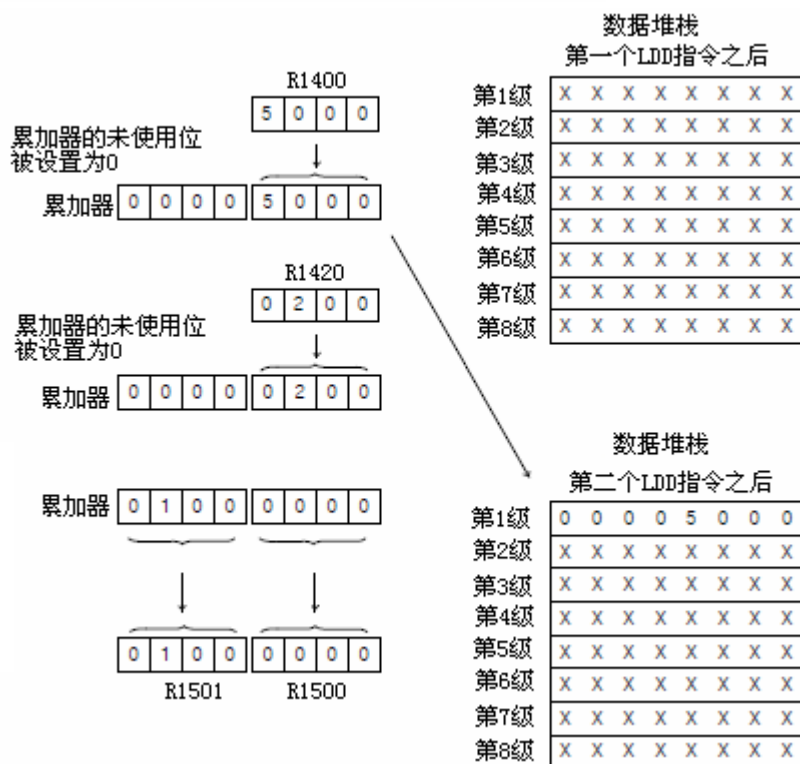
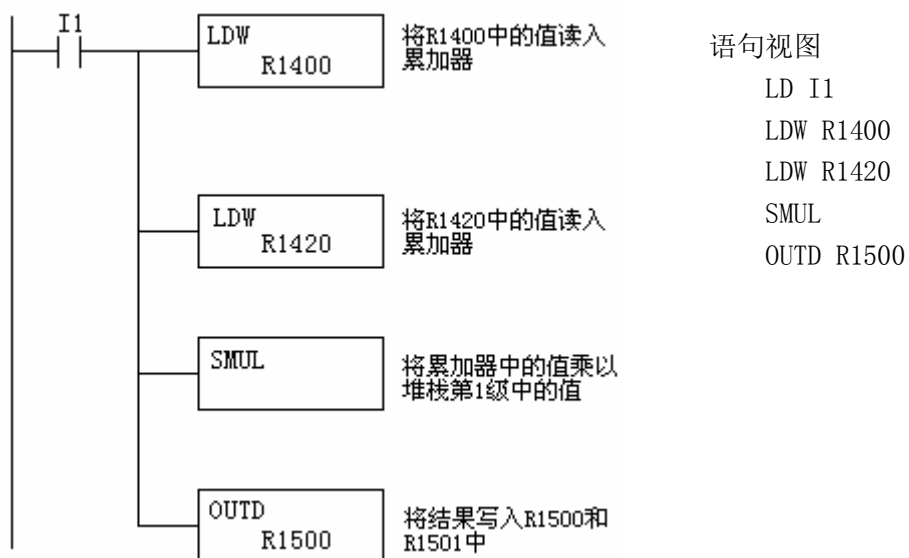


受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



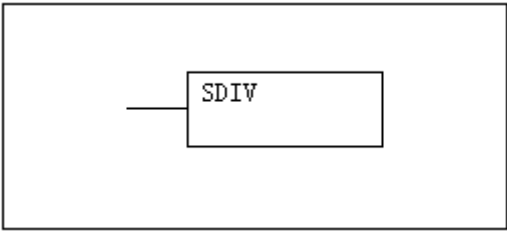
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，第一个 LDW 指令将 R1400 中的数据读入累加器，第二个 LDW 指令将 R1420 中的数据读入累加器，累加器中的原有数据被压入数据堆栈的第 1 级中。SMUL 指令将累加器中的数据乘以堆栈第 1 级中的数据，结果写入到 R1500 和 R1501 中。



9.48 堆栈除法指令（SDIV）

输入 SDIV 指令是一个 32 位指令，将累加器中的 8 位 BCD 数除以数据堆栈第 1 级中的 4 位 BCD 数，商存放在累加器中，余数存放在堆栈第 1 级中。

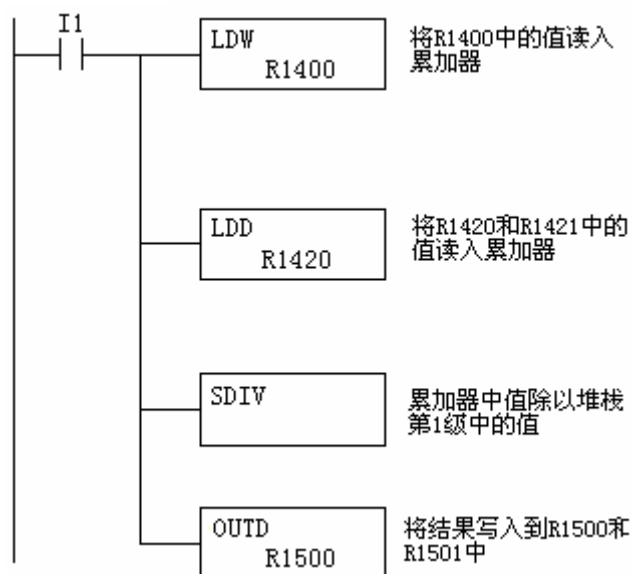


受影响的标志线圈	描述
SP53	操作数里的数据大于累加器中的数据以致不能运算时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON



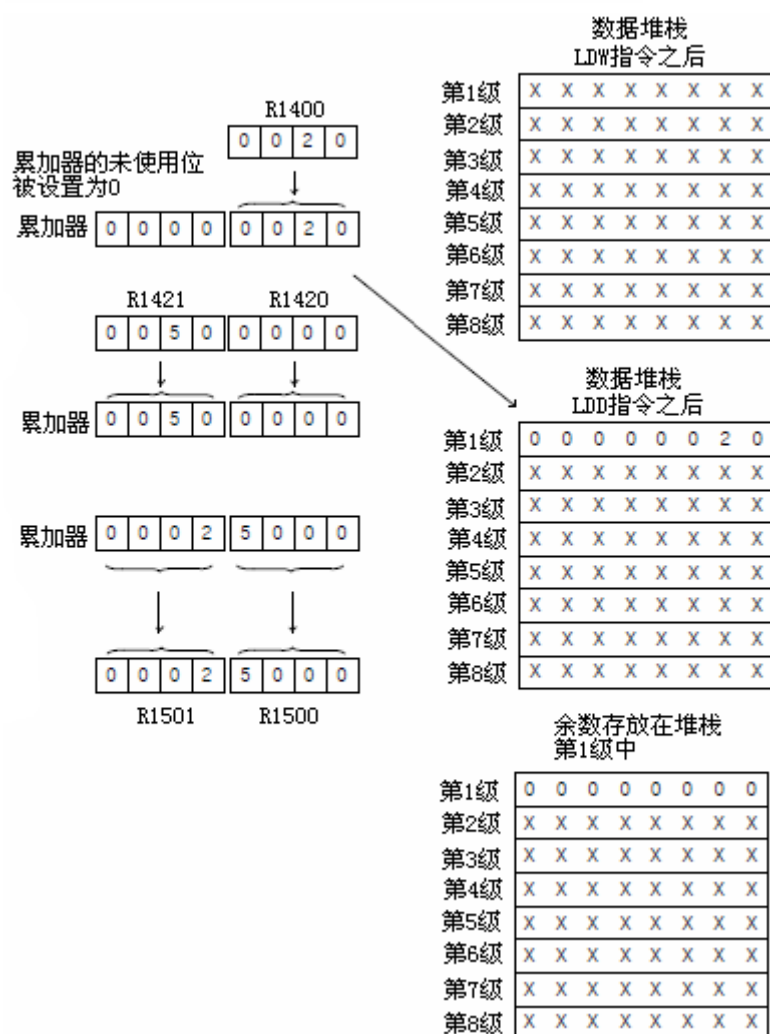
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，LDD 指令将 R1420 和 R1421 中的数据读入累加器，累加器中的原有数据被压入数据堆栈的第 1 级中。SDIV 指令将累加器中的数据除以堆栈第 1 级中的数据，结果写入到 R1500 和 R1501 中。



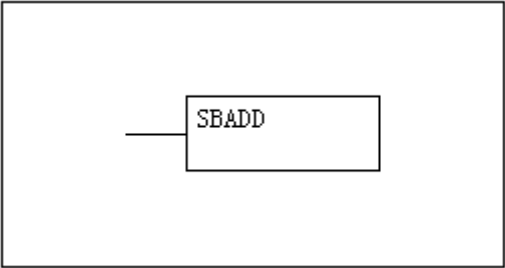
语句视图

```
LD I1
LDW R1400
LDD R1420
SDIV
OUTD R1500
```



9.49 堆栈二进制加法指令（SBADD）

输入 SBADD 指令是一个 32 位指令，将累加器中的二进制数加上堆栈第 1 级中的二进制数，堆栈第 1 级的数据移出，其他级均向上移动一级。

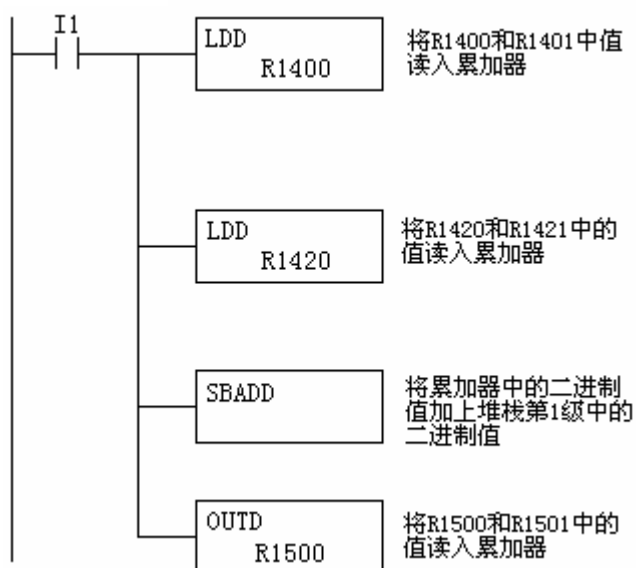


受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON



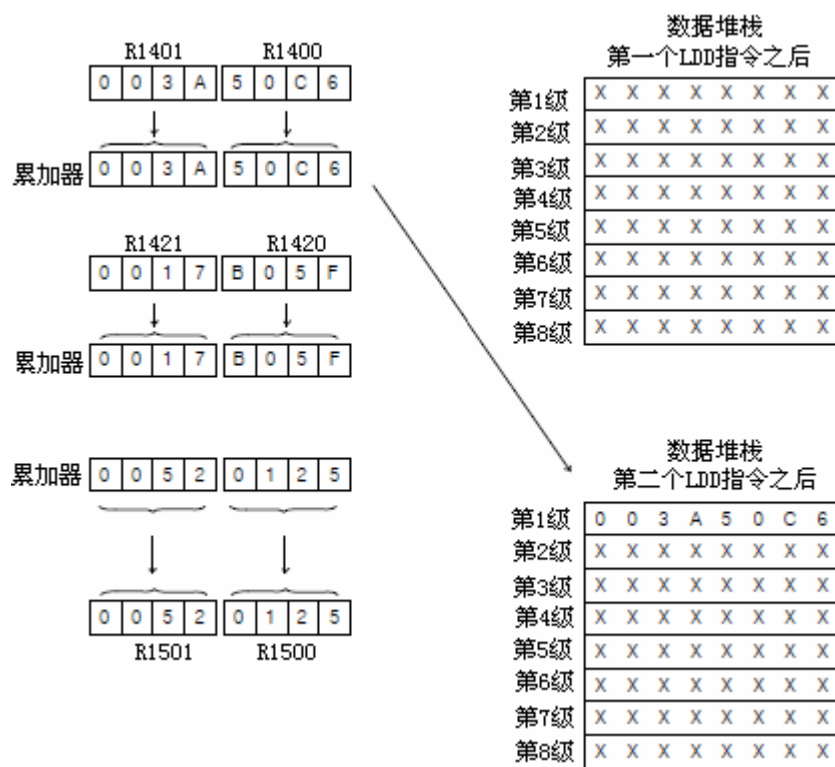
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，第一个 LDD 指令将 R1400 和 R1401 中的数据读入累加器，第二个 LDD 指令将 R1420 和 R1421 中的数据读入累加器，累加器原有数据被压入数据堆栈。SBADD 指令将累加器中的二进制数据加上堆栈第 1 级中的二进制数据，结果写入到 R1500 和 R1501 中。



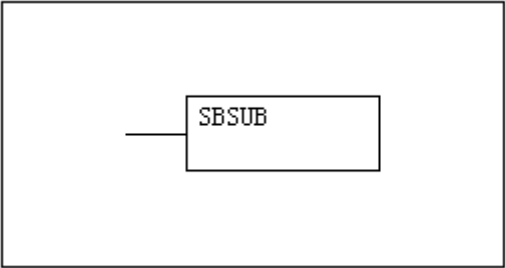
语句视图

```
LD I1
LDD R1400
LDD R1420
SBADD
OUTD R1500
```



9. 50 堆栈二进制减法指令（SBSUB）

 SBSUB 指令是一个 32 位指令，将累加器中的二进制数据减去堆栈第 1 级中的二进制数据，结果存放在累加器中。堆栈第 1 级的数据移出，其他级均向上移动一级。

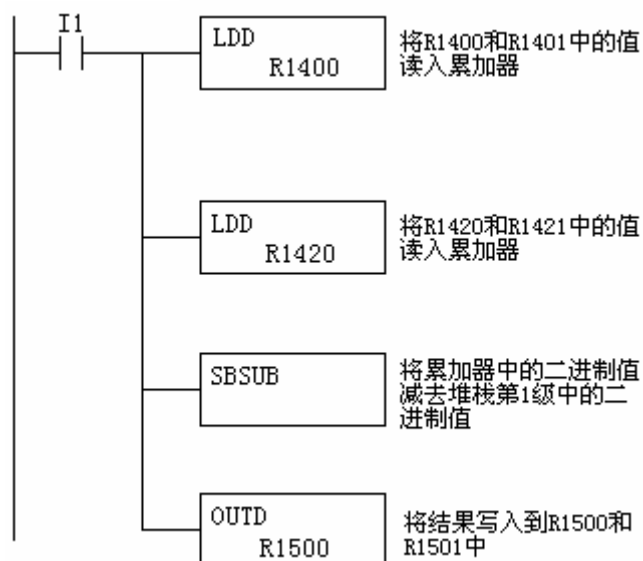


受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON



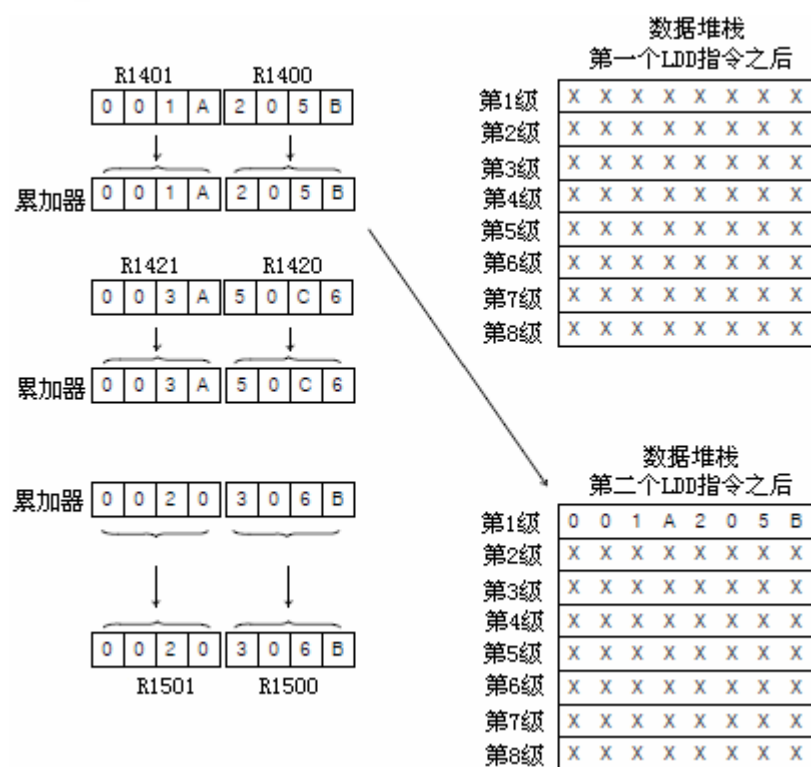
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，第一个 LDD 指令将 R1400 和 R1401 中的数据读入累加器，第二个 LDD 指令将 R1420 和 R1421 中的数据读入累加器，累加器原有数据被压入数据堆栈。SBSUB 指令将累加器中的二进制数据减去堆栈第 1 级中的二进制数据，结果写入到 R1500 和 R1501 中。



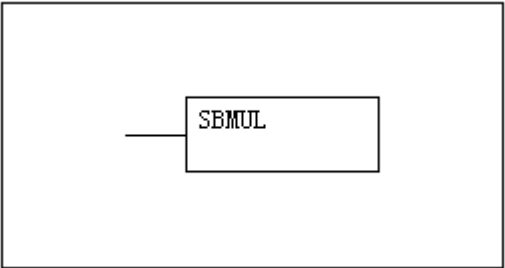
语句视图

```
LD I1
LDD R1400
LDD R1420
SBSUB
OUTD R1500
```



9.51 堆栈二进制乘法指令（SBMUL）

输入 SBMUL 指令是一个 16 位指令，将累加器中的二进制数据乘以堆栈第 1 级中的二进制数据，结果可能是 32 位（最大），存放在累加器中。堆栈第 1 级的数据移出，其他级均向上移动一级。

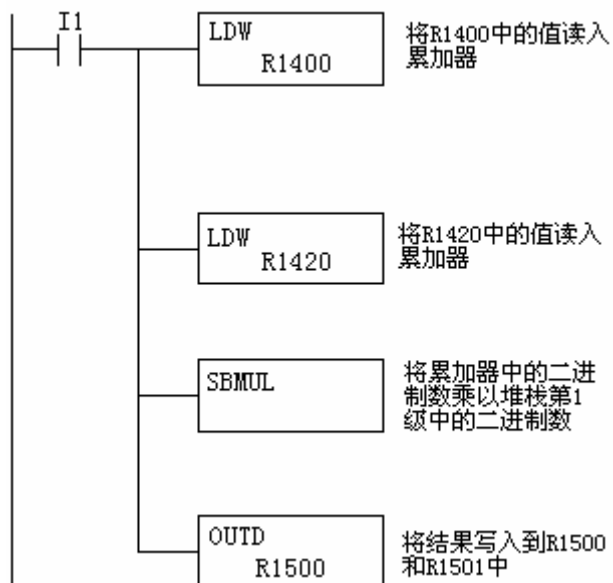


受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON



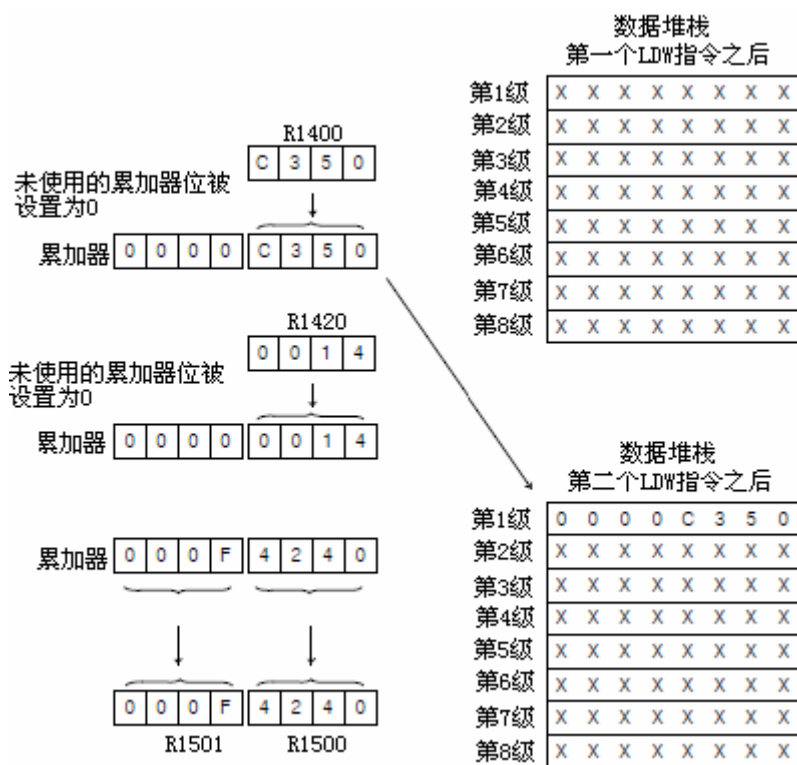
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，第一个 LDW 指令将 R1400 中的数据读入累加器，第二个 LDW 指令将 R1420 中的数据读入累加器，累加器原有数据被压入数据堆栈。SBMUL 指令将累加器中的二进制数据乘以堆栈第 1 级中的二进制数据，结果写入到 R1500 和 R1501 中。



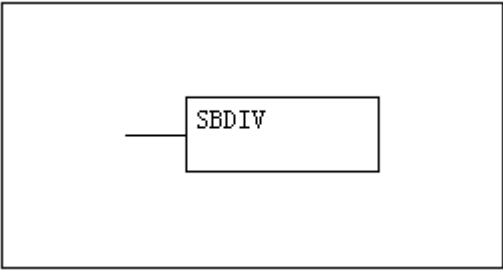
语句视图

```
LD I1
LDW R1400
LDW R1420
SBMUL
OUTD R1500
```



9.52 堆栈二进制除法指令（SBDIV）

输入 SBDIV 指令是一个 32 位指令，将累加器中的 32 位二进制数据除以堆栈第 1 级中的 16 位二进制数据，商存放在累加器中，余数存放在堆栈第 1 级中。

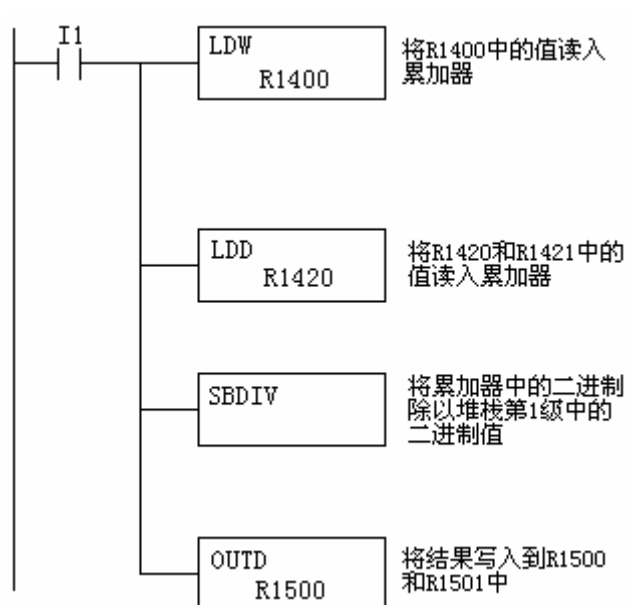


受影响的标志线圈	描述
SP53	不可进行运算处理时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON



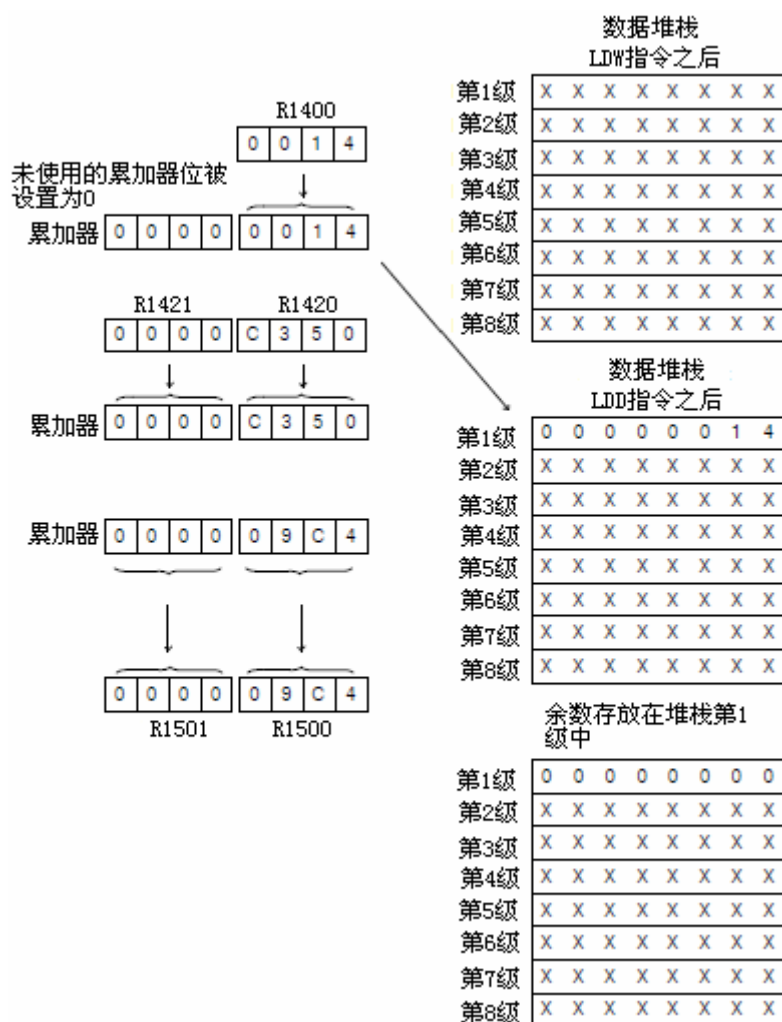
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，LDD 指令将 R1420 和 R1421 中的数据读入累加器，累加器原有数据被压入数据堆栈。SBDIV 指令将累加器中的二进制数除以堆栈第 1 级中的二进制数，结果写入到 R1500 和 R1501 中。



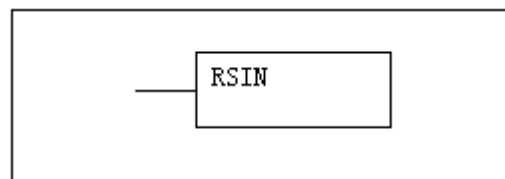
语句视图

```
LD I1
LDW R1400
LDD R1420
SBDIV
OUTD R1500
```



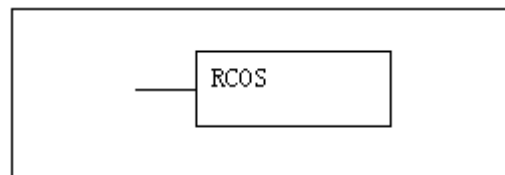
9.53 实数正弦指令（RSIN）

输入 RSIN 指令求出累加器中实数的正弦函数值，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



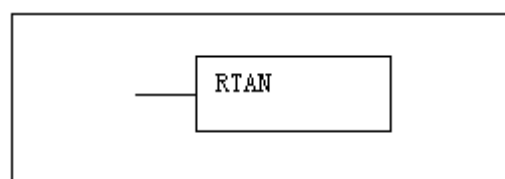
9.54 实数余弦指令（RCOS）

输入 RCOS 指令求出累加器中实数的余弦函数值，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



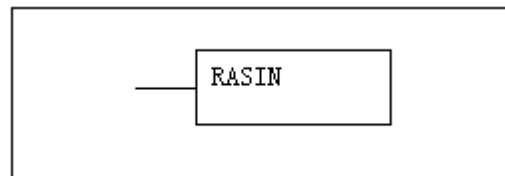
9.55 实数正切指令（RTAN）

输入 RTAN 指令求出累加器中实数的正切函数值，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



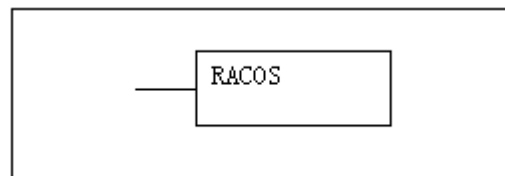
9.56 实数反正弦指令（RASIN）

输入 RASIN 指令将累加器中的正弦函数值转换成角度值，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



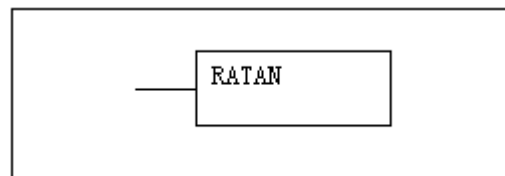
9.57 实数反余弦指令（RACOS）

输入 RACOS 指令将累加器中的余弦函数值转换成角度值，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



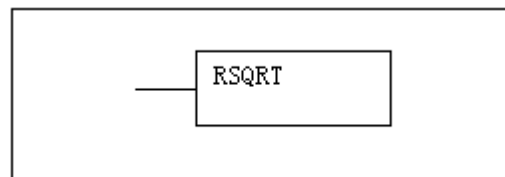
9.58 实数反正切指令（RATAN）

输入 RATAN 指令将累加器中的正切函数值转换成角度值，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



9.59 实数平方根指令（RSQRT）

输入 RSQRT 指令对累加器中的实数求平方根，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



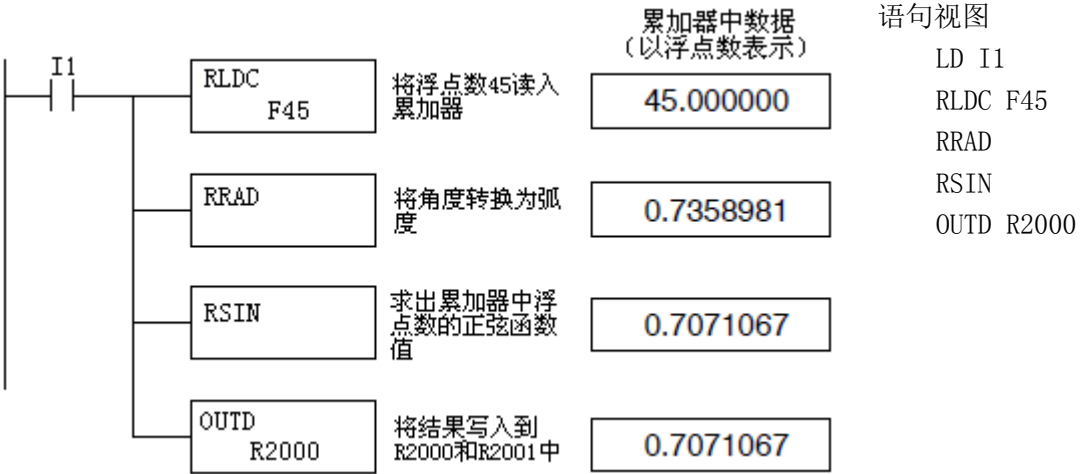
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

数学运算标志线圈	描述
SP53	不可进行运算处理时 ON



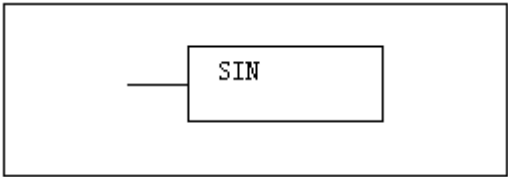
注意：平方根指令在有些场合很有用，比如将孔板测得的流量数据求取平方根用于 PID 回路的 PV 值，但是 PID 回路内置平方根功能，如果勾选，就不需再用平方根指令。

下面的例子中，求出 45 度的正弦函数值，由于这些高级功能指令的操作数是实数，因此使用 RLDC 指令读入实数。三角函数指令的运算对象须是弧度，因此用 RRAD 指令将角度转化为弧度，RSIN 指令求出正弦函数值，OUTD 指令将结果写入到 R2000 和 R2001 中，由于结果是 32 位的长度，因此需要双字节输出指令。



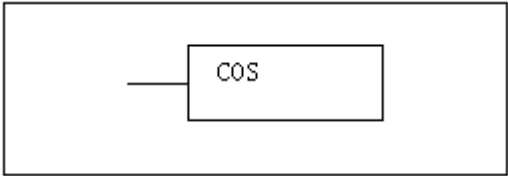
9.60 BCD 数正弦指令（SIN）

输入 SIN 指令求出累加器中 BCD 数的正弦函数值，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。



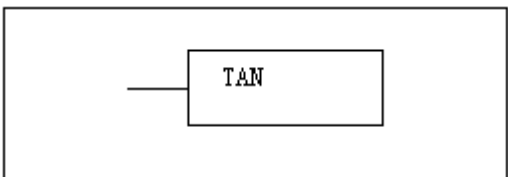
9.61 BCD 数余弦指令（COS）

输入 COS 指令求出累加器中 BCD 数的余弦函数值，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。



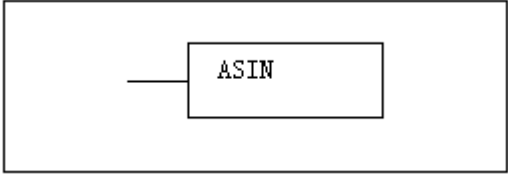
9.62 BCD 数正切指令（TAN）

输入 TAN 指令求出累加器中 BCD 数的正切函数值，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。



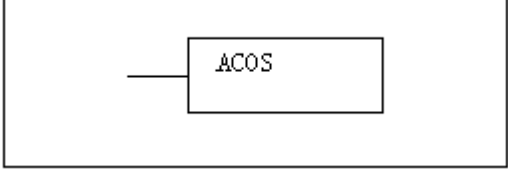
9.63 BCD 数反正弦指令（ASIN）

输入 ASIN 指令将累加器中的正弦函数值转换成角度值，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。



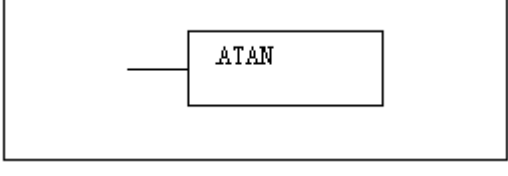
9.64 BCD 数反余弦指令（ACOS）

输入 ACOS 指令将累加器中的余弦函数值转换成角度值，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。



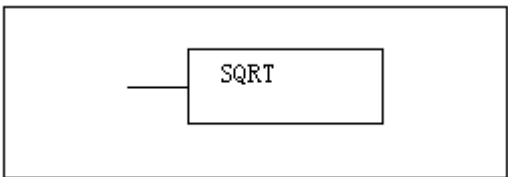
9.65 BCD 数反正切指令（ATAN）

输入 ATAN 指令将累加器中的正切函数值转换成角度值，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。



9.66 BCD 数平方根指令（SQRT）

输入 SQRT 指令对累加器中的 BCD 数求平方根，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。

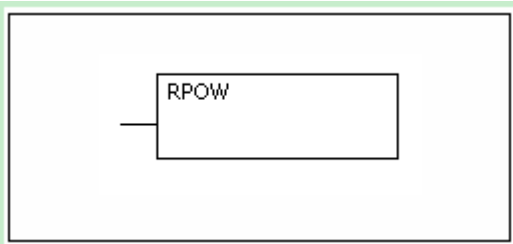


受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

受影响的数学运算标志线圈	描述
SP53	不可进行运算处理时 ON

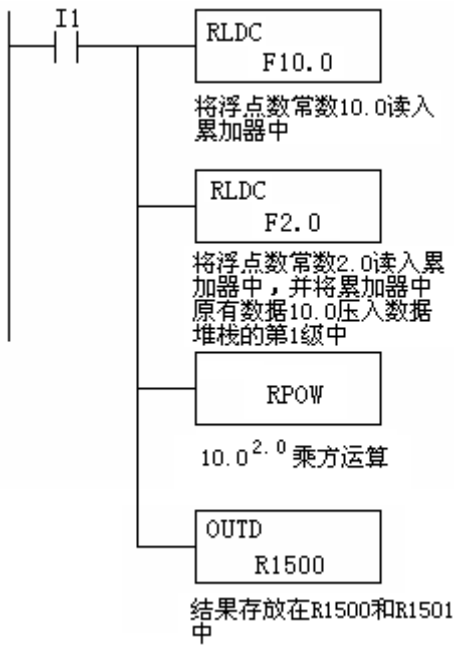
9.67 幂指令（RPOW）

输入 RPOW 指令是一个 32 位指令，将数据堆栈第 1 级中的 32 位实数进行 n 次方运算，n 是 32 位实数，存放在累加器中，结果也是 32 位实数，存放在累加器中。



受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON

下面的例子中，当 I1 为 ON 时，第一个 RLDC 指令实数常数 10.0 装入累加器，第二个 RLDC 指令将实数常数 2.0 装入累加器，并将累加器中原有数据 10.0 压入数据堆栈的第一级中，RPOW 指令进行 $10.0^{2.0}$ 运算，结果为 100.0，存放在 R1500 和 R1501 中。

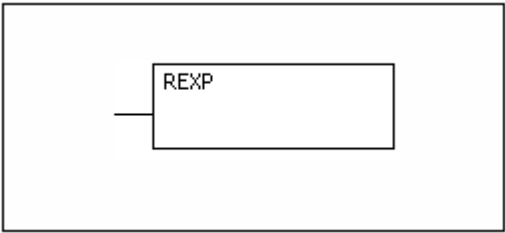


语句视图

```
LD I1
RLDC F10.0
RLDC F2.0
RPOW
OUTD R1500
```

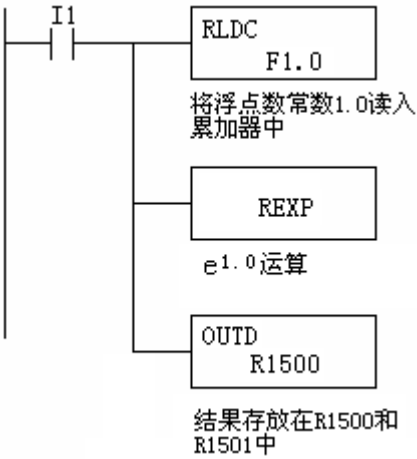
9.68 指数指令（REXP）

输入 REXP 指令是一个 32 位指令，将自然常数 e 进行 n 次方运算，n 是 32 位实数，存放在累加器中，结果也是 32 位实数，存放在累加器中。



受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON

下面的例子中，当 I1 为 ON 时，RLDC 指令实数常数 1.0 装入累加器，REXP 指令进行 $e^{1.0}$ 运算，结果为 $e=2.718282$ ，存放在 R1500 和 R1501 中。

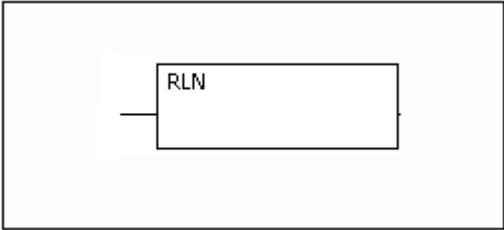


语句视图

```
LD I1
RLDC F1.0
REXP
OUTD R1500
```

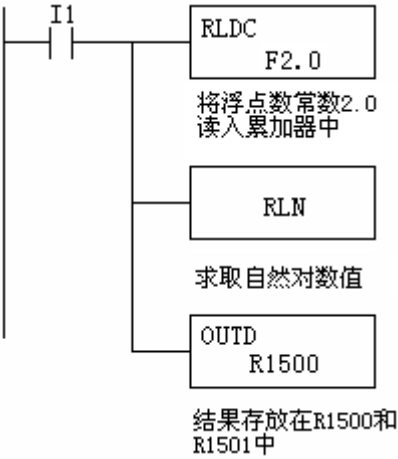
9.69 自然对数指令（RLN）

输入 RLN 指令是一个 32 位指令，求取累加器中实数的自然对数值，结果存放在累加器中。



受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON

下面的例子中，当 I1 为 ON 时，RLDC 指令实数常数 2.0 装入累加器，RLN 指令求取 2.0 的自然对数值，结果为 $\ln 2.0=0.693147$ ，存放在 R1500 和 R1501 中。

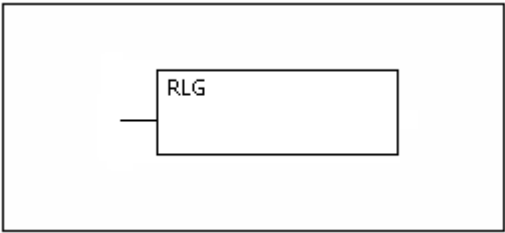


语句视图

```
LD I1
RLDC F2.0
RLN
OUTD R1500
```

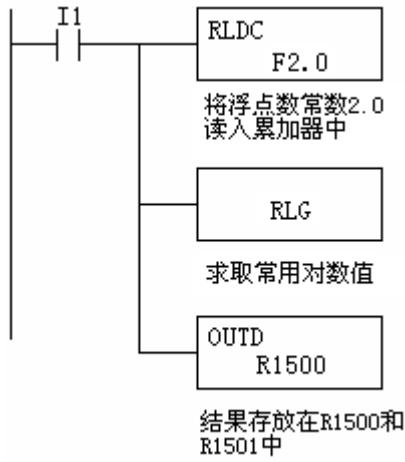

9.70 常用对数指令（RLG）

输入 RLG 指令是一个 32 位指令，求取累加器中实数的常用对数值，结果存放在累加器中。



受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON

下面的例子中，当 I1 为 ON 时，RLDC 指令实数常数 2.0 装入累加器，RLG 指令求取 2.0 的常用对数值，结果为 $\lg 2.0=0.301030$ ，存放在 R1500 和 R1501 中。



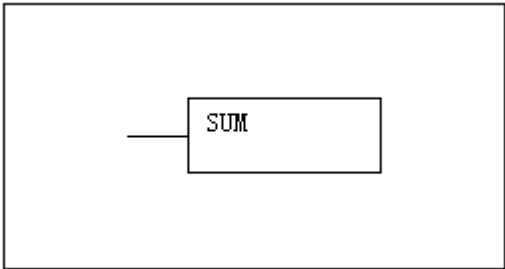
语句视图

```
LD I1
RLDC F2.0
RLG
OUTD R1500
```

第 10 章 位操作指令

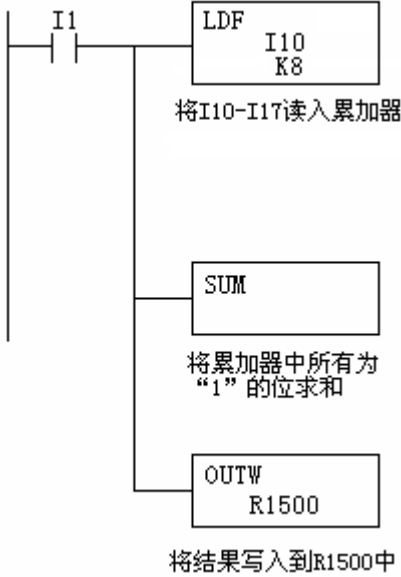
10.1 ON 位求和指令（SUM）

输入 求出累加器中所有为“1”（ON）的位的总数，结果以十六进制形式存放在累加器中。



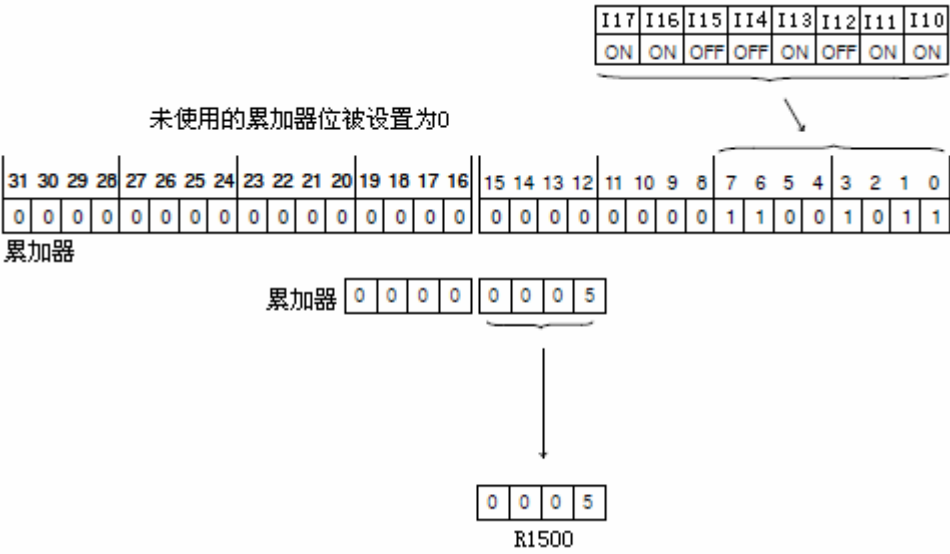
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON

下面的例子中，当 I1 为 ON 时，LDF 指令将 I10-I17 读入累加器，SUM 指令将累加器中所有为“1”的位求和，结果存放在累加器中，OUTW 指令将结果写入到 R1500 中。



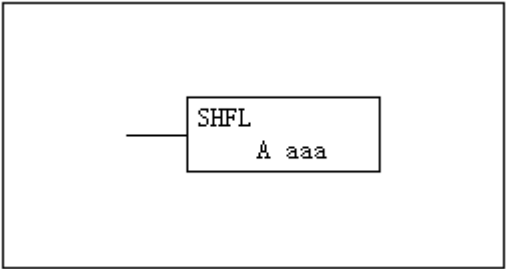
语句视图

```
LD I1
LDF I10 K8
SUM
OUTW R1500
```



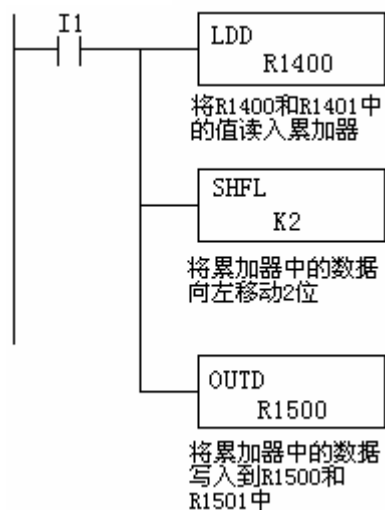
10.2 左移指令（SHFL）

输入 SHFL 指令是一个 32 位指令，将累加器中的数据向左移位，移动位数由指令指定，移动后空出的位以“0”填充，移出的位丢失。



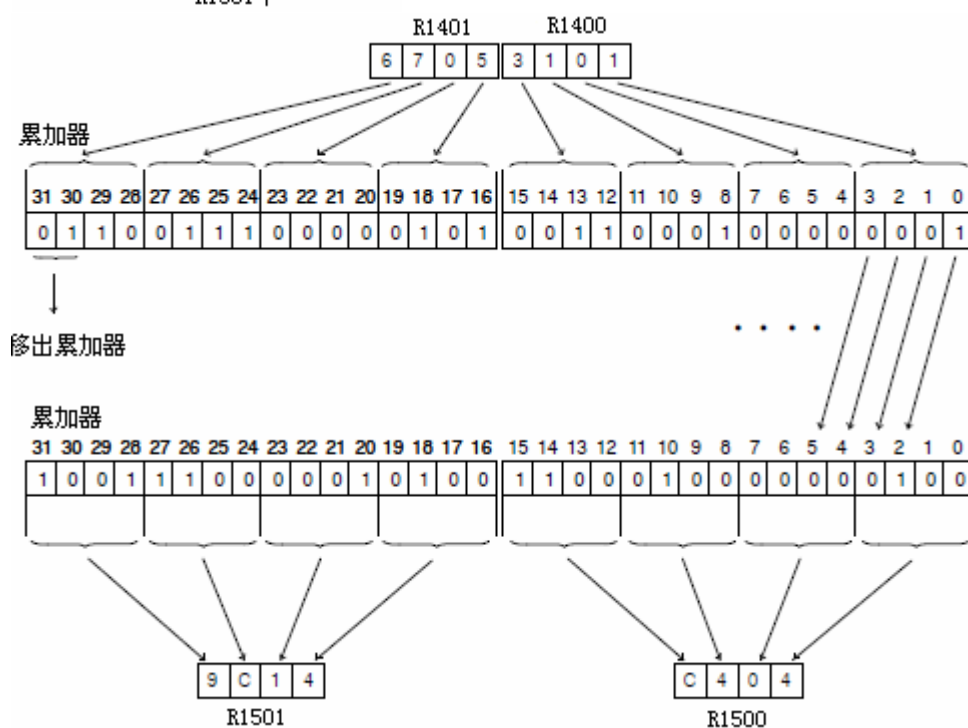
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	1-32

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器。SHFL 指令将累加器中的数据向左移动 2 位，OUTD 指令将累加器中的数据写入到 R1500 和 R1501 中。



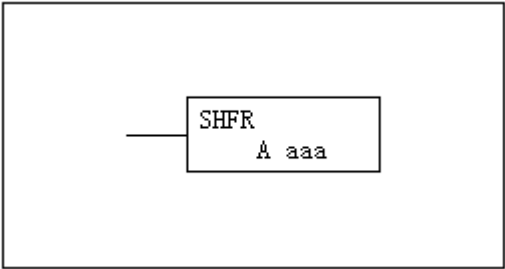
语句视图

```
LD I1
LDD R1400
SHFL K2
OUTD R1500
```



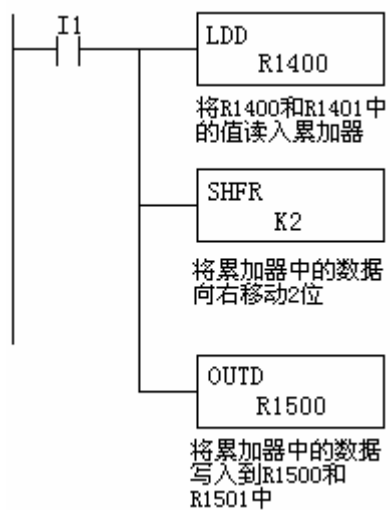
10.3 右移指令（SHFR）

输入 SHFR 指令是一个 32 位指令，将累加器中的数据向右移位，移动位数由指令指定，移动后空出的位以“0”填充，移出的位丢失。



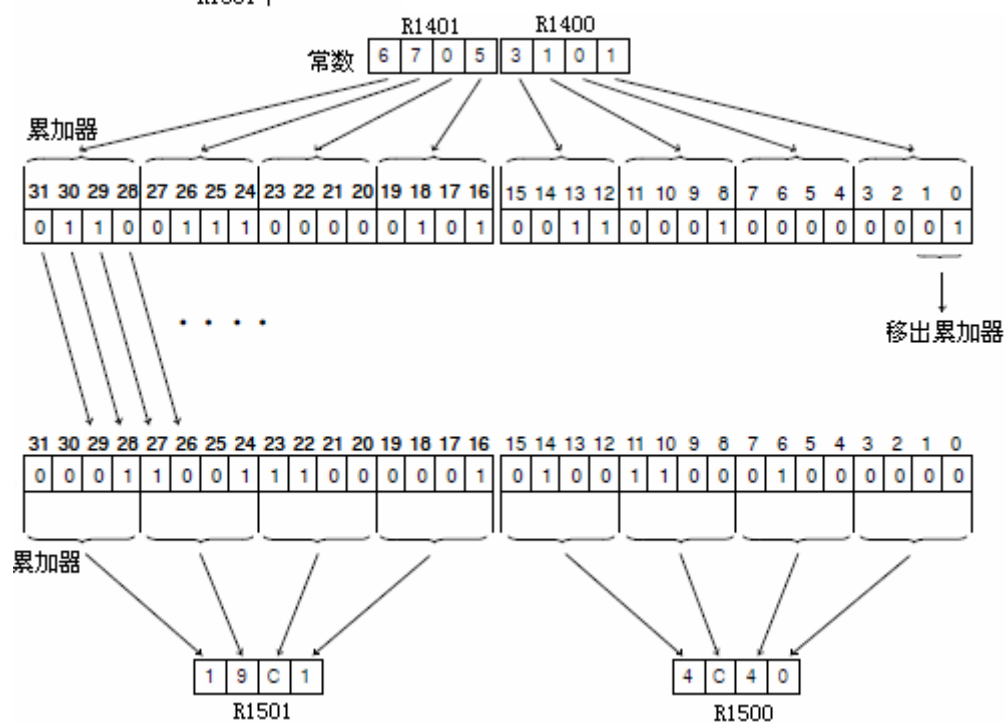
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	1-32

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器。SHFR 指令将累加器中的数据向右移动 2 位，OUTD 指令将累加器中的数据写入到 R1500 和 R1501 中。



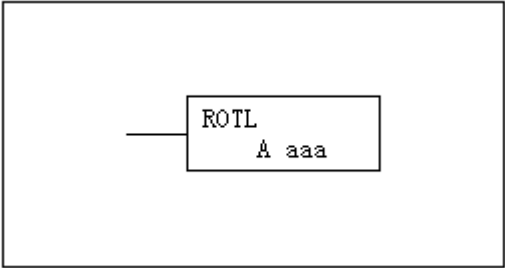
语句视图

```
LD I1
LDD R1400
SHFR K2
OUTD R1500
```



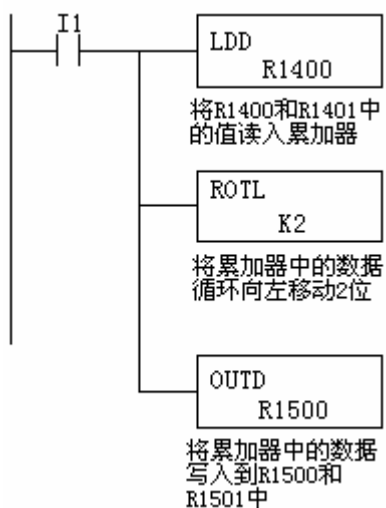
10.4 循环左移指令（ROTL）

输入 ROTL 指令是一个 32 位指令，将累加器中的数据循环向左移位，移动位数在指令中指定。



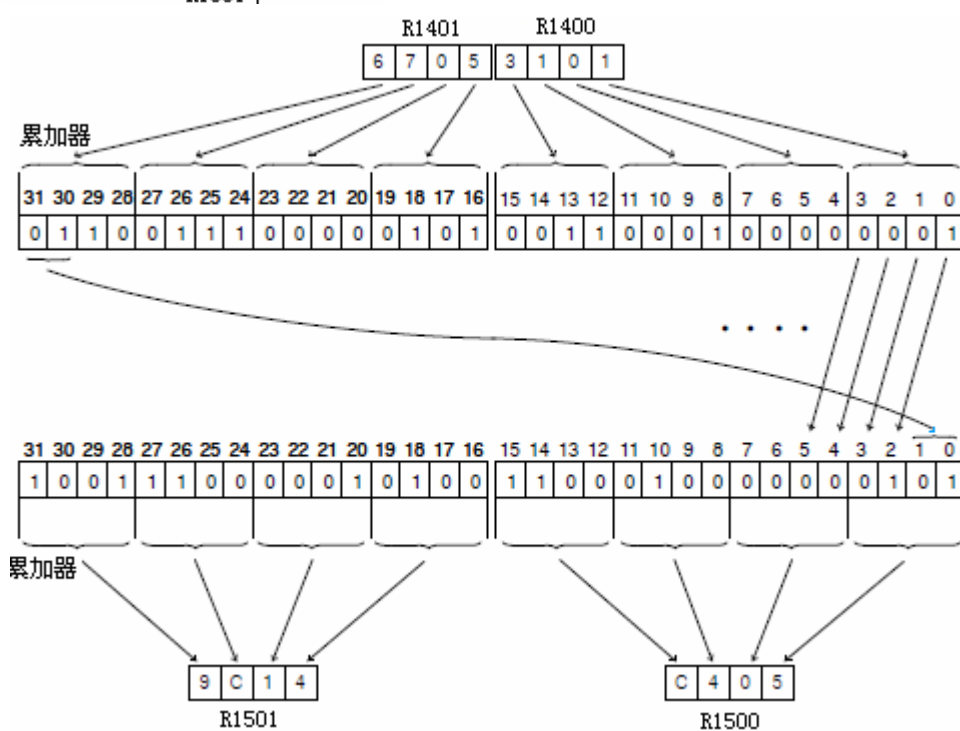
操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	1-32

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器。ROTL 指令将累加器中的数据循环向左移动 2 位，OUTD 指令将累加器中的数据写入到 R1500 和 R1501 中。



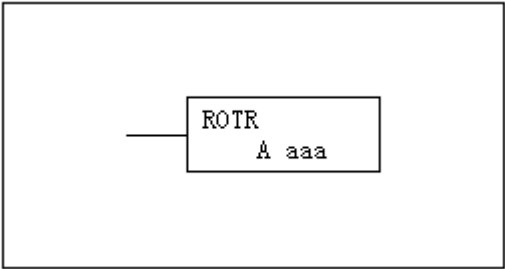
语句视图

```
LD I1
LDD R1400
ROTL K2
OUTD R1500
```



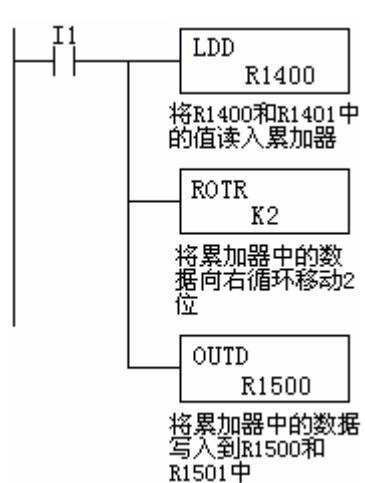
10.5 循环右移指令（ROTR）

 ROTR 指令是一个 32 位指令，将累加器中的数据循环向右移位，移动位数在指令中指定。



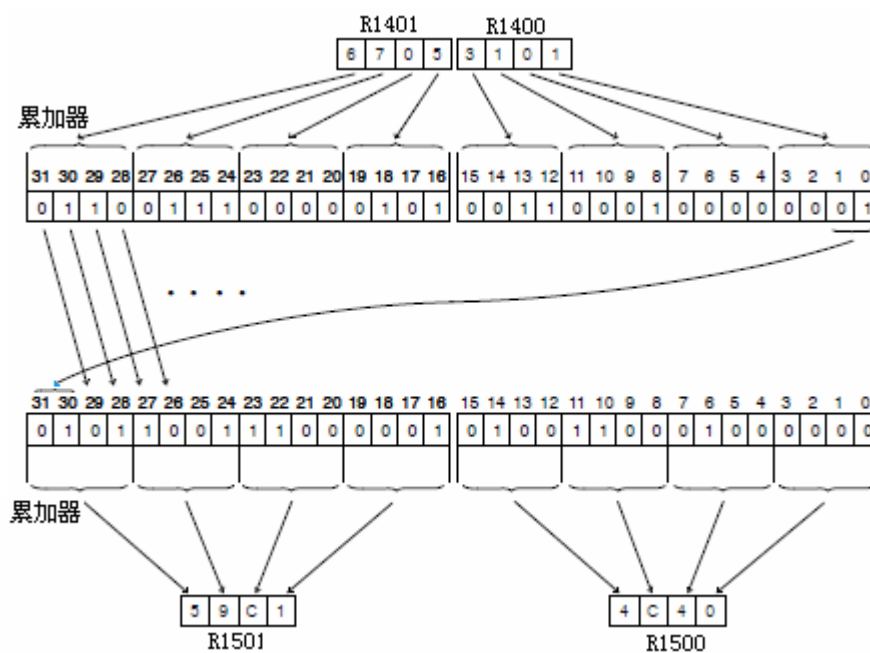
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	1-32

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器。ROTR 指令将累加器中的数据循环向右移动 2 位，OUTD 指令将累加器中的数据写入到 R1500 和 R1501 中。



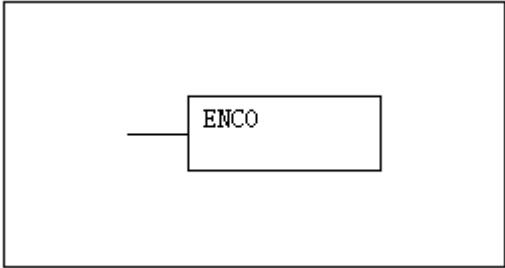
语句视图

```
LD I1
LDD R1400
ROTR K2
OUTD R1500
```



10.6 编码指令（ENCO）

输入 ENCO 指令将累加器中数值为“1”的一位的位号值转换成对应的 5 位二进制数。如果最高位是“1”（位号为 31），ENCO 指令就将十六进制数 1F（十进制数 31）放入累加器。如果被编码的数是 0000 或 0001，ENCO 指令就将 0 放入累加器。如果被编码的数据有多个位是“1”，则最低一个为“1”的位被编码，并且 SP53 被设置为 ON。累加器中的数据为 0 时，SP53 也被设置为 ON。

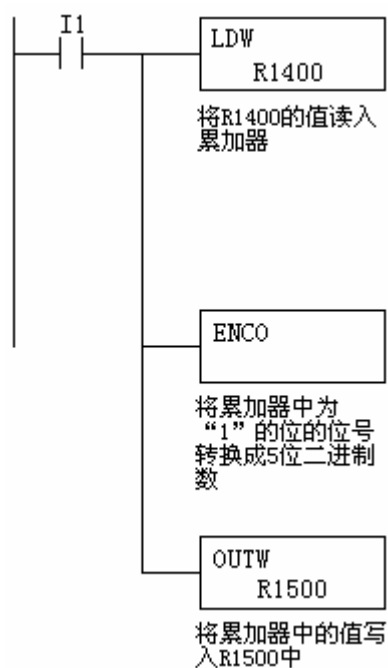


受影响的标志线圈	描述
SP53	不可进行运算处理时 ON



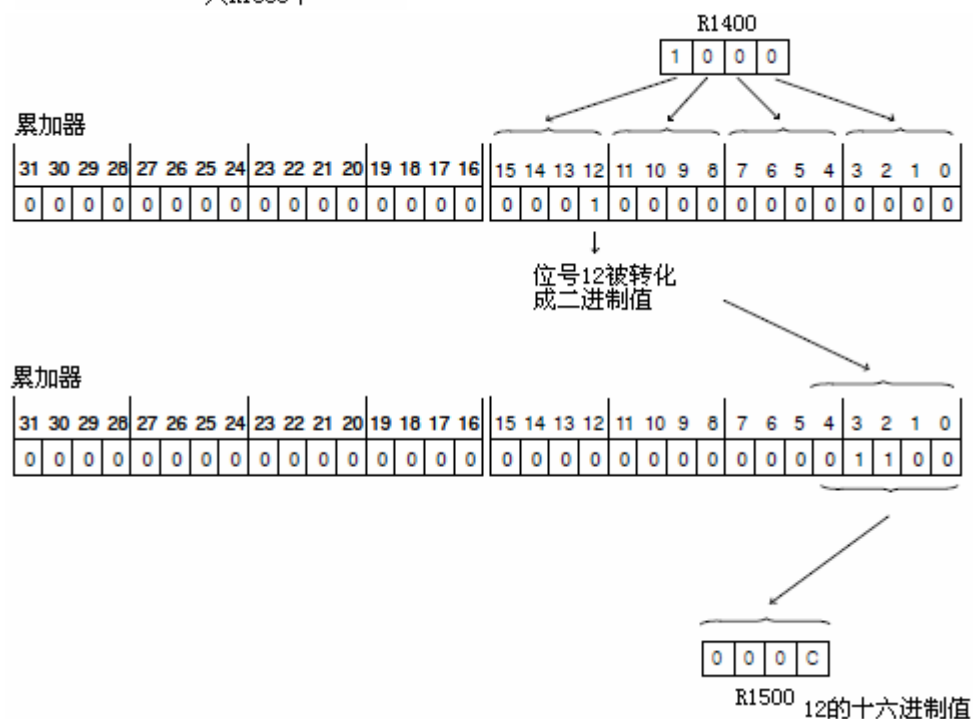
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，ENCO 指令将累加器中为“1”的位的位号值转换成对应的 5 位二进制数，放入累加器中，OUTW 指令将累加器中的数据写入到 R1500 中。



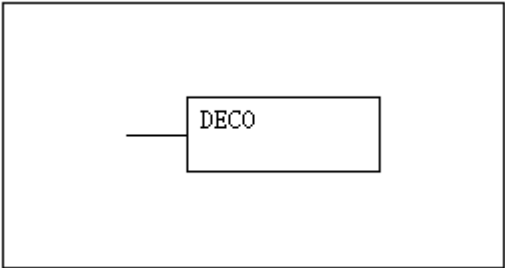
语句视图

```
LD I1
LDW R1400
ENCO
OUTW R1500
```

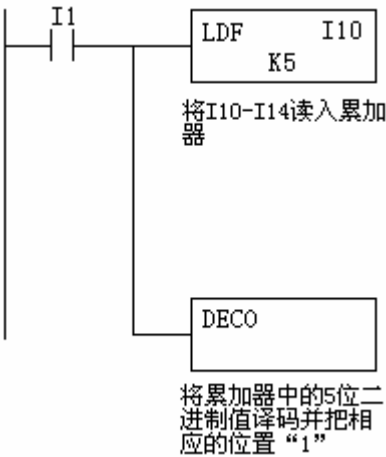


10.7 译码指令（DECO）

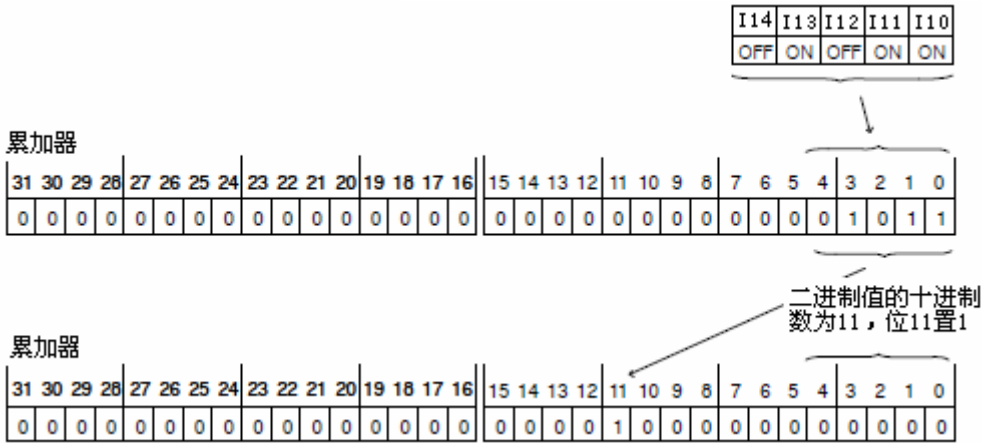
输入 DECO 指令将累加器中低 5 位的二进制编码转换成 0-31 之间的数，使累加器相应的位为“1”。如果累加器中数据是 F（十六进制），位 15 将被置位“1”。如果被译码的数据大于 31，则将其除以 32，直到其小于 32，然后再将其译码。



下面的例子中，当 I1 为 ON 时，LDF 指令将 I10-I14 读入累加器，DECO 指令将累加器中的 5 位二进制数据译码，将相应的位置“1”。



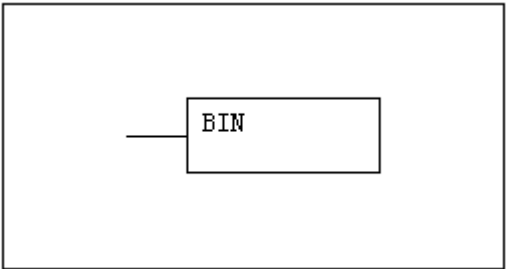
语句视图
LD I1
LDF I10 K5
DECO



第 11 章 数据变换指令

11.1 BIN 码变换指令（BIN）

输入 BIN 指令将累加器中的 BCD 数转换成对应的二进制数，结果存放在累加器中。

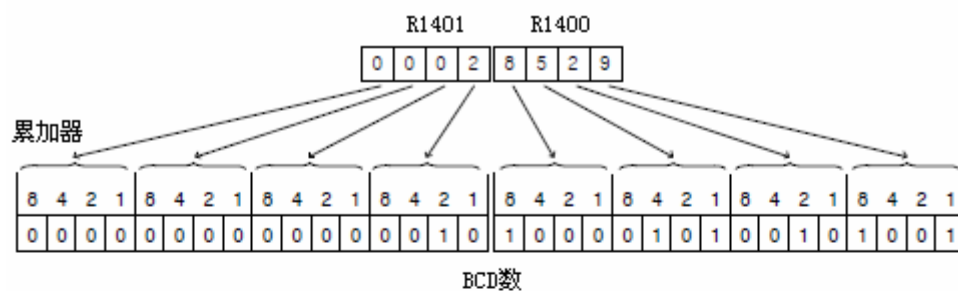


下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 的数据读入累加器，BIN 指令将累加器中的数据转换成二进制数(十六进制数)，OUTD 指令将结果写入到 R1500 和 R1501 中。查看 R1500 和 R1501 中的数据，显示的是十六进制数。



语句视图

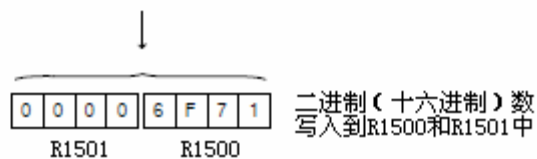
```
LD I1
LDD R1400
BIN
OUTD R1500
```



$$28529 = 16384 + 8192 + 2048 + 1024 + 512 + 256 + 64 + 32 + 16 + 1$$

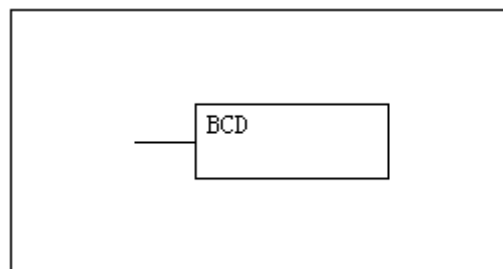
累加器 相等的二进制数

31 30 29 28				27 26 25 24				23 22 21 20				19 18 17 16				15 14 13 12				11 10 9 8				7 6 5 4				3 2 1 0			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0	1	1	1	0	0	0	1
2	1	5	2	1	6	3	1	8	4	2	1	5	2	6	1	3	1	8	4	2	1	5	2	1	6	3	1	8	4	2	1
1	0	3	6	3	7	3	3	1	1	0	0	2	6	3	5	2	6	1	0	0	0	0	0	0	4	2	6	0	2	6	0
4	7	6	8	4	1	5	7	8	9	9	4	4	2	1	0	7	3	9	2	4	0	9	2	0	2	2	8	1	4	2	1
7	3	8	4	2	0	5	7	4	7	8	2	1	0	3	6	6	8	6	6	6	8	6	6	6	8	6	6	6	8	6	6
4	7	7	3	1	8	4	7	6	3	1	5	4	7	2	4	4	2	4	4	4	2	4	4	4	2	4	4	4	2	4	4
4	4	0	5	7	6	4	2	0	0	5	7	6	4	2	6	6	4	6	6	6	4	6	6	6	4	6	6	6	4	6	6
8	1	9	4	7	7	3	1	8	3	0	4	2	6	1	0	4	2	4	4	4	2	4	4	4	2	4	4	4	2	4	4
3	6	1	5	2	6	4	2	0	0	8	2	6	4	2	6	6	4	6	6	6	4	6	6	6	4	6	6	6	4	6	6
6	2	2	4	8	4	2	6	4	2	6	8	4	2	6	4	4	2	4	4	4	2	4	4	4	2	4	4	4	2	4	4
4	4	8	6	4	2	6	4	2	6	4	2	6	4	2	6	6	4	6	6	6	4	6	6	6	4	6	6	6	4	6	6

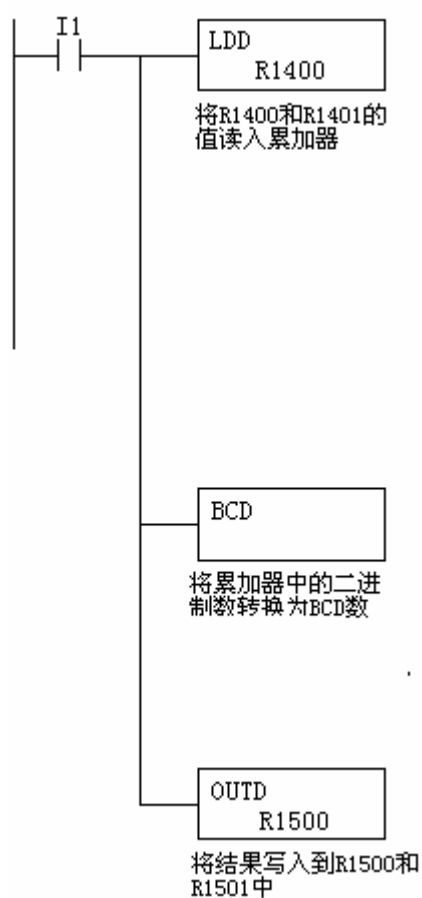


11.2 BCD 码变换指令（BCD）

输入 BCD 指令将累加器中的二进制数转换成对应的 BCD 数，结果存放在累加器中。

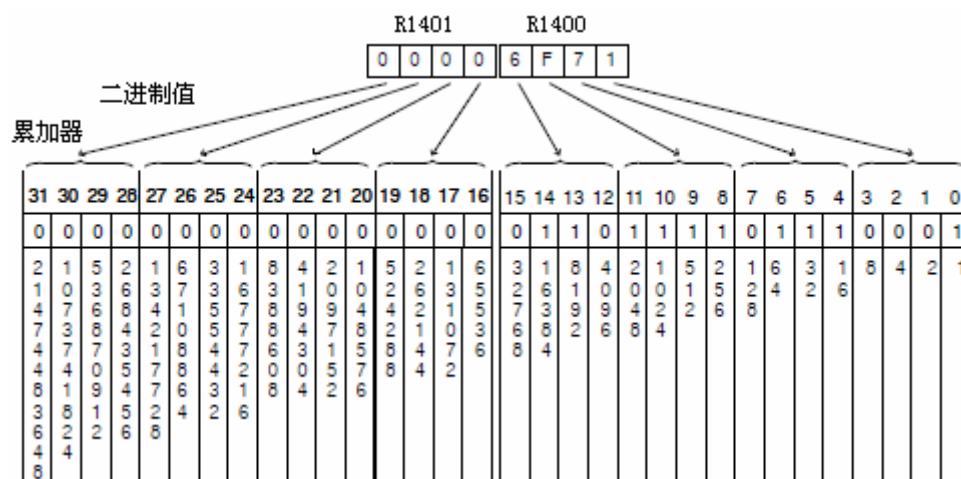


下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 的二进制数（十六进制数）读入累加器，BCD 指令将累加器中的二进制数据转换成 BCD 数，OUTD 指令将结果写入到 R1500 和 R1501 中。



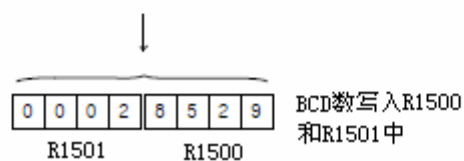
语句视图

```
LD I1
LDD R1400
BCD
OUTD R1500
```

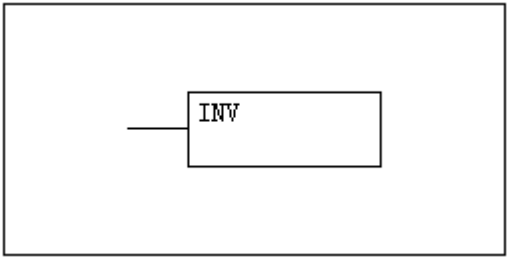
$$16384+8192+2048+1024+512+256+64+32+16+1=28529$$

累加器				相等的BCD数																											
8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	1	0	1	0	0	1	0	1

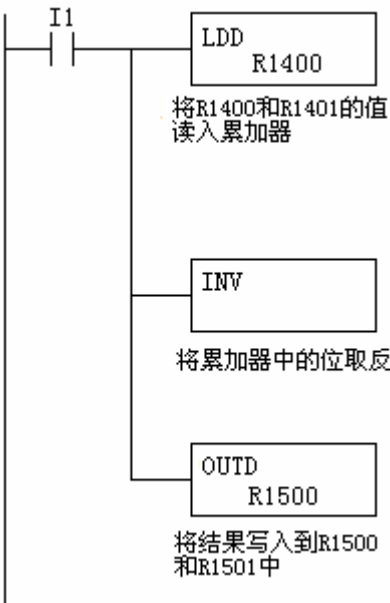


11.3 取反指令（INV）

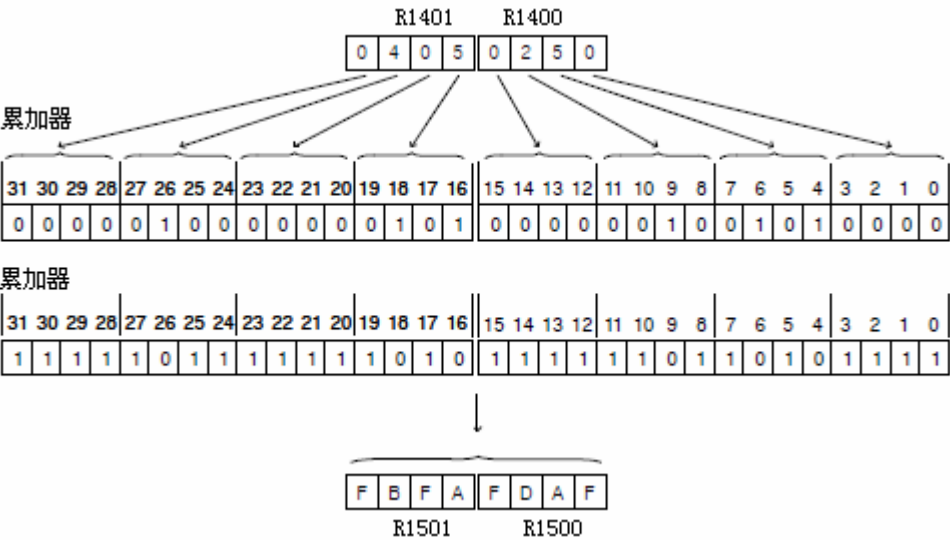
输入 INV 指令将累加器中的位取反，即 0 变 1，1 变 0，结果存放在累加器中。



下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 的数据读入累加器，INV 指令将累加器中的位取反，结果写入到 R1500 和 R1501 中。



语句视图
LD I1
LDD R1400
INV
OUTD R1500



11.4 十进制补码变换（BCDCPL）

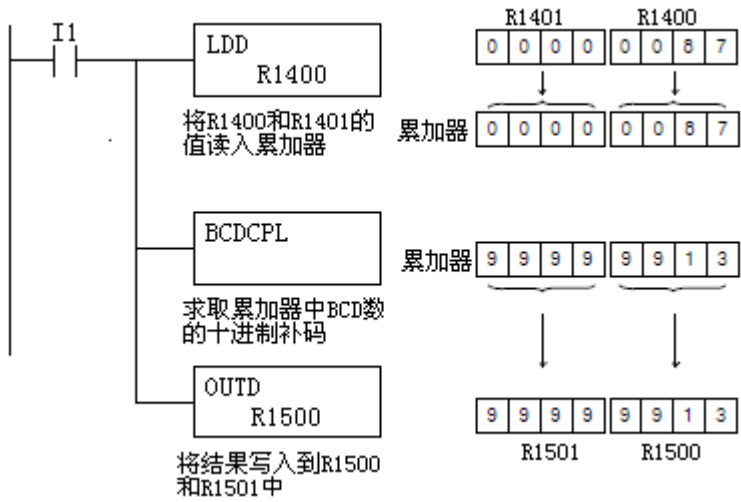
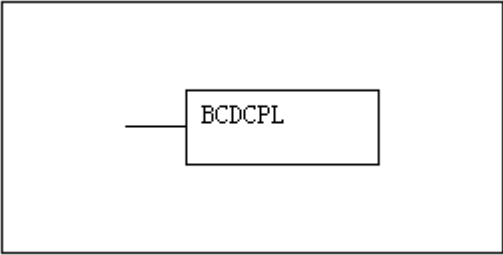
输入 BCDCPL 指令对累加器中的 8 位 BCD 数求取 100000000 的补数，结果存放在累加器中。指令的算式是：

100000000

- 累加器中数据

十进制补码值

下面的例子中，当 I1 为 ON 时，将 R1400 和 R1401 中的数据读入累加器，BCDCPL 指令求取累加器中数据的十进制补码，结果写入到 R1500 和 R1501 中。



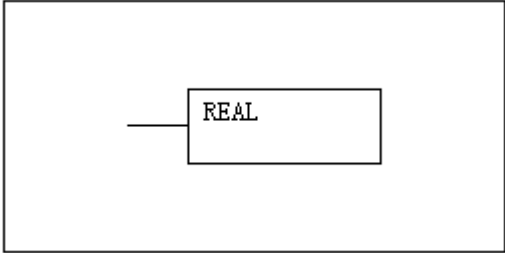
语句视图
LD I1
LDD R1400
BCDCPL
OUTD R1500



注意：如果程序中有减算指令导致借位的（有符号标志位标出），BCDCPL 指令可用来计算出两个数据之间的绝对差。

11.5 二进制到实数变换指令（REAL）

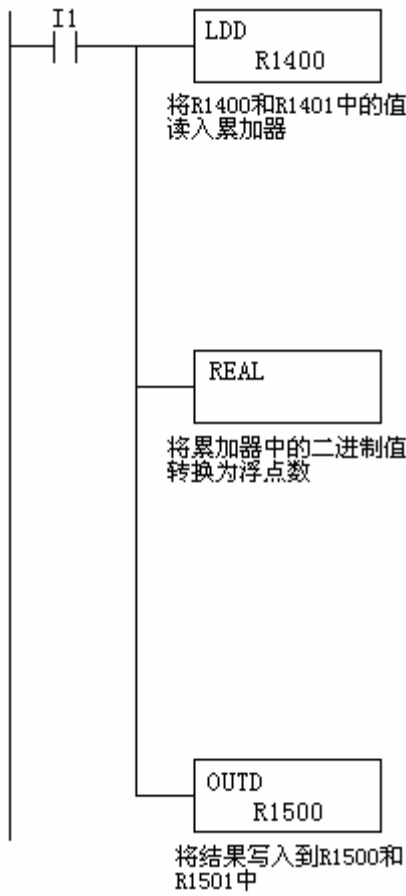
输入 将累加器中的二进制数转换为实数，结果存放在累加器中。二进制数和实数都使用累加器的 32 位。变换时根据 R7633.15 位状态决定是把被变换数看成带符号数进行变换，还是把被变换数看成无符号数进行变换。



注意：根据 R7633.15 位状态，变换结果可能有不同。

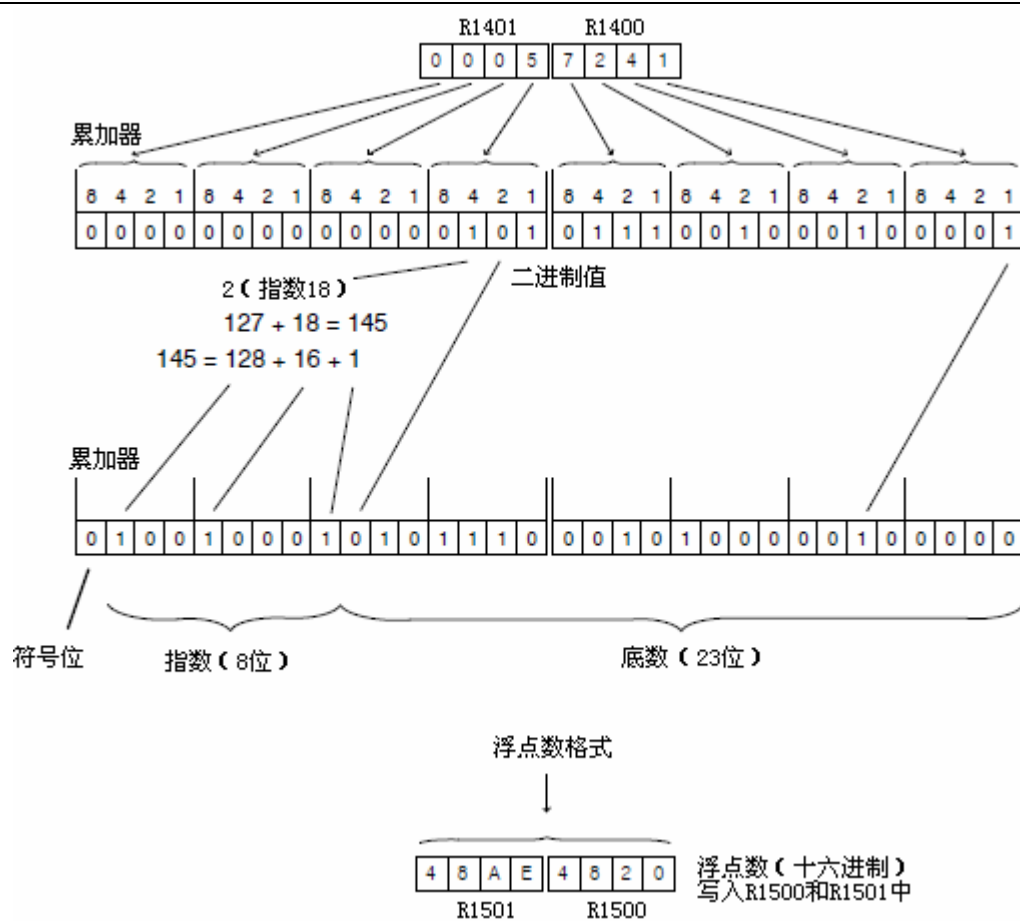
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

下面的例子中，当 I1 为 ON 时，将 R1400 和 R1401 的数据读入累加器，REAL 指令将累加器中的二进制数转换为实数。通过将二进制数的最高位位号加上 127（十进制），将其转换成实数指数，剩余的位是尾数，见下图。然后将结果写入到 R1500 和 R1501 中。查看 R1500 和 R1501 中的数，显示的是十六进制数。



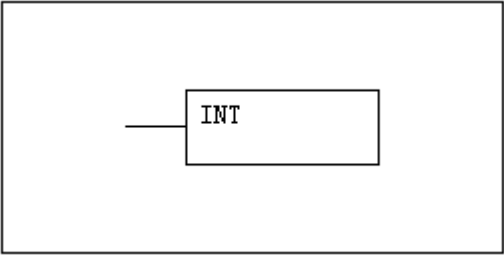
语句视图

```
LD I1
LDD R1400
REAL
OUTD R1500
```



11.6 实数到二进制数变换指令（INT）

输入 将累加器中的实数转换为二进制数。如果实数是负数，将转换成一个有符号数（以二进制补码格式存放），结果存放在累加器中。二进制数和实数都使用累加器的 32 位。转化过程中小数部分被删掉。

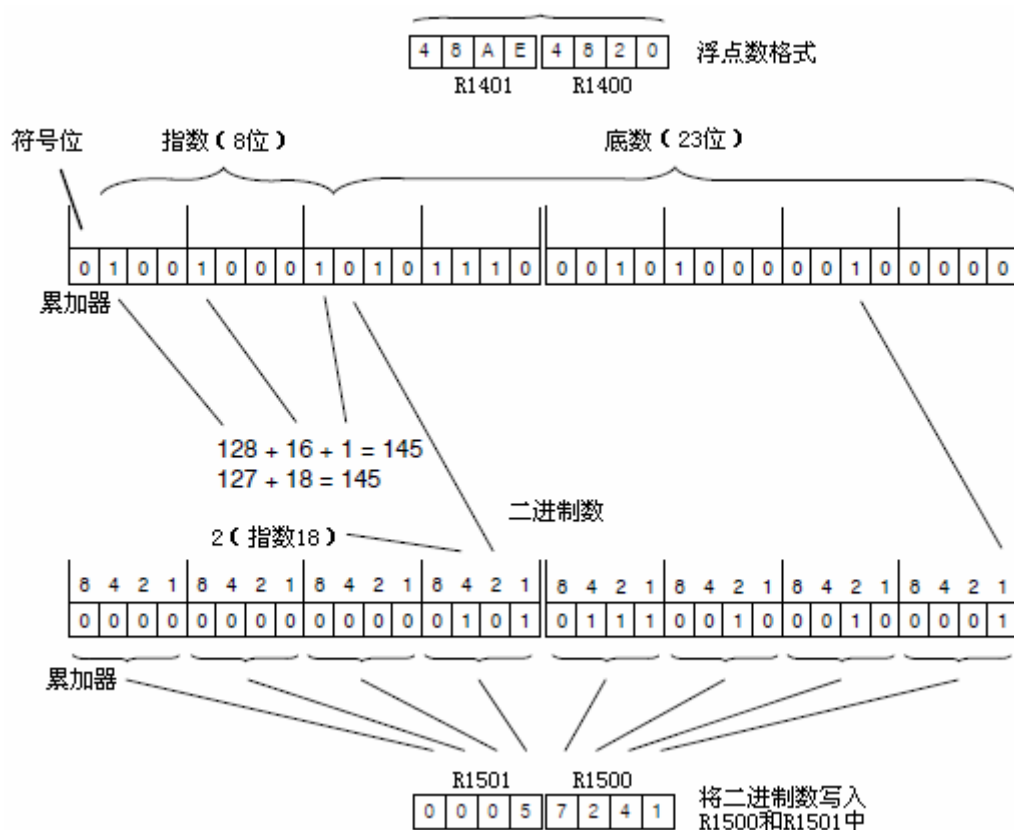
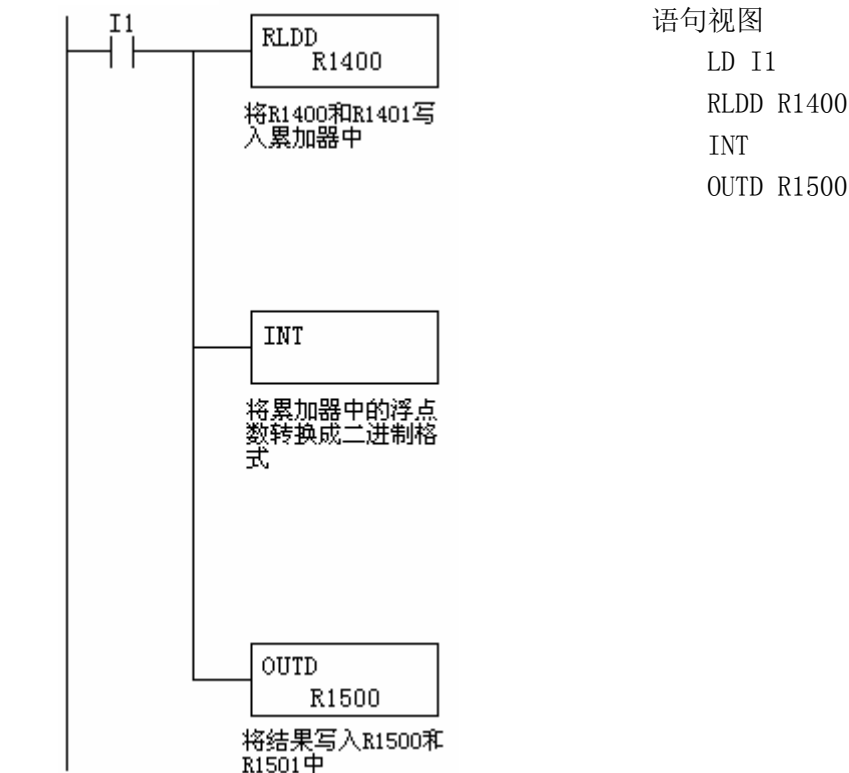


注意 1： 结果的小数部分将被四舍五入（例如，14.1→14 或-14.1→-15）。

注意 2： 如果实数是负数，将转化成有符号数（以二进制补码格式存放）。

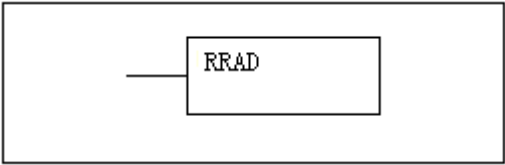
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON

下面的例子中，当 I1 为 ON 时，将 R1400 和 R1401 的数据读入累加器，INT 指令将累加器中的实数转换为二进制数。然后将结果写入到 R1500 和 R1501 中。查看 R1500 和 R1501 中的数，显示的是十六进制数。



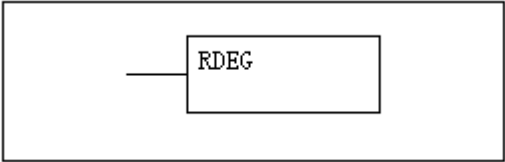
11.7 实数弧度变换指令（RRAD）

输入 RRAD 指令将累加器中实数格式的角度数据变换为相等的实数格式的弧度数据，结果存放在累加器中。



11.8 实数角度变换指令（RDEG）

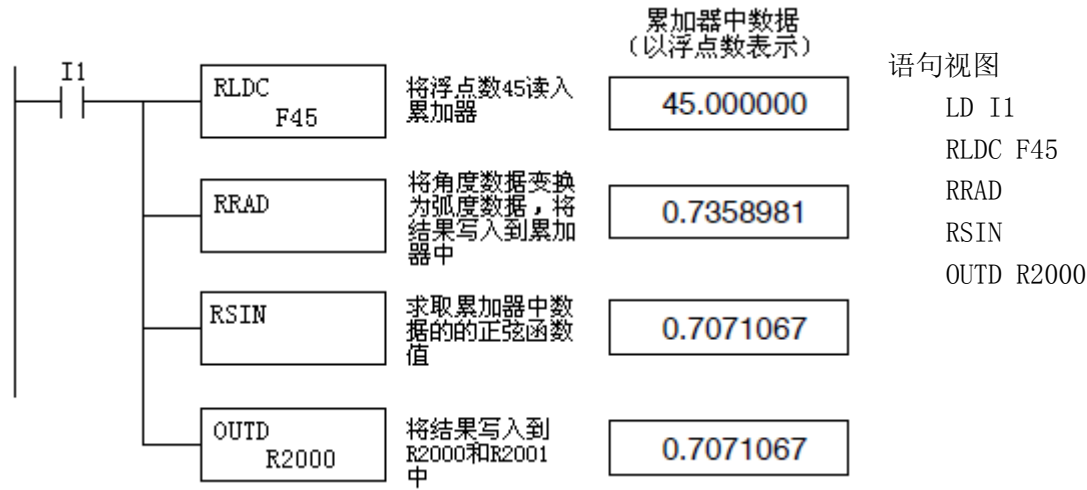
输入 RDEG 指令将累加器中实数格式的弧度数据变换为相等的实数格式的角度数据，结果存放在累加器中。



上面两个指令，是将累加器中的实数从角度变换为弧度，或是从弧度变换为角度。当是角度数据格式时，一个圆的角度是 360°；当是弧度数据格式时，一个圆的弧度是 2Π。被转换和转换后的数据都是正的或负的实数，大于一个圆的角度的数据也可以被转换，同三角函数指令配合使用，非常方便。

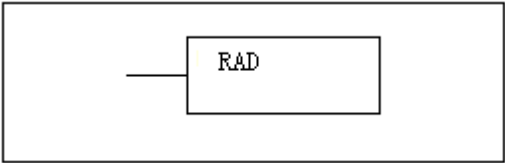
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP74	实数运算结果溢出时 ON

下面的例子中，求取 45° 的正弦函数。由于三角函数的运算对象只能是弧度，因此用 RRAD 指令将角度变换为弧度，RSIN 指令求取正弦值，将结果写入到 R2000 和 R2001 中。



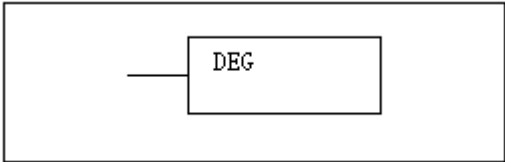
11.9 BCD 数弧度变换指令（RAD）

输入 RAD 指令将累加器中 BCD 格式的角度数据变换为相等的 BCD 格式的弧度数据，结果存放在累加器中。



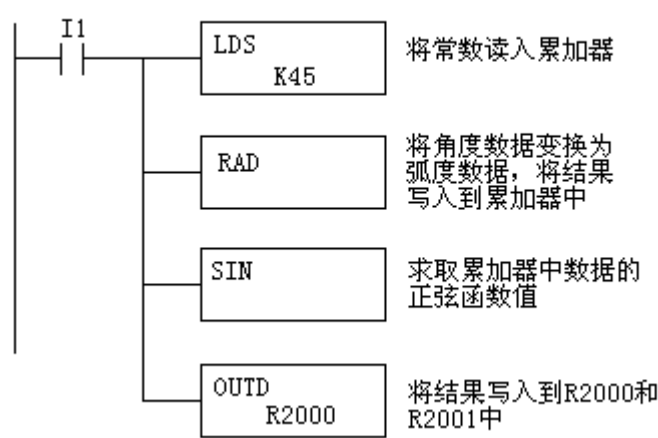
11.10 BCD 数角度变换指令（DEG）

输入 DEG 指令将累加器中 BCD 格式的弧度数据变换为相等的 BCD 格式的角度数据，结果存放在累加器中。



受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON

下面的例子中，求取 45° 的正弦函数。由于三角函数的运算对象只能是弧度，因此用 RAD 指令将角度变换为弧度，SIN 指令求取正弦值，将结果写入到 R2000 和 R2001 中。



语句视图

```
LD I1
LDS K45
RAD
SIN
OUTD R2000
```

11.11 ASC II →HEX 码变换指令（ATH）

输入

ATH 指令将一组 ASC II 数据转换成十六进制格式。ASC II 数据是 2 个字，而其相等的十六进制码是 1 个字。这就意味着一组 4 个 R 寄存器的 ASC II 数据变换成相等的十六进制数据只需要 2 个 R 寄存器。指令执行前，需将功能参数读入累加器和数据堆栈中，下面是将 ASC II 数据转换为十六进制数据的步骤：

第 1 步：将 ASC II 数据表的 R 寄存器的数目读入数据堆栈第 1 级中。

第 2 步：将 ASC II 数据表的起始 R 寄存器地址读入累加器中，参数必须是一个十六进制数。

第 3 步：在 ATH 指令中指定十六进制数据表的起始 R 寄存器地址。

ATH

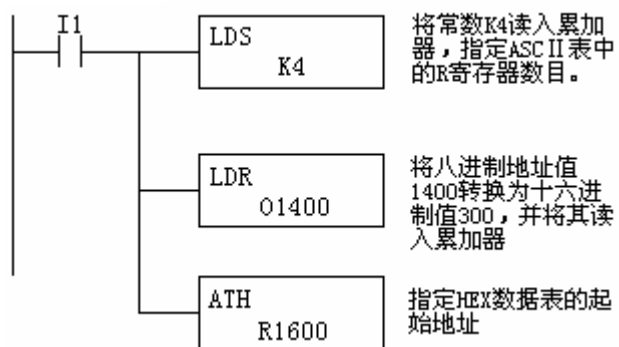
R aaa

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型		NK1 范围
		aaa
R 寄存器	R	所有（附录 1）

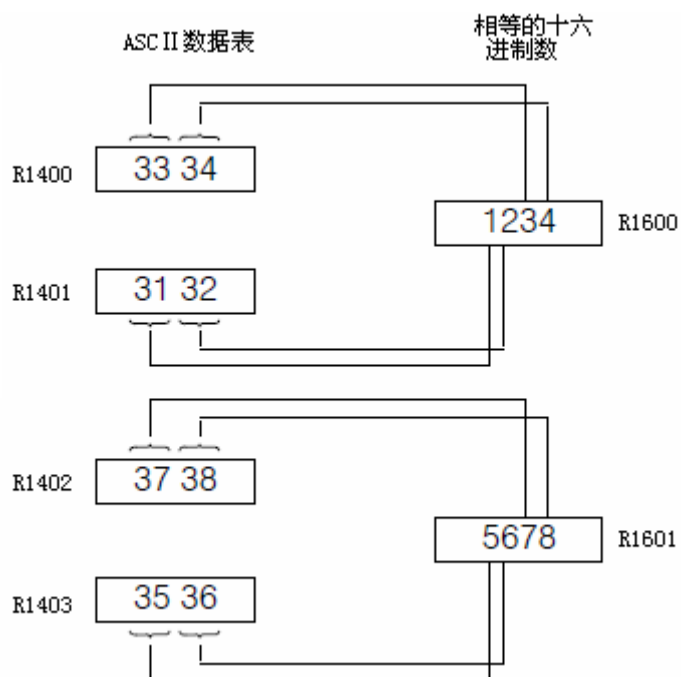
下面的例子中，当 I1 为 ON 时，常数 K4 被读入累加器，下一个读入指令执行时 K4 被压入数据堆栈的第 1 级中。ASC II 数据表的起始寄存器地址（R1400）被读入累加器。ATH 指令中指定了十六进制数据表的起始寄存器地址（R1600）。下表列出了 ATH 变换的有效 ASC II 值。

ATH 变换的有效 ASC II 码与 HEX 的对照表			
ASC II 值	HEX 值	ASC II 值	HEX 值
30	0	38	8
31	1	39	9
32	2	41	A
33	3	42	B
34	4	43	C
35	5	44	D
36	6	45	E
37	7	46	F



语句视图

```
LD I1
LDS K4
LDR 01400
ATH R1600
```



11.12 HEX→ASCII码变换指令（HTA）

输入

HTA 指令将一组十六进制数据转换成 ASCII 码。十六进制码是 1 个字，而其相等的 ASCII 数据是 2 个字。这就意味着一组 2 个 R 寄存器的十六进制数据变换成相等的 ASCII 数据需要 4 个 R 寄存器。指令执行前，需将功能参数读入累加器和数据堆栈中，下面是将十六进制数据转换为 ASCII 数据的步骤：

第 1 步：将十六进制数据表的 R 寄存器的数目读入数据堆栈第 1 级中。

第 2 步：将十六进制数据表的起始 R 寄存器地址读入累加器中，参数必须是一个十六进制数。

第 3 步：在 HTA 指令中指定 ASCII 数据表的起始 R 寄存器地址。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型	NK1 范围
	aaa
R 寄存器	R 所有（附录 1）

下面的例子中，当 I1 为 ON 时，常数 K2 被读入累加器，下一个读入指令执行时 K2 被压入数据堆栈的第 1 级中。十六进制数据表的起始寄存器地址（R1500）被读入累加器。HTA 指令中指定了 ASCII 数据表的起始寄存器地址（R1400）。

语句视图

```
LD I1
LDS K2
LDR 01400
HTA R1600
```

236

下表列出了 HTA 变换的有效 ASCII 值。

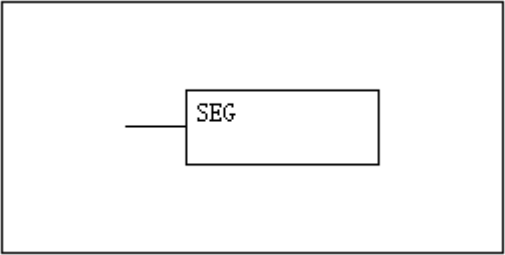
ATH 变换的有效 ASCII 码与 HEX 的对照表			
HEX 值	ASCII 值	HEX 值	ASCII 值
0	30	8	38
1	31	9	39
2	32	A	41
3	33	B	42
4	34	C	43
5	35	D	44
6	36	E	45
7	37	F	46

11.13 7 段译码指令（SEG）

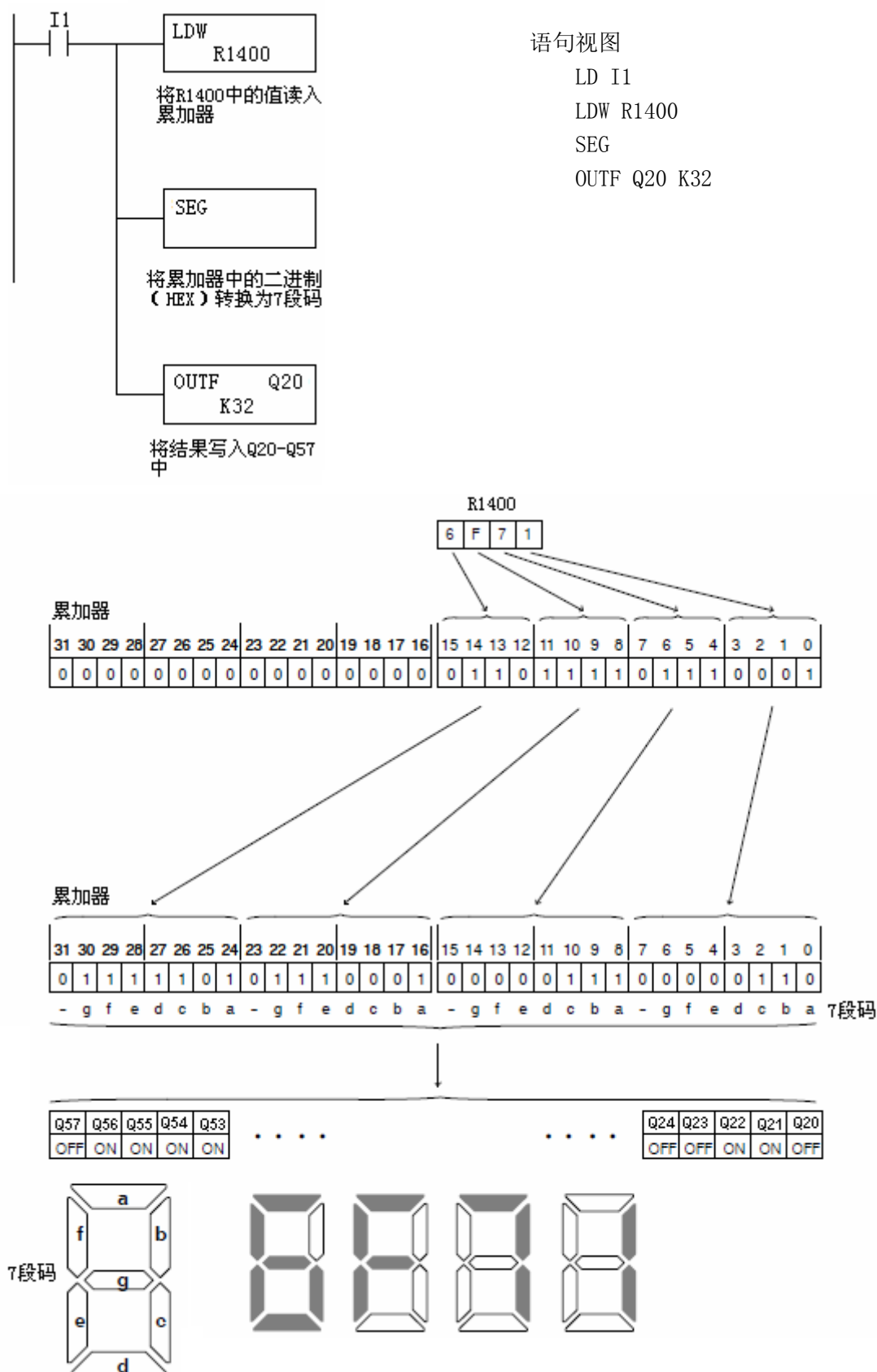
输入 BCD/SEG 指令将累加器中 4 字（16 位）十六进制数据转换成 7 段码显示格式，结果存放在累加器中。

下面是译码对应表：

ACC						显示 数据	7 段码								HEX 代码
							An								
							h	g	f	e	d	c	b	a	
	a ₃	a ₂	a ₁	a ₀		h	g	f	e	d	c	b	a		
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	3F
1	0	0	0	1	1	0	0	0	0	0	1	1	0		06
2	0	0	1	0	2	0	1	0	1	1	0	1	1		5B
3	0	0	1	1	3	0	1	0	0	1	1	1	1		4F
4	0	1	0	0	4	0	1	1	0	0	1	1	0		66
5	0	1	0	1	5	0	1	1	0	1	1	0	1		6D
6	0	1	1	0	6	0	1	1	1	1	1	0	1		7D
7	0	1	1	1	7	0	0	1	0	0	1	1	1		27
8	1	0	0	0	8	0	1	1	1	1	1	1	1		7F
9	1	0	0	1	9	0	1	1	0	1	1	1	1		6F
A	1	0	1	0	A	0	1	1	1	0	1	1	1		77
B	1	0	1	1	B	0	1	1	1	1	1	0	0		7C
C	1	1	0	0	C	0	0	1	1	1	0	0	1		39
D	1	1	0	1	D	0	1	0	1	1	1	1	0		5E
E	1	1	1	0	E	0	1	1	1	1	0	0	1		79
F	1	1	1	1	F	0	1	1	1	0	0	0	1		71

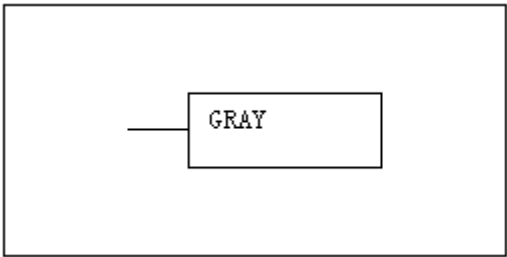


下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器。SEG 指令将累加器中的十六进制数（二进制数）转换为 7 段码格式。OUTF 指令将累加器中的数据写入到 Q20-Q57 中。

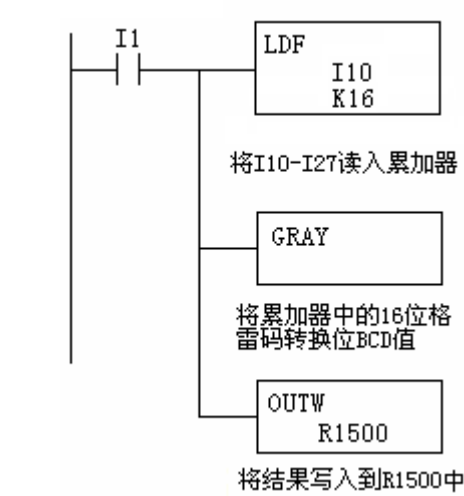


11.14 GRAY→BCD 码变换指令（GRAY）

输入 GRAY 指令将一个 16 位格雷码转换为一个 BCD 数。BCD 转换需要累加器中的低 10 位，高 22 位被设置为“0”。本指令用于使用格雷码的设备（通常是编码器），可将分辨率为 512 或 1024 的格雷码直接变换为对应的 BCD 码，而对分辨率为 360 和 720 的格雷码，转换后要分别减去 76 和 152，才能得到正确的 BCD 码。

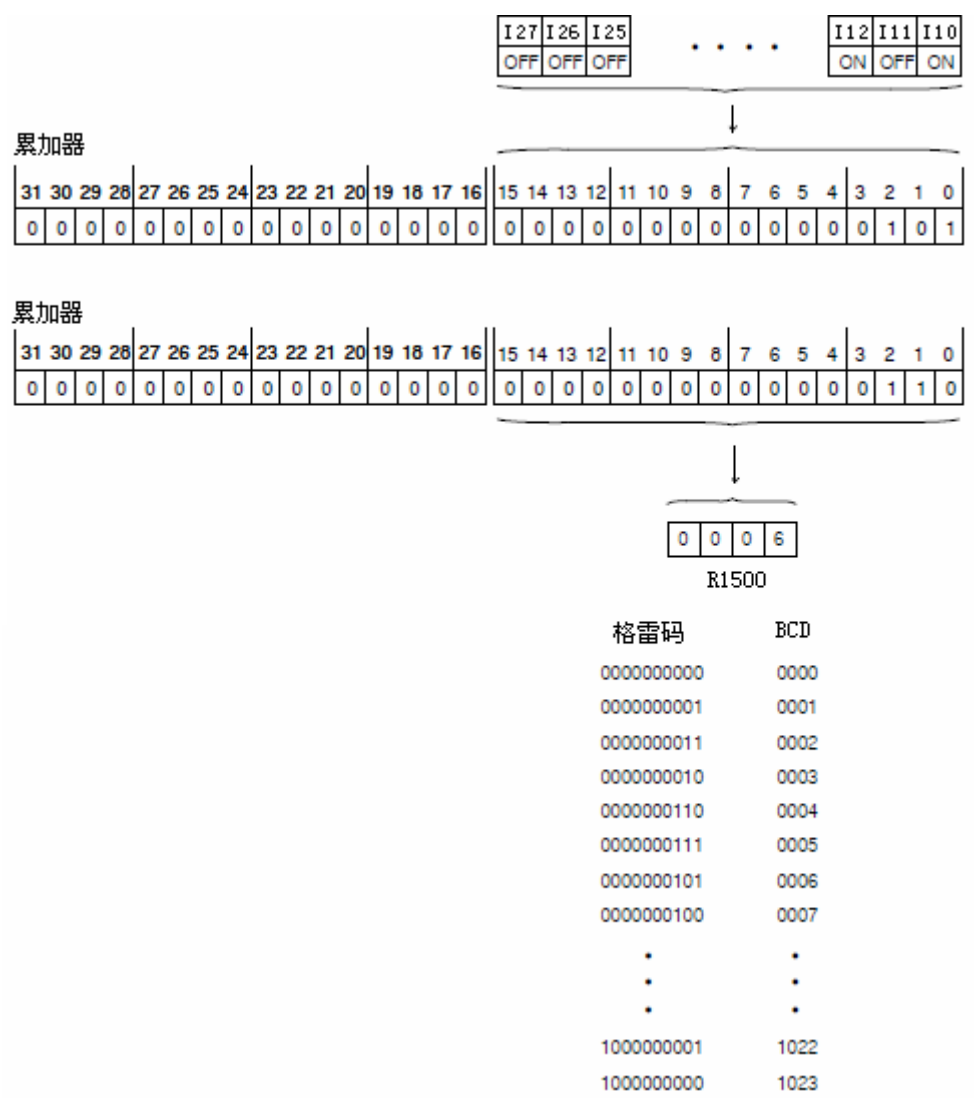


下面的例子中，当 I1 为 ON 时，LDF 指令将 I10-I27 读入累加器，GRAY 指令将累加器中的数据转换成 BCD 形式，将结果写入到 R1500 中。



语句视图

```
LD I1
LDF I10 K16
GRAY
OUTW R1500
```

11.15 位替换指令（SFLDGT）

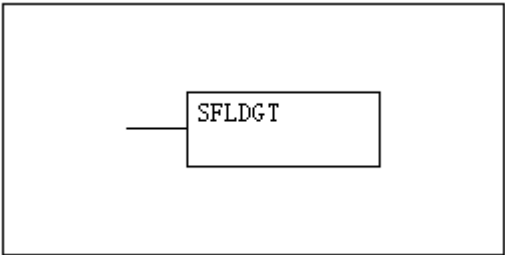
输入 SFLDGT 指令将数据堆栈第 1 级中存放的数据按累加器中存放的位指定数据进行排列处理。

SFLDGT 指令执行之前，需要将功能参数分别读入数据堆栈第 1 级和累加器中。下面是实现位替换功能的步骤：

- 第 1 步：将要进行位排列处理的数据读入数据堆栈的第 1 级中。
- 第 2 步：将位指定数据读入累加器，位指定数据的有效范围是 1-8（“1”是最低位，“8”是最高位）。
- 注意：如果位指定数据是 1-8 之外的数（0 或 9-F）时，相应位将被设置为 0，见下面的例子。
- 注意：当位指定数据有重复时，累加器的高位指定有效，见下面的例子。
- 第 3 步：写入 SFLDGT 指令。

位替换图解

最多有 8 位可被替换，数据堆栈第 1 级中存放的数据是要进行位排列处理的数据，这个数据按照累加器中的位指定数据进行重新排列，结果存入累加器中，见右图。

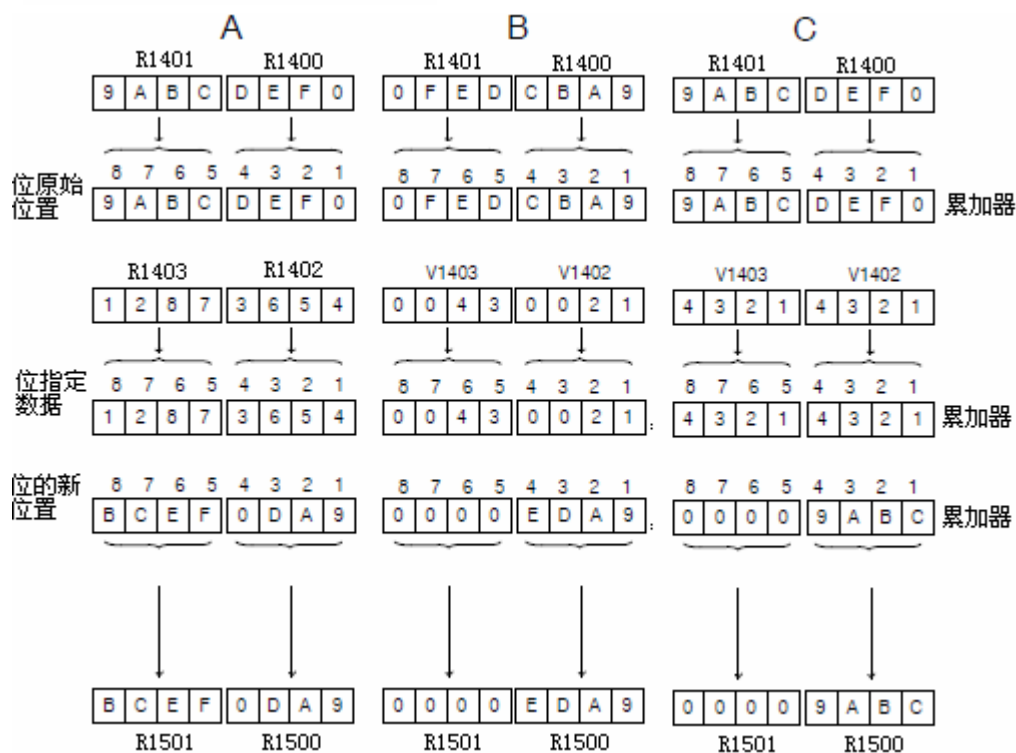
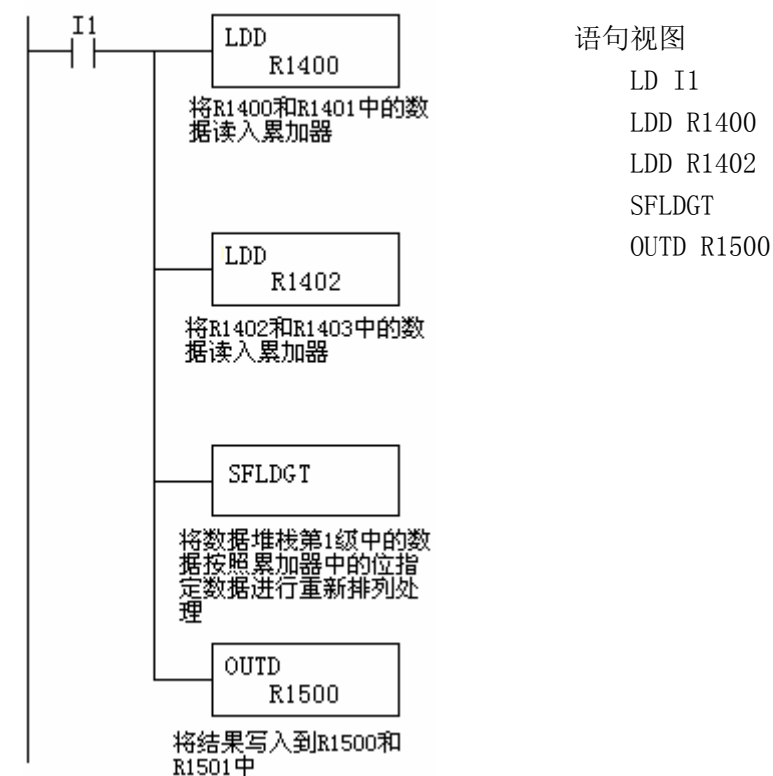


下面的例子中，当 I1 为 ON 时，数据堆栈第 1 级中的数据位按照累加器中的数据进行重新排列。

例 A 演示了当位指定数据没有 0 或 9-F，并且没有重复的数据时的位替换。

例 B 演示了位指定数据中有 0 或 9-F 时的位替换。注意，当替换指令执行时，相应位数据被设置为“0”。

例 C 演示了当位指定数据有重复时，当替换指令执行时，累加器中的高位指定有效。

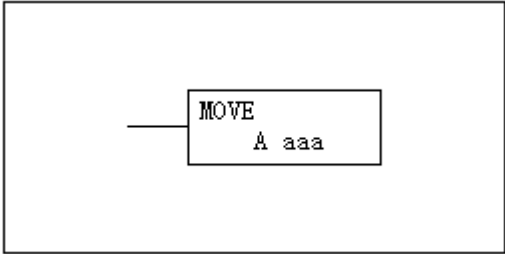


第 12 章 表检索指令

12.1 数据组传送指令（MOVE）

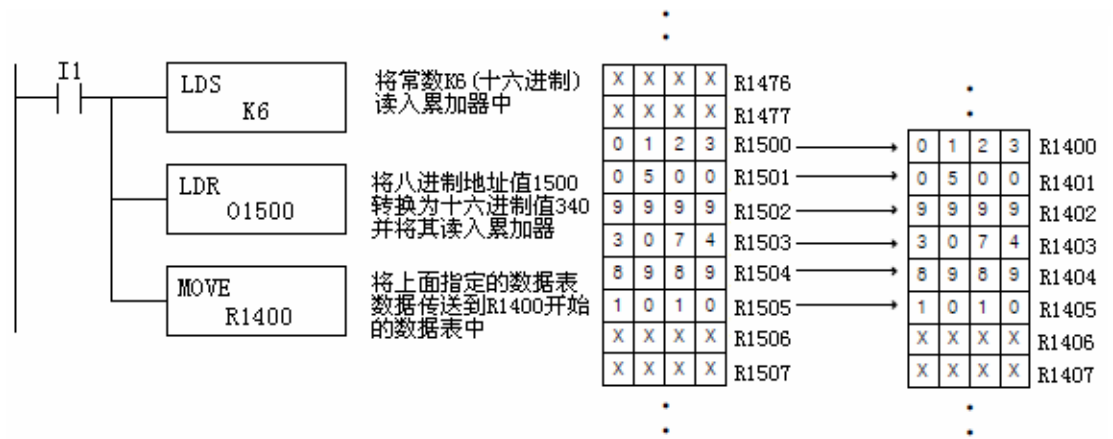
输入 MOVE 指令将一组最多 4095 个数据从一个 R 寄存器表传送到另一个相同长度的寄存器表中。指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：

- 第 1 步：将要被传送的数据表的长度读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0~FFF。
- 第 2 步：将数据表的起始寄存器地址读入累加器，这个数必须是十六进制数。
- 第 3 步：写入 MOVE 指令，指定目标数据表的起始寄存器地址。
- 小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。



操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
P 指针	P	所有（附录 1）

下面的例子中，当 I1 为 ON 时，常数 K6 被读入累加器，这个数据指定了数据表的长度，当第二个读入指令执行时，常数 K6 被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1500（R1500）读入累加器，这是数据表的起始地址。MOVE 指令中指定要传送的目标寄存器数据表的起始地址。



语句视图

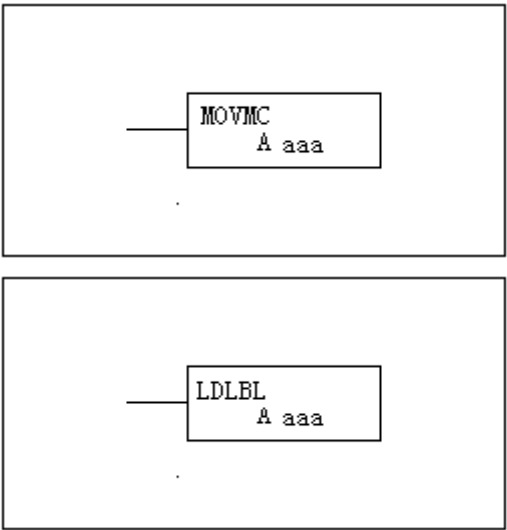
```
LD I1
LDS K6
LDR 01500
MOVE R1400
```

12.2 程序存储器—数据寄存器之间的数据传送指令（MOVMC）

12.3 数据标记地址读出指令（LDLBL）

输入 MOVMC 指令用于程序存储器向数据寄存器传送数据。当数据由程序存储器到数据寄存器传送时，LDLBL 指令同 MOVMC 指令一起使用。

程序存储器向数据寄存器传送数据之前，需将功能参数分别读入数据堆栈的第 1、第 2 级以及累加器中。下面是实现此指令功能的步骤：



- 第 1 步：将要传送的字的数目（最多 255 个）读入数据堆栈的第 2 级中，这个数必须是十六进制数，范围是 0-FF。
- 第 2 步：将数据源区域（十六进制）及目标数据表的起始寄存器地址的偏差值读入数据堆栈的第 1 级中。
- 第 3 步：将数据由程序存储器传送到 R 寄存器时，将数据源标记地址读入累加器。
- 第 4 步：写入 MOVMC 指令，指定目标数据表地址。

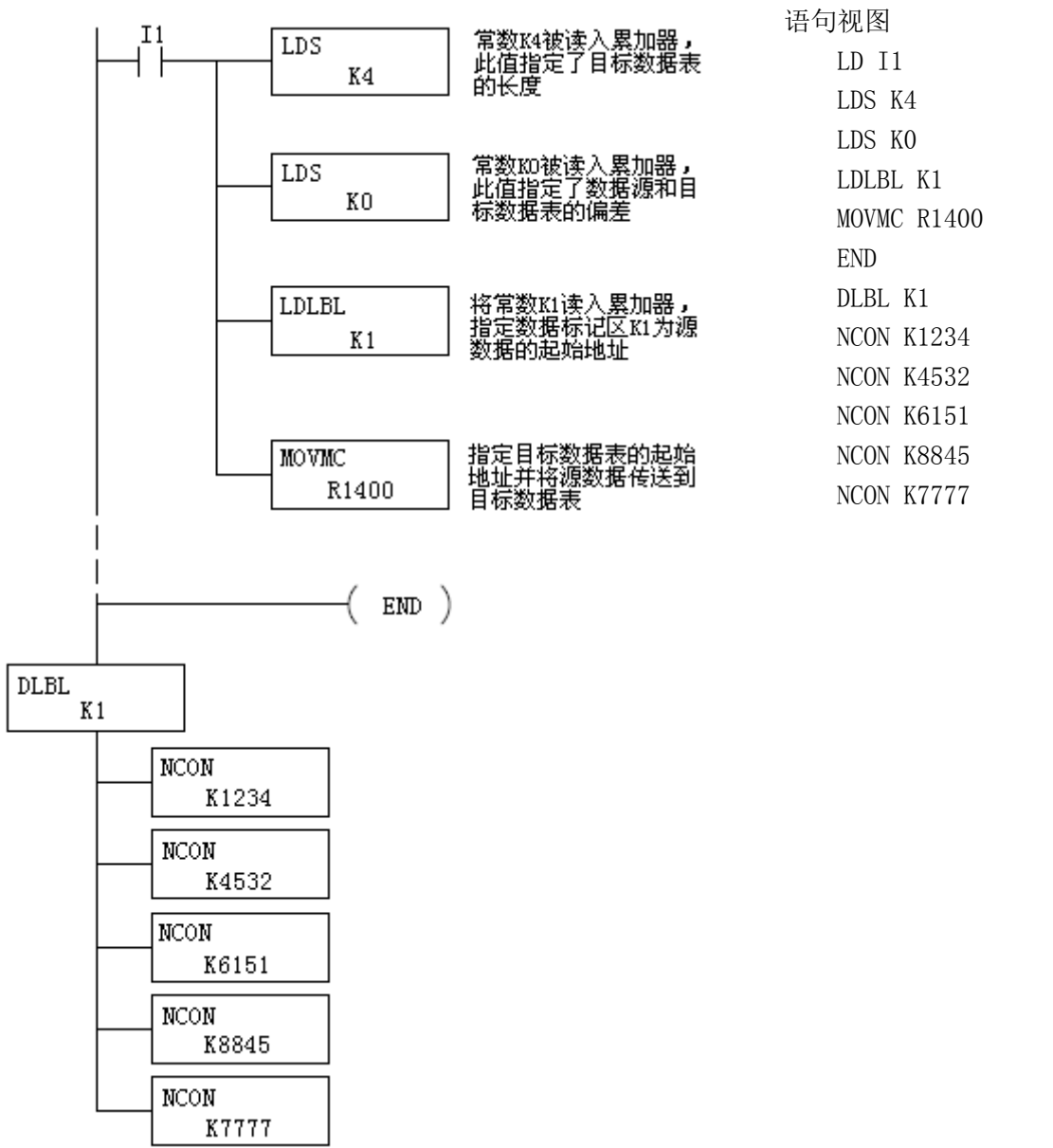
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	1-FFFF



警告：本指令的偏差值从 0 开始，也可以是不会导致数据源区域之外的数据被传送到目标数据表的任何数据。当偏差值超出数据源区域的范围，未知的数据将被传送到目标数据表中。

12.3.1 将数据从数据标记区传送到 R 寄存器

下面的例子中，数据从数据标记区传送到 R 寄存器。当 I1 为 ON 时，常数 K4 被读入累加器，此值指定了目标数据表的长度，当下面两个读入指令执行时，此值被压入数据堆栈的第 2 级中。常数 K0 被读入累加器，此值指定了数据源区域及目标数据表的偏移量，当 LDLBL 指令执行时，此值被压入数据堆栈的第 1 级中。LDLBL 指令将数据源标记地址读入累加器中。MOVMC 指令指定目标数据表的起始寄存器地址并将数据从数据源传送到 R 寄存器中。

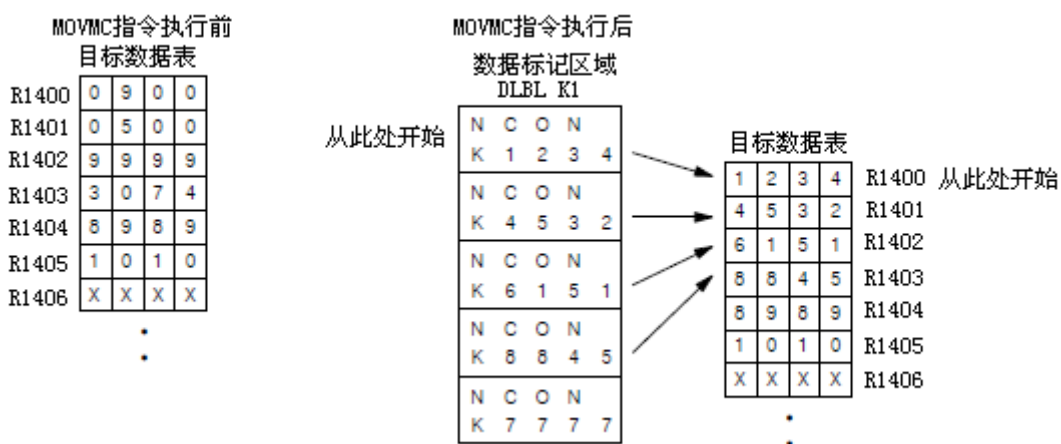


警告：上例中的偏差值是 0，也可以是不会导致数据源数据区之外的数据被传送到目标数据表的任何数据。当偏差值超出数据源数据区的范围时，未知的数据将被传送到目标数据表中。

下图给出了上例的运行结果。偏移量为 0，4 个字被传送。

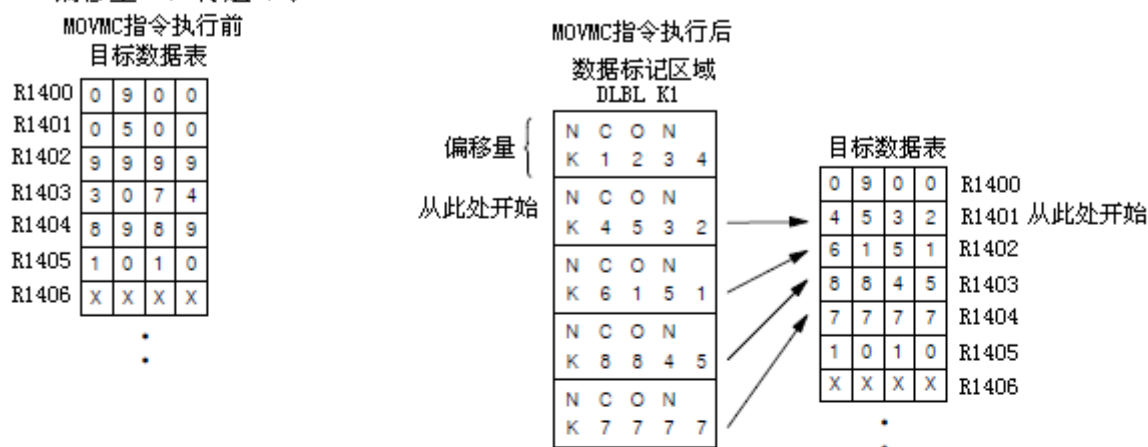
运行举例

偏移量=0，传送 4 字

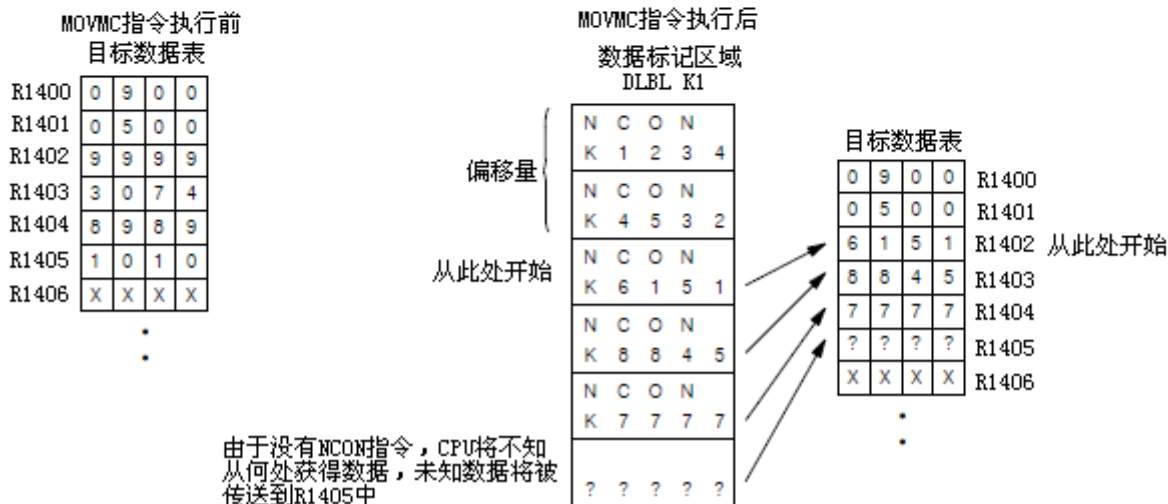


上图中很清楚的演示了偏移量为 0 时的程序运行结果。下图给出了当偏移量为其它数值时的程序运行结果，通过下图可以看到，偏移量是数据源区域及目标数据表的偏移量，并且，当使用了不恰当的偏移量（本例中为 2）时，会有未知的数据被传送到目标数据表。

偏移量=1，传送 4 字



偏移量=2，传送 4 字



12.4 登记数据寄存器传送指令（MOVAS）

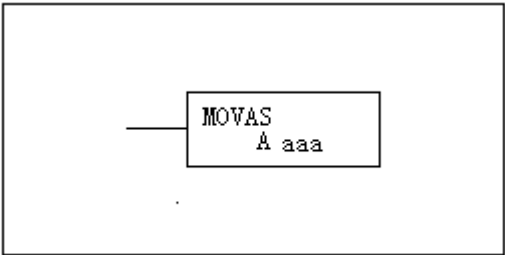
输入 MOVAS 指令用于将程序存储器中登记的数据，向数据寄存器中成组传送。

1.

使用此指令，首先要在程序存储器中有 DLBL 登记的标记名，在其后有用 ACON 或 NCON 登记的数据。
2.

执行此指令，在数据堆栈第 1 级存入传送字数，累加器中存入用 LDLBL 指令读入的数据标记名。
3.

程序存储器中登记的数据，1 个指令（1 个地址）是 16 位数据。
注意：程序存储器地址是 10 进制数，但传送字数用 8 进制设定。



操作数类型		NK1 范围
A		aaa
R 寄存器	R	所有（附录 1）

下面的例子中，条件 I0 成立时，数据标号 ABC 内登记的数据由 MOVAS 指令传送到指定的 R2000-R2002 中。

I0

LDR

03

LDLBL

KABC

MOVAS

R2000

END

DLBL

KABC

ACON

A KO

ACON

A YO

ACON

A OD

END

语句视图

LD I0

LDR 03

LDLBL KABC

MOVAS R2000

END

DLBL KABC

ACON A KO

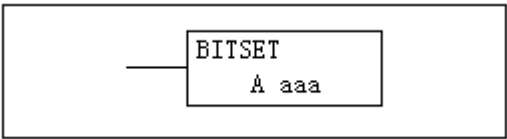
ACON A YO

ACON A OD

END

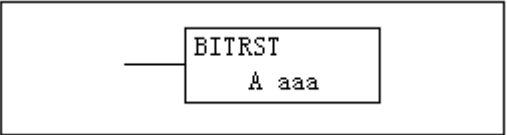
12.5 任意位置位指令（BITSET）

输入 BITSET 指令将R 寄存器表中的指定位置位。



12.6 任意位复位指令（BITRST）

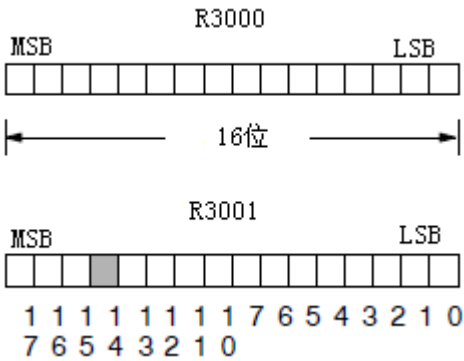
输入 BITRST 指令将R 寄存器表中的指定位置复。



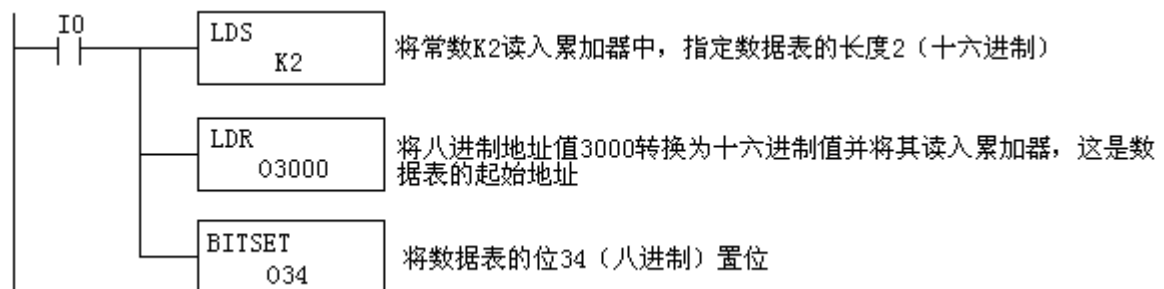
- 下面是实现这两个指令功能的步骤：
- 第 1 步：将数据表的长度（R 寄存器的个数）读入数据堆栈的第 1 级中。这个数必须是十六进制数，范围是 0-FF。
 - 第 2 步：将数据表的起始地址读入累加器中，这个数必须是十六进制数，可使用 LDR 指令将八进制地址值转换为十六进制数据。
 - 第 3 步：写入 BITSET 指令或 BITRST 指令，指定想要置位或复位的位号。位号按八进制编号，数据表第 1 个位的位号是“0”。
- 小技巧：每个 R 寄存器包含 16 位，因此，数据表的第 1 个寄存器位号范围是 0-17（八进制）。例如，假定数据表长度是 6 字=（6×16）位=96 位（十进制），或 140 位（八进制）。位的编号范围是 0-137。当指定的位超出这个范围时，SP53 将被置为 0N。

操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
八进制地址	0	0-77777

举例，假设数据表起始地址是 R3000，2 字长，如右图所示。每个字包含 16 位，或 0-17（八进制）。要将第二个字的位 12 置位，其八进制编号为 14。计算其在数据表中的编号 17+14=34（八进制），下面的程序演示了如何将这个位置位。



下面的例子中，使用 I0 来触发 BITSET 指令。首先，将数据表长度值（2 字）读入数据堆栈的第 1 级中。然后，将数据表的起始地址读入累加器中。由于 R3000 是一个八进制地址值，我们使用 LDR 指令，将地址读入累加器的同时还将其转换成了十六进制数据。最后，写入 BITSET（或 BISTRST）指令，指定位在数据表中的八进制编号（位 34）。



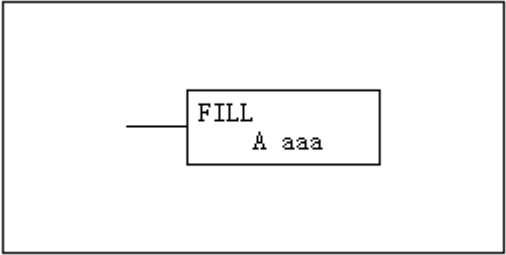
语句视图

```
LD I0
LDS K2
LDR 03000
BITSET 034
```

12.7 同一数据成组写入指令（FILL）

输入 FILL 指令将同一个数据（R 寄存器或 4 字常数）写入到一组（最多 255 个）R 寄存器中。

指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中。下面是实现这一指令功能的步骤：

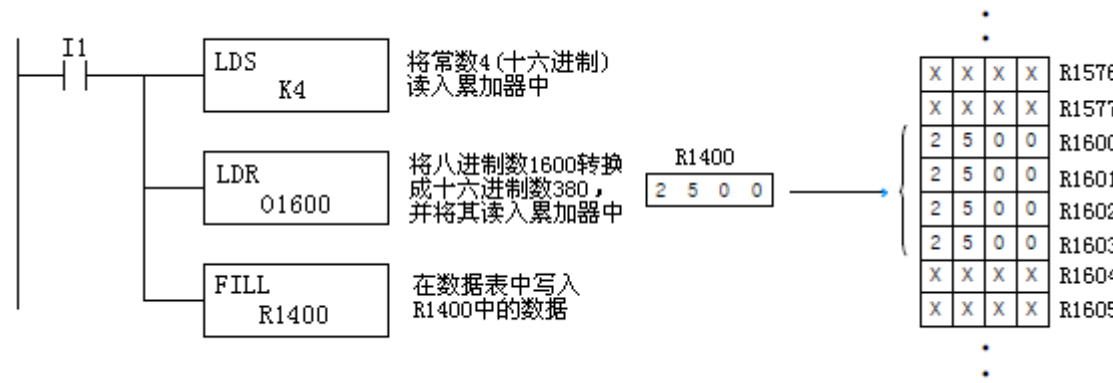


- 第 1 步：将被写入数据的 R 寄存器的数目读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。
- 第 2 步：将被写入数据表的起始 R 寄存器地址写入累加器，这个数必须是十六进制数。
- 第 3 步：写入 FILL 指令，并指定要写入的数据。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）
常数	K	0-FFFF

下面的例子中，当 I1 为 ON 时，LDS 指令将常数 K4 读入累加器中，此常数指定了被写入数据的 R 寄存器数目。LDR 指令将八进制地址值 1600 写入累加器，并将累加器中原有的数据常数 4（十六进制）压入堆栈的第 1 级中，这个地址值是被写入数据表的起始寄存器地址。FILL 指令指定要写入的数据并将其写入到数据表中。

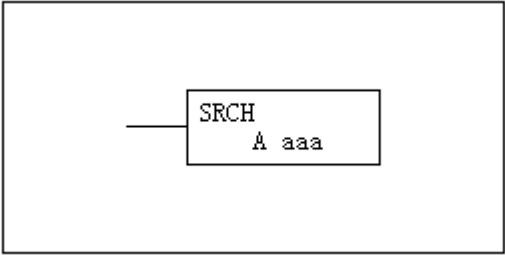


语句视图

LD I1
LDS K4
LDR 01600
FILL R1400

12.8 同一数据检索指令（SRCH）

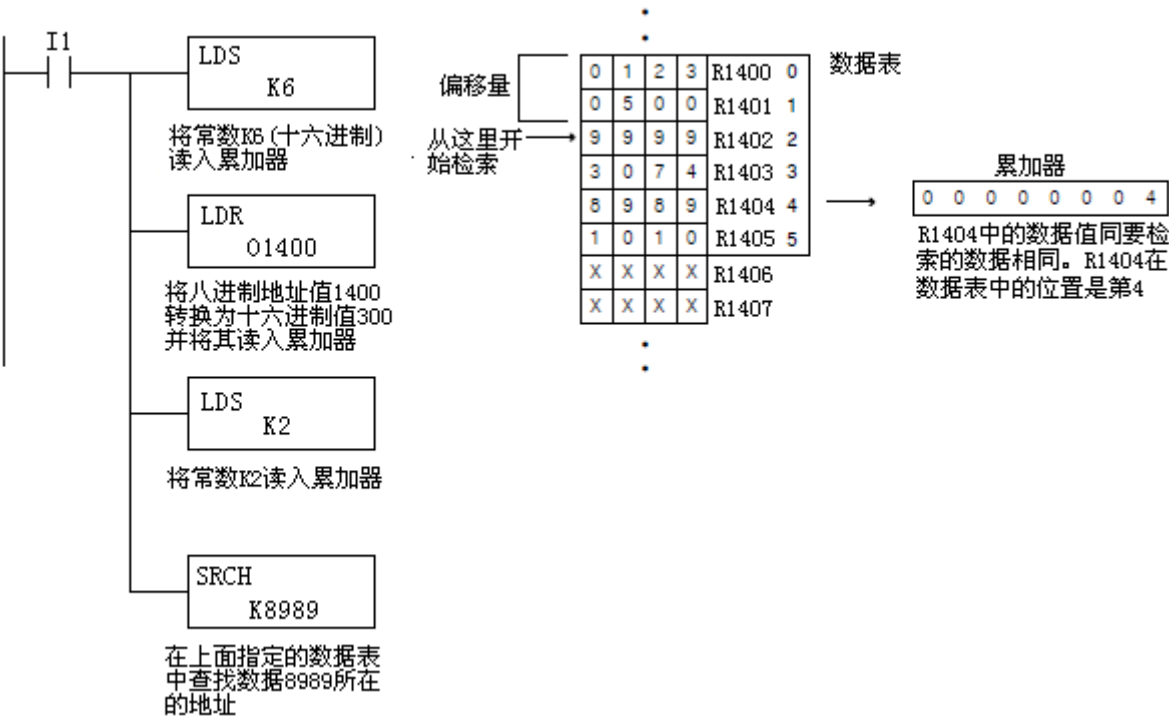
输入 SRCH 指令用于在一组（最多 255 个）R 寄存器中检索一个指定的数据。指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级、第 2 级和累加器中。下面是实现此指令功能的步骤：



- 第 1 步：将要被检索的数据表的长度读入数据堆栈的第 2 级中，这个数必须是十六进制数，范围是 0-FF。
- 第 2 步：将要被检索的数据表的起始 R 寄存器地址读入数据堆栈的第 1 级中，这个数必须是十六进制数。
- 第 3 步：将起始寄存器地址的偏移量读入累加器中，偏移量指定从何处开始检索，必须是十六进制数。
- 第 4 步：写入 SRCH 指令，并指定要检索的第一个数据。
- 执行结果：指令执行结果是包含被检索数据的寄存器在数据表中相对于起始地址的偏差量被写入累加器中。如果被检索数据所在的地址在偏移量指定检索的区域之外或是未找到要检索的数据，SP53 被置 ON。如果未找到要检索的数据，累加器中将被写入“0”。结果是一个十六进制数。
- 小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	0-FFFF

下面的例子中，当 I1 为 ON 时，常数 K6 被装入累加器，这个数据指定了数据表的长度，当第二个读入指令执行时，常数 K6 被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器，这是数据表的起始地址，当第三个读入指令执行时，这个地址值被压入数据堆栈的第 1 级中，第 1 级中的原有数据 K6 被压入第 2 级中。第三个读入指令将常数 K2 读入累加器，这个数据是数据表起始地址的偏移量。SRCH 指令中指定要检索的数据。如果被检索的数据等于某个寄存器中的数据，那这个寄存器的位置相对于数据表起始地址的偏差值被读入累加器中。

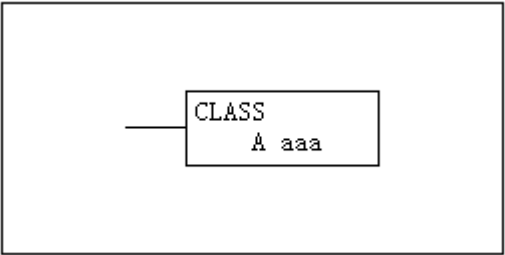


语句视图

```
LD I1
LDS K6
LDR 01400
LDS K2
SRCH K8989
```

12.9 数据级别分类指令（CLASS）

输入 CLASS 指令用于在指定的数据表中检索第一个大于指定值的数据在数据表中的位置。此指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：

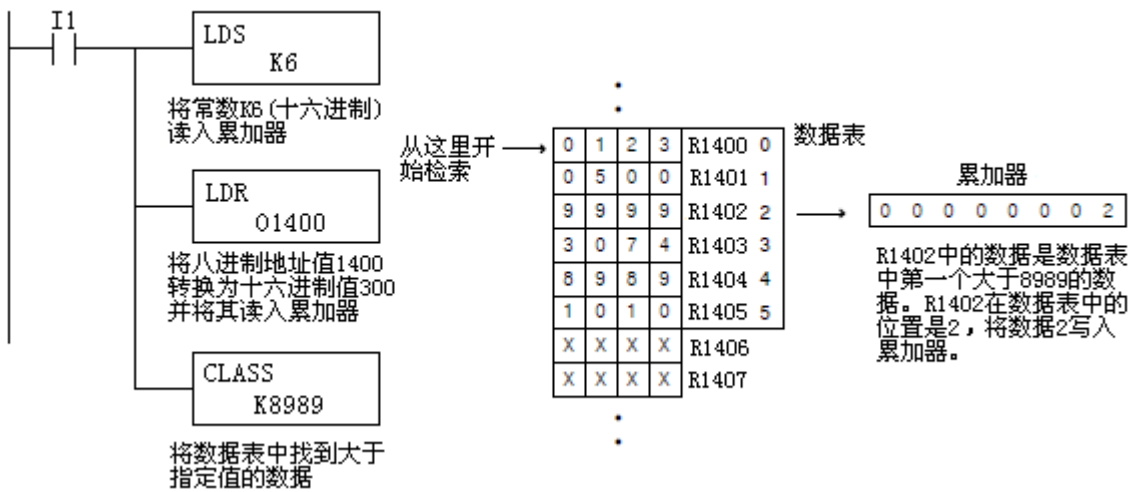


注意：本指令没有偏移量，与 SRCH 指令不同。

- 第 1 步：将要被检索的数据表的长度（最多 255 个寄存器）读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。
- 第 2 步：将数据表的起始寄存器地址读入累加器，这个数必须是十六进制数。
- 第 3 步：写入 CLASS 指令，指定要比较的数据。
- 执行结果：第一个包含比指定数据大的寄存器在数据表中的相对于起始地址的偏差值被写入累加器中。如果大于指定数据的数据没有找到，则在累加器中写入 0，结果是一个十六进制数。
- 小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	0-FFFF

下面的例子中，当 I1 为 ON 时，常数 K6 被读入累加器，这个数据指定了数据表的长度，当第二个读入指令执行时，常数 K6 被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器，这是数据表的起始地址。CLASS 指令中指定要比较的数据。如果寄存器中的数据大于要比较的数据，在这个寄存器在数据表中的相对于起始地址的偏差值被写入累加器中。如果数据表中的数据没有大于指定的数据，则在累加器中写入“0”，将 SP53 设置为 ON。

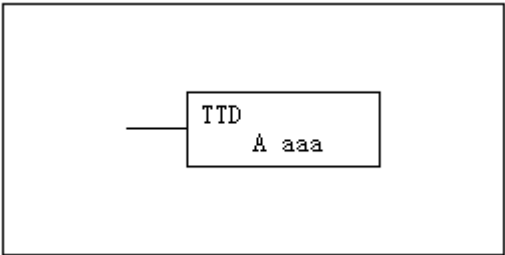


语句视图

```
LD I1
LDS K6
LDR O1400
CLASS K8989
```


12.10 指针加法取出指令（TTD）

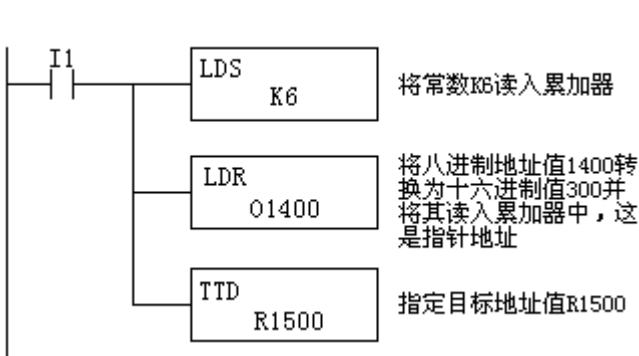
输入 TTD 指令将指定数据表中一个数据取出放入一个寄存器中，同时将指针值加 1。数据表的第一个寄存器存放的是一个指针值，用于指示将要被取出数据的寄存器。条件成立时，此指令每次扫描运行一次。当指针值等于数据表的最后一个地址时将复位为“1”。此指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：



- 第 1 步：将数据表的长度读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。
- 第 2 步：将数据表的起始寄存器地址读入累加器（注意，起始寄存器被用作指针），这个数必须是十六进制数。
- 第 3 步：写入 TTD 指令，指定目标寄存器地址。
- 小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。
- 小技巧：当条件成立时，此指令每次扫描执行一次。如果不想指令每次扫描都执行，可对条件使用一次扫描输出指令（PD）。
- 小技巧：指针中的值应设置为数据表操作开始的值。可使用特殊线圈 SP0 或一次扫描输出指令（PD），只在一次扫描中设置这个指针值而不影响指令的执行。

操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）

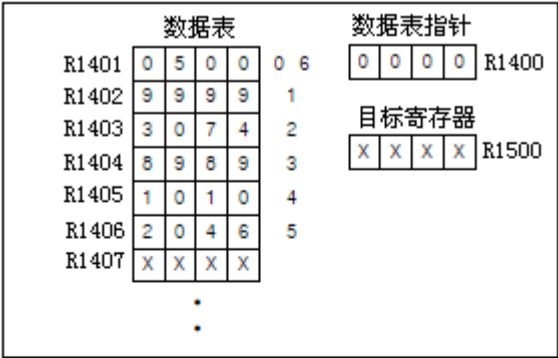
下面的例子中，当 I1 为 ON 时，第一个读入指令将常数 K6 读入累加器中，此值指定了数据表的长度，当第二个读入指令执行时，此值被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400 (R1400) 读入累加器中，R1400 是数据表的起始地址，注意，R1400 被用作指针地址，不是数据源的一部分。写入 TTD 指令，指定目标寄存器地址 R1500。TTD 指令每执行一次，数据表指针值就加“1”。



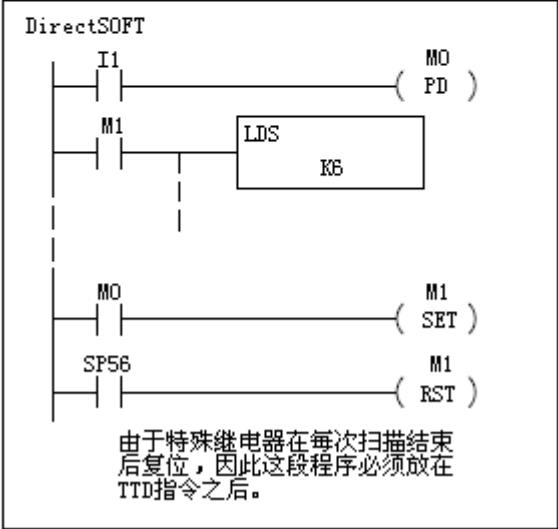
语句视图

```
LD I1
LDS K6
LDR 01400
TTD R1500
```

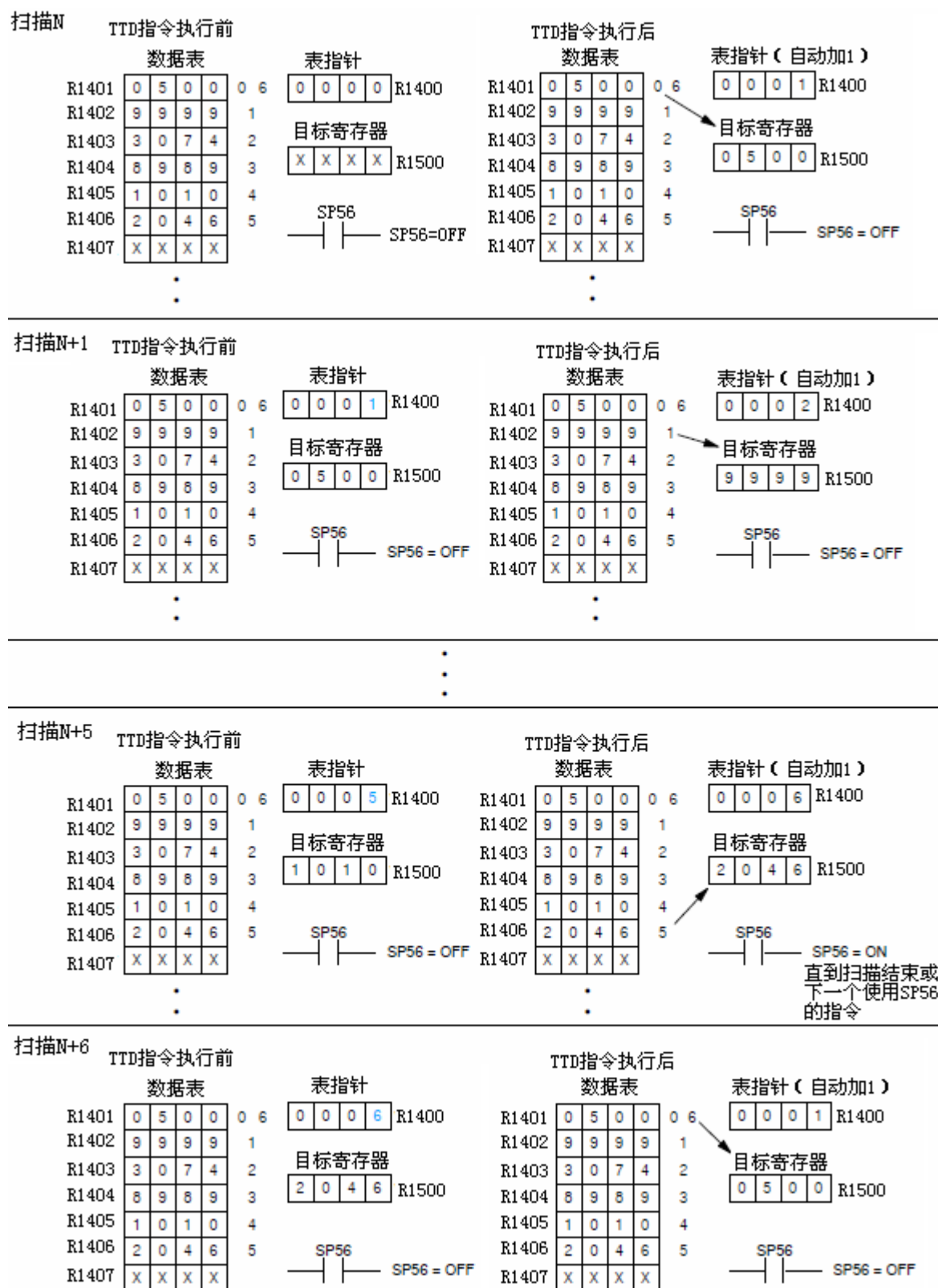
了解数据表是怎么编号的很重要，查看右边的图例，你会发现，当指针值等于 0 时，第一个寄存器地址 R1401 中的数据将被使用，而当指针值等于 6 时，R1401 中的数据也被使用。这是因为只有在 TTD 指令第一次执行之前，指针值才等于 0，第一次执行后，指针的范围就是从 1 到 6，然后再复位为 1。



右边的例子中，使用了一个常开输入点 (I1) 来控制执行。CPU 扫描速度非常快，由于指针值是自动增加的，因此数据表循环也非常快，如果不想这样，可以使用右图的方法，这样当数据表循环一次后就停下来。这段程序不是必须的，可以选用。

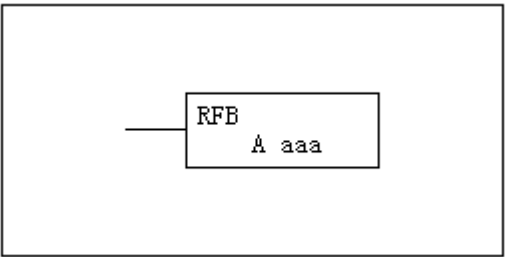


下面的图示演示了上页的程序例子中每次扫描后指针值如何从 0-6 自动循环, 然后从 1-6 循环, 注意, 扫描结束后, SP56 才被置为 ON。



12.11 指针减法取出指令（RFB）

输入 RFB 指令将指定数据表底部的一个数据取出放入一个寄存器中，同时将指针值减 1。数据表的第一个寄存器存放的是一个指针值，用于指示将要被取出数据的寄存器。条件成立时，此指令每次扫描运行一次。当指针值等于 0 时，指令停止运行。此指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：



- 第 1 步：将数据表的长度读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。
- 第 2 步：将数据表的起始寄存器地址读入累加器（注意，起始寄存器被用作指针），这个数必须是十六进制数。
- 第 3 步：写入 RFB 指令，指定目标寄存器地址。
- 小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。
- 小技巧：当条件成立时，此指令每次扫描执行一次。如果不想指令每次扫描都执行，可对条件使用一次扫描输出指令（PD）。
- 小技巧：指针中的值应设置为数据表操作开始的值。可使用特殊线圈 SP0 或一次扫描输出指令（PD），只在一次扫描中设置这个指针值而不影响指令的执行。

操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）

下面的例子中，当 I1 为 ON 时，第一个读入指令将常数 K6 读入累加器中，此值指定了数据表的长度，当第二个读入指令执行时，此值被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器中，R1400 是数据表的起始地址，注意，R1400 被用作指针地址，不是数据源的一部分。写入 RFB 指令，指定目标寄存器地址 R1500。RFB 指令每执行一次，数据表指针值就减“1”。

I1

LDS
K6

LDR
01400

RFB
R1500

将常数K6读入累加器

将八进制地址值1400转换为十六进制值300并
将其读入累加器中，这
是指针地址

指定目标地址值R1500

语句视图

LD I1
LDS K6
LDR 01400
RFB R1500

了解数据表是怎么编号的很重要，查看右边的图例，你会发现，当指针值等于 1 时，第一个寄存器地址 R1401 中的数据将被使用，当指针值等于 2 时，R1402 中的数据将被使用，以此类推。

数据表

R1401	0	5	0	0	1
R1402	9	9	9	9	2
R1403	3	0	7	4	3
R1404	8	9	8	9	4
R1405	1	0	1	0	5
R1406	2	0	4	6	6
R1407	X	X	X	X	

数据表指针

0	0	0	0	R1400

目标寄存器

X	X	X	X	R1500

右边的例子中，使用了一个常开输入点（I1）来控制执行。CPU 扫描速度非常快，由于指针值是自动减少的，因此数据表循环也非常快，如果不想这样，可以如右图的所示的方法使用一次扫描输出指令，这样当输入由 OFF→ON 时，数据仅读入一次。

I1

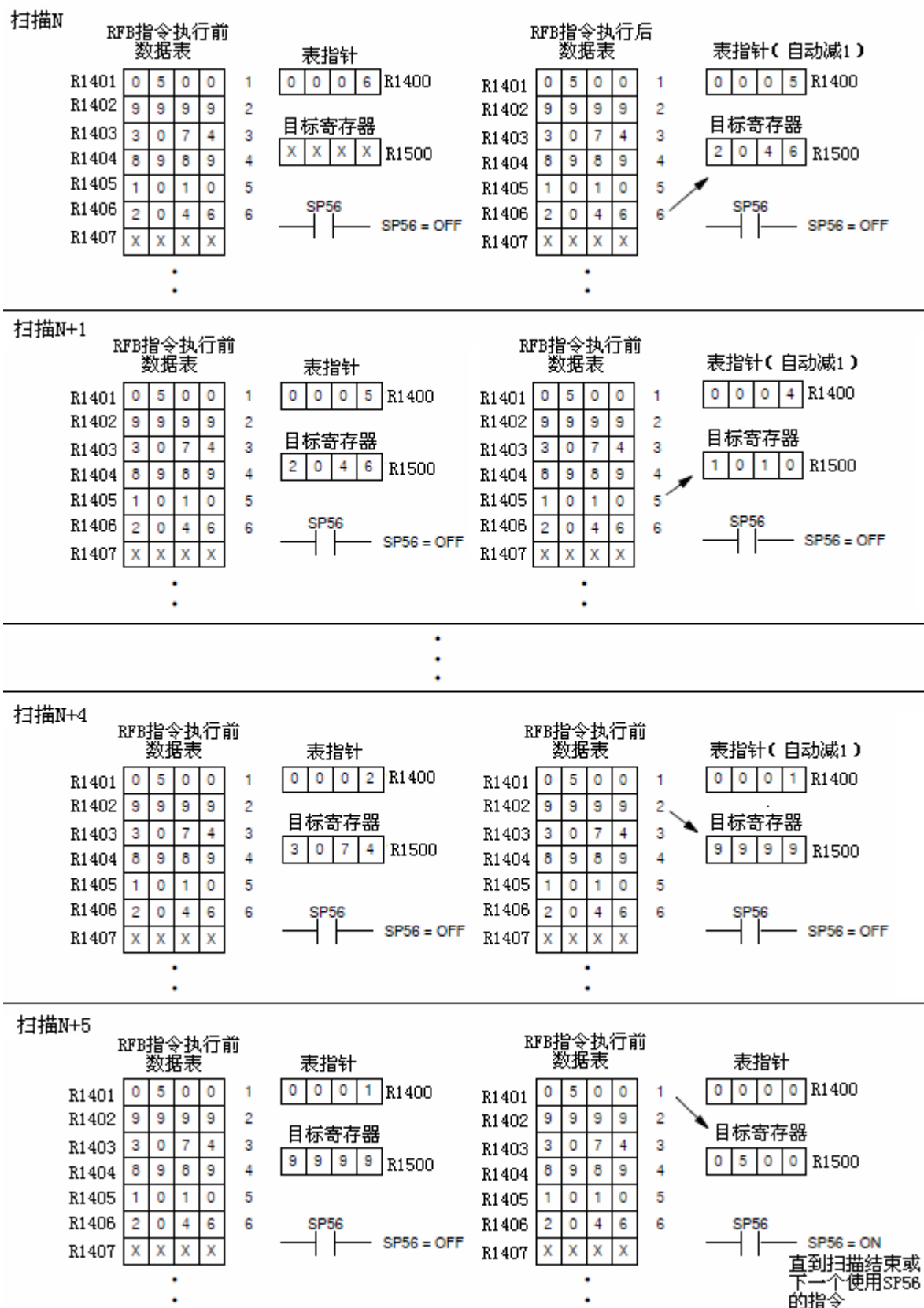
MO
PD

LDS
K6

LDR
01400

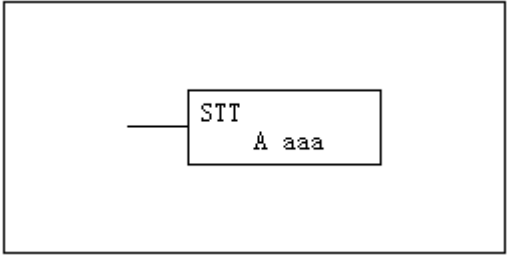
261

下面的图示演示了上页的程序例子中每次扫描后指针值如何从 6 到 0 自动循环, 注意, 扫描结束后, SP56 才被置为 ON。



12.12 指针加法存入指令（STT）

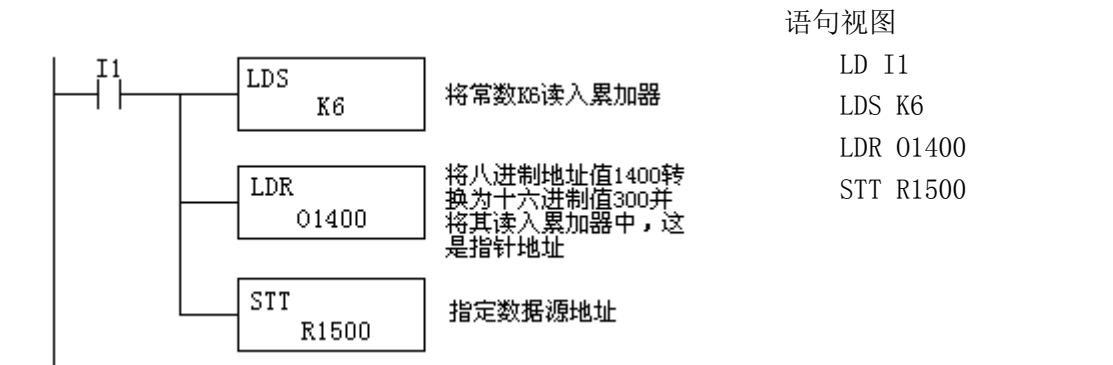
输入 STT 指令将一个常数或指定寄存器中的数据存入一个指定的数据表中，同时将数据表指针加 1。当指针值到达数据表的末端，就复位为“1”。数据表的第一个寄存器存放的是指针值，用于指示将要被存入数据的寄存器。条件成立时，此指令每次扫描运行一次。当指针值等于数据表的最后一个地址时将复位为“1”。此指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：



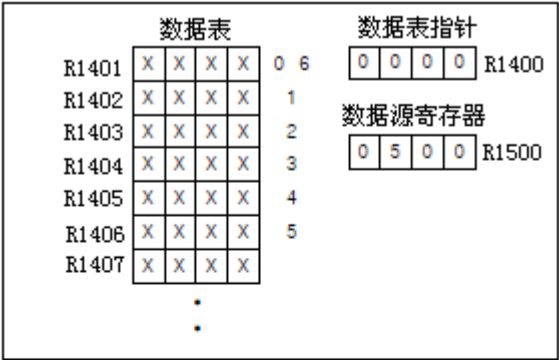
- 第 1 步：将数据表的长度读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。
- 第 2 步：将数据表的起始寄存器地址读入累加器（注意，起始寄存器被用作指针），这个数必须是十六进制数。
- 第 3 步：写入 STT 指令，指定数据源寄存器地址。
- 小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。
- 小技巧：当条件成立时，此指令每次扫描执行一次。如果不想指令每次扫描都执行，可对条件使用一次扫描输出指令（PD）。
- 小技巧：指针中的值应设置为数据表操作开始的值，而且它应小于等于数据表的长度。比如，如果数据表长度是 6 字长，那么指针的范围应为 0-6，如果超出这个范围，数据将不移动。可使用一次扫描输出指令（PD），只在一次扫描中设置这个指针值而不影响指令的执行。

操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	0-FFFF

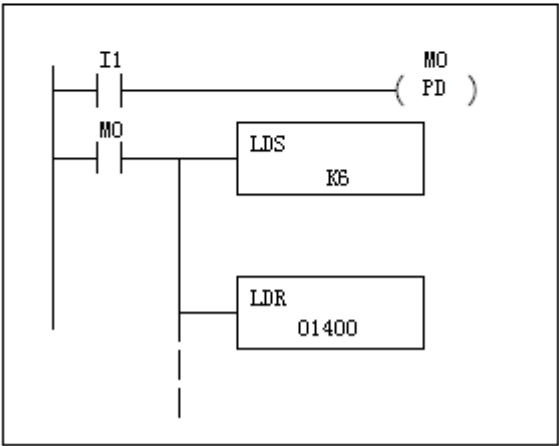
下面的例子中，当 I1 为 ON 时，第一个读入指令将常数 K6 读入累加器中，此值指定了数据表的长度，当第二个读入指令执行时，此值被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器中，R1400 是数据表的起始地址，注意，R1400 被用作指针地址。写入 STT 指令，指定数据源寄存器地址 R1500。STT 指令每执行一次，数据表指针值就加“1”。



了解数据表是怎么编号的很重要，查看右边的图例，你会发现，当指针值等于 0 时，源数据存入第一个寄存器 R1401 中，而当指针值等于 6 时，源数据也存入 R1401 中。这是因为只有在 STT 指令第一次执行之前，指针值才等于 0，第一次执行后，指针的范围就是从 1 到 6，然后再复位为 1。

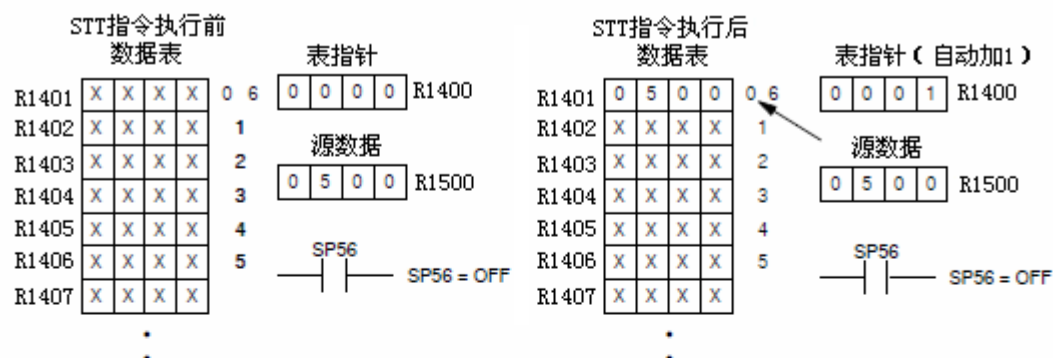


右边的例子中，使用了一个常开输入点（I1）来控制执行。CPU 扫描速度非常快，由于指针值是自动增加的，因此源数据存入数据表也非常快，如果不想这样，可以使用右图的方法，使用一次扫描输出指令，这样当输入由 OFF→ON 时，数据仅读入一次。

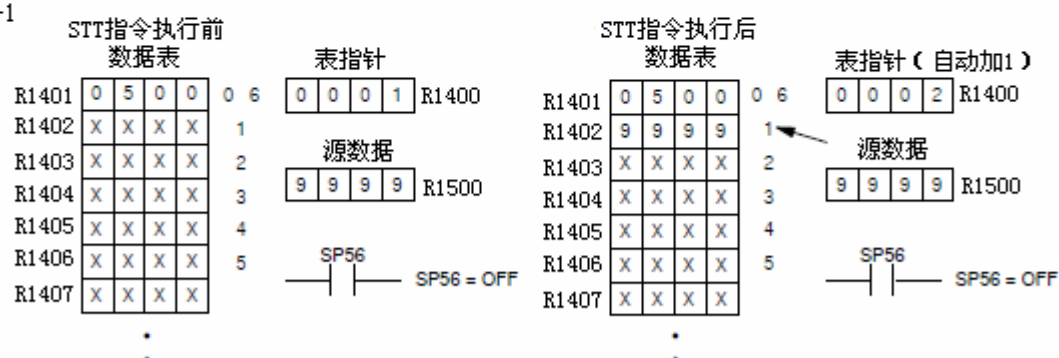


下面的图示演示了上页的程序例子中每次扫描后指令的执行结果，指针值如何从 0 到 6 自动循环，然后从 1 到 6 循环，注意，扫描结束后，SP56 才被置为 ON。本例假设有另外一段程序用来改变数据源 R1500 中的数据，并且其执行优先级高于 STT 指令。

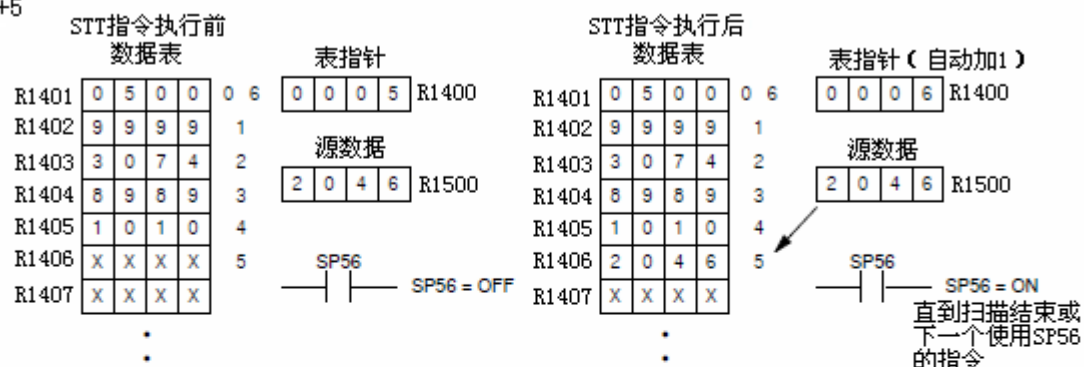
扫描N



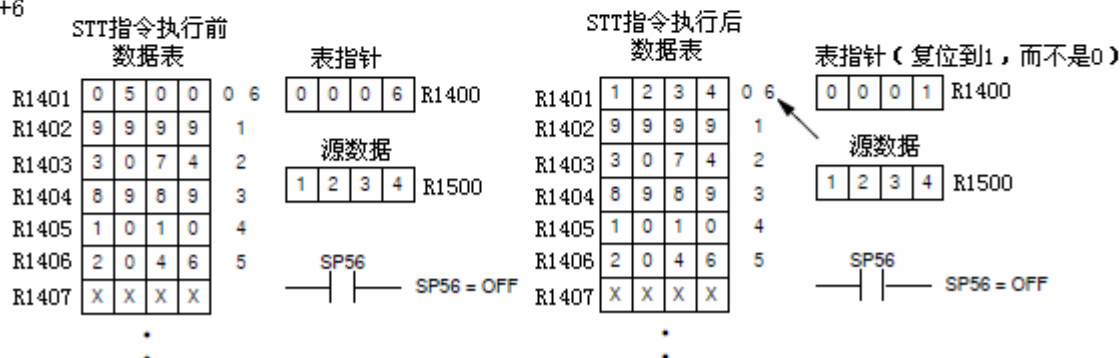
扫描N+1



扫描N+5

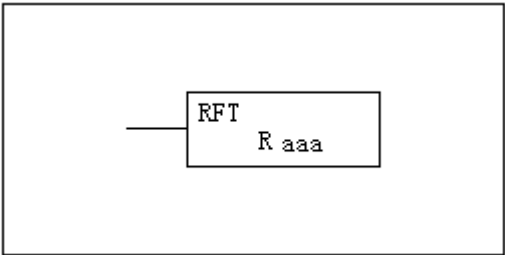


扫描N+6



12.13 堆栈上托取出指令（RFT）

输入 RFT 指令将指定数据表中的一个数据取出放入一个寄存器中。当数据表的一个数据移出后，其它数据都向上移动一个地址。数据表的第一个寄存器中存放的是数据表的长度，RFT 指令每运行一次，数据表的长度就减 1。如果数据表的长度等于 0 或大于数据表长度的最大值（由数据堆栈的第 1 级中的数据指定），指令将不执行，SP56 将被置位为 ON。

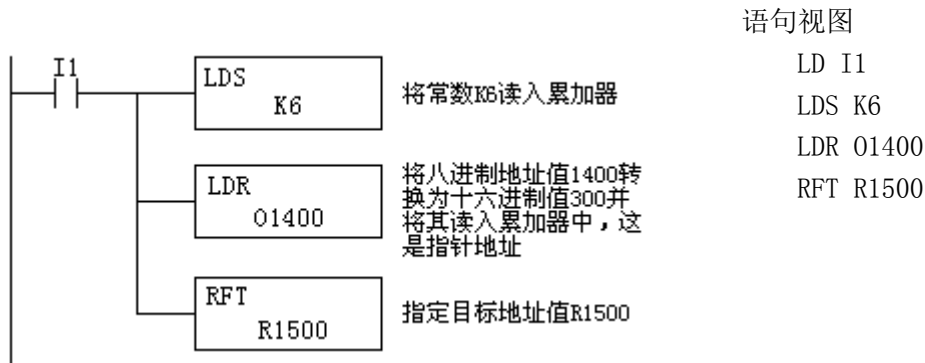


条件成立时，此指令每次扫描运行一次。此指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：

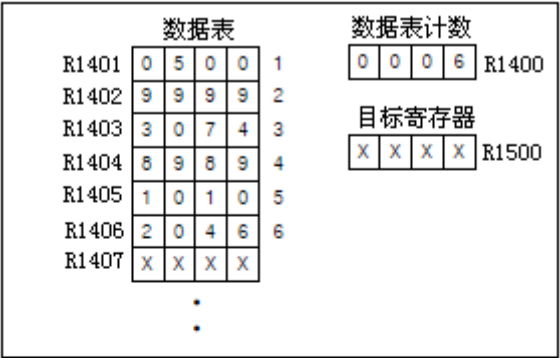
- 第 1 步：将数据表的长度读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。
- 第 2 步：将数据表的起始寄存器地址读入累加器（注意，起始寄存器中存放的是数据表的长度计数值），这个数必须是十六进制数。
- 第 3 步：写入 RFT 指令，指定目标寄存器地址。
- 小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。
- 小技巧：当条件成立时，此指令每次扫描执行一次。如果不想指令每次扫描都执行，可对条件使用一次扫描输出指令（PD）。
- 小技巧：数据表长度计数值应设置为指示数据表操作的起始点，而且它应小于等于数据表的长度。比如，如果数据表长度是 6 字长，那么数据表长度的计数值的范围应为 1-6，如果超出这个范围或等于 0，数据将不移动。可使用一次扫描输出指令（PD），只在一次扫描中设置这个指针值而不影响指令的执行。

操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）

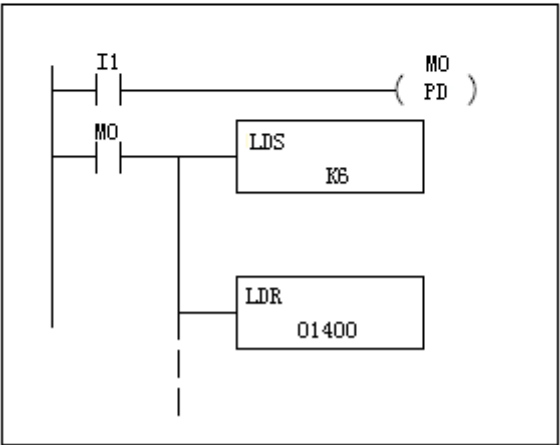
下面的例子中，当 I1 为 ON 时，第一个读入指令将常数 K6 读入累加器中，此值指定了数据表的长度，当第二个读入指令执行时，此值被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器中，R1400 是数据表的起始地址。写入 RFT 指令，指定目标寄存器地址 R1500。RFT 指令每执行一次，数据表长度值就减“1”。



了解数据表是怎么编号的很重要，查看右边的图例，你会发现，数据表编号从数据表顶端开始。例如，如果数据表长度计数值从 6 开始，则指令执行时，所有 6 个地址的值都会变动。

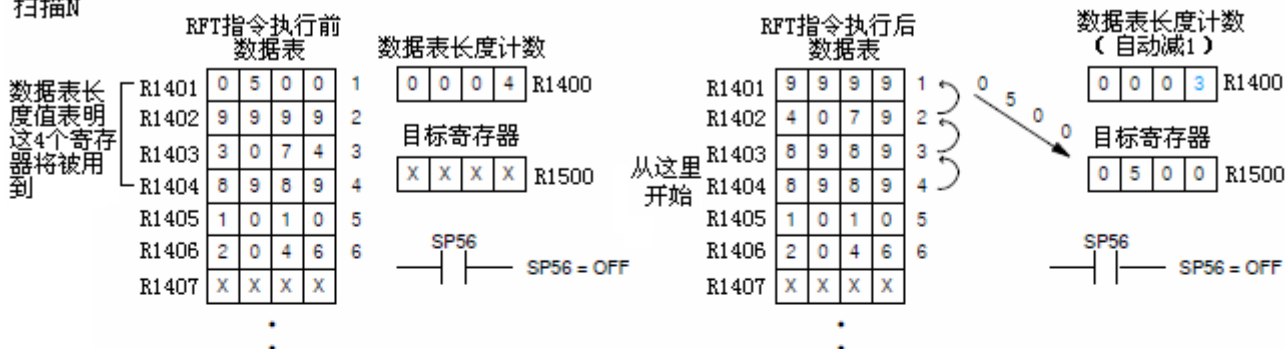


右边的例子中，使用了一个常开输入点（I1）来控制执行。CPU 扫描速度非常快，由于数据表长度计数值是自动减少的，因此数据表数据移出也非常快，如果不想这样，可以使用右图的方法，使用一次扫描输出指令，这样当输入由 OFF→ON 时，仅取出一个数据。

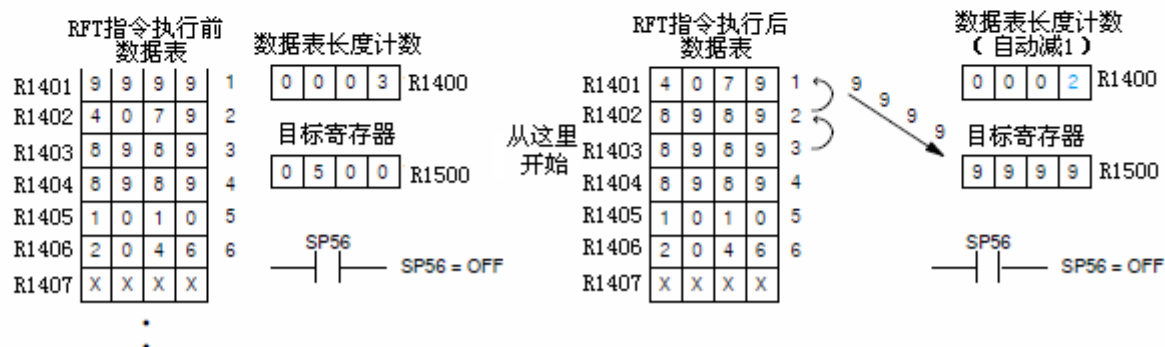


下面的图示演示了上页的程序例子中每次扫描后指令的执行结果，本例中，设置最初的数据表长度计数值为 4（注意，可将数据表长度计数值设置为任何不大于数据表长度的数值）。数据表长度计数值从 4 到 0 逐渐减少，指令每执行一次，就减少“1”。注意，数据表的最后两个位置，5 和 6，没有移动，还有，当长度计数值等于 0 时，SP56 才被置于 ON。

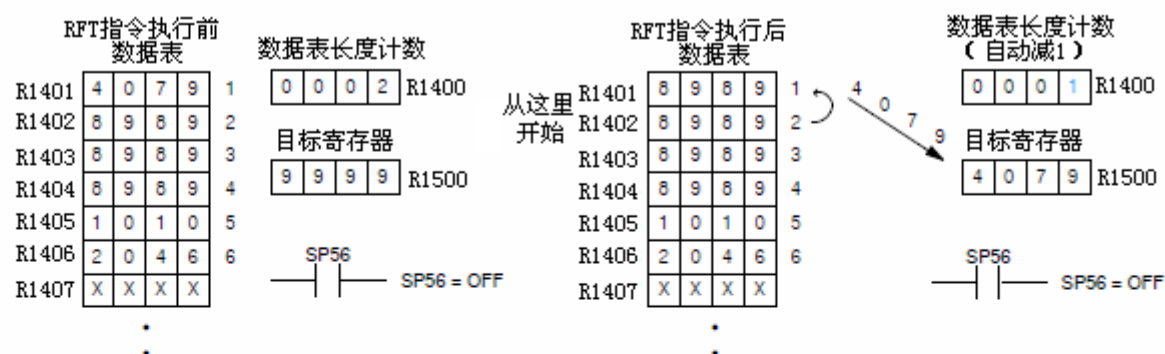
扫描N



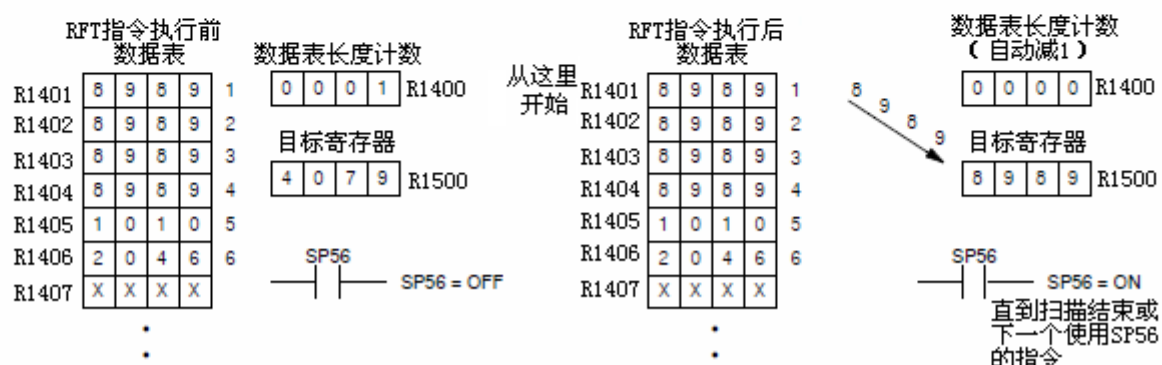
扫描N+1



扫描N+2



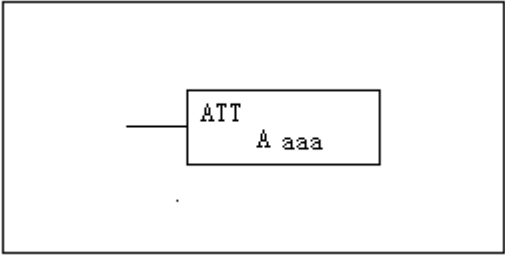
扫描N+3



12.14 堆栈下推存入指令（ATT）

输入 ATT 指令将一个指定寄存器中的数据存入一个指定的数据表中，数据表的其它数据均向下移动一个地址。

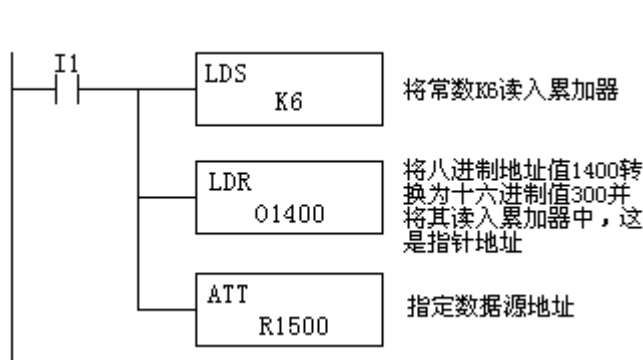
条件成立时，此指令每次扫描运行一次。此指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：



- 第 1 步：将数据表的长度读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。
- 第 2 步：将数据表的起始寄存器地址读入累加器（注意，起始寄存器中存放的是数据表计数值），这个数必须是十六进制数。
- 第 3 步：写入 ATT 指令，指定源数据寄存器地址。
- 小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。
- 小技巧：当条件成立时，此指令每次扫描执行一次。如果不想指令每次扫描都执行，可对条件使用一次扫描输出指令（PD）。
- 小技巧：数据表计数值应设置为指示数据表操作的起始点，而且它应小于等于数据表的长度。比如，如果数据表长度是 6 字长，那么数据表计数值的范围应为 1-6，如果超出这个范围或等于 0，数据将不移动。可使用一次扫描输出指令（PD），只在一次扫描中设置这个指针值而不影响指令的执行。

操作数类型	NK1 范围	
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	所有（附录 1）

下面的例子中，当 I1 为 ON 时，第一个读入指令将常数 K6 读入累加器中，此值指定了数据表的长度，当第二个读入指令执行时，此值被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器中，R1400 是数据表的起始地址，里面存放的数据表计数值。写入 ATT 指令，指定源数据寄存器地址 R1500。ATT 指令每执行一次，数据表计数值就加“1”。



语句视图

```
LD I1
LDS K6
LDR 01400
ATT R1500
```

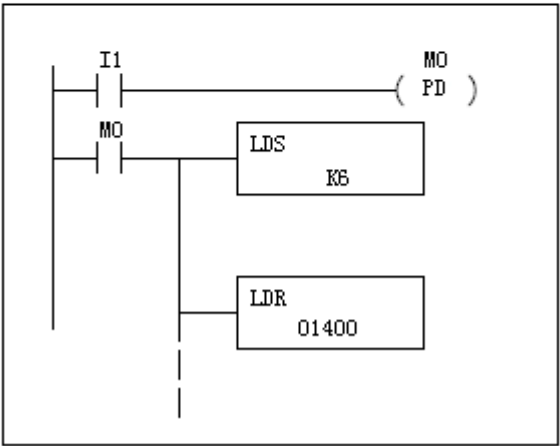
对于 ATT 指令，数据表计数值确定了指令停止执行之前还要将数据读入数据表的次数，因此，数据表计数值有助于理解系统是如何运用它来控制指令执行的。

右边的例子中，如果数据表计数值被设置为 2，数据表长度是 6 字，那么指令停止执行前还要执行 4 次，很容易计算出：

数据表长度－数据表计数值=执行次数

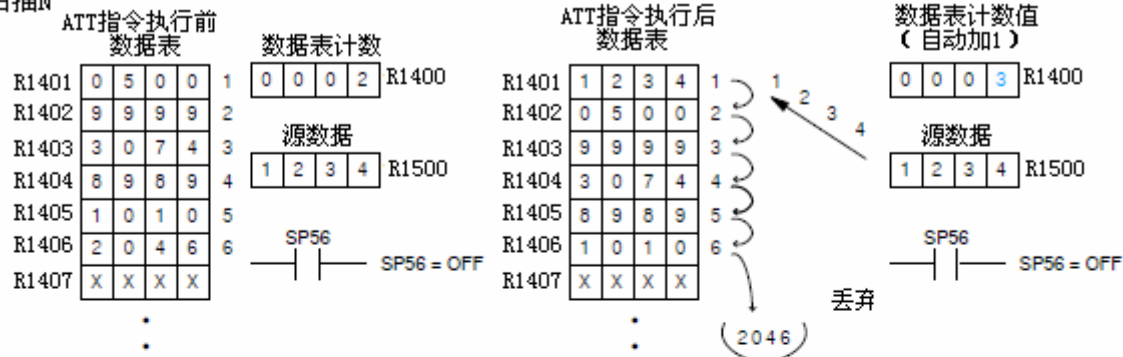
数据表					数据表计数值					
R1401	0	5	0	0	1	<table><tr><td>0</td><td>0</td><td>0</td><td>2</td></tr></table> R1400	0	0	0	2
0	0	0	2							
R1402	9	9	9	9	2					
R1403	3	0	7	4	3					
R1404	8	9	8	9	4					
R1405	1	0	1	0	5					
R1406	2	0	4	6	6					
R1407	X	X	X	X		<table><tr><td>X</td><td>X</td><td>X</td><td>X</td></tr></table> R1500	X	X	X	X
X	X	X	X							
⋮										
					6-2=4					

右图的例子中使用了一个常开输入点（I1）来控制执行。CPU 扫描速度非常快，由于数据表计数值是自动增加的，因此源数据存入数据表也非常快，如果不想这样，可以使用右图的方法，使用一次扫描输出指令，这样当输入由 OFF→ON 时，数据仅读入一次。

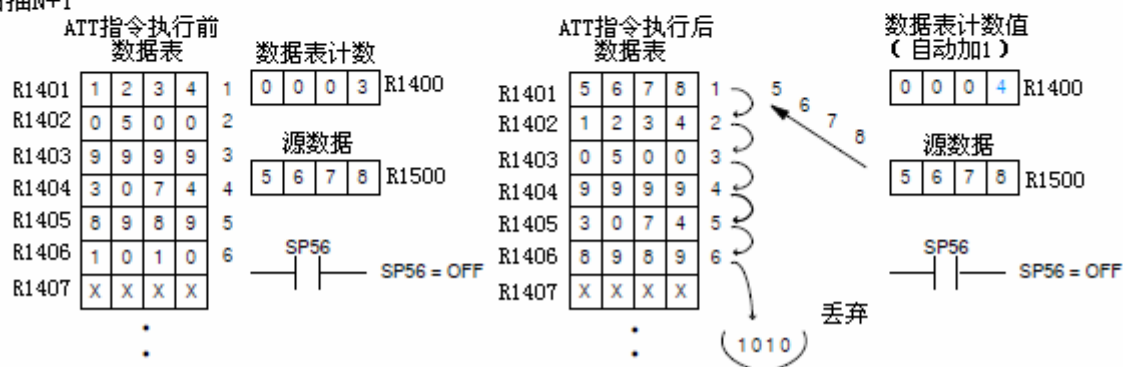


下面的图示演示了上页的程序例子中每次扫描后指令的执行结果，数据表计数值的初始值被设置为 2，指令执行后它的值自动从 2 增加到 6。注意，当数据表计数值等于 6，也就是等于数据表长度时，SP56 被设置为 ON。本例假设有另外一段程序用来改变数据源 R1500 中的数据，并且其执行优先级高于 ATT 指令。

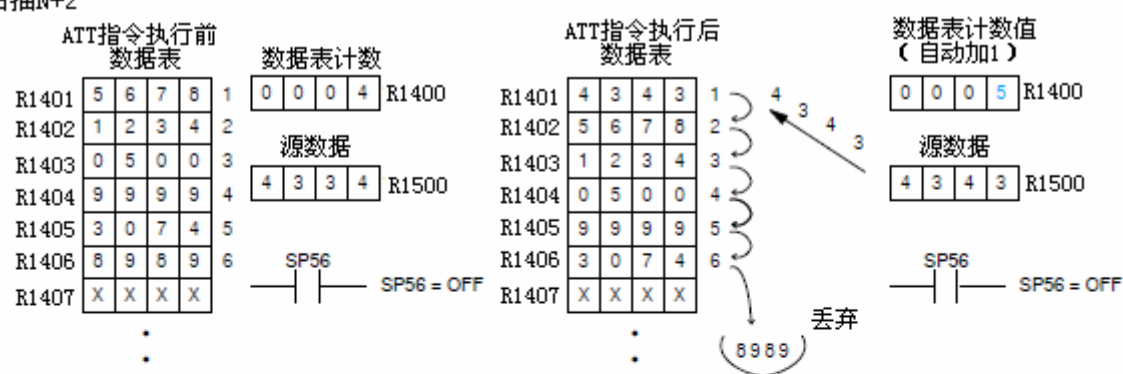
扫描N



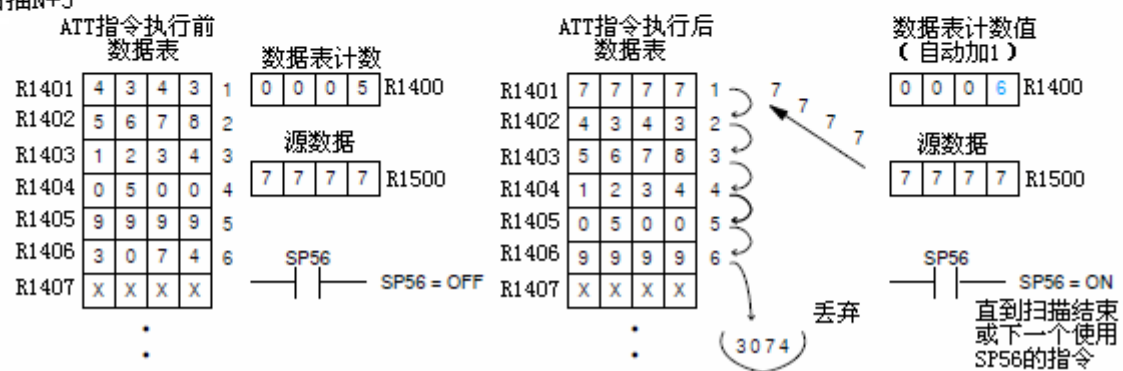
扫描N+1



扫描N+2

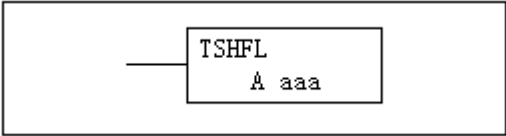


扫描N+3



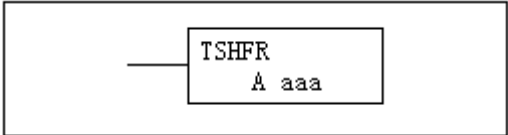
12.15 表左移指令（TSHFL）

输入 TSHFL 指令将 R 寄存器数据表中的所有位左移指定位数。

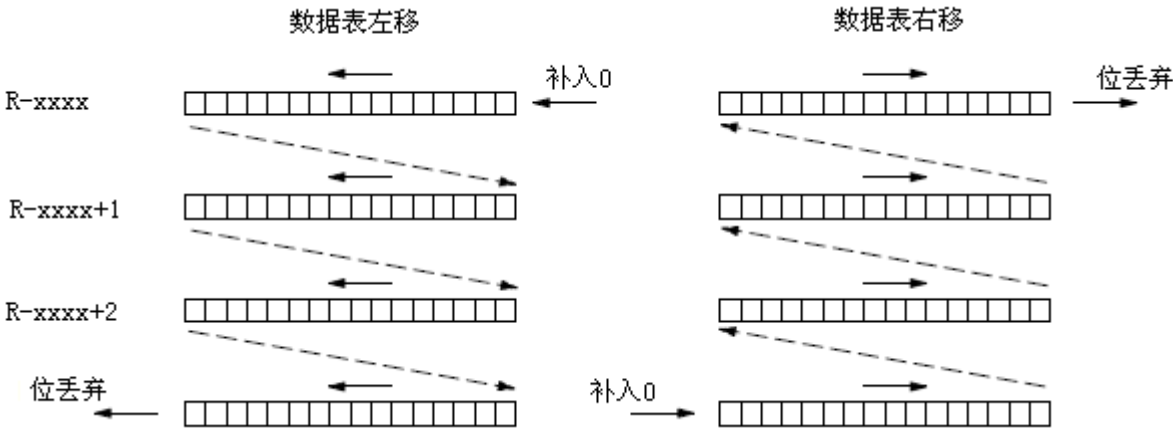


12.16 表右移指令（TSHFR）

输入 TSHFR 指令将 R 寄存器数据表中的所有位右移指定位数。



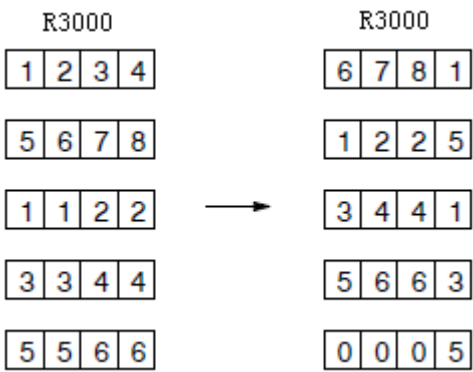
下面的说明同时适用于 TSHFL 指令和 TSHFR 指令，数据表就是一系列的 R 寄存器。这两个指令通过整个数据表连续移位。位移出字的边缘，进入相邻字的另一端中。在数据表的两端，位要么被丢弃，要么在数据表中补入 0。下图中数据表是 4 字长。



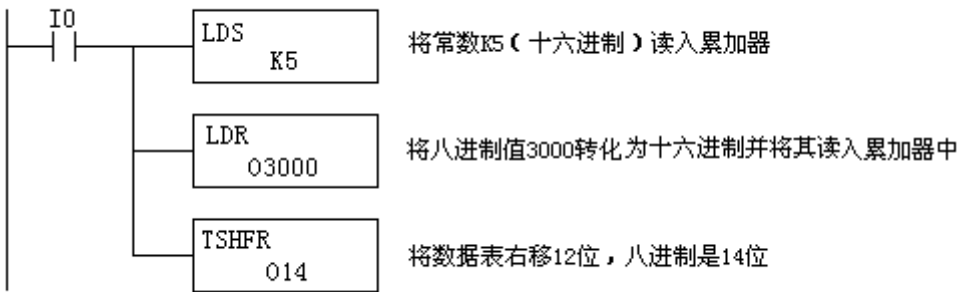
- 第 1 步：将数据表的长度（R 寄存器的个数）读入数据堆栈的第 1 级中。这个数必须是十六进制数，范围是 0-FF。
- 第 2 步：将数据表的起始地址读入累加器中，这个数必须是十六进制数，可使用 LDR 指令将八进制地址值转换为十六进制数据。
- 第 3 步：写入 TSHFL 和 TSHFR 指令，指定要移动的位数。这个数必须是八进制。
- 小技巧：每个 R 寄存器包含 16 位，因此，数据表的第 1 个寄存器位号范围是 0-17（八进制）。如果想移动 20 位，八进制就是 24 位。当指定要移动的位数超出数据表的位数时，SP53 将被置为 0N。如果最后移出的位为“1”，SP67 将被置为 0N。

操作数类型	NK1 范围	
A	aaa	
R 寄存器	R	所有（附录 1）
八进制地址	0	0-7777

右边的例子中，包含的是 BCD 数。假设要将数据表右移 3 个 BCD 字（12 位）。将 12 转换为八进制数 14。使用 TSHFR 指令并指定移动位数是 14，运行结果见右图。注意到 2-3-4 三个数字被丢弃，底部补入了三个 0。



下面的例子中假设 R3000-R3004 中的数据就如上图所示。使用 I0 来触发 TSHFR 指令。首先，将数据表长度值（5 字）读入堆栈的第 1 级中。其次，将数据表的起始地址读入累加器。由于 R3000 是一个八进制数，我们使用 LDR 指令将其转换为十六进制数。最后，写入 TSHFR 指令，并指定要移动的位数，十进制数是 12，八进制数是 14。

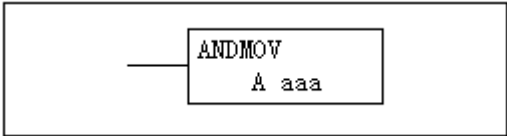


语句视图

```
LD I0
LDS K5
LDR 03000
TSHFR 014
```

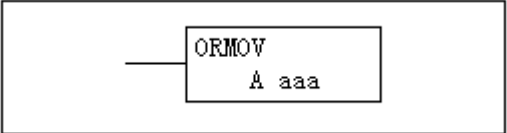
12.17 ACC 逻辑与传送指令（ANDMOV）

输入 ANDMOV 指令将一个数据表中的每个字同累加器中的数据逻辑与运算，然后将运算结果传送到指定地址。



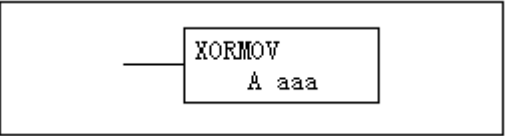
12.18 ACC 逻辑或传送指令（ORMOV）

输入 ORMOV 指令将一个数据表中的每个字同累加器中的数据逻辑或运算，然后将运算结果传送到指定地址。



12.19 ACC 逻辑异或传送指令（XORMOV）

输入 XORMOV 指令将一个数据表中的每个字同累加器中的数据逻辑异或运算，然后将运算结果传送到指定地址。



下面的说明同时适用于 ANDMOV 指令、ORMOV 和 XORMOV 指令，数据表就是一系列的 R 寄存器。这些指令将累加器中的内容与每个字进行逻辑运算，然后将运算结果传送到指定的地址中。

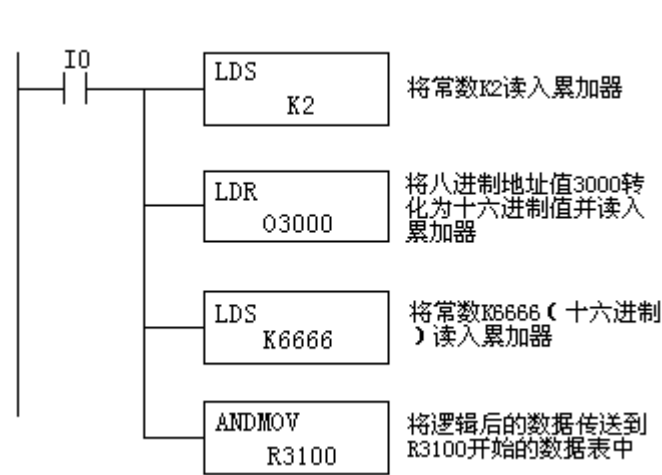
- 第 1 步：将数据表的长度（R 寄存器的个数）读入数据堆栈的第 2 级中。这个数必须是十六进制数，范围是 0-FF。
- 第 2 步：将数据表的起始地址读入数据堆栈的第 1 级中，这个数必须是十六进制数，可使用 LDR 指令将八进制地址值转换为十六进制数据。
- 第 3 步：将要同数据表中数据逻辑运算的 BCD/十六进制数读入累加器。
- 第 4 步：写入 ANDMOV、ORMOV、XORMOV 指令，指定目标数据表地址，这个新数据表长度自动等于原始数据表。

操作数类型	NK1 范围
A	aaa
R 寄存器	R 所有（附录 1）

右边的例子中，将 R3000 开始的两字数据表同 K6666 逻辑与并将结果传送到以 R3100 开头的



下面的程序完成了上例中的 ANDMOV 操作，假设 R3000-R3001 中的数据已经存在。首先，将数据表长度值（2 字）读入累加器中。其次，将数据表的起始地址读入累加器中，使用 LDR 指令。然后，把要与数据表数据逻辑与的数据读入累加器。在 ANDMOV 指令中，指定目标数据表的起始地址 R3100。



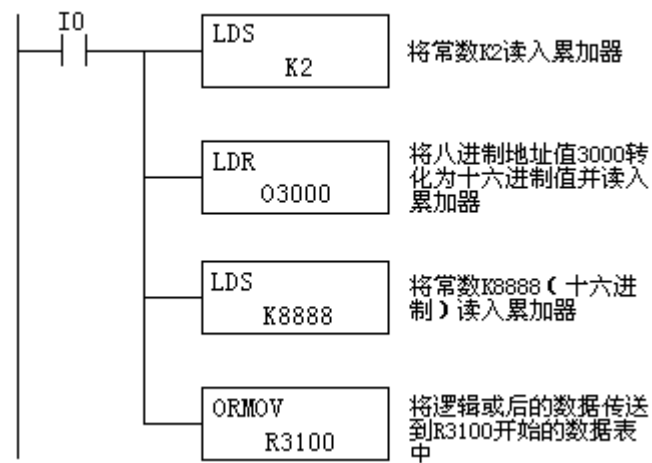
语句视图

```
LD I0
LDS K2
LDR 03000
LDS K6666
ANDMOV R3100
```

右边的例子中，将 R3000 开始的两字数据表同 K8888 逻辑或并将结果传送到以 R3100 开头的



下面的程序完成了上例中的 ORMOV 操作，假设 R3000-R3001 中的数据已经存在。首先，将数据表长度值（2 字）读入累加器中。其次，将数据表的起始地址读入累加器中，使用 LDR 指令。然后，把要与数据表数据逻辑或的数据读入累加器。在 ORMOV 指令中，指定目标数据表的起始地址 R3100。



语句视图

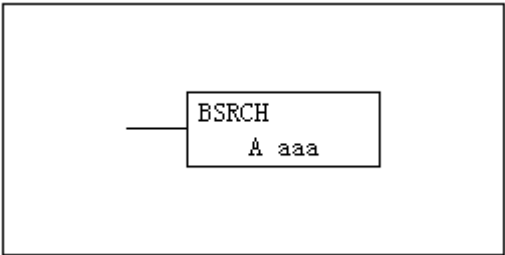
```
LD I0
LDS K2
LDR 03000
LDS K8888
ORMOV R3100
```

右边的例子中，将 R3000 开始的两字数据表同 K3333 逻辑异或并将结果传送到以 R3100 开头的



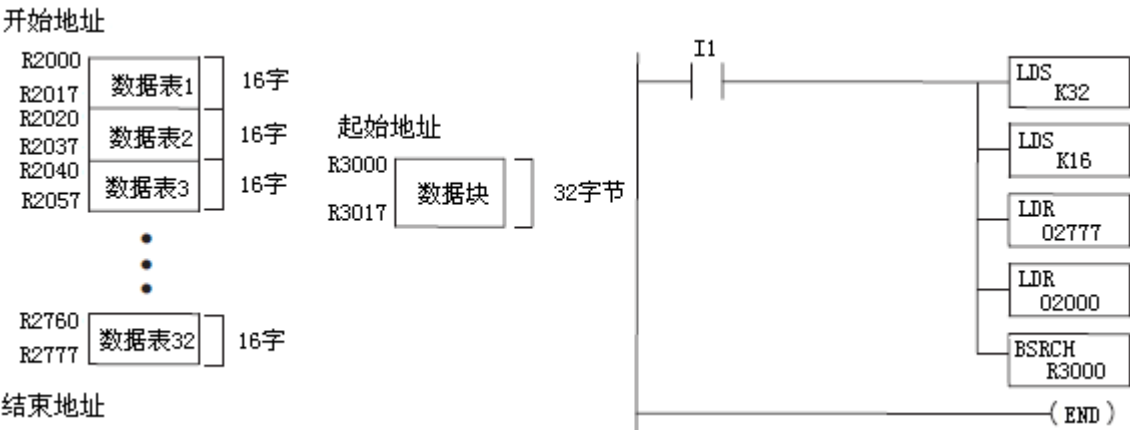
12.20 多字节数据检索指令（BSRCH）

输入 BSRCH 指令在一个数据表中检索一个数据块。指令执行前，要将功能参数分别读入堆栈的第 1 级、第 2 级、第 3 级和累加器中。如果要检索的数据块被找到，它的起始地址将被存放在累加器中。如果没有找到，SP53 将被置位为 ON。



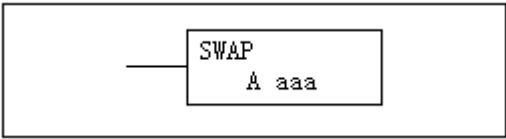
操作数类型		NK1 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

- 下面是实现这个指令功能的步骤：
- 第 1 步：读入要检索的数据块字节数，这个数必须是十六进制数，范围是 0-FF。
 - 第 2 步：读入每个数据表的长度（字数），BSRCH 指令将检索相邻的多个数据表，这个参数必须是十六进制数，范围是 0-FF。
 - 第 3 步：读入检索对象数据表的结束地址。这个参数必须是十六进制数，可以使用 LDR 指令将八进制地址值转化为十六进制数据。
 - 第 4 步：读入检索对象数据表的开始地址。这个参数必须是十六进制数，可以使用 LDR 指令将八进制地址值转化为十六进制数据。
 - 第 5 步：写入 BSRCH 指令，指定要检索数据块的起始地址。



12.21 交换指令（SWAP）

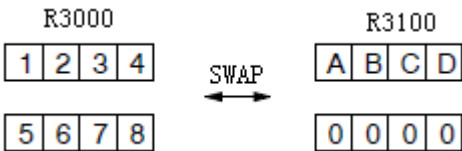
输入 SWAP 指令将两个相同长度的数据表数据互换。



- 下面是实现此指令功能的步骤：
- 第 1 步：将数据表的长度（R 寄存器的个数）读入数据堆栈的第 1 级中。这个数必须是十六进制数，范围是 0-FF，注意，两个数据表的长度必须相等。
 - 第 2 步：将第一个数据表的起始地址读入累加器中，这个数必须是十六进制数，可使用 LDR 指令将八进制地址值转换为十六进制数。
 - 第 3 步：写入 SWAP 指令，指定第二个数据表的起始地址。
- 小技巧：数据交换只能执行一个扫描周期，如果指令在多个连续的扫描周期内执行，就会很难知道数据表中的数据。因此，要注意 SWAP 指令只能执行一个扫描周期。

操作数类型	NK1 范围
A	aaa
R 寄存器	R
	所有（附录 1）

右边的例子中，将以 R3000 开始的 2 字数据表同 R3100 开始的 2 字数据表进行数据交换，下面是程序。



下面的程序使用了一个 PD 触点（IO 由 OFF 到 ON 时接通一个扫描周期）。首先，将数据表的长度值（2 字）读入累加器，然后将第一个数据表的起始地址读入累加器，使用 LDR 指令，将八进制地址值转换为十六进制数据。注意谁是第一个数据表没有关系，因为 SWAP 指令执行结果是一样的。



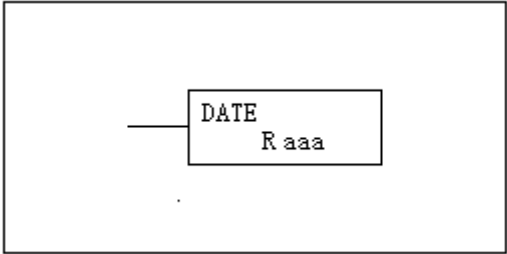
语句视图

LDPD IO
LDS K2
LDR 03000
SWAP R3100

第 13 章 日历/时钟指令（NK0 不支持）

13.1 日期设定指令（DATE）

输入 DATE 指令用来设定 CPU 的日期，指令需要两个连续的 R 寄存器地址来设置日期。如果指定的地址无效，则日期将不进行设定。当前日期可从 4 个连续的 R 寄存器地址（R7771-R7774）中读出。

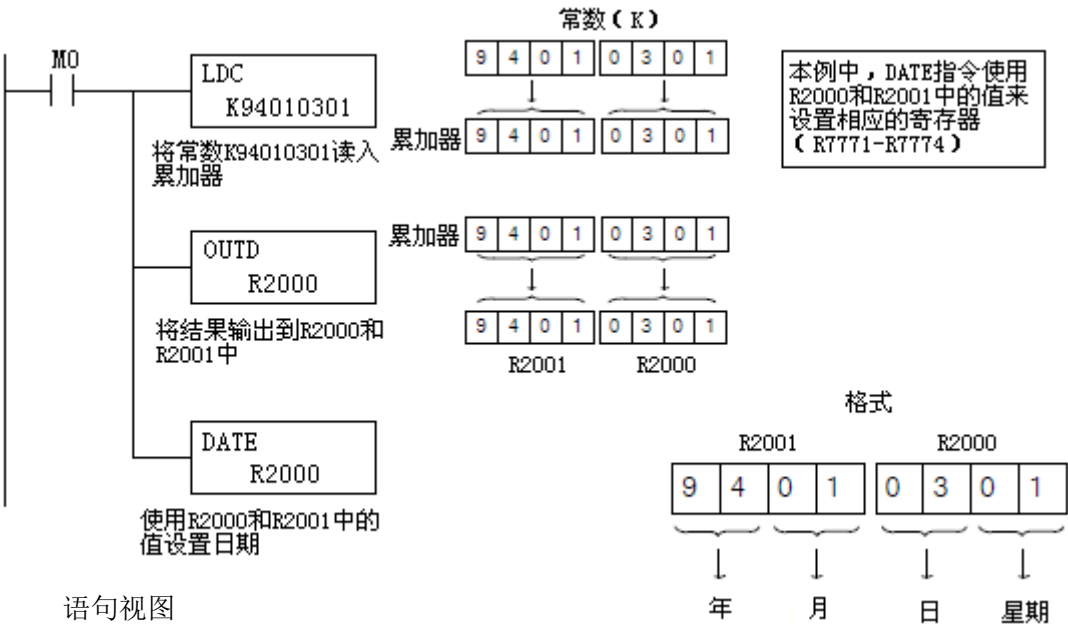


日期	范围	R 寄存器地址（BCD） （只读）
年	0-99	R7774
月	1-12	R7773
日	1-31	R7772
星期	0-06	R7771

星期：0=星期日，1=星期一，2=星期二，3=星期三，4=星期四，5=星期五，6=星期六

操作数类型	NK1 范围
	aaa
R 寄存器	R 所有（附录 1）

下面的例子中，当 M0 为 ON 时，常数 K94010301 被读入累加器中，M0 应是一个一次输出(PD)指令产生的触点。将累加器中的数据写入到 R2000 和 R2001 中。DATE 指令使用 R2000 中的数据来设置日期。

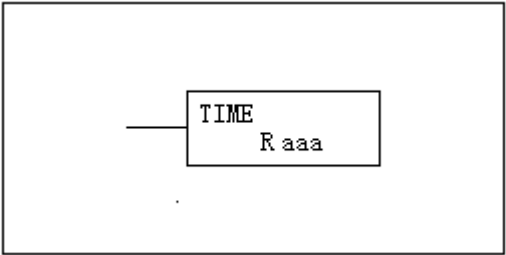


语句视图

```
LD M0
LDC K94010301
OUTD R2000
DATE R2000
```

13.2 时钟设定指令（TIME）

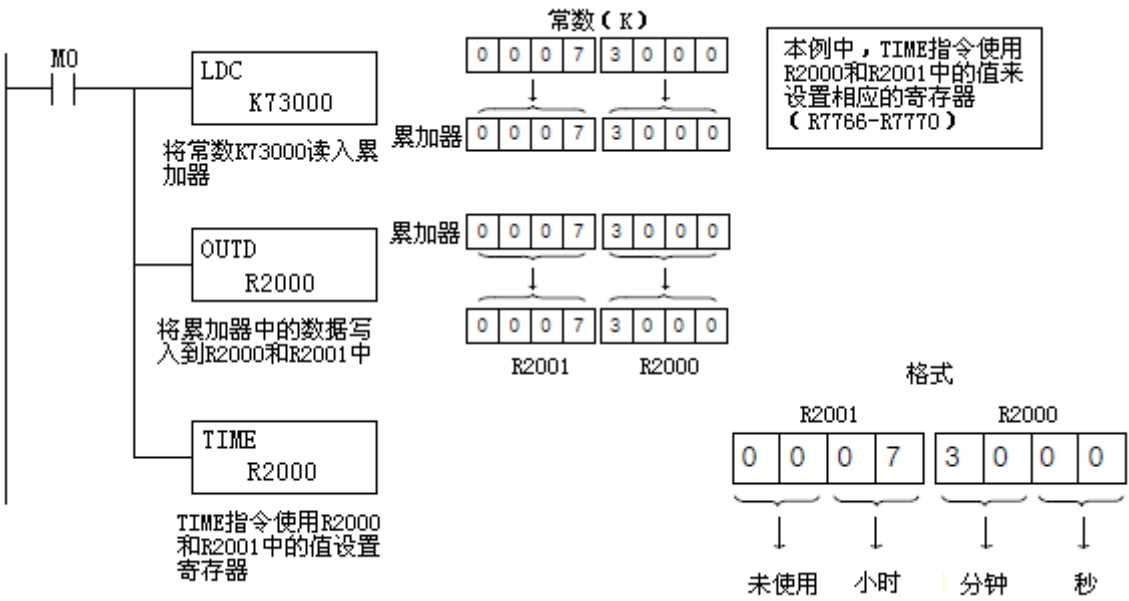
输入 TIME 指令用来设置时间（24 小时时钟）。指令需要两个连续的寄存器地址来设置时间。如果指定的地址无效，则时间将不进行设定。当前时间可从 **R7747** 和 R7766-R7770 中读出。



日期	范围	R 寄存器地址 (BCD) (只读)
1/100s (10ms)	0-99	R7747
秒	0-59	R7766
分	0-59	R7767
小时	0-23	R7770

操作数类型	NK1 范围
	aaa
R 寄存器	R 所有 (附录 1)

下面的例子中，当 M0 为 ON 时，常数 K73000 被读入累加器，OUTD 指令将累加器中的数写入 R2000 和 R2001 中。TIME 指令使用 R2000 和 R2001 中的数据来设置时间。



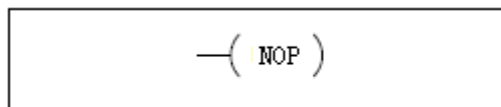
语句视图

```
LD M0
LDC K73000
OUTD R2000
TIME R2000
```

第 14 章 CPU 控制指令

14.1 空操作指令（NOP）

输入 NOP 指令是一个空操作指令。用该指令可在程序中设置空间，对以后追加程序很方便。

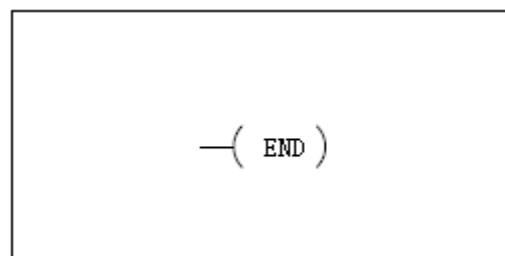


语句视图
NOP

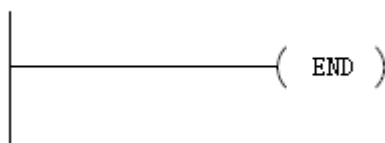


14.2 程序结束指令（END）

输入 END 指令是主程序的扫描终点标志。END 指令要放在主程序的最后。如果主程序后没有 END 指令，将会产生错误，CPU 将不会进入运行模式。数据标记、子程序和中断程序要写在 END 指令之后。END 指令是无条件指令，前面不能添加触点。

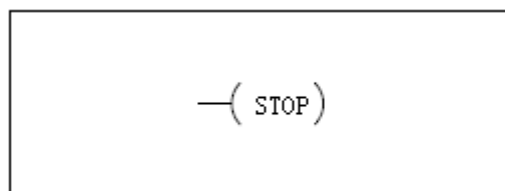


语句视图
END



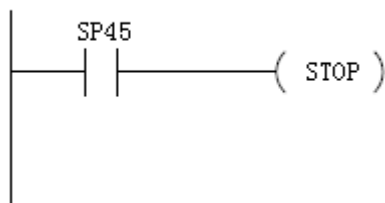
14.3 扫描停止指令（STOP）

输入 STOP 指令将 CPU 强制从运行模式转换为停止模式。该指令通常通过一个停机条件（比如 I/O 模块错误）来停止 PLC 运行。



下面的例子中，当 SP45 被置为 ON，表明有一个 I/O 模块错误，CPU 将停止运行，转换为编程模式。

语句视图
LD SP45
STOP



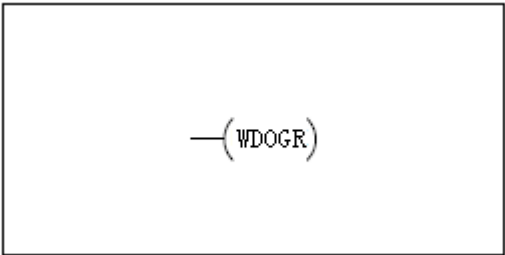
14.4 监视定时器复位指令（WDOGR）

输入 WDOGR 指令复位监视定时器的时间。监视定时器的缺省设置是 200ms，一般的扫描时间都不会超过 200ms，但不是不可能超过。如果程序中有 For/next 回路、子程序、中断程序和表指令等，扫描时间有可能超过 200ms。

如果扫描时间超过监视定时器中设定的时间，将产生一个软件时间溢出错误（E003）并且 CPU 将进入编程模式。如果扫描时间很长，在程序中加入 WDOGR 指令是很重要的，在扫描时间超过监视定时器设置的时间之前，必须执行 WDOGR 指令。

如果扫描时间总是超过监视定时器设定的时间，则需要加长监视定时器的设定值。KPPsoft 编程软件中，“PLC”菜单中可直接改变设置。

下面的例子中，当 WDOGR 指令执行时，CPU 扫描时间被复位。



语句视图
WDOGR



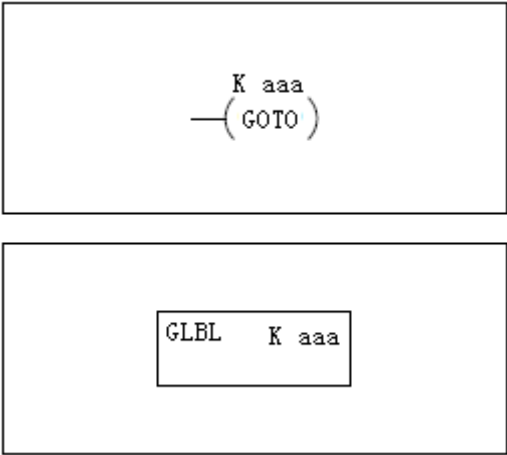
第 15 章 程序控制指令

15.1 跳转指令（GOTO）

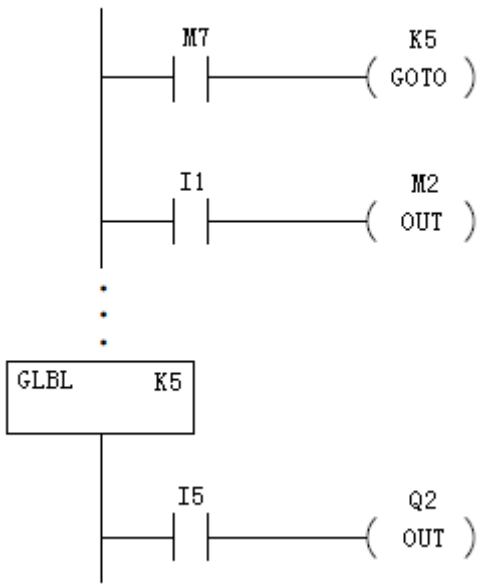
15.2 跳转目标指令（GLBL）

输入 GOTO/GLBL 指令将这两个指令之间的所有指令跳过。GOTO 指令和 GLBL 指令的标号必须相同。GOTO 指令执行时，GOTO 和 GLBL 之间的指令不执行。

操作数类型		NK1 范围
		aaa
常数	K	1~FFFF



下面的例子中，当 M7 为 ON 时，GOTO 和 GLBL（有相同的标号）之间所有的指令将被跳过，CPU 不执行被跳过的指令。



语句视图

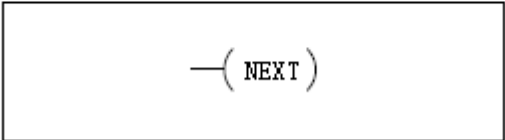
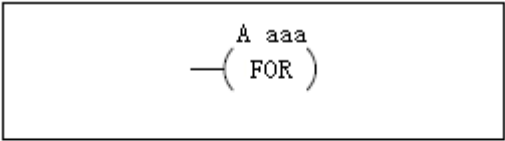
```
LD M7
GOTO K5
LD I1
OUT M2
...
GLBL K5
LD I5
OUT Q2
```

15.3 循环执行开始指令（FOR）

15.4 循环体最后指令（NEXT）

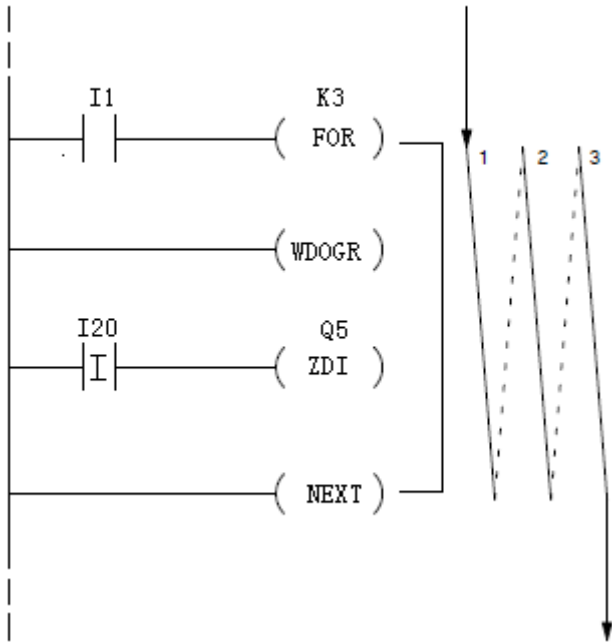
输入 FOR 和 NEXT 指令用于按指定次数执行两个指令之间的程序。FOR 指令的条件成立时，FOR 和 NEXT 两个指令之间的程序按指定次数执行；如果 FOR 指令的条件不成立，则两个指令之间的程序不执行。

FOR/NEXT 指令不能被嵌套。程序中最多可写入 64 个 FOR/NEXT 回路，如果超出，将产生一个错误 E413。FOR/NEXT 回路执行时，正常的 I/O 更新和 CPU 的存储器管理任务被暂停。除了直接 I/O 指令以外，I/O 指令都不会被更新直到程序执行完成。如果程序执行时间很长，需要加入 WDOGR 指令将监视定时器复位。



操作数类型		NK1 范围
		aaa
寄存器	R	所有（附录 1）
常数	K	1-9999

下面的例子中，当 I1 为 ON 时，FOR/NEXT 回路将被循环执行三次。如果 I1 为 OFF，则回路不执行。根据应用需要使用直接指令和 WDOGR 指令。



语句视图

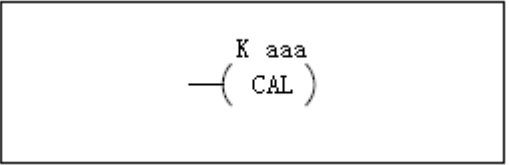
```
LD I1
FOR K3
WDOGR
LDDI I20
ZDI Q5
NEXT
```

15.5 子程序调用指令（CAL）

输入 CAL 指令允许在主程序之后写入一段程序，仅在需要时调用。程序中最多可使用 256 个 CAL 指令。

子程序执行完成后，返回到 CAL 指令后的主程序中继续执行。子程序可以有 8 级嵌套，如果超过 8 级，将会产生一个错误 E412。子程序通常用于一个程序块不需要每次扫描都执行的场合。

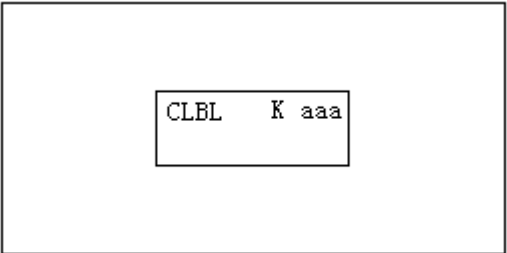
操作数类型		NK1 范围
		aaa
常数	K	1-FFFF



15.6 子程序开始标记（CLBL）

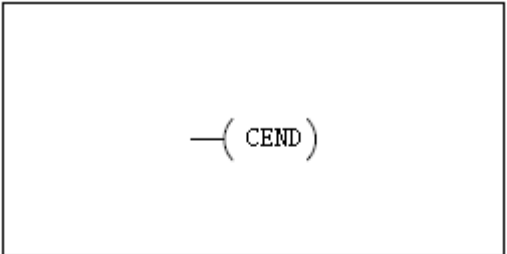
输入 CLBL 指令写在主程序 END 指令之后。当 CAL 指令执行时，CPU 将执行同 CAL 有相同标号的子程序。

操作数类型		NK1 范围
		aaa
常数	K	1-FFFF



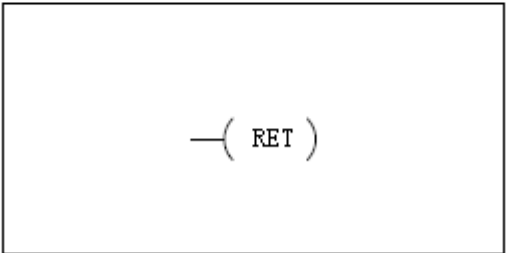
15.7 无条件复归指令（CEND）

输入 子程序中的 CEND 指令执行时，CPU 将返回到主程序中子程序被调用的地方。CEND 指令用作子程序的结束指令，必须是子程序的最后一个指令，指令前不可添加条件触点。

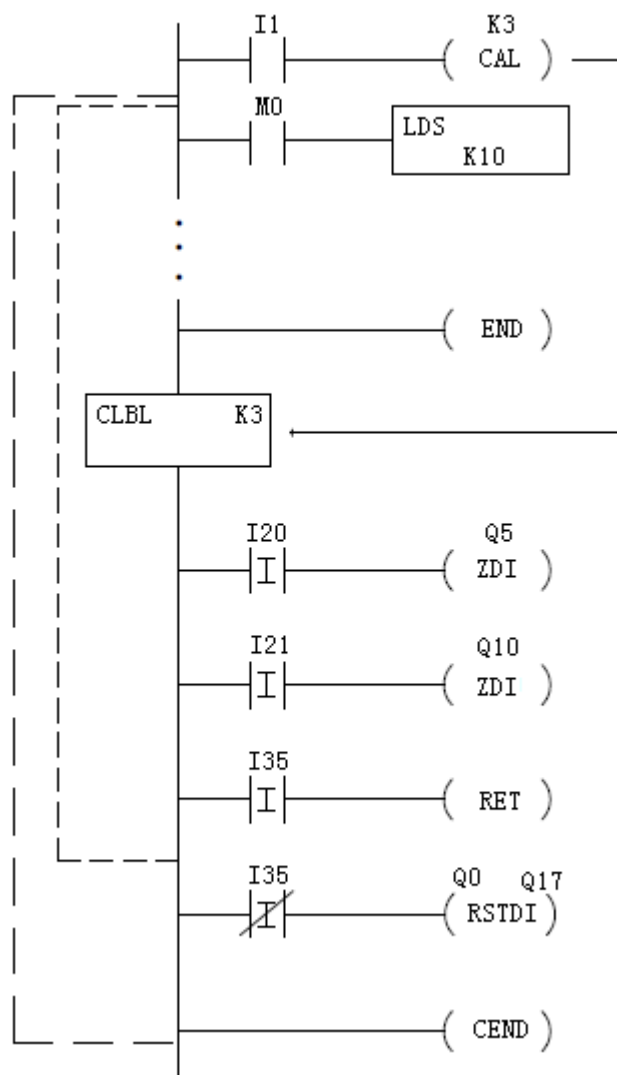


15.8 条件复归指令（RET）

输入 RET 指令是一个可选用指令，同一个条件触点配合使用来完成从子程序有条件返回。使用 RET 指令时，仍然需要结束子程序的 CEND 指令。



下面的例子中，当 I1 为 ON 时，子程序 K3 将被调用。CPU 将跳去执行标记号为 K3 的子程序。如果 I35 为 ON，CPU 将返回主程序中。如果 I35 为 OFF，则 Q0-Q17 将被复位，然后 CPU 返回到主程序中。



语句视图

```

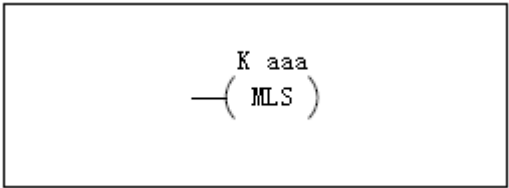
LD I1
CAL K3
LD M0
LDS K10
...
END
CLBL K3
LDDI I20
ZDI Q5
LDDI I21
ZDI Q10
LDDI I35
RET
LDNDI I35
RSTDI Q0 Q17
CEND

```

15.9 新母线开始指令（MLS）

输入 MLS 指令是定义 LD 指令（或 LD 开始的指令群）条件新母线的指令。主母线编号一直是 0，MLR 指令依次指定 1-7 的母线编号。

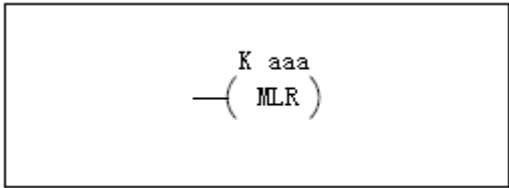
操作数类型		NK1 范围
		aaa
常数	K	1-7



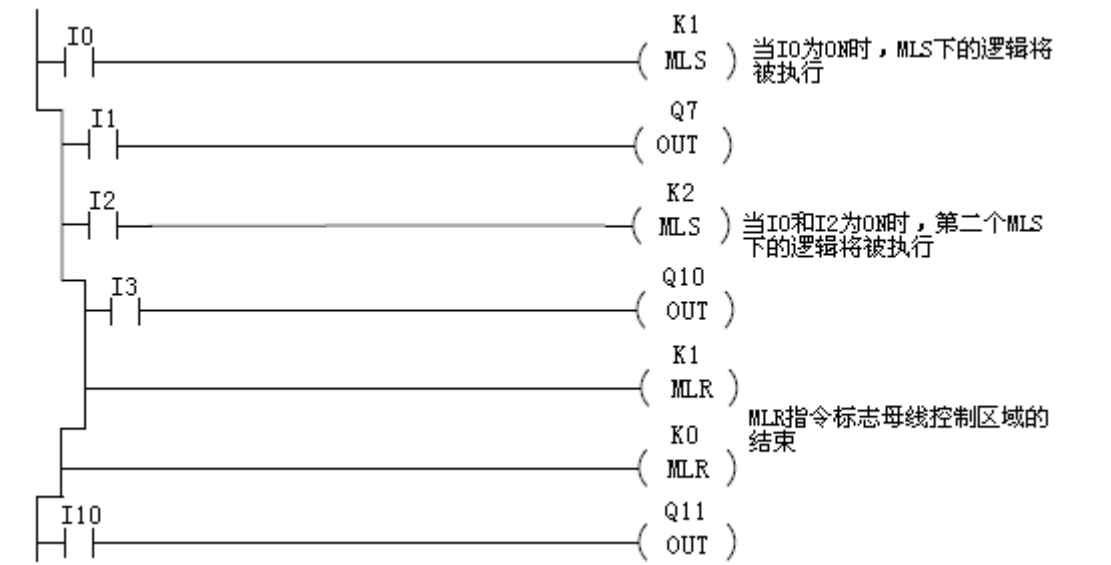
15.10 母线复归指令（MLR）

输入 MLR 指令根据母线号的指定，返回到前面的母线上。MLR 指令的标记号比相应的 MLS 指令的标记号小 1。

操作数类型		NK1 范围
		aaa
常数	K	0-7



MLS 和 MLR 指令允许用户快速控制程序段的通断，使程序控制更加灵活，下面的例子演示了如何使用 MLS 和 MLR 指令。

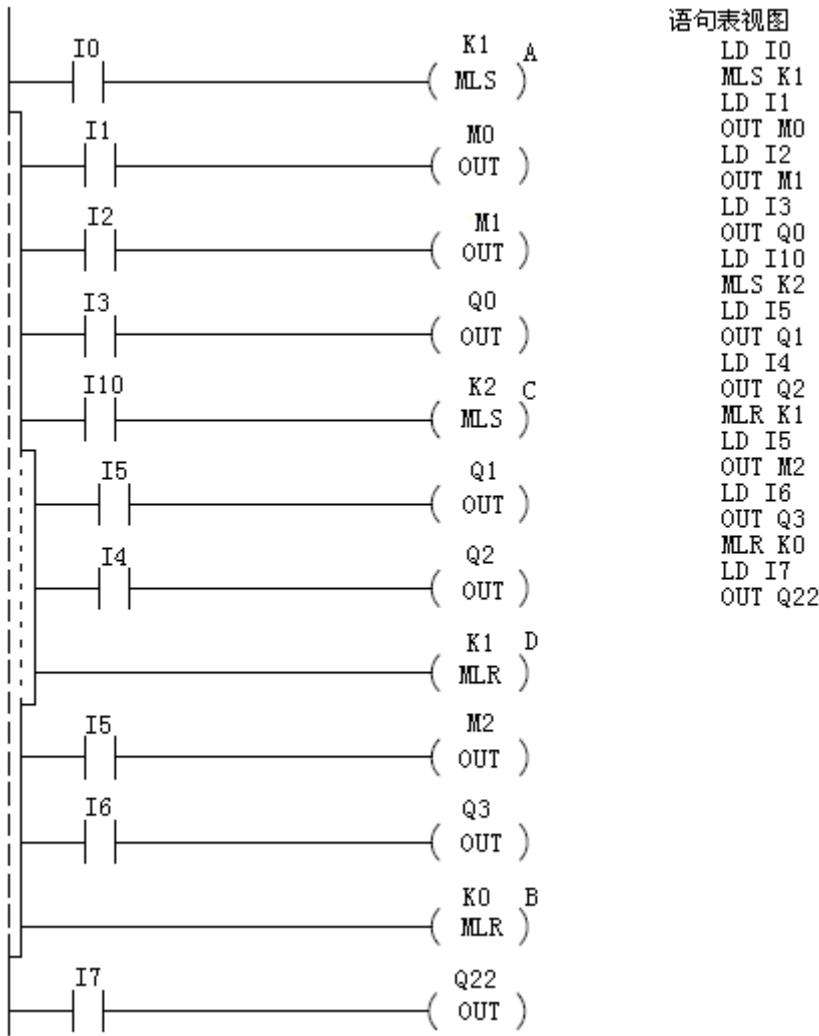


下面的例子中，当 I0 为 ON 时，第 1 个 MLS K1 (A) 和 MLR K0 (B) 之间的逻辑程序才能接通。（注意，如果 I0 为 OFF，程序仍被扫描，但是没有接通。）当 I10 和 I0 为 ON 时，MLS K2 (C) 和 MLR K1 (D) 之间的逻辑程序将接通。最后一行程序不受任何 MLS 线圈控制。

注意，MLS/MLR 指令控制程序段是否接通，但是不控制指令是否执行。指令始终是执行的，但由于没有接通，程序不能将输出线圈置为 ON，在例子中，考虑下面的情况：

- 1. I0 为 OFF，意味着第二行的程序没有接通。
- 2. 使用编程设备将 M0 置为 ON。

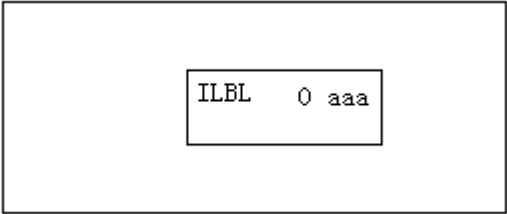
因为第二行程序 (LD I1, OUT M0) 没有接通，如果希望 M0 被接通，使用编程设备将 M0 置为 ON。但是，MLS 指令并不意味着控制区域的指令没有执行，实际上，它们被执行了，但是没有接通。所以这种情况下，当这行程序被执行时，M0 将被置为 OFF。这是因为 CPU 检测到此行程序没有接通。当 CPU 执行这行程序时，就将 M0 置为 OFF。



第 16 章 中断指令（NK0 不支持）

16.1 中断子程序指令（ILBL）

输入 ILBL 指令允许在主程序外写入一段中断程序，需要的时候执行。中断可在程序中调用，也可通过中断模块的外部中断调用。



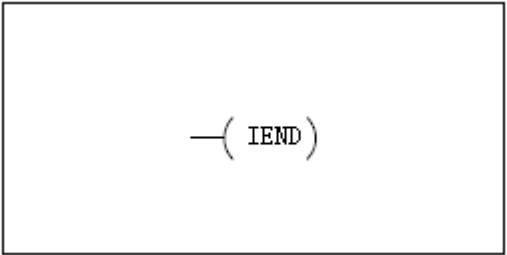
通常情况下，中断程序用于需要对一个输入快速响应的场合或者需要执行的程序段要比普通 CPU 执行快的场合。中断标记以及相关中断程序必须放在END指令之后。当中断子程序通过中断模块或软件中断调用后，CPU 将结束目前处理的指令，转而执行相应的中断子程序。中断模块的中断以八进制编号，同相应的硬件输入信号相对应（例如，I10 将调用中断子程序 ILBL10）。一旦中断程序执行完成，程序执行将从中断发生前的地方继续执行。

软件中断可在 R7634 中设置中断时间。有效范围是 1-9999ms。数据必须是 BCD 格式。如果数据超出有效范围，中断将不执行。

操作数类型		NK1 范围
		aaa
常数	0	0-20

16.2 中断子程序无条件复归指令（IEND）

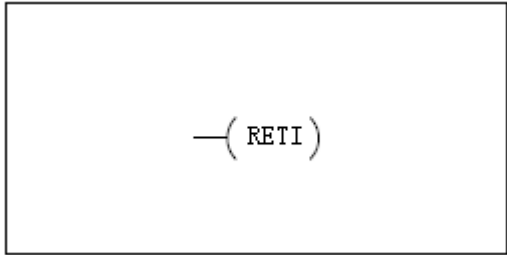
输入 IEND 指令执行时，CPU 结束中断程序，返回到主程序原先中断的地方继续执行。IEND 指令是中断子程序的最后一个指令，前面不可添加条件触点。



—(IEND)

16.3 中断子程序条件复归指令（RETI）

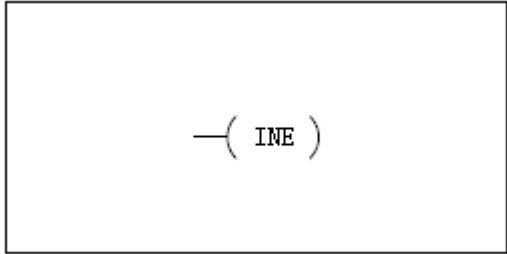
输入 RETI 指令是一个可选用指令，同一个条件触点配合使用来完成从中断子程序有条件返回。使用 RETI 指令时，仍然需要结束中断子程序的 IEND 指令。



—(RETI)

16.4 中断许可指令（INE）

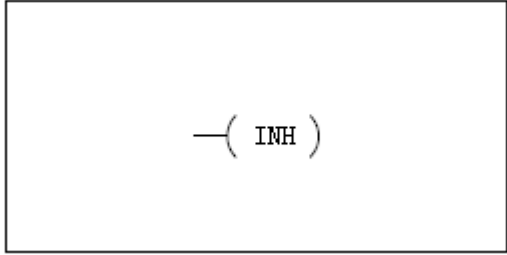
输入 INE 指令是写在主程序中，END 指令之前。INE 指令执行时，硬件或软件中断被许可，直到中断禁止指令 INH 执行时，中断禁止。



—(INE)

16.5 中断禁止指令（INH）

输入 INH 指令是写在主程序中，END 指令之前。INH 指令执行时，硬件或软件中断被禁止，直到中断许可指令 INE 执行时，中断被许可。

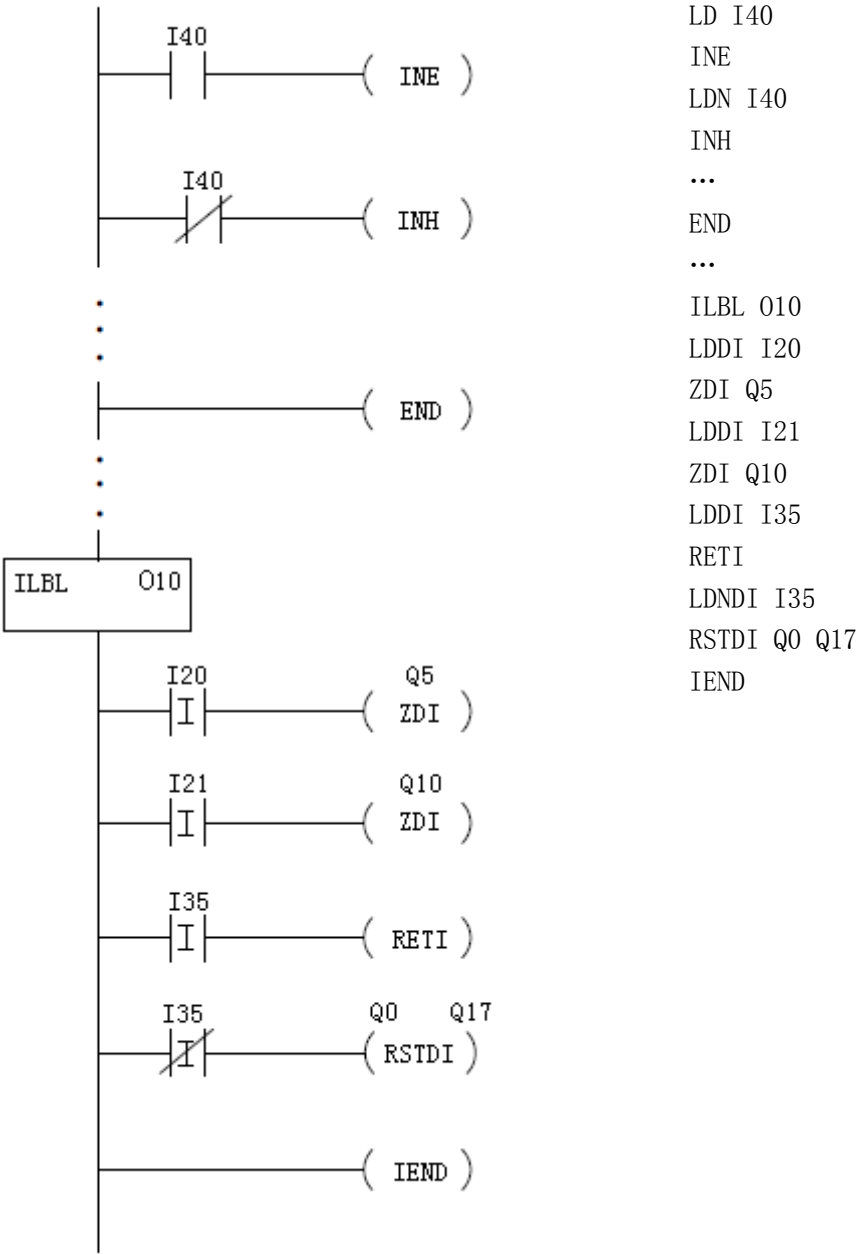


—(INH)

16.6 硬件中断举例

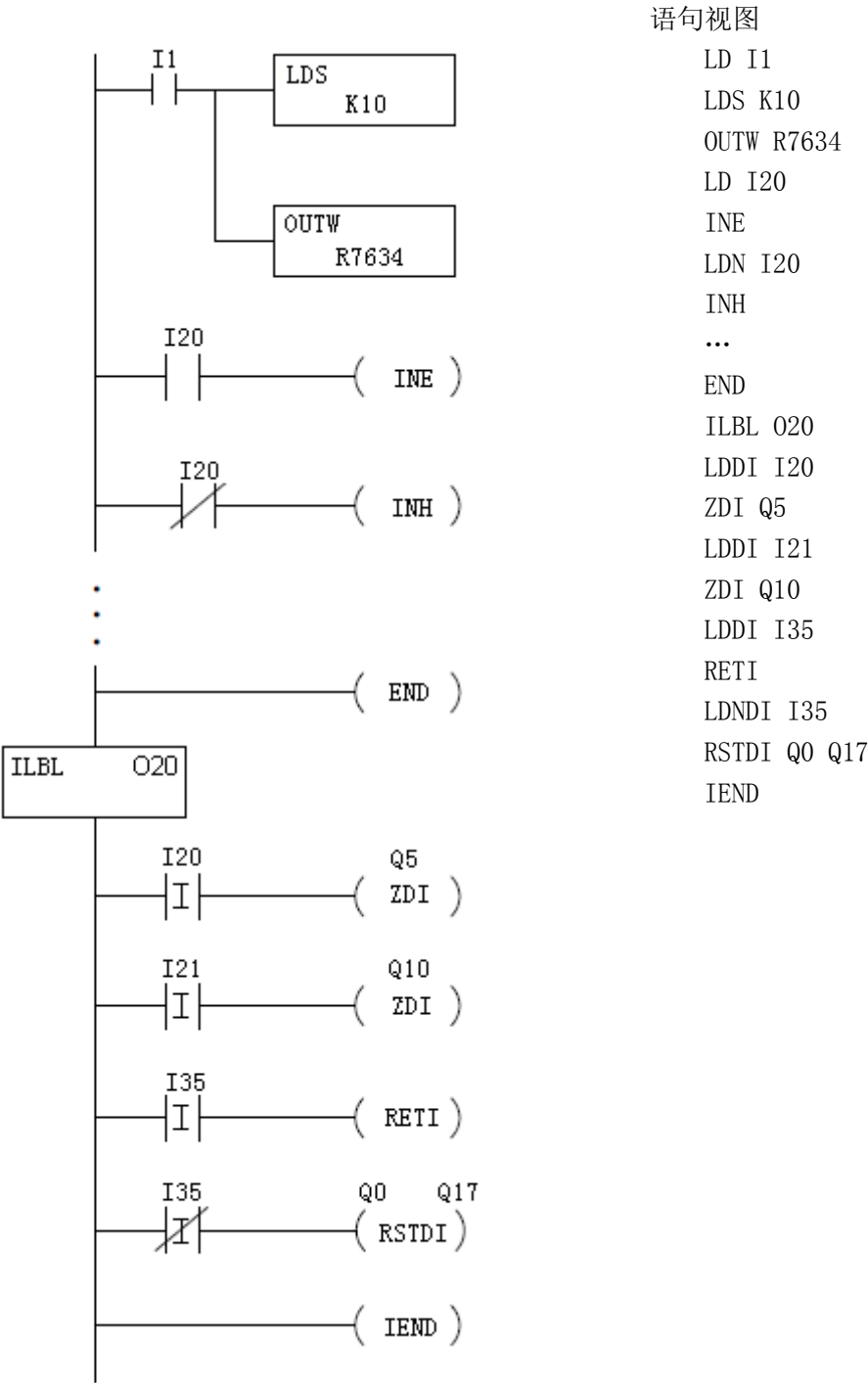
下面的例子中，当 I40 为 ON 时，中断许可；当 I40 为 OFF 时，中断禁止。当接收到中断信号 I10 时，CPU 将跳去执行中断子程序 ILBL 010。如果 I35 为 ON，CPU 将返回到主程序；如果 I35 为 OFF，Q0-Q17 将被复位，然后 CPU 结束中断子程序，返回到主程序中。

语句视图



16.7 软件中断举例

下面的例子中，当 I1 为 ON 时，常数 K10 被读入累加器，然后写入到 R7634 中。此值将软件中断时间设置为 10ms。当 I20 为 ON 时，中断被允许，当 I20 为 OFF 时，中断被禁止。每隔 10ms，CPU 将跳去执行中断程序 ILBL 020。当 I35 为 ON 时，CPU 将返回到主程序中。当 I35 为 OFF 时，Q0-Q17 被复位。软件中断的中断时间范围是 1-9999ms。



第 17 章 通讯指令

17.1 读出指令（RX）

输入 RX 指令用于网络中主局 CPU 读取子局 CPU 中的数据块。指令执行前，需要将功能参数分别读入数据堆栈的第 1、第 2 级和累加器中。

下面是实现该指令功能的步骤：

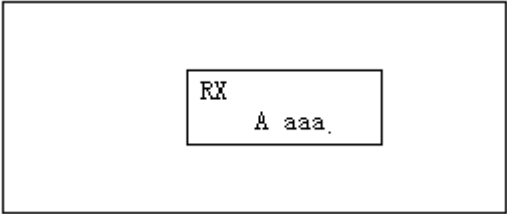
第 1 步：将所用通讯口号 (P0-P2) 和子局地址读入堆栈的第 2 级中（注意，地址 0 仅同等站有效）。

第 2 步：将要被读出的字节数（0-128 BCD）读入堆栈的第 1 级中。

第 3 步：将主局中用来存放读出数据的起始地址写入累加器中，这个地址必须是十六进制数。

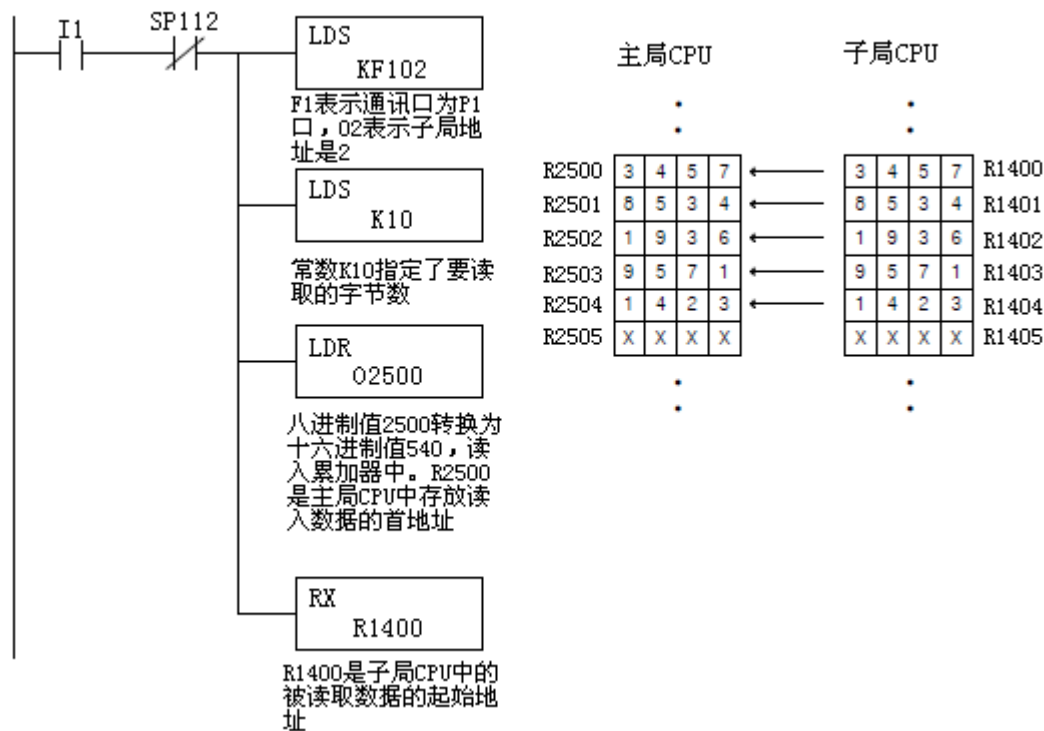
第 4 步：写入 RX 指令，指定子局 CPU 中存放被读出的数据的起始地址。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。



操作数类型		NK1 范围
	A	aaa
寄存器	R	所有（附录 1）
指针	P	所有（附录 1）
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777
级	S	0-1777
定时器	T	0-777
计数器	C	0-777
特殊线圈	SP	0-777
通讯 I/O	GI/GQ	0-3777

下面的例子中，当 I1 为 ON 并且 SP112（0：P1 口不在通讯中；1：P1 口在通讯中）为 OFF 时，RX 指令将从子局中读取连续 10 个字节的数据（R1400~R1404），并将数据写入到主局 CPU 的 R2500~R2504 中。主局通讯口为 P1，子局地址为 2。

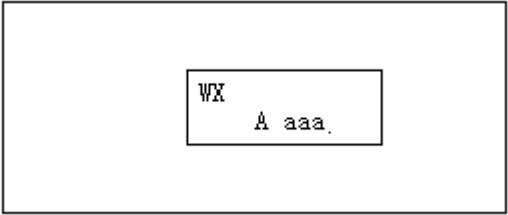


语句视图

```
LD I1
LDN SP112
LDS KF102
LDS K10
LDR 02500
RX R1400
```

17.2 写入指令（WX）

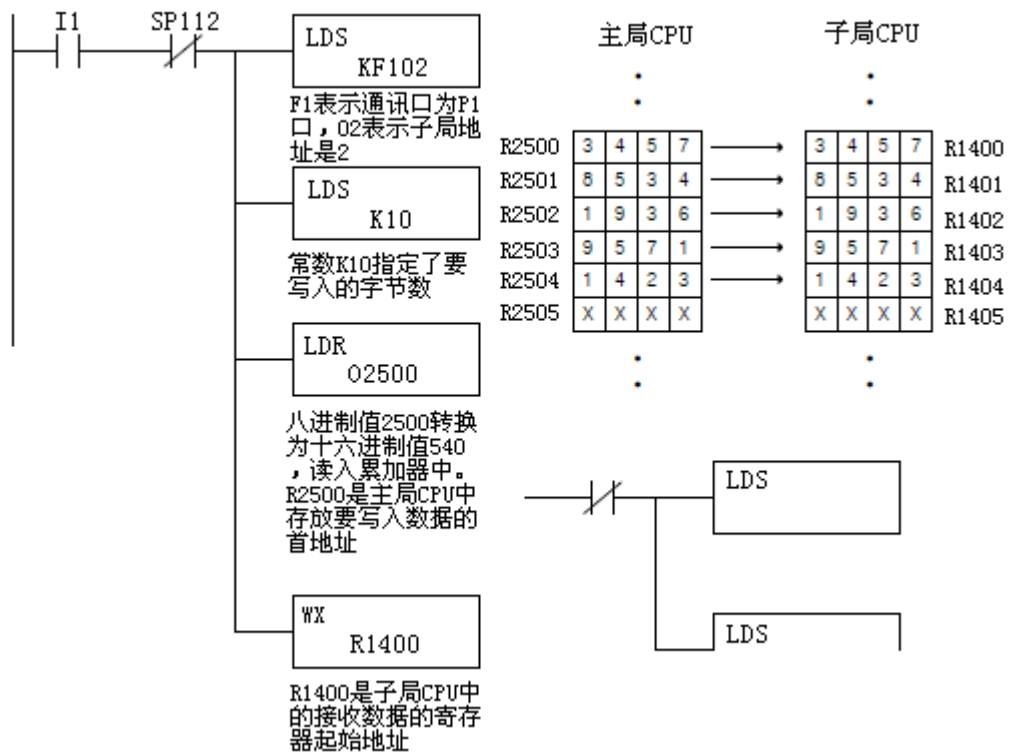
输入 WX 指令用于网络中主局 CPU 向子局 CPU 中写入一个数据块。指令执行前，需要将功能参数分别读入数据堆栈的第 1、第 2 级和累加器中。



- 下面是实现该指令功能的步骤：
- 第 1 步：将通讯口（P0-P2）和子局地址读入堆栈的第 2 级中。
 - 第 2 步：将要被写入的字节数（0-128 BCD）读入堆栈的第 1 级中。
 - 第 3 步：将主局中存放被写出数据的起始地址写入累加器中，这个地址必须是十六进制数。
 - 第 4 步：写入 WX 指令，指定子局 CPU 中要接收数据的起始地址。
- 小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型	NK1 范围	
	A	aaa
寄存器	R	所有（附录 1）
指针	P	所有（附录 1）
输入	I	0-1777
输出	Q	0-1777
内部线圈	M	0-3777
级	S	0-1777
定时器	T	0-777
计数器	C	0-777
特殊线圈	SP	0-777
通讯 I/O	GI/GQ	0-3777

下面的例子中，当 I1 为 ON 并且 SP112（0：P1 口不在通讯中；1：P1 口在通讯中）为 OFF 时，WX 指令将主局中的连续 10 字节的数据（R2500-R2504）写入到子局 CPU 的中 R1400-R1404 中。主局通讯口为 P1，子局地址为 2。



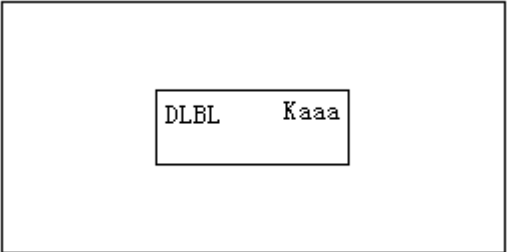
语句视图

```
LD I1
LDN SP112
LDS KF102
LDS K10
LDR 02500
WX R1400
```

第 18 章 信息指令

18.1 数据区标号指令（DLBL）

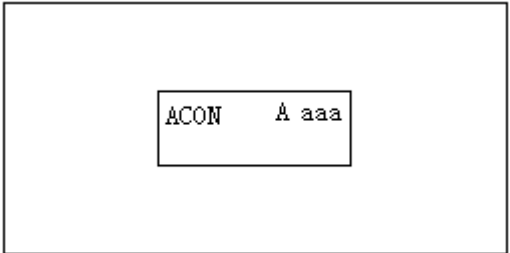
输入 DLBL 指令标记 ASCII/数字数据区域的开始, 通常同 ACON 和 NCON 指令一起使用。DLBL 指令写在 END 指令的后面。程序中最多可写入 256 个 DLBL 指令。DLBL 区域中可使用多个 NCON 和 ACON 指令。



操作数类型		NK1 范围
		Aaa
常数	K	1~FFFF

18.2 ASCII 数据登记指令（ACON）

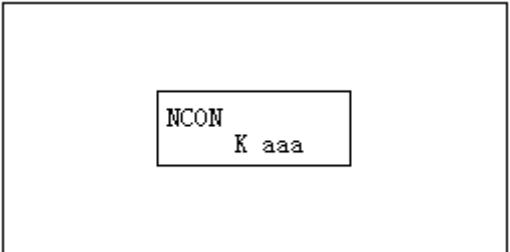
输入 ACON 指令同 DLBL 指令一起使用, 用来存放 ASCII 文本。两个 ASCII 字符存放在一个 ACON 指令中。如果只有一个字符, 在故障信息表中字符前会有一个空格。



操作数类型		NK1 范围
		Aaa
ASCII	A	0~9 A~Z

18.3 数值数据登记指令（NCON）

输入 NCON 指令同 DLBL 指令一起使用, 用来存放十六进制的 ASCII 等效数字。指令中可存放两个数字。



操作数类型		NK1 范围
		Aaa
常数	K	0~FFFF

第 19 章 Modbus RTU 指令

19.1 Modbus 网络读指令（MRX）

输入 MRX 指令用于 NK1 作 MODBUS 网络主局时，将网络子局的一个数据块读入主局中的功能存储器区内。该指令可让用户指定 MODBUS 的功能码、子局号、主局和子局的存储器开始地址，传送数据量、MODBUS 数据类型和异常响应缓冲寄存器。

- **CPU 端口号：**指定使用的 NK1 的通信口，可选 K0~K3，K0 表示使用 P0 口，K1 表示使用 P1 口等等。
- **子局号地址：**指定对方子局号（0~247）。该参数对 P3（以太网）端口无效。
- **功能码：**指定 MRX 指令使用的 MODBUS 功能码。MRX 指令支持下列 MODBUS 功能码：
 - 01——读一组线圈
 - 02——读一组输入信号
 - 03——读保持寄存器
 - 04——读输入寄存器
 - 07——读异常状态
- **子局开始地址：**指定要读取子局数据的存储器开始地址，为标准 MODBUS 地址。
- **主局开始地址：**指定存放读入数据的 NK1 主局存储器开始地址。根据 MODBUS 功能码的不同，支持的 NK1 功能存储器类型也不同。
- **数据大小：**指定要读入的线圈/输入点/保持寄存器/输入寄存器数量。
- **超时（ms）：**指定等待通信对方应答的超时时间。
- **IP 地址：**仅对 P3 口（以太网口）有效的参数，指定通信对方的 IP 地址。
- **TCP 端口号：**仅对 P3 口（以太网口）有效的参数，指定通信对方的 TCP 端口号。（注意：使用 NK1 做 MRX 通信子局时，其 TCP 端口地址固定为 502。）
- **MODBUS 数据类型：**指定使用何种 MODBUS 数据格式，可以选择 584/984 或 484 数据格式。
- **通信实行中标志：**启动 MRX 指令通信后置为 ON，通信结束后变为 OFF。
- **通信完成标志：**启动 MRX 指令通信时复位该标志，MRX 指令通信完成后置为 ON 并保持。
- **通信错误标志：**启动 MRX 指令通信时复位该标志，MRX 指令通信出错时置为 ON 并保持。
- **异常响应缓冲区：**指定主局存放通信异常响应码的存储器地址。
 - 异常响应码内容及其含义如下：
 - 异常响应码 = 1 表示指定的功能码不支持。
 - 异常响应码 = 2 表示指定的地址、范围出错。

异常响应码 = 3 表示要求的数据内容异常。可能指定了当前 PLC 动作模式下是禁止读写的内容。

异常响应码 = 4 表示通信处理过程异常。

1) MRX 子局存储器地址范围

功能码	MODBUS 数据格式	子局地址范围
01—读线圈	484 方式	1 ~ 999
	584/984 方式	1 ~ 65536
02—读输入状态	484 方式	1001 ~ 1999
	584/984 方式	10001 ~ 19999 (5 位) 或 100001 ~ 165536 (6 位)
03—读保持寄存器	484 方式	4001 ~ 4999
	584/984 方式	40001 ~ 49999 (5 位) 或 400001 ~ 465536 (6 位)
04—读输入寄存器	484 方式	3001 ~ 3999
	584/984 方式	30001 ~ 39999 (5 位) 或 300001 ~ 365536 (6 位)
07—读异常状态	484 和 584/984 方式	N/A

2) MRX 主局存储器地址范围

操作数类型	NK1 范围
全局输入 GI	GI0 ~ GI3777
输入 I	I0 ~ I1777
全局输出 GQ	GQ0 ~ GQ3777
输出 Q	Q0 ~ Q1777
中间继电器 M	M0 ~ M3777
级 S	S0 ~ S1777
定时器 T	T0 ~ T777
计数器 C	C0 ~ C777
特殊继电器 SP	SP0 ~ SP1777
数据寄存器 R	R2000 ~ R7377 R10000~36777

3) MRX 指令传送数据量指定

操作数类型	NK1 范围
数据寄存器 R	R2000 ~ R7377 R10000~36777
常数 K	传送单个位时: 1~2000 传送寄存器时: 1~125

4) MRX 异常响应缓冲寄存器范围

操作数类型	NK1 范围
数据寄存器 R	R2000 ~ R7377 R10000~36777

19.2 Modbus 网络写指令（MWX）

输入 MWX 指令用于将 NK1（MODBUS 网络主局）中的一块数据写到网络中某个子局的指定存储区中。该指令可让用户指定 MODBUS 的功能码、子局号、主局和子局的存储器开始地址，传送数据量、MODBUS 数据类型和异常响应缓冲寄存器。

- **CPU 端口号：**指定使用的 NK1 的通信口，可选 K0~K3，K0 表示使用 P0 口，K1 表示使用 P1 口等等。
- **子局号地址：**指定对方子局号(0~247)。该参数对 P3（以太网）端口无效。
- **功能码：**指定 MWX 指令使用的 MODBUS 功能码。MWX 指令支持下列 MODBUS 功能码：
 - 05——强制置位/复位单个线圈
 - 06——改写单个寄存器
 - 15——强制置位/复位一组线圈
 - 16——改写一组寄存器
- **子局开始地址：**指定要写入数据的子局存储器开始地址，为标准 MODBUS 地址。
- **主局开始地址：**指定要读出数据的 NK1 主局存储器开始地址，根据 MODBUS 功能码的不同，支持的 NK1 功能存储器也不同。
- **数据大小：**指定要写出的线圈/输入点/保持寄存器/输入寄存器数量。
- **超时（ms）：**指定等待通信对方应答的超时时间。
- **IP 地址：**仅对 P3 口（以太网口）有效的参数，指定通信对方的 IP 地址。
- **TCP 端口号：**仅对 P3 口（以太网口）有效的参数，指定通信对方的 TCP 端口号。（注意：使用 NK1 做 MWX 通信子局时，其 TCP 端口地址固定为 502。）
- **MODBUS 数据类型：**指定使用何种 MODBUS 数据格式，可以选择 584/984 或 484 数据格式。
- **通信实行中标志：**启动 MWX 指令通信后置为 ON，通信结束后变为 OFF。
- **通信完成标志：**启动 MWX 指令通信时复位该标志，MWX 指令通信完成后置为 ON 并保持。
- **通信错误标志：**启动 MWX 指令通信时复位该标志，MWX 指令通信出错时置为 ON 并保持。
- **异常响应缓冲区：**指定主局存放通信异常响应码的存储器地址。

异常响应码内容及其含义如下：

 - 异常响应码 = 1 表示指定的功能码不支持。
 - 异常响应码 = 2 表示指定的地址、范围出错。
 - 异常响应码 = 3 表示要求的数据内容异常。可能指定了当前 PLC 动作模式下是禁止读写的内容。
 - 异常响应码 = 4 表示通信处理过程异常。

1) MWX 子局存储器地址范围

功能码	MODBUS 数据格式	子局地址范围
05—强制置位/复位单个线圈	484 方式	1 ~ 999
	584/984 方式	1 ~ 65536
06—改写单个寄存器	484 方式	4001 ~ 4999
	584/984 方式	40001 ~ 49999 (5 位) 或 400001 ~ 465536 (6 位)
15—强制置位/复位一组线圈	484 方式	1 ~ 999
	584/984 方式	1 ~ 65536
16—改写一组寄存器	484 方式	4001 ~ 4999
	584/984 方式	40001 ~ 49999 (5 位) 或 400001 ~ 465536 (6 位)

2) MWX 主局存储器地址范围

操作数类型	NK1 范围
全局输入 GI	GI0 ~ GI3777
输入 I	I0 ~ I1777
全局输出 GQ	GQ0 ~ GQ3777
输出 Q	Q0 ~ Q1777
中间继电器 M	M0 ~ M3777
级 S	S0 ~ S1777
定时器 T	T0 ~ T777
计数器 C	C0 ~ C777
特殊继电器 SP	SP0 ~ SP1777
数据寄存器 R	全部

3) MWX 指令传送数据量指定

操作数类型	NK1 范围
数据寄存器 R	R2000 ~ R7377 R10000~36777
常数 K	传送单个位时: 1~2000 传送寄存器时: 1~125

4) MWX 异常响应缓冲寄存器范围

操作数类型	NK1 范围
数据寄存器 R	R2000 ~ R7377 R10000~36777

19.3 KPPsoft 软件中使用 MRX/MWX 的例子

1、仅利用通信实行中标志的通信程序段

下面的程序段使用 NK1 的 P1 通信口（RS-485 口）来实现 NK1 与支持 MODBUS RTU 通信协议的带 RS-485 串行通信口的设备之间的数据交换功能。

程序中用 MODBUS 协议读通信指令 MRX 的功能码 03，读出局号为 2 的子局的保持寄存器 40001~40002 的内容，保存到 NK1 的 R3000，R3001 中；用写通信指令 MWX 的功能码 06，把 NK1 主局寄存器 R2500 中的值写入局号为 2 的子局的保持寄存器 40010 中。

由于在一次扫描中只能有一个通信指令被执行，因此，为了保证读、写通信程序都能被执行到，这里使用了互锁线圈。

注意：本段程序利用了 MRX/MWX 指令的通信实行中标志（M0）来作为 MRX/MWX 指令的执行条件之一，以保证仅在 P1 通信口处于空闲状态时才启动一次通信。你还可以利用另外 2 个通信标志（通信完成，通信错误）来判断通信是否正常结束，通信数据是否已经正常发送/接收了。



2、利用通信实行中、通信完成、通信错误等 3 个标志的通信程序段

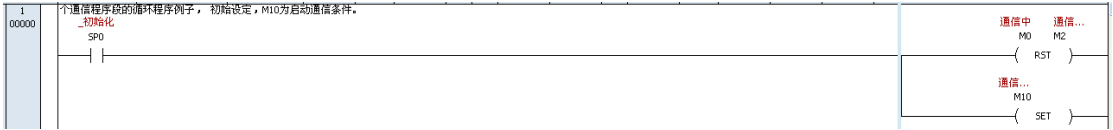
下面的程序段利用 NK1 的以太网通信口与带有标准 MODBUS TCP/IP 通信口的设备进行通信，循环执行 2 条通信指令。例子中假设通信对方的 IP 地址为 192.168.10.100(注意：NK1 的以太网口的通信地址必须设定在同一个网段地址中，例如 192.160.10.101 等)，MODBUS 子局地址号为 2。

通信程序启动后，不断循环执行 2 条通信写指令，把 NK1 的 R2000 和 R2001 的数据送到通信对方的 416705 地址处（对应 NK1 的 RQ0），实现数据的交替输出。

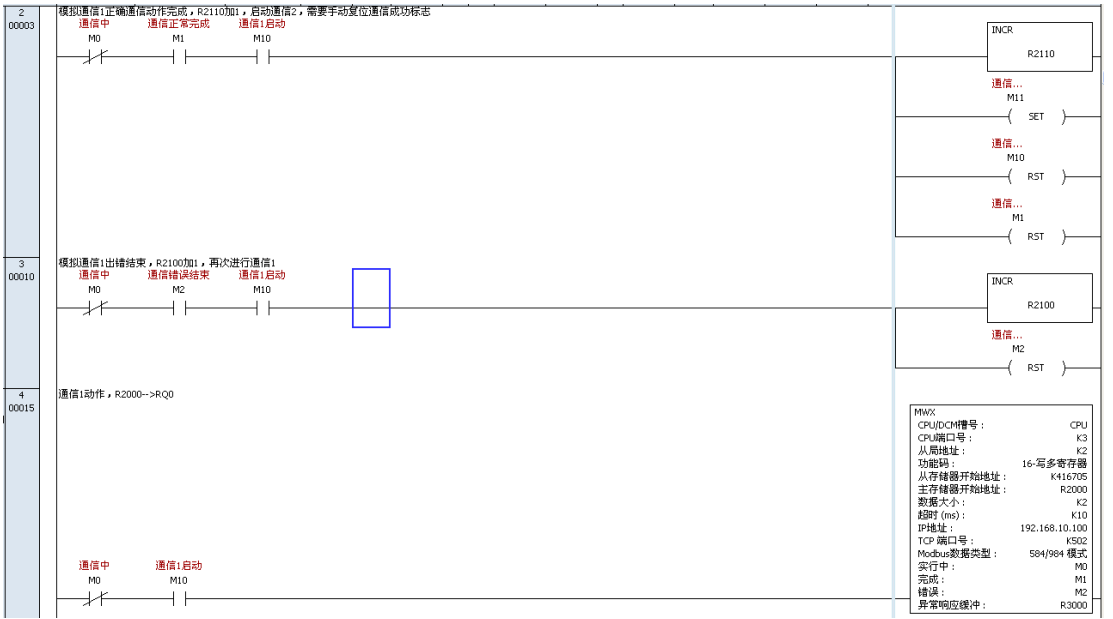
程序中使用了指令的所有 3 个通信标志。通信中标志（M0）用于判别通信口是否处于空闲状态；通信正常完成标志（M1）用于判断通信完成，实现通信指令的互锁执行；通信错误标志（M1）用于判断进行通信错误处理，并重新执行原来的通信指令，直到正确完成通信。

例子程序中用对计数器加 1 来模拟对通信的后续处理。R2110，R2112 统计正常的通信次数；R2100，R2102 统计错误的通信次数。

1) 初始程序段，复位通信标志，启动通信（SET M10）



2) 通信指令一处理，正常通信完成，R2110 加 1，启动下一条通信指令；通信出错，R2100 加 1，重新执行通信指令一，直到正常通信完成。



3) 通信指令二处理，正常通信完成，R2112 加 1，循环启动通信指令一执行；通信出错，R2102 加 1，重新执行通信指令二，直到正常通信完成。



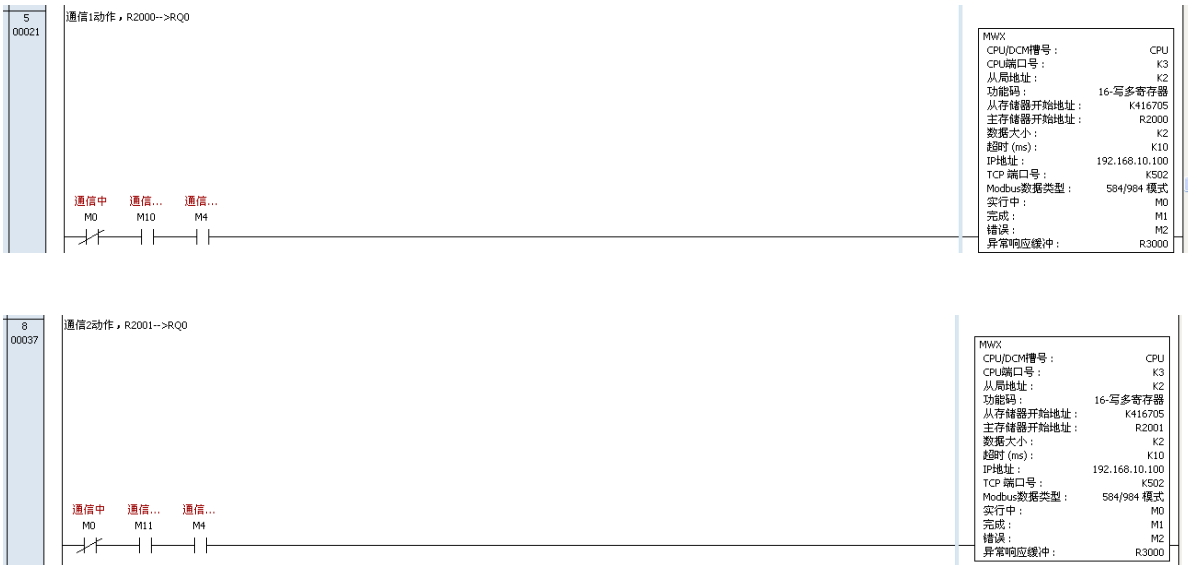
上面的通信程序利用通信标志来实现通信指令的循环执行，能够实现最快的通信指令执行速度。但有时候，我们并不需要那么快的通信指令执行速度。

为了延迟每条通信指令的执行时间，可以在每条通信指令执行条件中与上一个时间条件，例如，利用下面的指令回路就可以实现每隔一秒左右执行一次通信指令。

1) 初始程序段后添加如下回路（M4 为时间条件）



2) 每条通信指令条件中‘与’上 M4 条件



第 20 章 ASCII 指令

20.1 ASCII 输入指令（AIN）

输入 AIN 指令允许 CPU 通过指定的通讯端口接收 ASCII 字符串，并将其写入一串指定的 R 寄存器中。ASCII 数据可以以固定长度或可变长度带一个特定终止符的格式接收。本指令的其它功能包括字节交换、字符超时和用户自定义用于忙、完成和超时错误的标志位。

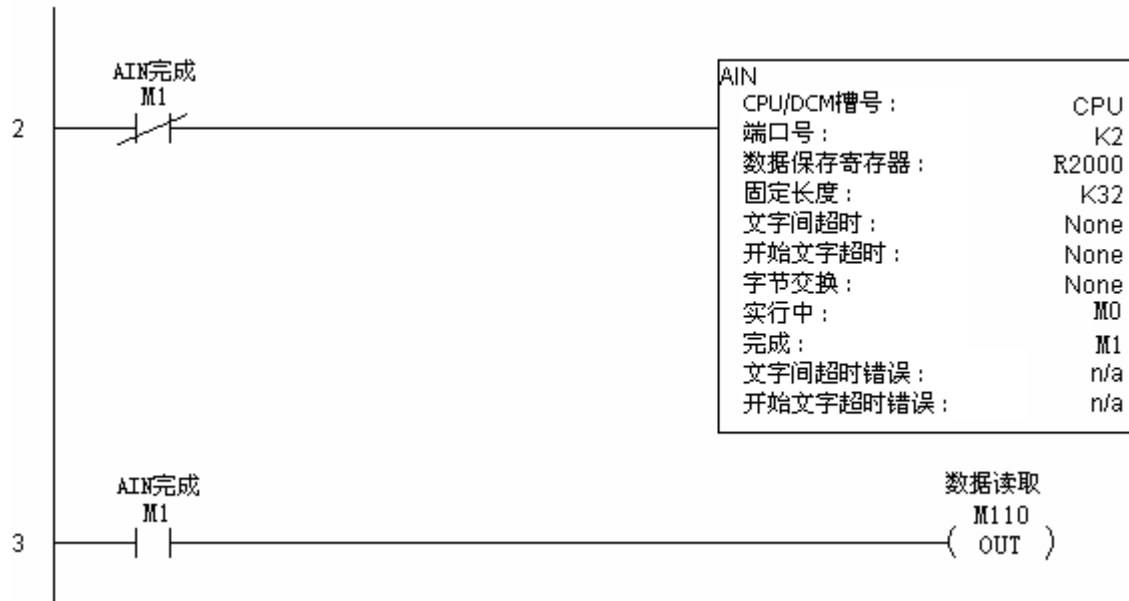
AIN 固定长度配置

- 文字列类型：选择固定长度，ASCII 字符串基于这个长度被送入 CPU 端口中。
- 端口号：0-3。
- 目标数据：指定存放 ASCII 字符串的寄存器地址。
- 固定长度：指定端口接收 ASCII 字符串的长度，单位是字节。
- 文字间超时：如果传入的 ASCII 字符之间的时间超出了设置值，指定的超时错误位将被置位。数据目标寄存器中将没有数据。当 AIN 指令许可位被禁用时此位被复位，选择 None 将禁用此功能。
- 开始文字超时：如果从 AIN 许可到首字符传入的时间超过了设定值，首字符超时位将被置位。当 AIN 指令许可位被禁用时此位被复位，选择 None 将禁用此功能。
- 字节交换：将每个 R 寄存器中的固定长度 ASCII 字符进行高位和低位交换。详细介绍可查看 SWAPB 指令的介绍。
- 执行中：当 AIN 指令正在接受 ASCII 数据时此位为 ON。
- 完成标志：当固定长度的 ASCII 数据被接受后此位被置为 ON；当 AIN 指令的许可位被禁用时此位被复位。
- 文字间超时错误：当字符间接收超时此位被置位。
- 开始文字超时错误：当首字符接收超时此位被置位。

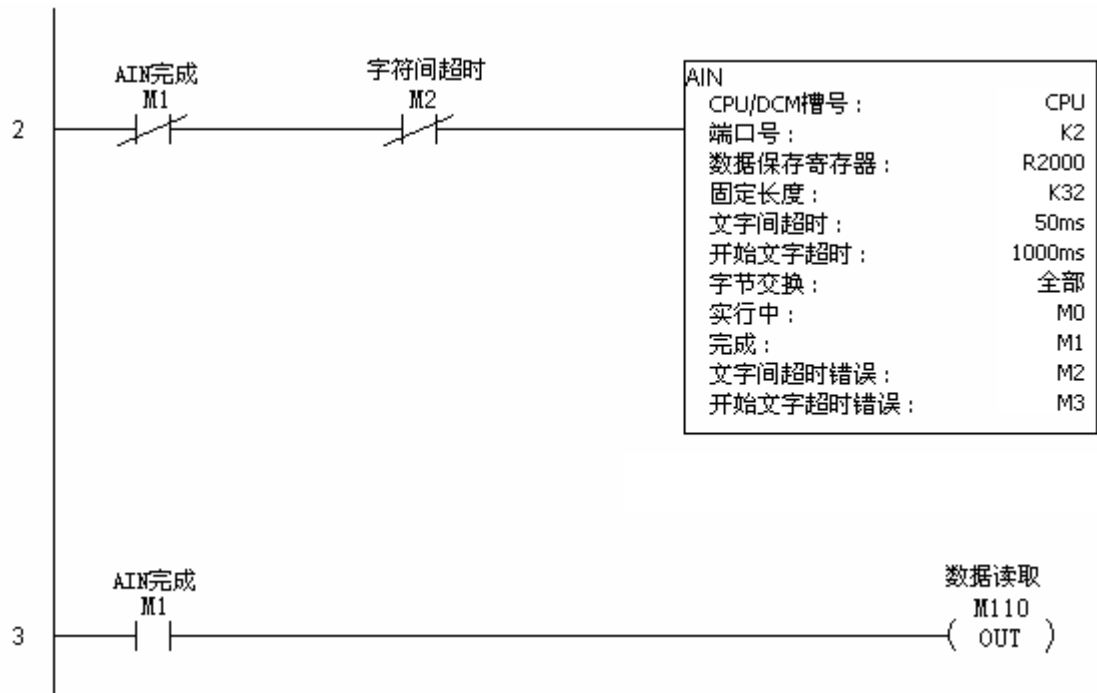
参数	NK1 范围
数据目标地址	R2000-R36777
固定长度	K1-128
位：执行中、完成、超时错误	M0-3777

AIN 固定长度程序举例

下面的程序用于 PLC 从端口连续读取数据并且时间不是很关键时。



下面的程序用于字符间时间很关键时。



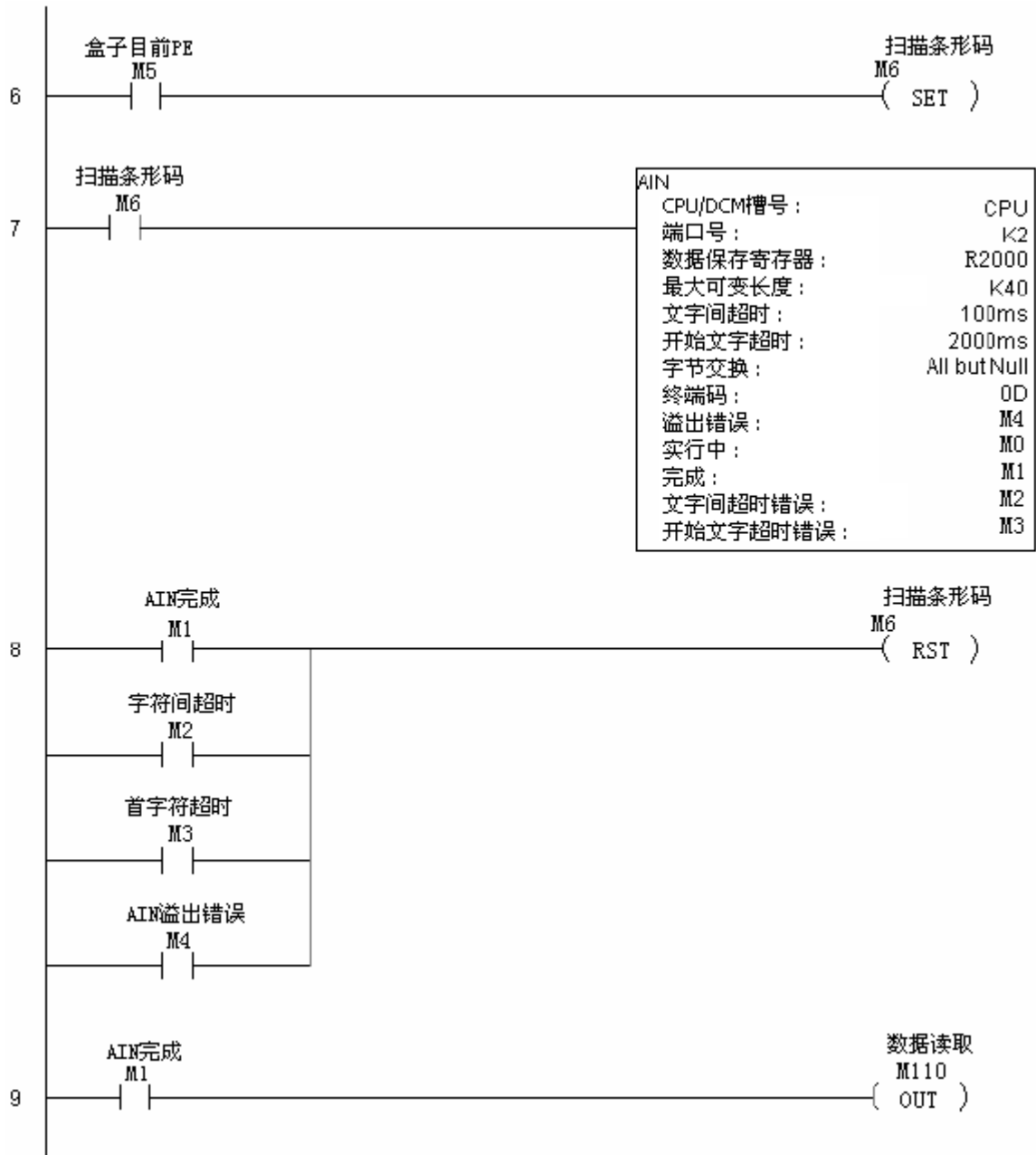
AIN 可变长度配置

- 文字列类型：如果后面带有终止字符的 ASCII 字符串长度不一致，要选择可变长度。
- 端口号：0-3。
- 目标数据：指定存放 ASCII 字符串的寄存器。
- 最大可变长度：指定端口接收 ASCII 字符串的最大长度，单位是字节。
- 文字间超时：如果传入的 ASCII 字符之间的时间超出了设置值，指定的超时错误位将被置位。数据目标寄存器中将没有数据。当 AIN 指令许可位被禁用时此位被复位，选择 None 将禁用此功能。
- 开始文字超时：如果从 AIN 许可到首字符传入的时间超过了设定值，首字符超时位将被置位。当 AIN 指令许可位被禁用时此位被复位，选择 None 将禁用此功能。
- 字节交换：将每个 R 寄存器中的固定长度 ASCII 字符进行高位和低位的交换。详细介绍可查看 SWAPB 指令的介绍。
- 执行中：当 AIN 指令正在接受 ASCII 数据时此位为 ON。
- 完成标志：当可变长度的 ASCII 数据被接受后此位被置为 ON；当 AIN 指令的许可位被禁用时此位被复位。
- 文字间超时错误：当字符间接收超时此位被置位。
- 开始文字超时错误：当首字符接收超时此位被置位。

参数	NK1 范围
数据目标地址	R2000-R36777
最大可变长度	K1-128
位：忙、完成、超时错误	M0-3777

AIN 可变长度程序举例

下面的程序用于读取盒子上的条形码（PE=光电传感器）。



20.2 ASCII 码打印指令（PRINTV）

输入 PRINTV 指令将指定的一组 R 寄存器中的指定字节数的 ASCII 字符串输出到指定的通讯端口中。其它功能包括放在字符串后的需要指定终止符的附加字符选项，字节交换选项以及用户指定忙标志位和完成标志位。

- 端口号：0-3。
- 开始地址：指定包含要打印的 ASCII 码字符串的 R 寄存器组的起始地址。
- 字节数：指定要打印字符串的长度。
- 结束码指定：指定需要指定终止符的附加在字符串后面的 ASCII 字符。
- 字节交换：将正在打印的 R 寄存器中的字符串的高位和低位互换。详细信息请查看 SWAPB 指令介绍。
- 执行中：当指令正在打印 ASCII 数据时 ON。
- 完成标志：一旦 ASCII 数据被打印完成就被置为 ON；当禁用 PRINTV 指令的允许位时此位被复位。

参数	NK1 范围
端口号	0-3
起始地址	R2000-R36777
字节数	R2000-R36777 或 K1-128
位：忙，完成	M0-3777

第 21 章 定位指令（NK0 不支持）

21.1 原点搜索指令（MT_ORG）（IB-2200）

输入 原点搜索指令用于实现 NK1 脉冲输出的回原点功能。完成原点搜索后，当前位置值自动恢复为 0，当前速度也自动降为 0。

MT_ORG (IB-2200)	
原点搜索	
动作轴：	K1
原点搜索方法：	K1
原点搜索速度：	K0
爬行速度：	K0

参数	含义	可用操作数	说明	运行中更改
Par1	指定动作轴	K	1：第一轴 2：第二轴	不可
Par2	原点搜索方法设定	K, R, P	1： 方法 1（近点信号方法） 2： 方法 2（无近点信号方法）	不可
Par3	原点搜索速度	K, R, P	1Hz ~ 最高速度（32 位 BCD）	不可
Par4	原点爬行速度	K, R, P	1Hz ~ 最高速度（32 位 BCD）	不可

使用原点搜索功能时的注意点：

1)、原点搜索动作中需要用到正向极限信号、反向极限信号、近点信号和原点信号 4 个位置信号，需要确保近点信号、原点信号在正向极限信号和反向极限信号之间；

2)、原点搜索动作中，需要用到搜索速度和爬行速度 2 个速度，从搜索速度到爬行速度的变换减速度由脉冲输出功能基本参数中的减速时间决定。

3)、原点搜索完成后的停止动作为立即停止动作。

4)、原点搜索动作正常完成后，系统自动置位“原点搜索完成”标志（M1703/M1723），该标志保持到执行下一条定位指令时，被自动复位。

5)、原点搜索过程中，如果在检测到反向限位（正向限位）信号并进行动作反向后，直到检测到正向限位（反向限位）信号，都没有检测到近点信号或原点信号，则 NK1 停止原点搜索动作，置位“定位异常标志”，并把错误信息写入“错误警告寄存器”中，原点搜索完成标志不置位。

根据是否使用近点信号，NK1 的原点搜索分成带近点信号和不带近点信号 2 种搜索方法。另外，再根据启动原点搜索时的搜索方向设定，NK1 脉冲输出功能的原点搜索共有 4 种搜索方式。下面分别说明这 4 种原点搜索方式的搜索动作。

方法一、带近点信号原点搜索方法

该原点搜索方法中需要用到正向极限信号、反向极限信号、近点信号和原点信号 4 个位置信号。其原点搜索过程示意图如下：

在以方法一执行原点搜索指令前，先要根据原点信号和近点信号的相对位置关系，指定好原点搜索的方向，原点搜索方向由“原点搜索方向指定”特殊线圈（M1704/M1724）决定。

当原点信号在近点信号和正向限位信号之间时，原点搜索方向要设定为正向，M1704/M1724 = 0；

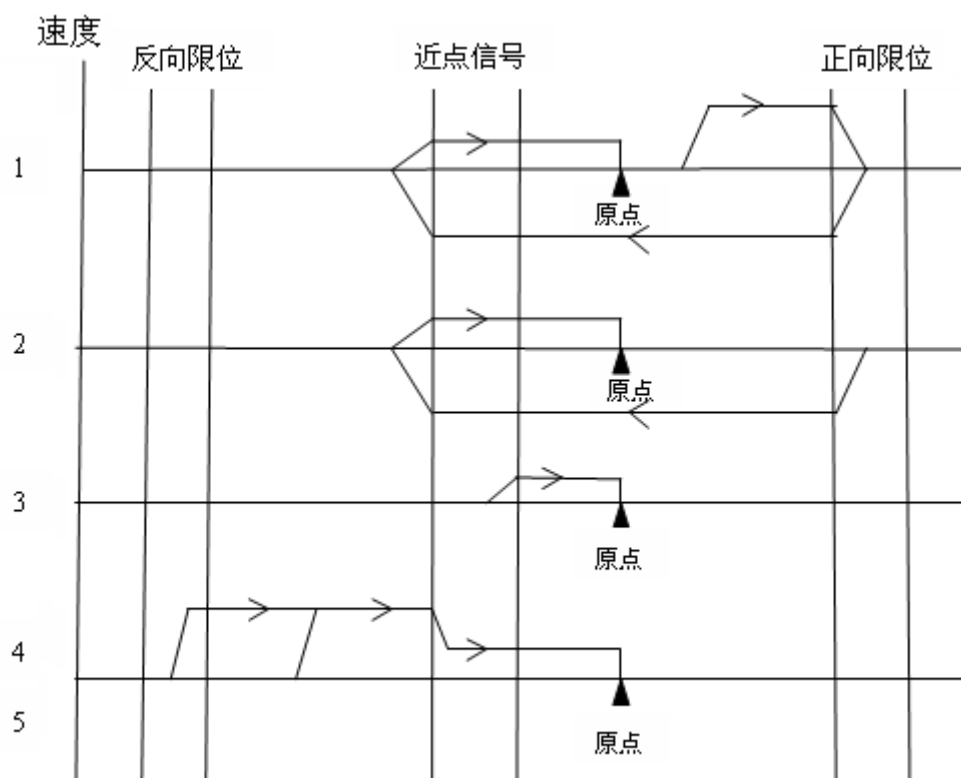
当原点信号在反向限位信号和近点信号之间时，原点搜索方向要设定为反向，M1704/M1724 = 1；

若原点搜索方向指定不正确，执行原点搜索指令后，系统会报原点搜索失败错误。

方法一中若近点信号与原点信号指定相同的输入，则不使用原点信号。当近点信号从 ON 到 OFF（近点信号逻辑反转特殊辅助线圈为 0）时，立即停止脉冲输出，完成原点搜索动作。

执行原点搜索动作时，根据开始原点搜索时的位置不同，原点搜索动作也各不相同。下面说明方法一下执行原点搜索指令后的各原点搜索动作。

搜索方式 1 原点信号在近点信号和正向限位信号之间，正向搜索时的搜索动作。



情形 1、原点搜索开始位置在近点信号与正向限位之间的时候

- ① 原点搜索启动后，以原点搜索速度，向正方向（原点搜索方向）开始移动；
- ② 正向限位信号 ON 后，即减速反转，以原点搜索速度向反方向移动；
- ③ 近点信号由 ON 变为 OFF 后减速反转，以爬行速度向正方向移动；
- ④ 近点信号由 ON 变为 OFF 后，原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

作。

情形 2、原点搜索开始位置在正向限位上的时候

- ① 原点搜索启动后，以原点搜索速度向反方向移动；
- ② 近点信号由 ON 变为 OFF 后减速反转，以爬行速度向正方向移动；
- ③ 近点信号由 ON 变为 OFF 后，原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

作。

情形 3、原点搜索开始位置在近点信号上的时候

- ① 原点搜索启动后，以爬行速度向正方向移动；
- ② 近点信号由 ON 变为 OFF 后，原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

作。

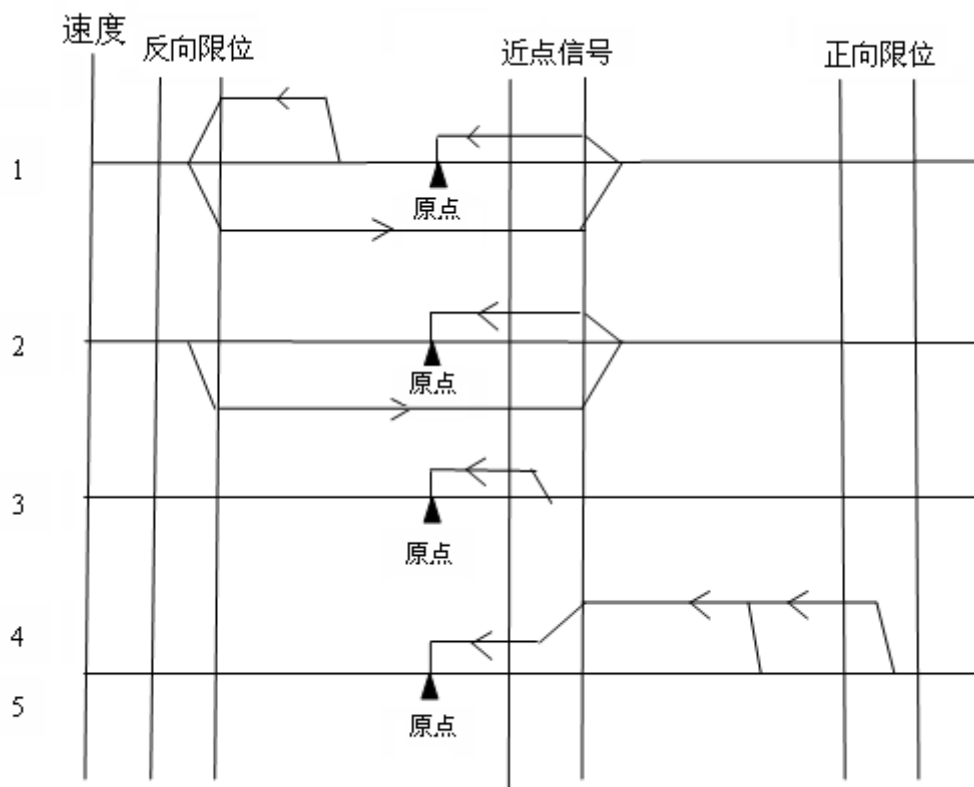
情形 4、原点搜索开始位置在反向限位信号和近点信号之间的时候

情形 5、原点搜索开始位置在反向限位上的时候

- ① 原点搜索启动后，以原点搜索速度向正方向移动；
- ② 近点信号由 OFF 变为 ON 后减速，以爬行速度继续向正方向移动；
- ③ 近点信号由 ON 变为 OFF 后，原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

情形 4/情形 5 中，如果在搜索速度转为爬行速度的过程中，检测到原点信号由 OFF 变为 ON，则立即停止，完成原点搜索动作。

搜索方式 2 原点信号在反向信号和近点限位信号之间，反向搜索时的搜索动作。



情形 1、原点搜索开始位置在反向限位和近点信号之间的时候

- ① 原点搜索启动后，以原点搜索速度，向反方向（原点搜索方向）开始移动；
- ② 反向限位信号 ON 后，即减速反转，以原点搜索速度向正方向移动；
- ③ 近点信号由 ON 变为 OFF 后减速反转，以爬行速度向反方向移动；
- ④ 近点信号由 ON 变为 OFF 后，原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

作。

情形 2、原点搜索开始位置在反向限位上的时候

- ① 原点搜索启动后，以原点搜索速度向正方向移动；
- ② 近点信号由 ON 变为 OFF 后减速反转，以爬行速度向反方向移动；
- ③ 近点信号由 ON 变为 OFF 后，原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

作。

情形 3、原点搜索开始位置在近点信号上的时候

- ① 原点搜索启动后，以爬行速度向反方向移动；
- ② 近点信号由 ON 变为 OFF 后，原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

情形 4、原点搜索开始位置在近点信号和正向限位信号之间的时候

情形 5、原点搜索开始位置在正向限位上的时候

- ① 原点搜索启动后，以原点搜索速度向反方向移动；
- ② 近点信号由 OFF 变为 ON 后减速，以爬行速度继续向反方向移动；
- ③ 近点信号由 ON 变为 OFF 后，原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

情形 4/情形 5 中，如果在搜索速度转为爬行速度的过程中，检测到原点信号由 OFF 变为 ON，则立即停止，完成原点搜索动作。

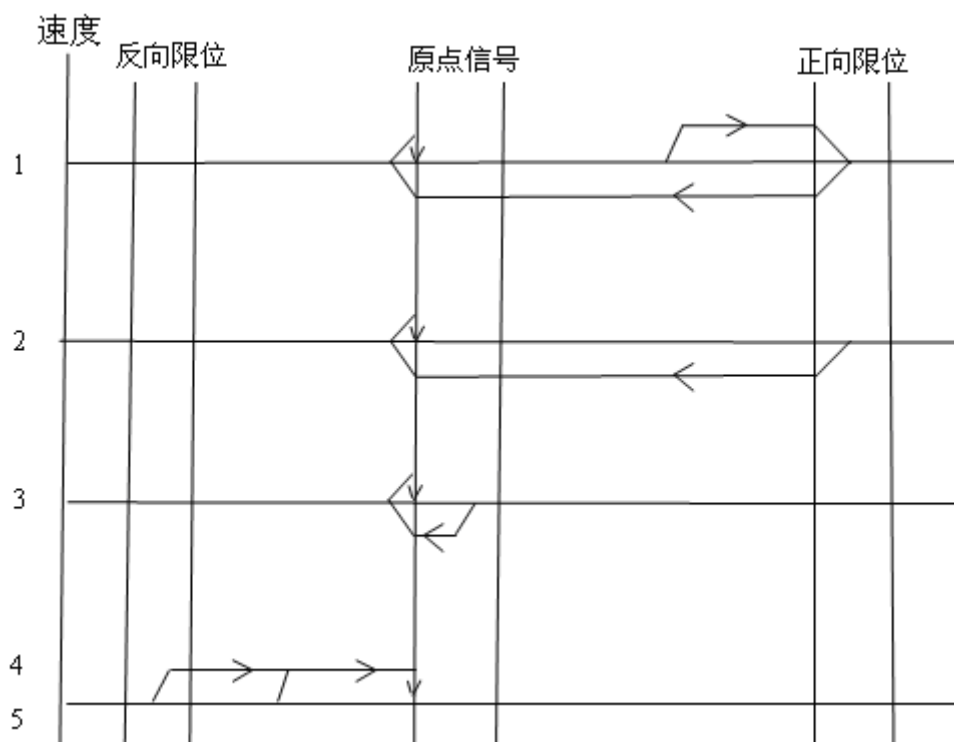
方法二、无近点信号原点搜索方法

该原点搜索方法适用于只有原点信号，没有近点信号的情形。

使用该原点搜索方法时需要用到正向极限信号、反向极限信号和原点信号 3 个位置信号。

根据启动原点搜索时的原点搜索方向，方法二又分以下 2 种搜索方式。

搜索方式 3 无近点信号下，开始原点搜索方向为正方向时的搜索动作



情形 1、原点搜索开始位置在原点信号与正向限位之间的时候

- ① 原点搜索启动后，以爬行速度，向正方向（原点搜索方向）开始移动；
- ② 正向限位信号 ON 后，即减速反转，以爬行速度向反方向移动；
- ③ 原点信号由 ON 变为 OFF 后减速反转，以爬行速度向正方向移动；

- ④ 原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

情形 2、原点搜索开始位置在正向限位上的时候

- ① 原点搜索启动后，以爬行速度向反方向移动；
- ② 原点信号由 ON 变为 OFF 后减速反转，以爬行速度向正方向移动；
- ③ 原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

情形 3、原点搜索开始位置在原点信号上的时候

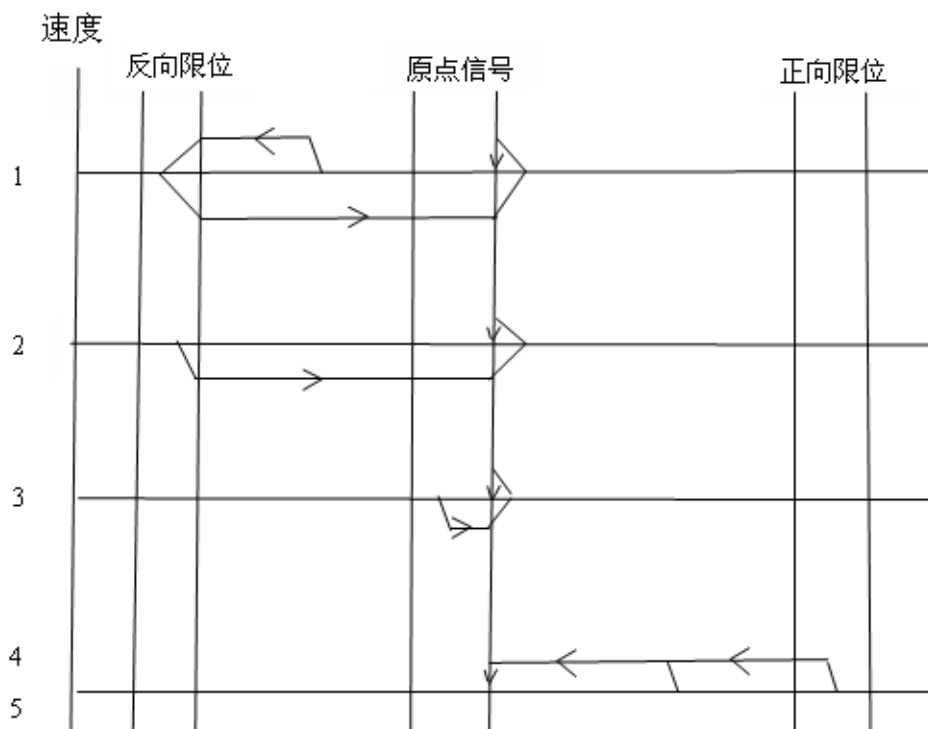
- ① 原点搜索启动后，以爬行速度向反方向移动；
- ② 原点信号由 ON 变为 OFF 后减速反转，以爬行速度向正方向移动；
- ③ 原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

情形 4、原点搜索开始位置在反向限位信号和原点信号之间的時候

情形 5、原点搜索开始位置在反向限位上的时候

- ① 原点搜索启动后，以爬行速度向正方向移动；
- ② 原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

搜索方式 4 无近点信号下，开始原点搜索方向为反方向时的搜索动作



情形 1、原点搜索开始位置在反向限位与原点信号之间的時候

- ① 原点搜索启动后，以爬行速度，向反方向（原点搜索方向）开始移动；
- ② 反向限位信号 ON 后，即减速反转，以爬行速度向正方向移动；
- ③ 原点信号由 ON 变为 OFF 后减速反转，以爬行速度向反方向移动；
- ④ 原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

情形 2、原点搜索开始位置在反向限位上的时候

- ① 原点搜索启动后，以爬行速度向正方向移动；
- ② 原点信号由 ON 变为 OFF 后减速反转，以爬行速度向反方向移动；
- ③ 原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

情形 3、原点搜索开始位置在零点信号上的时候

- ① 原点搜索启动后，以爬行速度向正方向移动；
- ② 原点信号由 ON 变为 OFF 后减速反转，以爬行速度向反方向移动；
- ③ 原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

情形 4、原点搜索开始位置在零点信号和正向限位信号之间的时候

情形 5、原点搜索开始位置在正向限位上的时候

- ① 原点搜索启动后，以爬行速度向反方向移动；
- ② 原点信号由 OFF 变为 ON 时停止，完成原点搜索动作。

21.2 绝对位置定位指令（MT_ABS）（IB-2201）

输入

MT_ABS 指令是以规定的速度运动到指定位置的动作指令。动作参数中的目标位置设定值是 32 位数据，其数据格式可以是 BCD 或 HEX，最高位为 1 表示负数设定值（BCD 格式和 HEX 格式下不同）。指令中的定位速度如果是用 R/P 寄存器指定的话，则可以在定位动作中通过改变寄存器的值而更改定位动作速度。

MT_ABS (IB-2201)

绝对位置定位

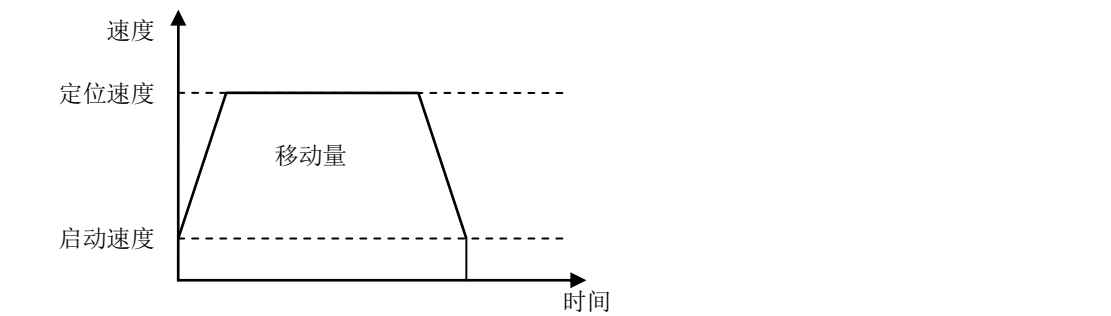
动作轴：K1

目标位置：K0

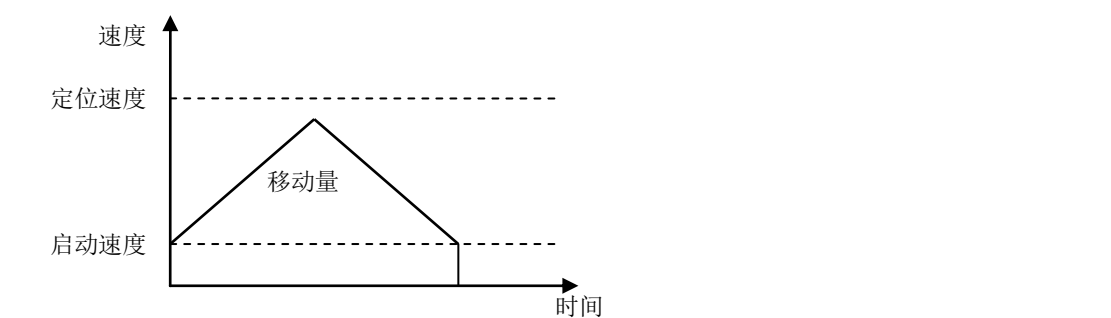
定位速度：K0

参数	含义	可用操作数	说明	运行中更改
Par1	指定动作轴	K	1：第一轴 2：第二轴	不可
Par2	目标位置设定	K, R, P	89999999 ~ 9999999（BCD 数格式时） 0x80000000~0x7FFFFFFF（HEX 数格式时）	不可
Par3	定位速度	K, R, P	1Hz ~ 最高速度（32 位 BCD）	可

执行该指令时，NK1 系统会根据当前位置值、指令目标位置值、定位速度值、以及基本参数中的加减速时间设定内容，自动计算出需要的运动方向、移动位置量数据，进行阶梯形位置控制。



当根据设定的参数所计算出来的加速距离和减速距离之和大于总的移动量（系统自动计算目标位置值和当前位置值之差）时，系统自动变为三角定位运行方式（未达到定位速度，或者正好达到定位速度），如下图：



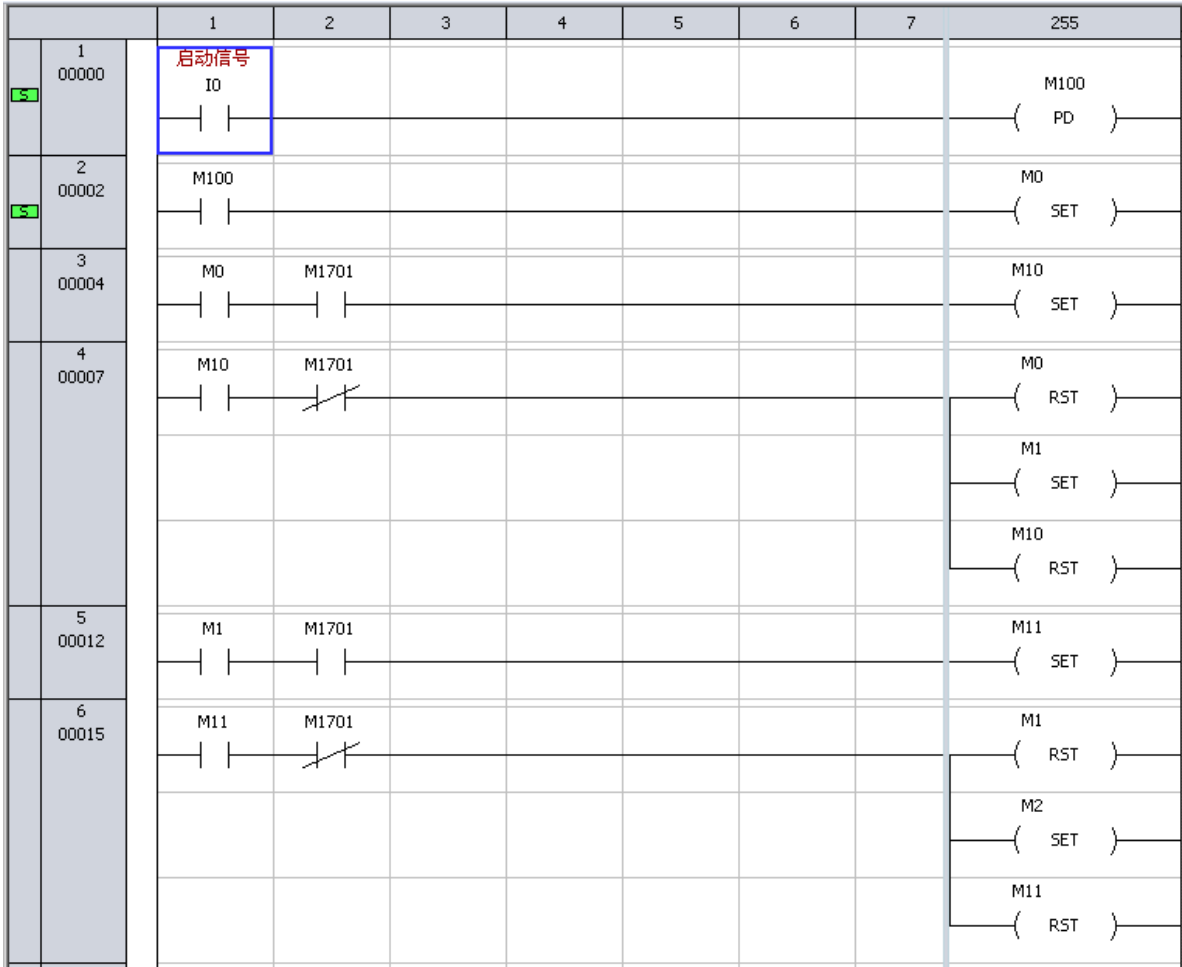
下面给出一个多个绝对位置定位指令连续动作的例子

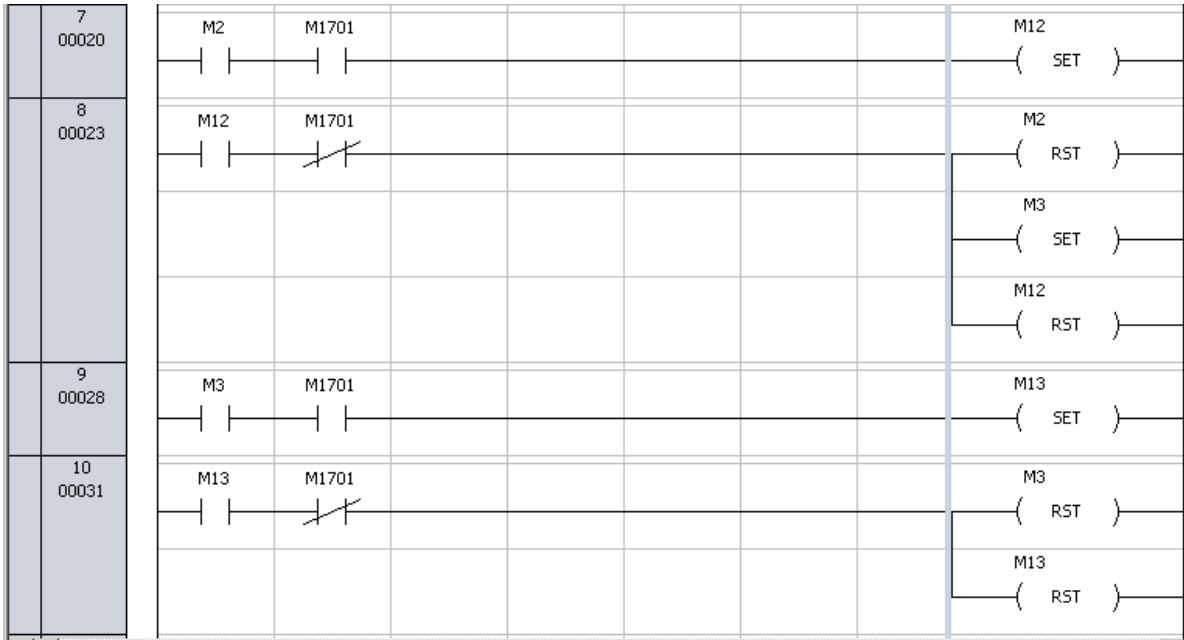
各 MT_ABS 指令动作参数以及程序控制用控制线圈如下表:

指令序号	指定动作轴	目标位置设定	定位速度	程序中使用控制线圈
1	1	10000	2000	M0 M10
2	1	20000	2000	M1 M11
3	1	80010000 (-10000)	2000	M2 M12
4	1	0	2000	M3 M13

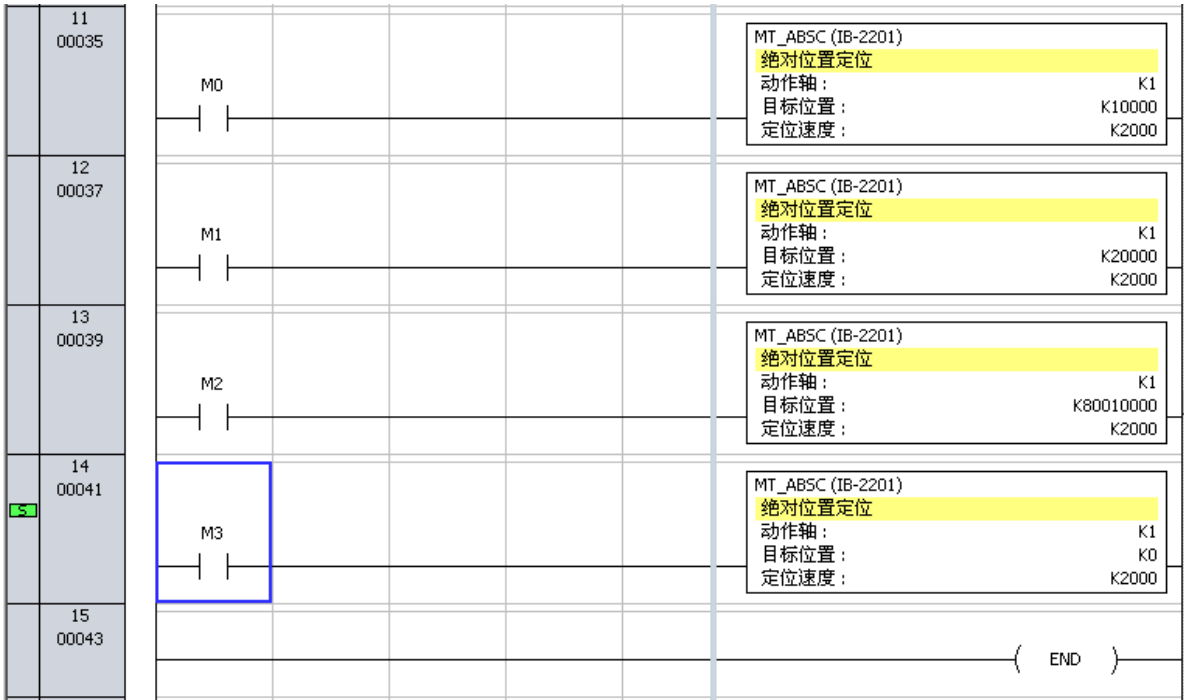
编制的用户程序如下:

1) 动作条件控制程序段, 使用 I0 信号启动一次定位动作。程序段中用到定位动作中标志, 来有效控制各定位指令的执行时序。

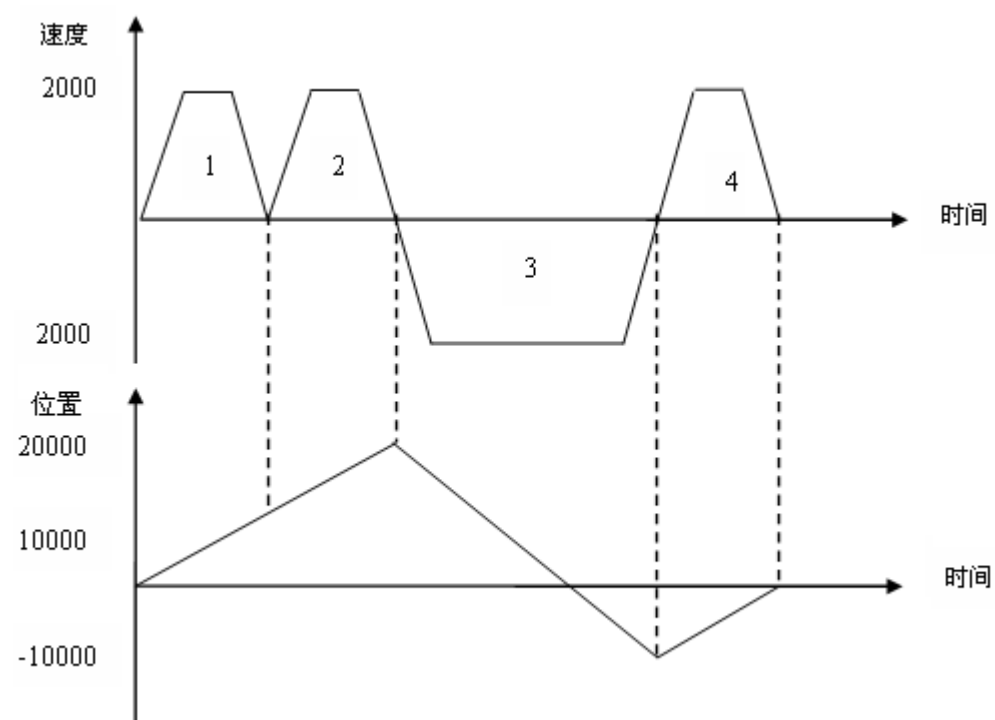




2) 定位指令程序段，给出实现连续绝对值定位指令动作的程序段。



执行以上定位指令段一次，其实现的动作时序图如下。例子中假设定位动作开始时，当前位置值为0，执行一次定位指令段后，当前位置又回到0。



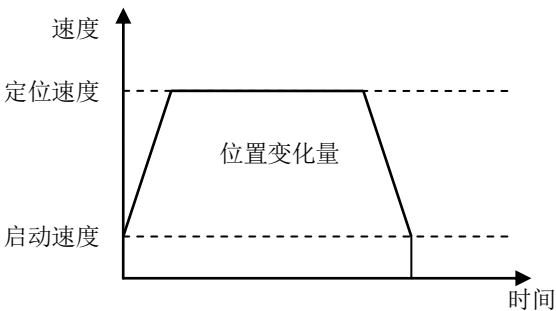
21.3 相对位置定位指令（MT_INCC）（IB-2202）

输入 MT_INCC 指令是以规定的速度移动指定的位置移动量的动作。动作参数中的相对位置设定值是 32 位数据，其数据格式可以是 BCD 或 HEX，最高位为 1 表示负数设定值（BCD 格式和 HEX 格式下不同），表示向反方向移动指定的位置移动量。指令中的定位速度如果是用 R/P 寄存器指定的话，则可以在定位动作中通过改变寄存器的值而更改定位动作速度。

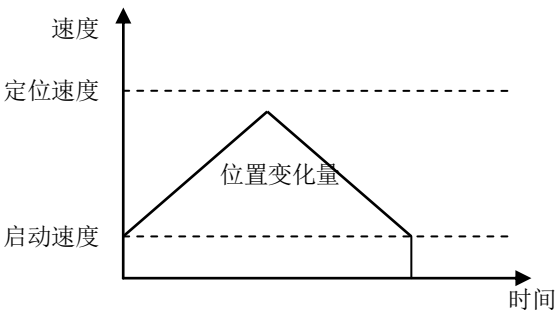
MT_INCC (IB-2202)	
相对位置定位	
动作轴：	K1
位置变化值：	K0
定位速度：	K0

参数	含义	可用操作数	说明	运行中更改
Par1	指定动作轴	K	1：第一轴 2：第二轴	不可
Par2	相对位置设定	K, R, P	89999999 ~ 99999999（BCD 数格式时） 0x80000000~0x7FFFFFFF（HEX 数格式时）	不可
Par3	定位速度	K, R, P	1Hz ~ 最高速度（32 位 BCD）	可

执行该指令时，NK1 系统会根据指令相对位置设定值、定位速度值、以及基本参数中的加减速时间设定内容，自动计算得出需要的运动方向、移动位置量数据，进行阶梯形位置控制。



当根据设定的参数所计算出来的加速距离和减速距离之和大于目标移动量时，系统自动变为三角定位运行方式（未达到定位速度，或者正好达到定位速度），如下图：

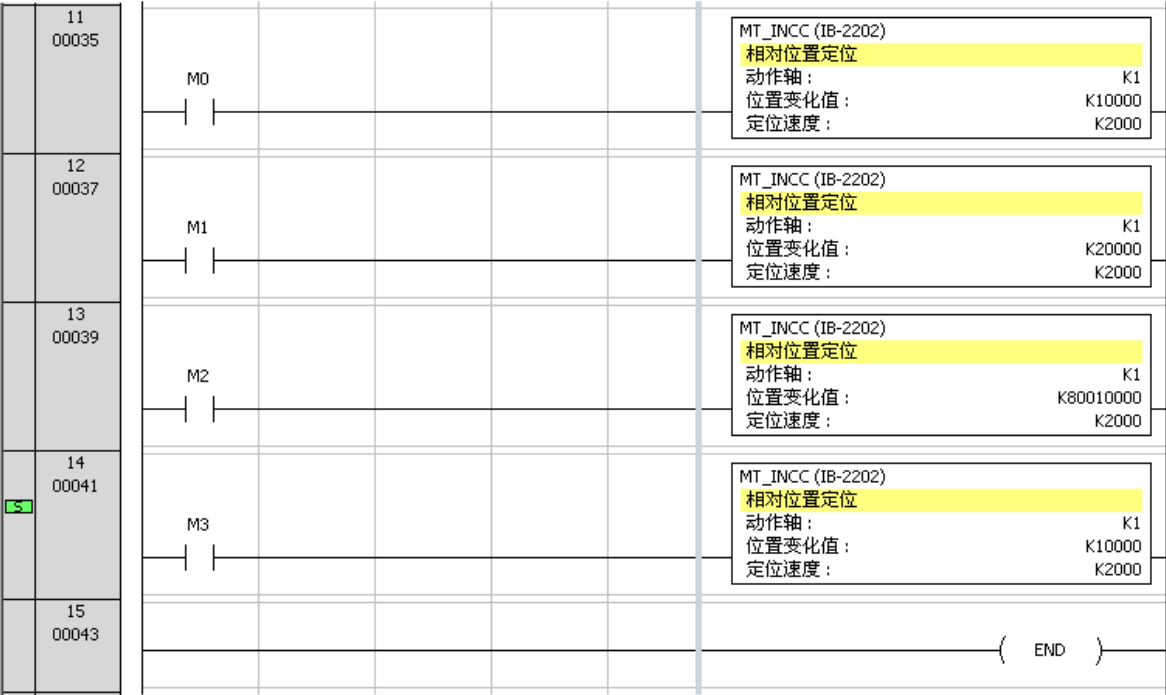


下面给出一个多个相对位置定位指令连续动作的例子

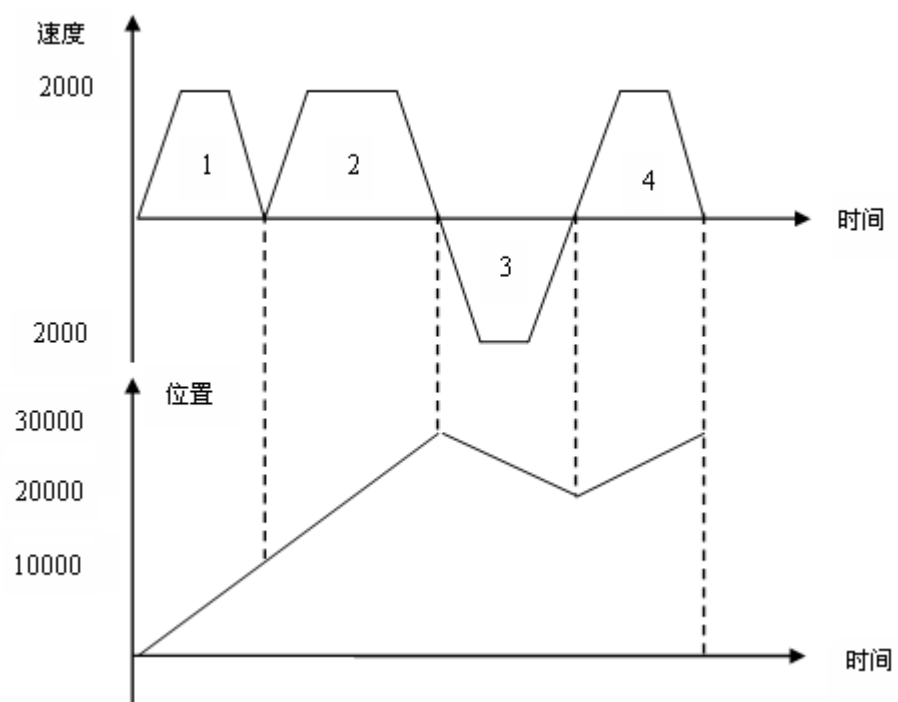
各 MT_INCC 指令动作参数以及程序控制用控制线圈如下表:

指令序号	指定动作轴	相对位置设定	定位速度	程序中使用控制线圈
1	1	10000	2000	M0 M10
2	1	20000	2000	M1 M11
3	1	80010000 (-10000)	2000	M2 M12
4	1	10000	2000	M3 M13

- 1) 定位指令动作条件控制程序段与绝对位置定位指令一样。
2) 连续多个相对位置定位指令程序段如下:



执行以上定位指令段一次，其实现的动作时序图如下。例子中假设定位动作开始时，定位指令开始执行时，当前位置值为 0，则执行一次定位指令段后，当前位置变为 30000；再执行一次，当前位置变为 60000，依次类推。



21.4 速度控制指令（MT_SPDC）（IB-2203）

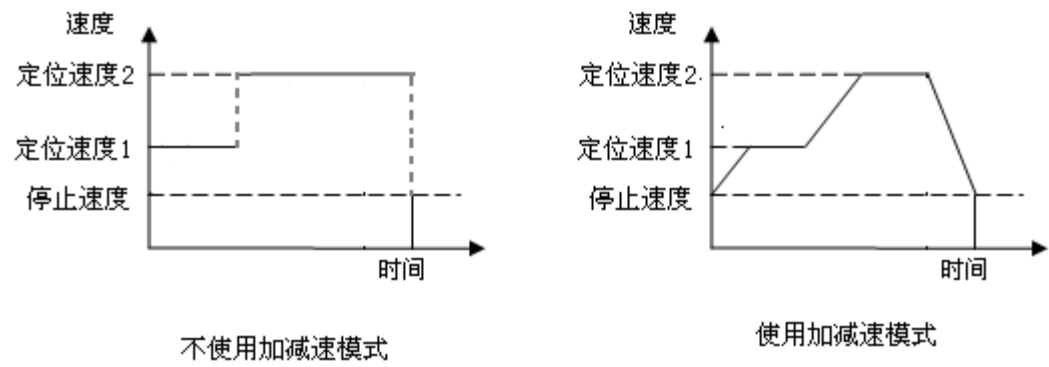
输入 该指令用于实现速度控制的目的。启动该指令后，指定轴按指定的定位速度向指定的方向输出脉冲。指令动作过程中，通过修改指令定位速度参数来实现速度控制。

MT_SPDC (IB-2203)	
速度控制	
动作轴：	K1
方向：	K0
目标速度：	K0

参数	含义	可用操作数	说明	运行中更改
Par1	指定动作轴	K	1：第一轴 2：第二轴	不可
Par2	方向	K, R, P	0：正转 1：反转	可
Par3	定位速度	K, R, P	1Hz ~ 最高速度（32 位 BCD）	可

速度控制指令使用注意点：

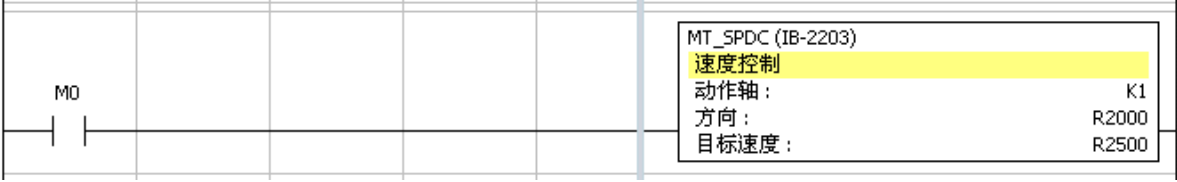
- 1) 在速度控制指令动作中，当前位置值会改变，当其当前位置值范围到达软件正限位（负限位）参数寄存器设置的值时，速度控制指令仍然有效执行，动作轴上仍然有正向（反向）脉冲输出，但当前位置值不再增加（减小）。
- 2) 在速度控制指令动作中，正反向限位特殊线圈仍然有效，速度控制指令动作中，一旦检测到正向或反向限位特殊线圈成立，则系统停止速度控制动作，置位“定位异常标志”，并把错误信息写入“错误警告寄存器”中。轴 1 的正向限位特殊线圈为 M1705，反向限位特殊线圈为 M1706；轴 2 的正向限位特殊线圈为 M1725，反向限位特殊线圈为 M1726。
- 3) 速度控制指令动作中，如果其方向和定位速度参数是用 R 或 P 寄存器指定的，则可以通过修改寄存器内容，来实现动作中方向和定位速度参数的改变。如果修改参数有错误，则系统停止速度控制动作，置位“定位异常标志”，并把错误信息写入“错误警告寄存器”中。
- 4) 使用特殊线圈“速度控制模式中是否带加减速”（轴 1：M1711，轴 2：M1731）可定义速度改变的方式：指令动作中修改定位速度时，如果选择不使用加减速，则直接从原来速度跳变到新的速度；如果指定使用加减速时，则按基本动作参数规定的加/减速时间从原来的速度加速/减速改变到新的速度。



- 5) 改变方向参数时，如果指定不使用加减速，则动作轴速度先从当前速度跳变到 0，然后从 0 跳变到定位速度。如果指定使用加减速，则动作轴先按设定减速时间减速到 0，然后按加速时间加速到定位速度。

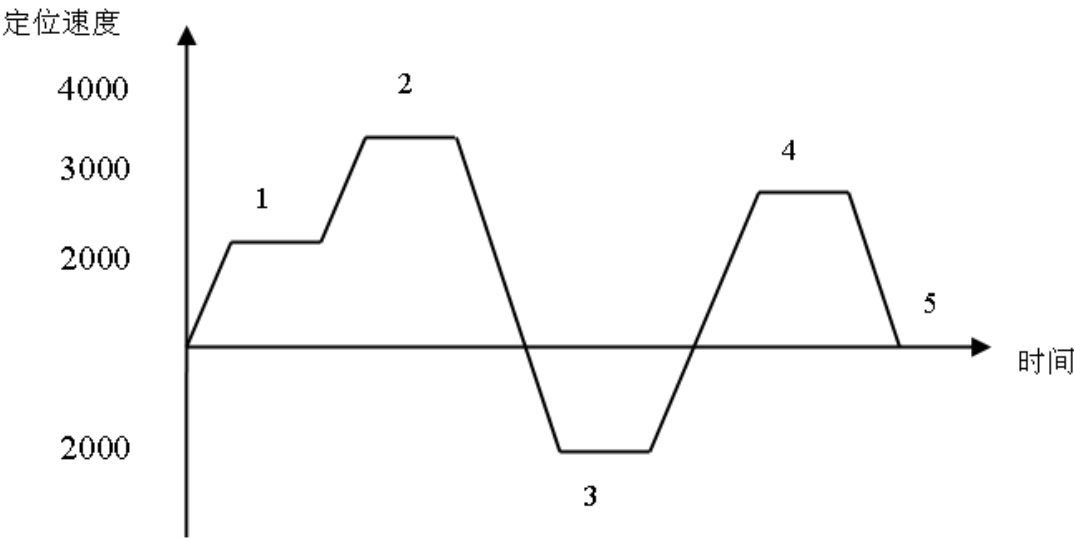
6) 可以使用 MT_STPM 指令或指定定位速度为 0 的方式来停止速度控制指令的动作。使用 MT_STPM 指令时的停止动作模式，由指令参数决定。

下面给出速度控制指令执行中修改定位速度和方向的例子，例子中假设使用加减速控制，执行的速度控制指令为：

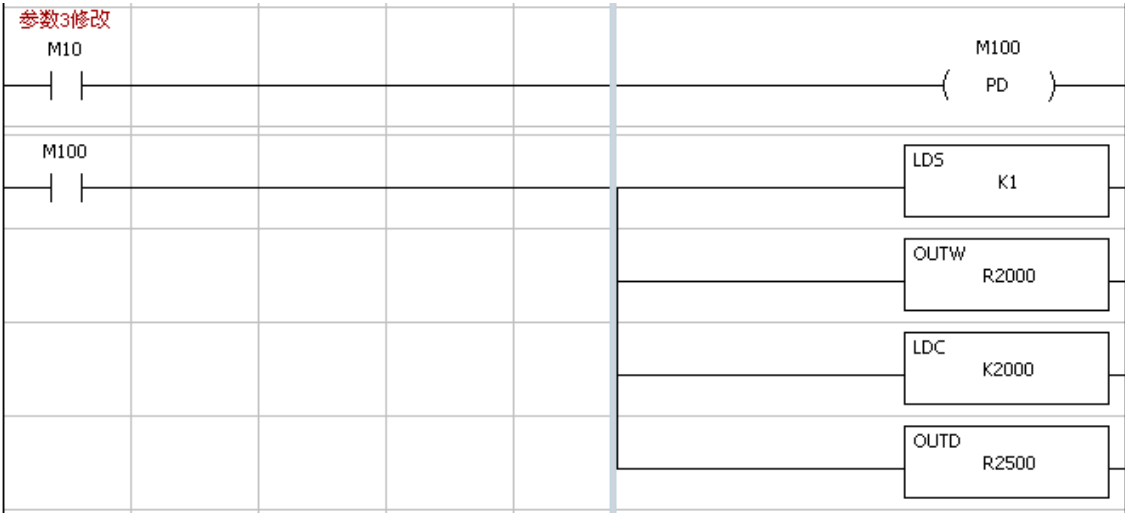


修改指令参数的步骤及实现的动作时序图如下：

参数修改步骤	指定动作轴	方向 (R2000)	定位速度 (R2500)
1	1	0	2000
2	1	0	4000
3	1	1	2000
4	1	0	3000
5	1	0	0



指令参数的修改可以通过用户程序对相关寄存器的赋值来实现，例如上面参数修改步骤 3 可以通过以下程序段实现。



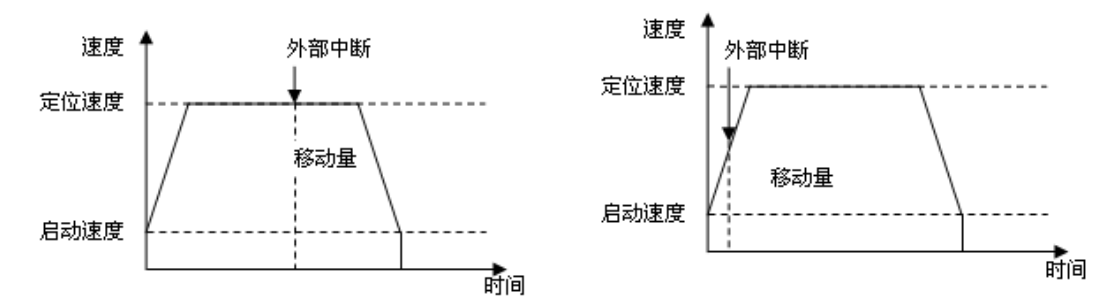
21.5 中断定位指令（MT_INTC）（IB-2204）

输入 MT_INTC 是中断定位指令。

MT_INTC (IB-2204)	
中断定位	
动作轴：	K1
方向：	K0
目标速度：	K0
中断定位距离：	K0

参数	含义	可用操作数	说明	运行中更改
Par1	指定动作轴	K	1：第一轴 2：第二轴	不可
Par2	方向	K, R, P	0：正转 1：反转	可
Par3	定位速度	K, R, P	1Hz ~ 最高速度（32 位 BCD）	可
Par4	中断定位距离 （相对距离）	K, R, P	89999999 ~ 99999999（BCD 数格式时） 0x80000000~0x7FFFFFFF（HEX 数格式时）	不可

指令启动后，按指定方向、设定加速时间加速到定位速度后进行速度控制动作，这时通过修改指令中速度参数可以实现速度的改变，修改方向参数可以实现动作方向的改变。速度控制动作过程中，检测到“中断信号输入”特殊线圈（轴 1：M1712，轴 2：M1732），则自动把控制方式改变到位置控制方式，输出参数“中断定位距离”规定的脉冲数后减速停止，完成定位动作。



- 中断定位指令使用注意点：
- 1) 中断信号来以前的动作与速度控制指令一样，为速度控制动作。
 - 2) 在中断定位指令动作中，当前位置值会改变，其当前位置值的范围必须在软件正、负限位参数寄存器设置值的范围之内。当前位置值超出该范围，系统停止定位动作，置位“定位异常标志”，并把错误信息写入“错误警告寄存器”中。
 - 3) 中断定位指令动作中，如果其方向和定位速度参数是用 R 或 P 寄存器指定的，则可以通过修改寄存器内容来实现动作中方向和定位速度参数的改变。如果修改参数有错误，则系统停止中断定位动作，置位“定位异常标志”，并把错误信息写入“错误警告寄存器”中。
 - 4) 中断定位指令中的“中断定位距离”参数指的是相对位置量。
 - 5) “中断信号输入”特殊线圈（M1712/M1732）可以由设置成中断定位用外部中断输入点（轴 1：I6，轴 2：I7）来自动控制产生，也可以由用户自己通过用户程序控制，灵活实现中断定位功能。
 - 6) 完成中断定位动作后，系统自动复位“中断信号输入”特殊线圈（M1712/M1732）。
 - 7) 利用“中断信号逻辑反转”特殊线圈（M1710/M1730），可以实现“中断输入信号”

的逻辑反转，从正逻辑有效转为负逻辑有效。该逻辑反转特殊线圈设定对“中断输入信号”特殊线圈（M1712/M1732）和 I06，I07 都有效。

- 8) 检测到中断输入信号，开始中断定位时，当当前动作方向与“中断定位距离”参数方向一致时，输出指定脉冲数；当 2 者方向不一致时，系统先减速停止，然后向反方向运动，输出脉冲数从速度 0 时开始计算。
- 9) 如果在加速阶段产生中断信号，则系统继续加速到定位速度后完成中断定位动作。

21.6 当前位置变更指令（MT_PSET）（IB-2212）

输入 MT_PSET 是当前位置设定指令，用于修改当前位置值到指定的数值。注意只有在指定轴处于停止状态下（M1701 或 M1721 为 OFF 状态）才能修改其当前位置值。

MT_PSET (IB-2212)	
当前位置设定	
动作轴：	K1
位置设定值：	K0

参数	含义	可用 操作数	说明	运行中 更改
Par1	指定动作轴	K	1： 第一轴 2： 第二轴	不可
Par2	目标位置设定	K, R, P	89999999 ~ 99999999（BCD 数格式时） 0x80000000~0x7FFFFFFF（HEX 数格式时）	不可

21.7 复位指令（MT_RST）（IB-2210）

输入 MT_RST 是轴复位指令,当指定轴在某个定位指令执行中发生错误时,就会停止指令执行,置位定位异常标志,报告错误警告信息。当对错误进行一定处理,解除错误发生的条件后,需要使用复位指令对指定轴进行复位处理,以使指定轴可以继续执行新的定位指令。

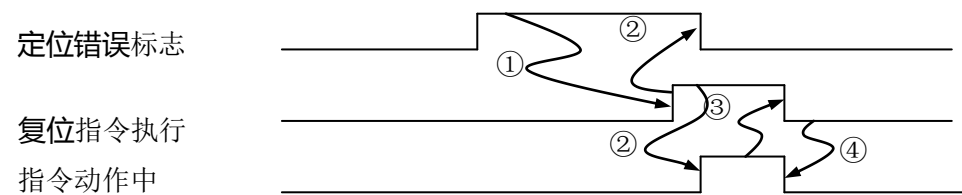
MT_RST (IB-2210)

复位

动作轴: K1

参数	含义	可用操作数	说明	运行中更改
Par1	指定动作轴	K	1: 第一轴 2: 第二轴	不可

使用复位指令的动作时序图如下：
②为复位指令中，清除定位错误标志



上图中，①为发生定位错误，系统置位并保持定位错误标志；在对错误进行处理后，执行复位指令。②为执行复位指令，置位定位指令动作中标志，清除定位错误标志。③为复位指令执行完成。④为指令动作中标志复位，可以继续使用定位指令。

21.8 停止指令（MT_STPM）（IB-2211）

输入 MT_STPM 是轴停止指令，停止指定轴的定位动作，本指令可以在其他定位指令动作中执行。

MT_STPM (IB-2211)	
停止	
动作轴：	K1
停止代码：	K0

参数	含义	可用操作数	说明	运行中更改
Par1	指定动作轴	K	1：第一轴 2：第二轴	不可
Par2	停止代码	K, R, P	停止代码（16 位 BCD） 0：非常停止 1：立刻停止	不可

使用停止指令停止定位动作时，可以选择以下 2 种停止方式：

方式一：非常停止，停止指令“停止代码”参数为‘0’时采用的停止方式。采用该种方式执行停止指令时，动作轴减速停止，减速时间由“非常停止时间”基本参数决定。

方式二：立即停止，停止指令“停止代码”参数为‘1’时采用的停止方式。采用该种方式执行停止指令时，动作轴立即停止脉冲输出动作。

21.9 多段定位指令（MT_MTBL）（IB-2214）

输入 MT_MTBL 是多段定位指令，用于实现多个连续的阶梯形定位动作，每段动作可以选择使用绝对定位或相对定位方式，本指令最多可以实现 30 个连续阶梯形定位动作。

MT_MTBL (IB-2214)	
批指令处理	
保存到：	K1

参数	含义	可用操作数	说明	运行中更改
Par1	指定动作轴	K	1：第一轴 2：第二轴	不可
Par2	设定开始寄存器	R, P	寄存器号	不可
Par3	多段段数	K, R, P	1 ~ 30	不可

多段定位动作的每段需要 8 个定位参数寄存器，在使用多段定位指令前，先要设置好每段定位动作参数寄存器。多段定位指令启动后，系统从第一段开始按段检查定位参数，如果某段参数正确，则执行该段定位动作，执行完该段定位动作后，进入下一段处理，一直到执行后所有段的定位动作。如果某段参数有问题，系统执行到该段后，立即停止，并置位“定位异常发生”标志（M1702/M1722），把错误代码存入“错误警告寄存器”，同时复位“定位动作中”标志（M1701/M1721）。另外，该错误段后的定位指令段将被忽略而不能执行。

在执行多段定位指令时，你可以随时修改没有被执行到的段的动作参数。

多段定位指令参数寄存器组成如下：

寄存器	参数名称	范围	说明
0 段	Rn+00	目标位置 89999999 ~ 9999999 (BCD 数格式时) 0x80000000~0x7FFFFFFF (HEX 数格式时)	32 位数据
	+02	定位速度 1000 ~ 100000Hz	32 位 BCD
	+04	加速时间 10 ~ 9990ms	16 位 BCD
	+05	减速时间 10 ~ 9990ms	16 位 BCD
	+06	停留时间 0 ~ 9990ms	16 位 BCD
	+07	多段指令字	HEX 数
1 段	+10	目标位置 89999999 ~ 9999999 (BCD 数格式时) 0x80000000~0x7FFFFFFF (HEX 数格式时)	32 位数据
	+12	定位速度 1000 ~ 100000Hz	32 位 BCD
	+14	加速时间 10 ~ 9990ms	16 位 BCD
	+15	减速时间 10 ~ 9990ms	16 位 BCD
	+16	停留时间 0 ~ 9990ms	16 位 BCD
	+17	多段指令字	HEX 数
最大 30 段			

参数说明

1)、 目标位置

设定该段定位动作中移动到的目标位置（绝对值方式时）或输出的脉冲数量（相对值方式时）。

2)、 定位速度

设定完成该段定位动作使用的定位速度。

3)、加速时间/减速时间

每段定位动作可以设置单独的加速（减速）时间。当该段的加速时间（减速时间）设置为 0 时，系统自动按照基本参数设置中的加速（减速）时间进行动作。

4)、停留时间

停留时间是指段与段之间的间隔时间，即本段定位动作结束，开始下一段定位动作前等待的时间。断续模式下或连续模式下下一段动作方向与当前段动作方向不一致时，停留时间段内速度为 0；连续模式下且下一段动作方向与当前段动作方向一致时，停留时间段内保持速度为当前段定位速度，停留时间段内输出的脉冲数既不包含于当前段目标位置内，也不算在下一段目标位置内。

5)、多段指令字

多段指令字寄存器用于指定本段定位动作的模式，以及本段定位动作结束后如何启动下一段定位动作或结束本多段定位指令动作。

多段指令字为 16 位寄存器，其设定如下（HEX）：

Bit15 Bit14	Bit13 Bit12	Bit11 Bit10	Bit9 Bit8	Bit7 Bit0
预留	结束方式	定位方式	预留	预留

00: 指令结束 00: 绝对值

10: 断续 01: 相对值

11: 连续

定位方式 (Bit11 Bit10): 00 表示为绝对值定位方式；01 表示相对值定位方式。

结束方式 (Bit13 Bit12): 00 表示结束当前段定位动作后就结束整个指令的定位动作；10 表示当前段为断续模式结束，完成本段定位动作后，其定位速度自动变为 0；11 表示当前段为连续模式结束，若下一段动作方向与当前段动作方向一致，那么当前段结束时速度保持为当前段定位速度，下一段动作时从本段定位速度加速（减速）到下一段定位速度；如果下一段动作方向与当前段动作方向不一致，则当前段结束时速度变为 0，下一段动作时速度从 0 加速到下一段定位速度。

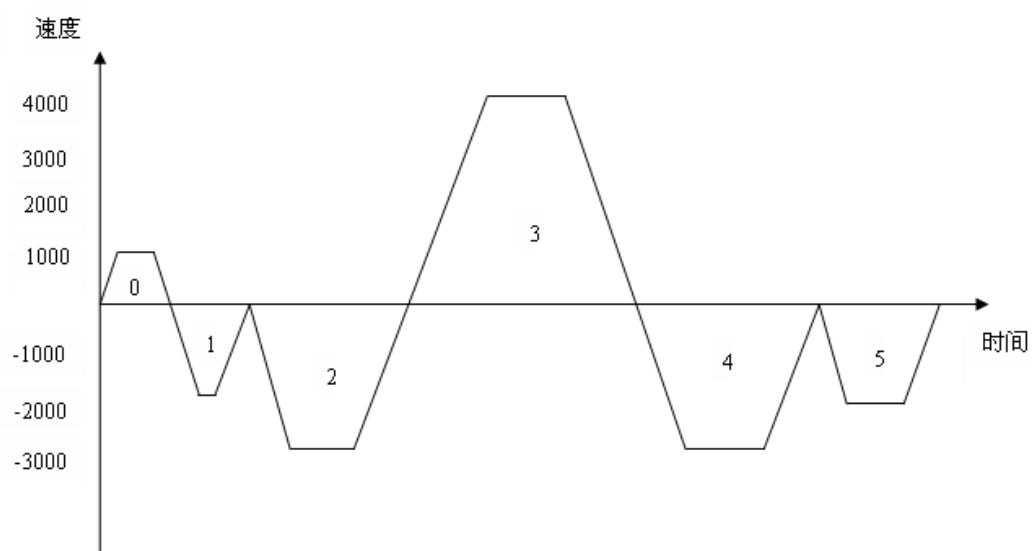
下面给出 2 个使用多段定位指令实现的动作时序图。

例一、 断续绝对值定位模式

需要设定的多段定位指令的各段参数如下表。

例一（断续绝对值模式）						
段号	位置	速度	加速时间	减速时间	停留时间	多段指令字
0	10000	1000	150	150	0	2000
1	-10000	2000	150	150	0	2000
2	-100000	3000	150	150	0	2000
3	800000	4000	150	150	0	2000
4	300000	3000	150	150	0	2000
5	0	2000	150	150	0	0000

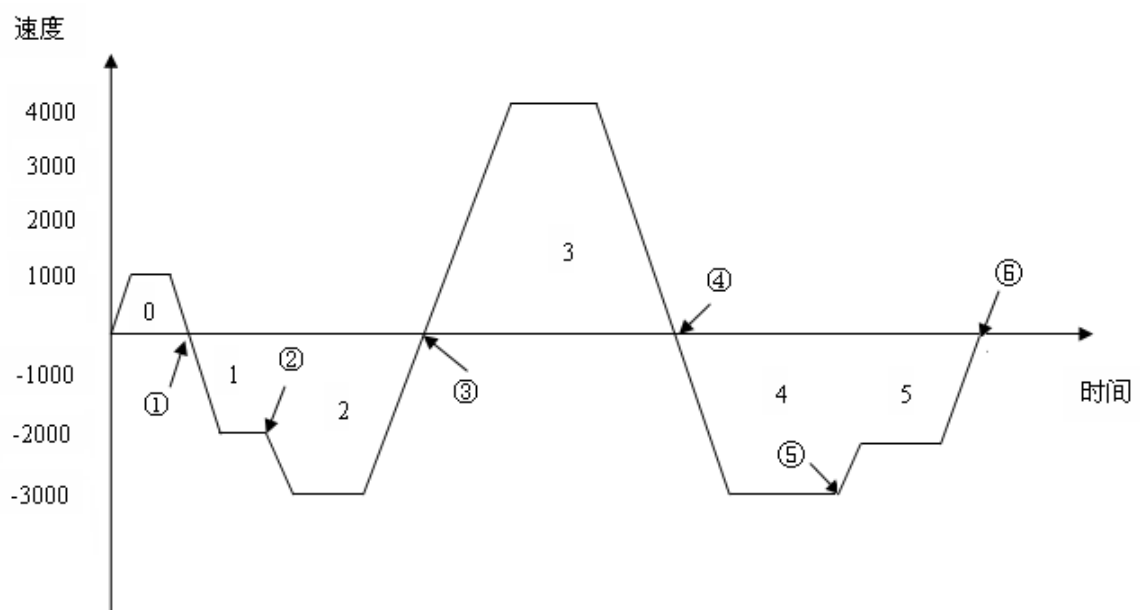
其实现的动作时序图如下，图中速度曲线与时间轴的每个交点即为 2 段定位动作的交接点。



例二、连续绝对值定位模式

需要设定的多段定位指令的各段参数如下表。

例二（连续绝对值模式）						
段号	位置	速度	加速时间	减速时间	停留时间	多段指令字
0	10000	1000	150	150	0	3000
1	-10000	2000	150	150	0	3000
2	-100000	3000	150	150	0	3000
3	800000	4000	150	150	0	3000
4	300000	3000	150	150	0	3000
5	0	2000	150	150	0	0000



上图中，第 0 段的目标位置 10000（移动量 10000）是指从定位动作开始到①时，所走的距离；

第 1 段的目标位置 -10000（移动量 -20000）是指①到②时，所走的距离；

第 2 段的目标位置 -100000（移动量 -90000）是指②到③时，所走的距离；

第 3 段的目标位置 800000（移动量 180000）是指③到④时，所走的距离；

第 4 段的目标位置 300000（移动量 -50000）是指④到⑤时，所走的距离；

第 5 段的目标位置 0（移动量 -30000）是指⑤到⑥时，所走的距离。

第 22 章 级式编程方法

22.1 级式编程介绍

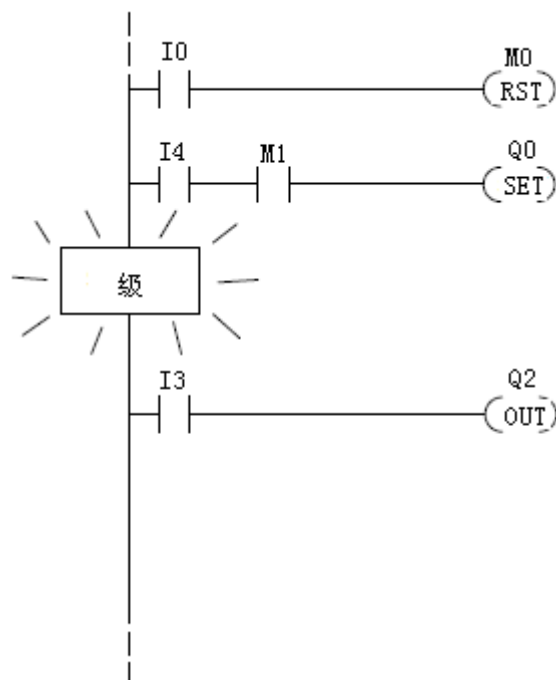
级式编程提供了比单纯的梯形图编程相对容易的方法来组织和编写复杂的应用程序。级式语言编程不能替代或否定传统的梯形图编程方法，这也是为什么级式编程也叫 RLL^{plus}。级式语言只是允许将一个梯形图程序以级为单位进行分解和重新组织，这样编程比传统的梯形图程序更快更直观。

许多用户习惯用传统的梯形图语言进行编程，对于新的编程方法比如级式语言编程往往持怀疑态度，甚至害怕去尝试。虽然传统的梯形图程序在解决各种逻辑关系方面功能很强大，但是它也有不足之处：

- 由于没什么结构，大型程序很难管理。
- 梯形图中，必须创建很多闭锁。
- 当进程被卡，很难找出错误的程序段。
- 由于不直观，程序后期修正很困难。

级式语言编程弥补了这些不足，因此，学习级式语言编程是很有必要的。本章详细介绍了级式语言的编程方法供用户学习，为了达到最佳学习效果，请按如下的要求做：

- 从头开始学，不要跳过任何内容。
- 通过例子学习级式语言编程概念。



22.2 画状态转移图

CPU 一遍一遍的重复扫描梯形图程序，扫描的基本步骤是：

- 1) 读输入
- 2) 执行梯形图程序
- 3) 写输出

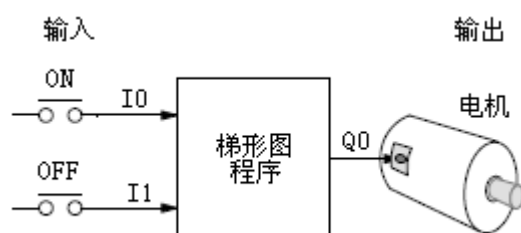
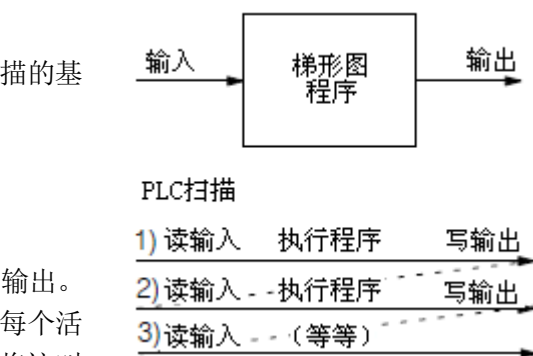
这几毫秒的时间里，输入的变化就能影响输出。

许多制造过程包含一系列的活动或条件，每个活动或条件持续数秒、数分钟甚至数小时，我们将这叫做“进程状态”，它在某个特定的时间有效或无效。梯形图程序面临的一个挑战是，一个特定的输入有可能只持续很短的时间，通常情况下，我们需要创建闭锁来保持这个输入的状态，以在需要的时间段维持一个进程状态。

可将一个梯形图程序分解成多个代表“进程状态”的“级”，然后将它们重新组织。在详细介绍级之前，首先介绍状态转移图。

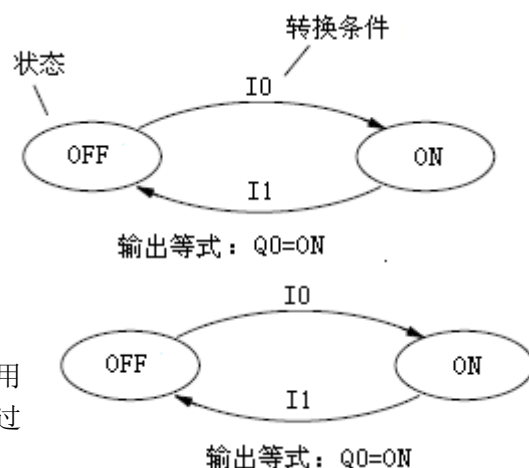
有时我们需要暂时忽略 PLC 的扫描性质，将思维集中在需要确定的进程状态上。清晰的思维和简要的分析有助于我们编写出高效、无漏洞的程序。状态图是一个帮助我们画出进程的工具，你会发现，如果我们的状态图正确，我们的程序也将是正确的。

右图是一个简单进程的例子，控制一个工业电机。使用一个绿色的瞬时 SPST 按钮来开启电机，一个红色的按钮将其关闭。操作者按相应的按钮 1 秒左右。进程的两个状态是 ON 和 OFF。



下一步是画一个状态转移图，如右图所示，显示两种状态 OFF 和 ON，用转换线条连接起来。当 I0 为 ON 时，状态转移是 OFF→ON，当 I1 为 ON 时，状态转移是 ON→OFF。

控制器的输出是 Q0，Q0=ON 状态。下面我们首先使用梯形图来执行状态图，然后再

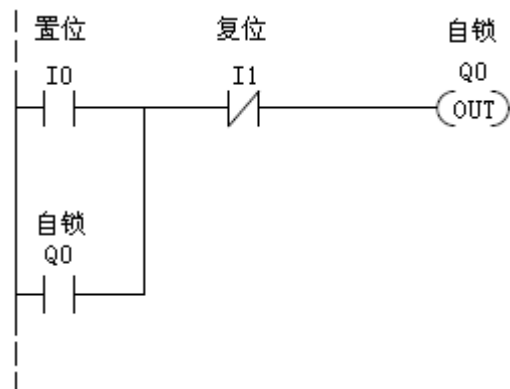


用级式编程来执行状态图，这样有助于用户看清两种方法的区别。

我们需要绘制右边的状态转移图，它的作用是：不使用编程语言描述问题，换句话说，通过绘制状态转化图，我们已经解决了控制问题。

首先，我们使用传统的梯形图程序将上述的状态图编程，然后使用级式语言将其编程。

梯形图程序：右图是梯形图程序，输出 Q0 有双重作用。当瞬时开关 I0 被按下时，线圈 Q0 被置为 ON，第二行的触点 Q0 也为 ON，进行了自锁，因此，当 I0 被按下后，Q0 就一直为 ON，电机始终处于运行状态。



当关闭按钮 I1 被按下时，常闭触点 I1 断开，将 Q0 的自锁复位，输出 Q0 被置为 OFF。

级式程序：级式程序如右图所示。两个级 S0 和 S1 对应两个状态 OFF 和 ON。级中的程序对应各自的状态，这就意味着 PLC 只需扫描状态为 ON 的那些级。

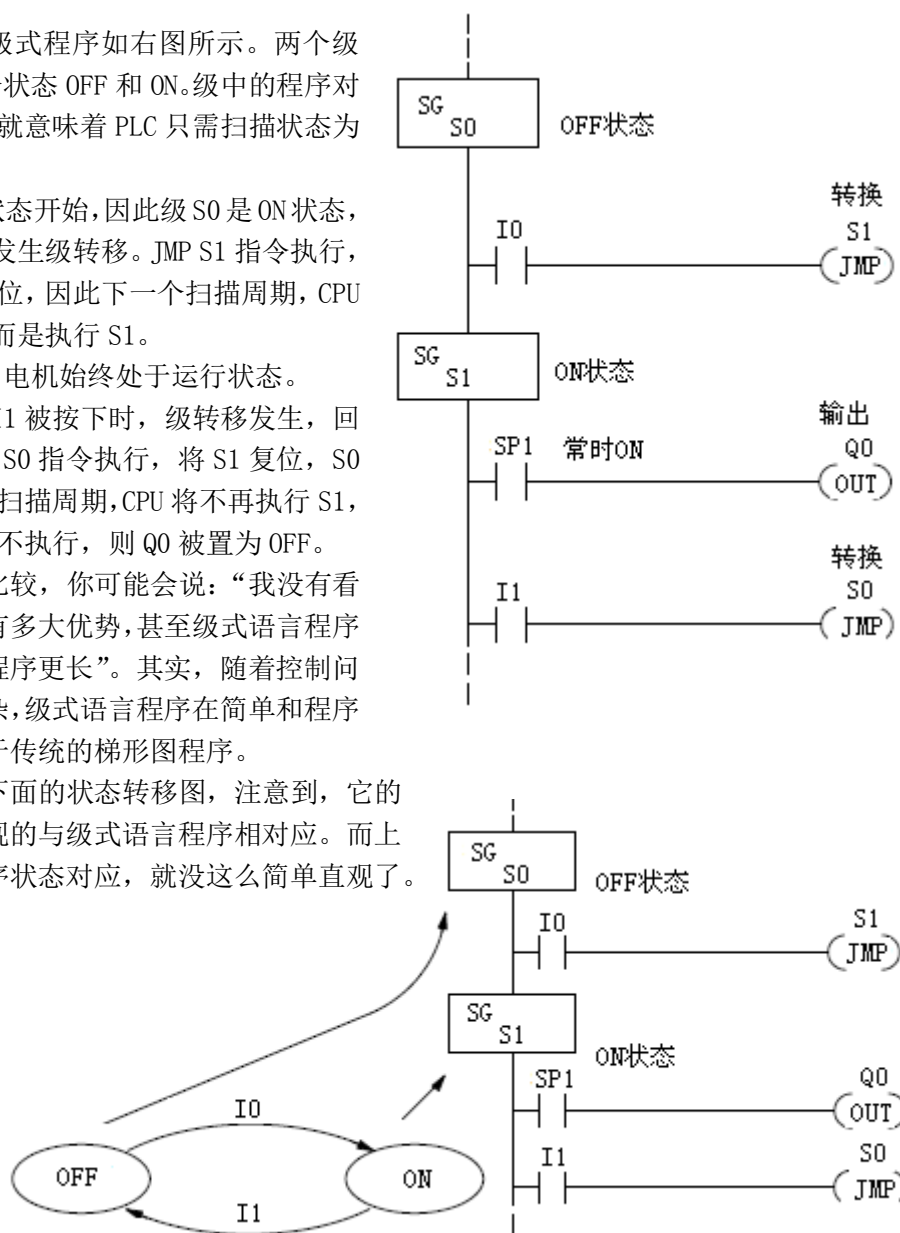
假设从 OFF 状态开始，因此级 S0 是 ON 状态，当 I0 被按下时，发生级转移。JMP S1 指令执行，将 S0 复位，S1 置位，因此下一个扫描周期，CPU 将不再执行 S0，而是执行 S1。

S1 为 ON 时，电机始终处于运行状态。

当关闭按钮 I1 被按下时，级转移发生，回到 OFF 状态。JMP S0 指令执行，将 S1 复位，S0 置位，因此下一个扫描周期，CPU 将不再执行 S1，而是执行 S0。S1 不执行，则 Q0 被置为 OFF。

两种语言做比较，你可能会说：“我没有看出级式语言编程有多大优势，甚至级式语言程序比普通的梯形图程序更长”。其实，随着控制问题变得越来越复杂，级式语言程序在简单和程序大小等方面都优于传统的梯形图程序。

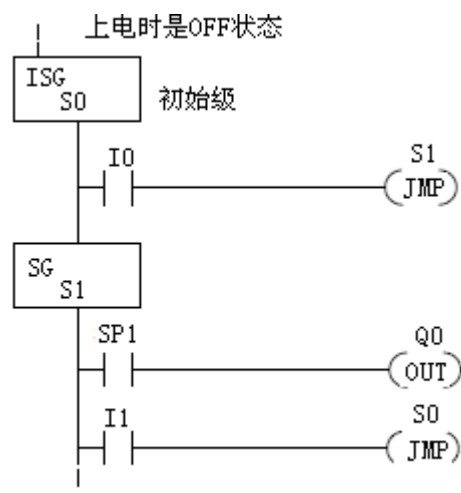
例如，对于下面的状态转移图，注意到，它的状态非常简单直观的与级式语言程序相对应。而上页中的梯形图程序状态对应，就没这么简单直观了。



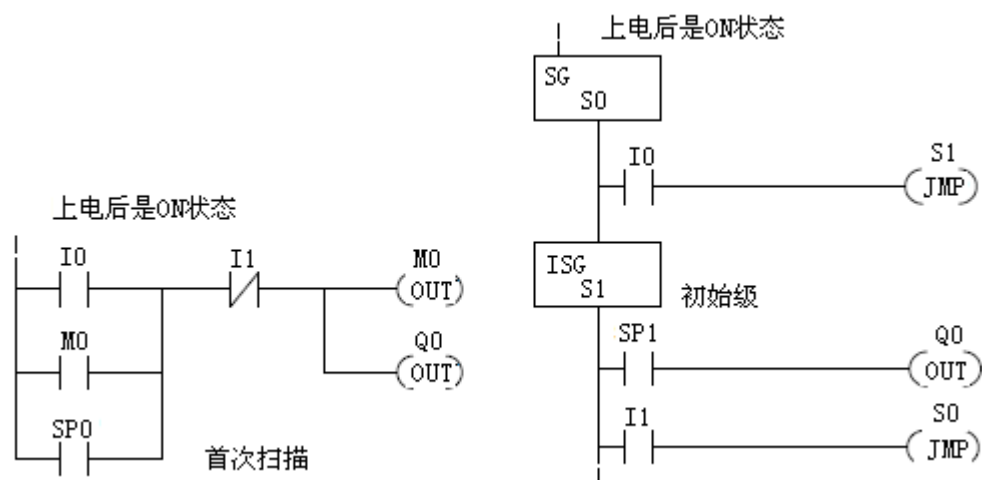
初始级

PLC 在上电以及编程→运行模式的转换后，程序中的所有级（SG）都是 OFF 状态。因此，如果没有初始级的话，级中的程序没办法扫描（因为 PLC 只扫描状态为 ON 的级中的程序）。

假设希望电机上电后的起始状态是 OFF。初始级是 PLC 上电后，定义状态为 ON 的级。右边是修改后的程序，将 SG S0 改为 ISG S0，这就确保了上电时 PLC 将扫描 I0，因为 S0 为 ON。上电以后，初始级就同其它级一样运行。

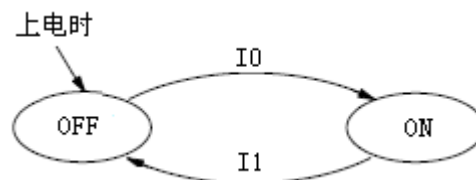


如果希望电机上电后的起始状态为 ON。下面的梯形图程序中，必须添加一个一次扫描线圈 SP0，首次扫描时将 M0 置为 ON。在下面的级式语言程序中，只需将初始级由 S0 改为 S1 即可。



注意：如果初始级被设置了停电保持，那么上电后，初始级在首次扫描时将保持其断电前的状态而不是被置为 ON。

可以在状态转移图上标记上电时的状态，见右图，这样有助于我们在编写级式语言程序时记得初始级是哪些级。程序中允许使用多个初始级。



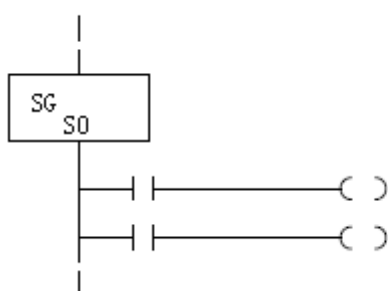
级状态位

一个级就是一个程序段，这个程序段在某个特定的时间级的状态是 ON 或是 OFF。PLC 存储器中的所有级状态位（S0-Sxxx）都是独立的状态位，每个级状态值都是开关量 0 或 1。

程序执行总是从上到下，从左到右。下图演示了级状态位的作用，级登记指令下方的程序持续运行直到到达下一个级登记指令或者直到级 S0 的程序结束。右边是等效功能图。当 S0 为 ON 时，级登记指令下面的所有 S0 中的程序执行。

- 当 S0=0 时，S0 的程序不扫描（执行）。
- 当 S0=1 时，S0 的程序扫描（执行）。

实际程序图



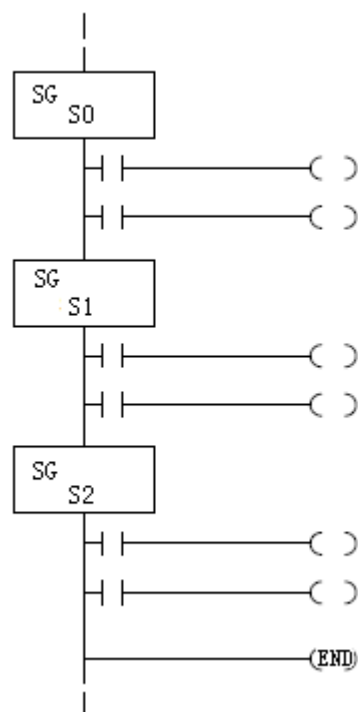
等效功能图



级式指令特性

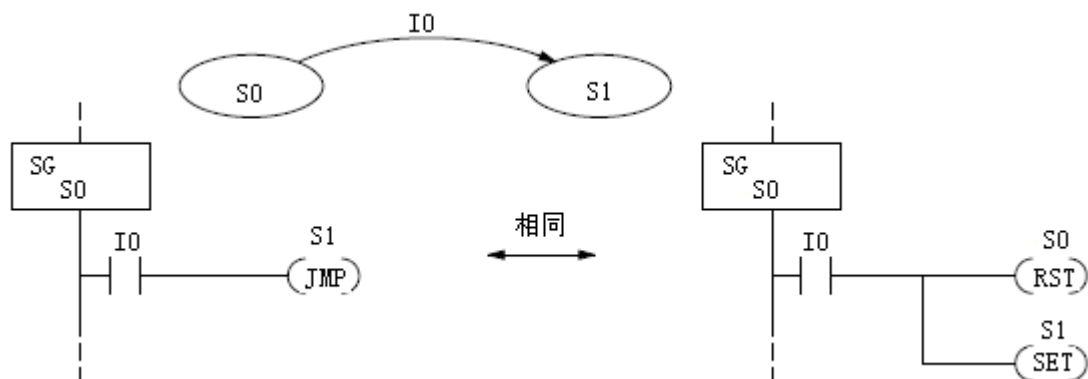
级式指令将程序划分为若干级，下面是一些级的规则：

- 执行：只有状态为 ON 的级中的程序每次扫描被执行。
- 级转移：再次扫描到转移到的级时，就执行这个级。
- 级编号：级号是八进制，和 I/O 一样，因此“S8”是无效级号。
- 级数：程序中最多可使用的级数根据 CPU 的不同而不同。
- 级号不能重复：级号是唯一的，不能重复使用。
- 级号顺序：级号可任意分配，可跳过任意数字，也可以以任意顺序排列级号。
- 最后一级：最后一个级包含的程序是级登记指令到 END 线圈之间所有的程序。



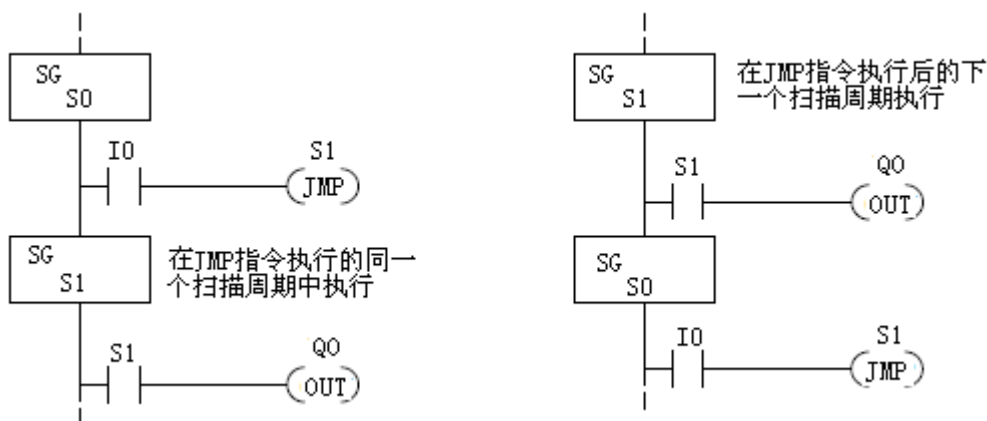
22.3 级转移指令的使用方法

级转移指令 JMP 将其所属级复位，将 JMP 指令中的指定的级置位。见下面的状态转移图。当 I0 为 ON 时，状态 S0→S1 转换发生。下面的两段程序功能是相同的。



JMP 指令很容易出现理解偏差，它不像 GOTO 和 CAL 指令那样当执行时立即跳转，它是这样工作的：

- JMP 指令执行时将其所属级复位。当前扫描周期中级中的程序都要被扫描完成，即使 JMP 指令后面的程序也一样。
- 复位作用在下一个扫描周期，先前执行 JMP 指令的级被复位，PLC 执行程序时将其跳过。
- JMP 指令执行时，JMP 指令指定的级将被立即置位，因此，当程序执行到这个指定的级时，这个级就被执行。下面左边的程序例子中，S0 中的 JMP 指令执行后，S1 中程序将在同一个扫描周期被执行。下面右边的程序例子中，S0 中的 JMP 指令执行后，S1 中的程序要到下一个扫描周期才被执行，因为级 S1 在 S0 的上方。

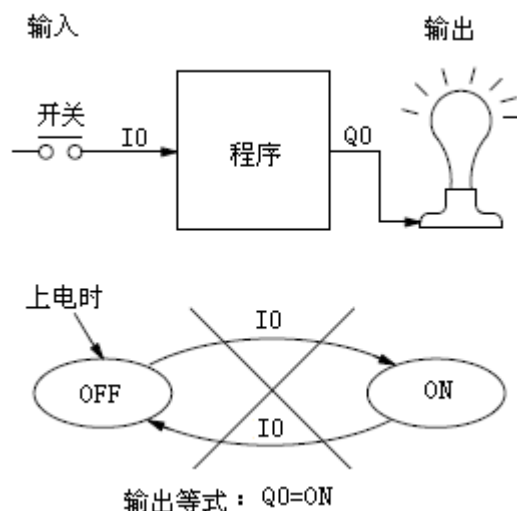


注意：上面两个例子中假定 S0 状态为 ON，S1 状态为 OFF。

22.4 级式语言编程举例：开/关灯控制

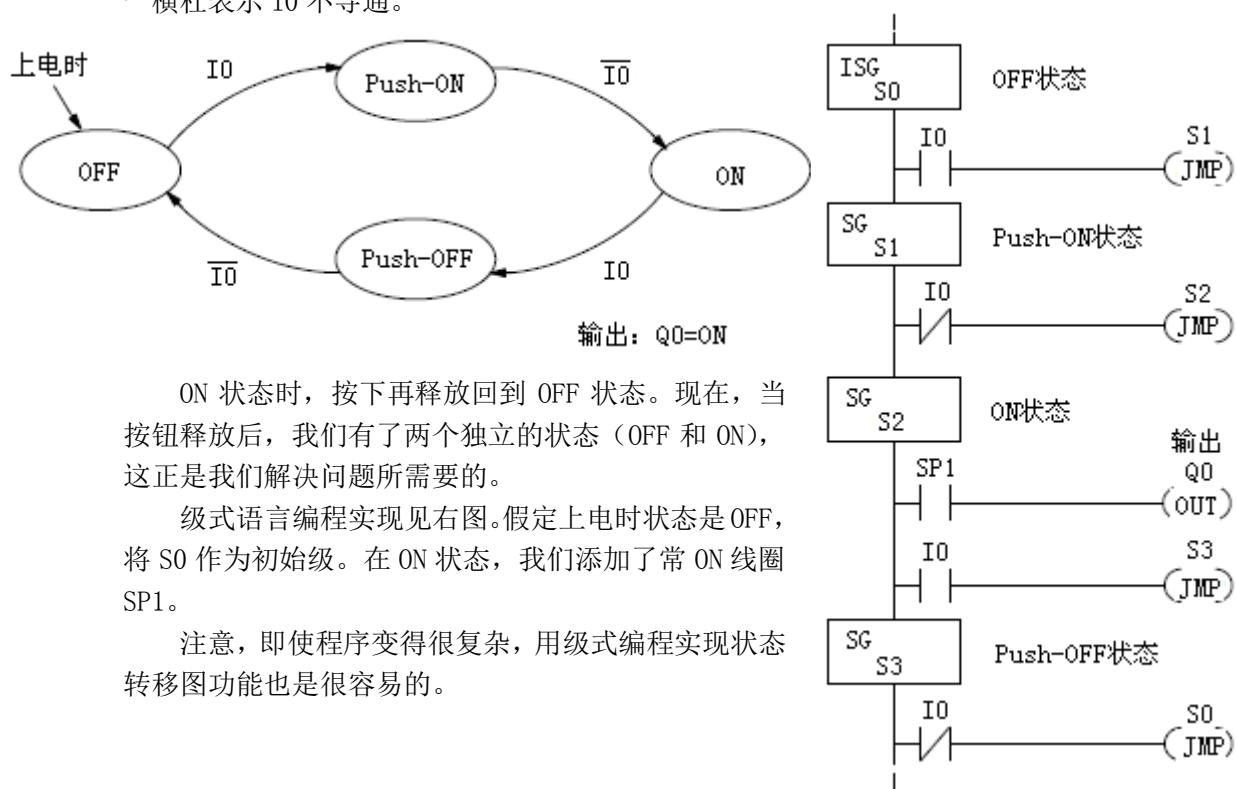
右图的例子中，使用一个普通的瞬时按钮来控制一个灯泡的开关。程序将开关输入闭锁，因此我们将开关按下然后释放可将灯打开，再次按下然后释放将灯关闭。

下面画出状态转移图。如右图所示，两个状态都使用 I0，但是，这样画是不正确的。



注意本例与上面电机例子不同，因为我们只有一个按钮。当按下这个按钮时，两个转换条件都得到满足。如果使用上面的状态转移图，用级式语言编程实现，结果是在每个扫描周期灯 ON/OFF 闪烁，这显然是不行的。

解决方法是将按下和释放按钮作为两个独立的事件。见下面新的状态转移图。上电时是 OFF 状态。当开关 I0 被按下，输入 Push-ON 状态。释放后，输入 ON 状态。注意，I0 上面有一横杠表示 I0 不导通。



ON 状态时，按下再释放回到 OFF 状态。现在，当按钮释放后，我们有了两个独立的状态（OFF 和 ON），这正是我们解决问题所需要的。

级式语言编程实现见右图。假定上电时状态是 OFF，将 S0 作为初始级。在 ON 状态，我们添加了常 ON 线圈 SP1。

注意，即使程序变得很复杂，用级式编程实现状态转移图功能也是很容易的。

22.5 级式语言四步编程

你可能已经注意到，用级式语言编程来解决控制问题的步骤都是一样的。如果你熟悉了本章的所有例子，等到实际应用时，自然而然会按照这些步骤来做。下面是这些步骤：

第 1 步：写一个书面进程描述

书面描述进程的所有功能。按顺序列出每个功能。如果进程比较复杂，可将其划分为多个进程。注意，可以将各个进程的之间的联系描述出来。

第 2 步：画出框图

输入是进程处理所需要的全部信息，输出连接到进程要控制设备。

- 列出进程所需的输入输出清单。
- 将 I/O 点分配给物理设备。

第 3 步：画出状态转移图

状态转移图描述了框图的核心功能，读取输入生成输出。

- 标示和命名进程中的状态。
- 标示状态之间转换所需的事件。
- 确保进程能够重新启动，或者说是循环的。
- 为进程选择上电时的状态。
- 写出输出等式。

第 4 步：用级式指令编程

用级式指令编程实现状态转移图的功能。

- 每个状态作为一个级。注意级编号是八进制的。NK1 CPU 最多能使用 1024 个级。
- 每个级中编写转换程序。
- 上电时需要为 ON 的级要使用初始级登记指令（ISG）。
- 将输出或动作放在相应的级中。

注意到，步骤 1-3 是步骤 4 的准备工作。下面是级式编程的完整的例子。

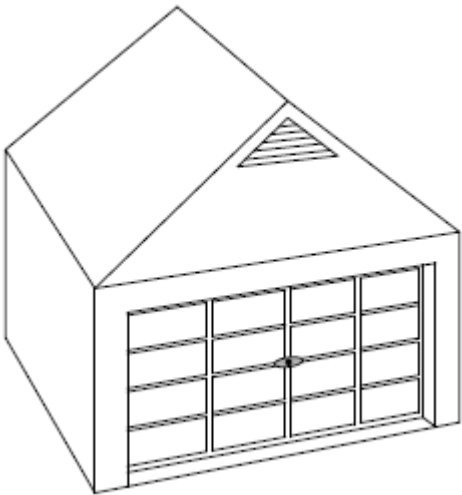
22.6 级式编程举例：车库门开关控制器

下面的例子中，我们创建一个车库门开启控制器。

功能描述

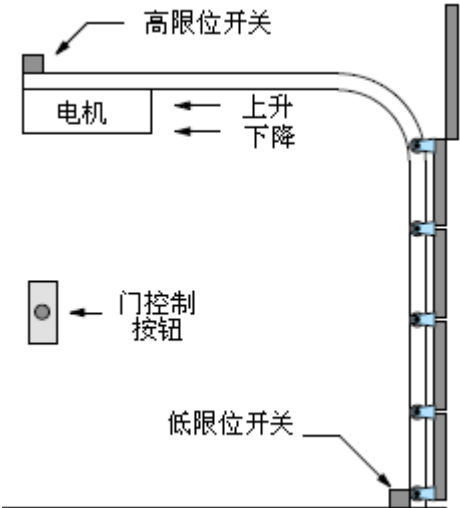
第一步必须描述门开启控制器是如何工作的。首先完成基本功能，然后在里面添加特别的功能（级式程序很容易修改）。

车库门开启控制器有一个电机，在得到命令时将门升起或下降。将一个瞬时按钮按下然后释放将门开启。门开启后，再将按钮按下然后释放可将门关闭。



为了在系统中标示输入和输出，有时需要画出系统的主要组成部分，见右图。门有一个高、低限位开关。当门开启或关闭到位后限位开关闭合。门在开启或关闭的过程中，限位开关都是断开的。

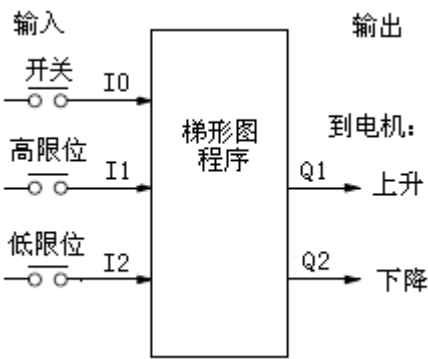
电机有两个命令输入：开启和关闭。没有命令时，电机是停止状态。门的命令是一个简单的按钮。无论是右图中的安装在墙壁上的按钮，还是遥控式的按钮，其内部都是一对或逻辑触点。



画出框图

控制器的框图见右图。输入 I0 来自门控制按钮，输入 I1 是高限位开关信号，输入 I2 是低限位开关信号。

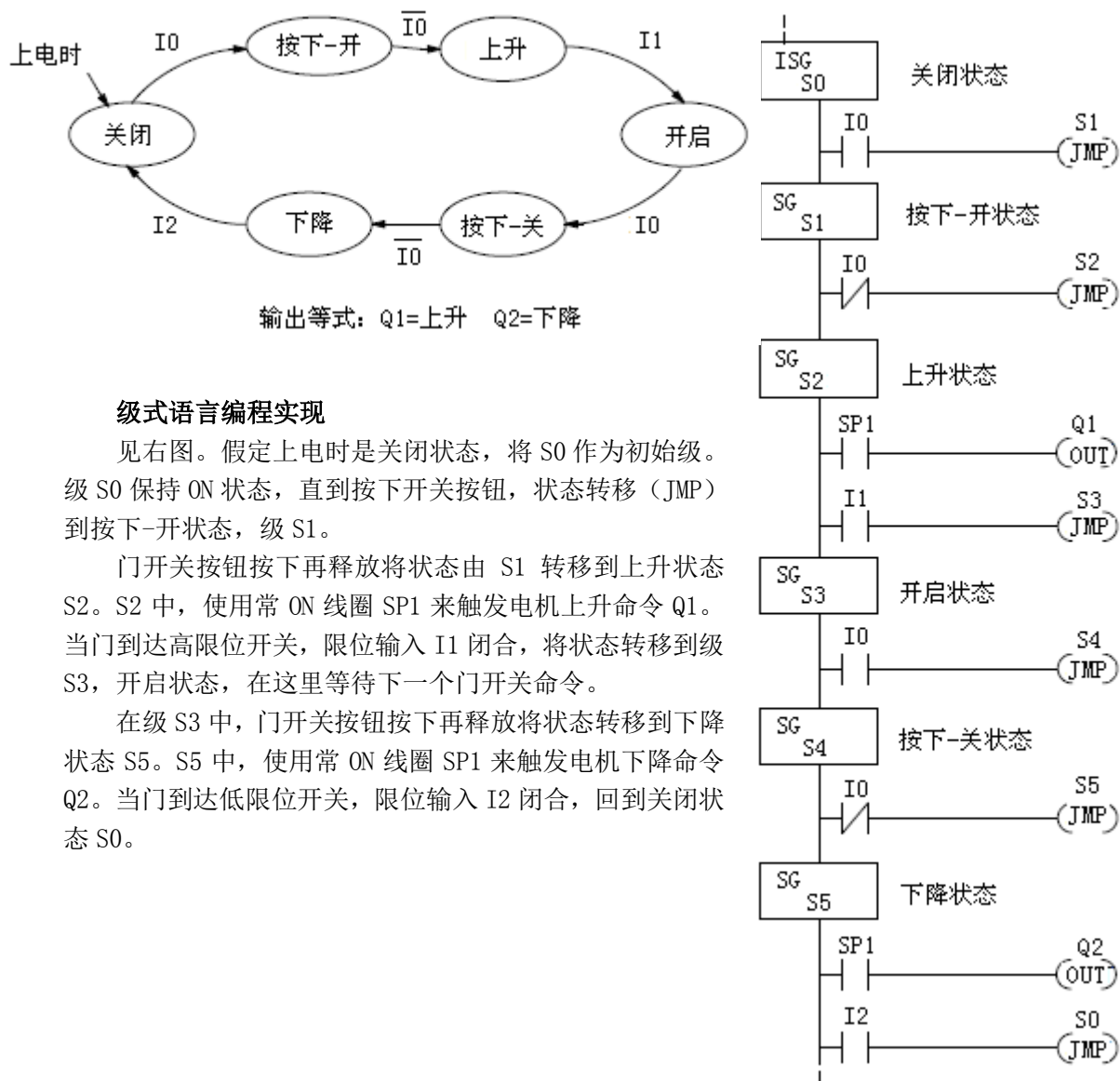
控制器有两个输出来驱动电机。Q1 是开（上升）命令，Q2 是关（下降）命令。



画出状态转移图

现在画出状态转移图，同上面的灯开关控制一样，本例也是只有一个开关。

- 当门是关闭状态时，按下然后释放按钮（I0）后，转换到上升状态，此时 Q1 被置为 ON，电机将门升起（开启）。
- 当限位开关 I1 闭合后，状态转移为开启状态，此时电机关闭。
- 再次按下并释放按钮（I0）后，转换到下降状态，此时 Q2 被置为 ON，电机将门下降（关闭）。当限位开关 I2 闭合后，状态又回到关闭状态。



级式语言编程实现

见右图。假定上电时是关闭状态，将 S0 作为初始级。级 S0 保持 ON 状态，直到按下开关按钮，状态转移（JMP）到按下-开状态，级 S1。

门开关按钮按下再释放将状态由 S1 转移到上升状态 S2。S2 中，使用常 ON 线圈 SP1 来触发电机上升命令 Q1。当门到达高限位开关，限位输入 I1 闭合，将状态转移到级 S3，开启状态，在这里等待下一个门开关命令。

在级 S3 中，门开关按钮按下再释放将状态转移到下降状态 S5。S5 中，使用常 ON 线圈 SP1 来触发电机下降命令 Q2。当门到达低限位开关，限位输入 I2 闭合，回到关闭状态 S0。



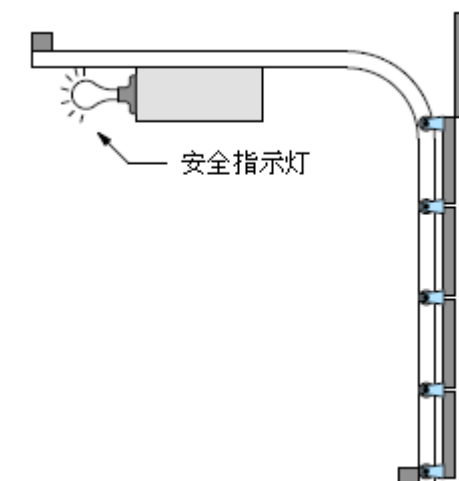
注意：上电时初始级（ISG）自动为 ON，其后，和其它级一样运行。

加入安全指示灯功能

下面给系统添加安全指示灯功能。最好的做法是首先完成主要功能的编程，然后再添加次要功能，本例中我们就是这种做法。

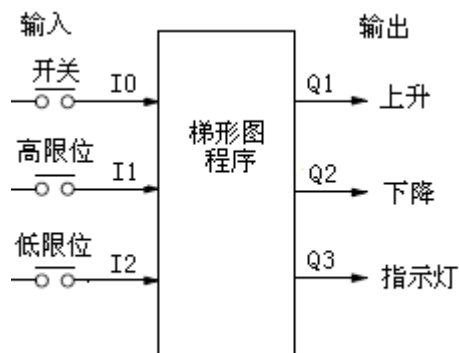
许多市售的车库门都安装有安全指示灯，见右图，安装在电机外壳上，当门有动作时，安全指示灯被点亮，然后持续亮约 3 分钟。

这部分牵涉到并行进程的用法，不使用 JMP 指令，使用 SET 和 RST 指令将其代替。

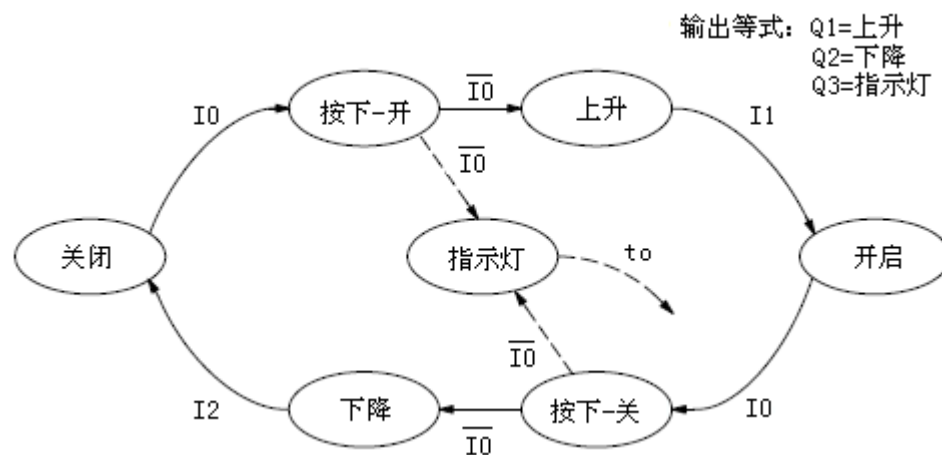


修改框图和状态转移图

为了控制安全指示灯，我们在框图中添加了一个输出 Q3，见右图。下面的状态转移图中，我们添加了辅助状态“指示灯”，只要按下再释放门开关按钮，上升或下降状态被激活的同时，指示灯状态也被激活。画到状态“指示灯”的线是虚线，因为这不是一个主要路径。



可认为指示灯状态是一个同上升和下降状态并行的进程。到指示灯的路径不是一个状态转移 (JMP)，而是一个状态置位 (SET)。在指示灯的级中，我们将使用一个 3 分钟定时器，定时时间到了以后，定时器位 T0 被置为 ON，指示灯的级被复位。指示灯状态不转移到任何地方，只是指示灯级被置为 OFF，指示灯熄灭。



在级中使用定时器

修改后的程序见右边，阴影部分是新添加的程序。

在级 S1 中，添加 SET 指令将级 S6 置位。当触点 I0 断开，状态由 S1 转移到状态 S2 和 S6。在级 S4 中，我们做了同样的改动。因此，任何时候按下门开关按钮，指示灯都会亮。

你可能会关心在哪里添加指示灯级，并且如何给其编号，其实这些都不重要。

- 只需选择一个不使用的级号用于新添加的级即可。
- 放在程序中的位置也不是关键，本例中将其放在最后。

你可能会认为，级应该按转移顺序排列，这是很好的做法，但没必要一定这样做，有时也做不到，例如，本例中，有两处转换到级 S6 的指令。

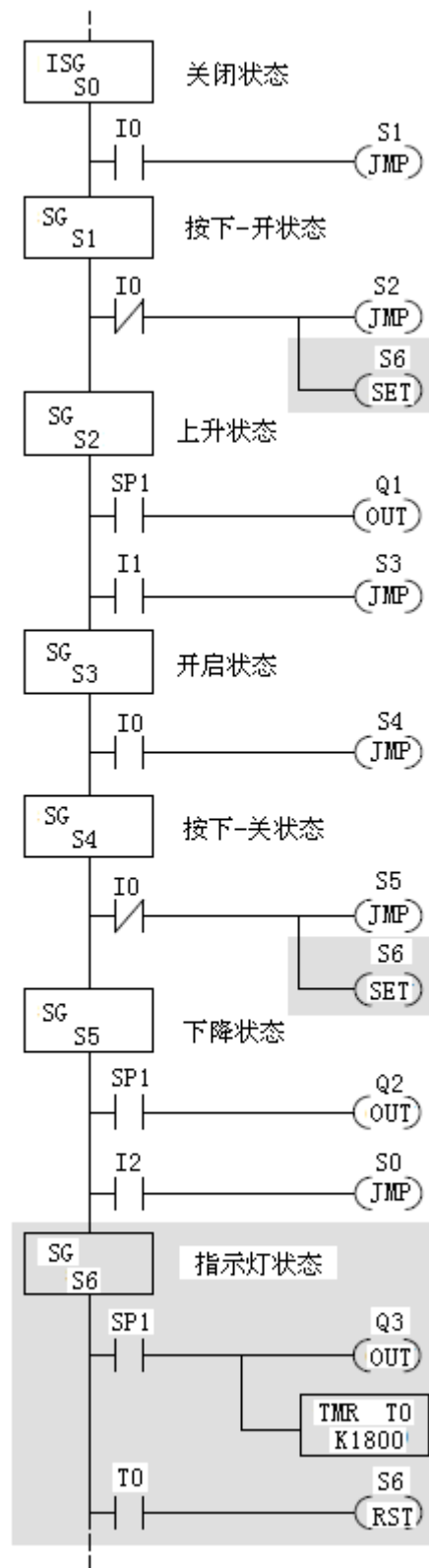
级 S6 中，通过激活 Q3 将安全指示灯点亮。SP1 是常 ON 线圈。定时器 T0 以 0.1 秒增计时，要完成 3 分钟计时，K 的计算算式为：

$$K = \frac{3\text{分} * 60\text{秒/分}}{0.1\text{秒}}$$

$$K=1800$$

当 S6 为 ON 时，定时器就增计时。当定时时间达到设定值，相应的定时器状态位被置为 ON。因此，3 分钟后，T0=1，指令 RST 将 S6 复位。

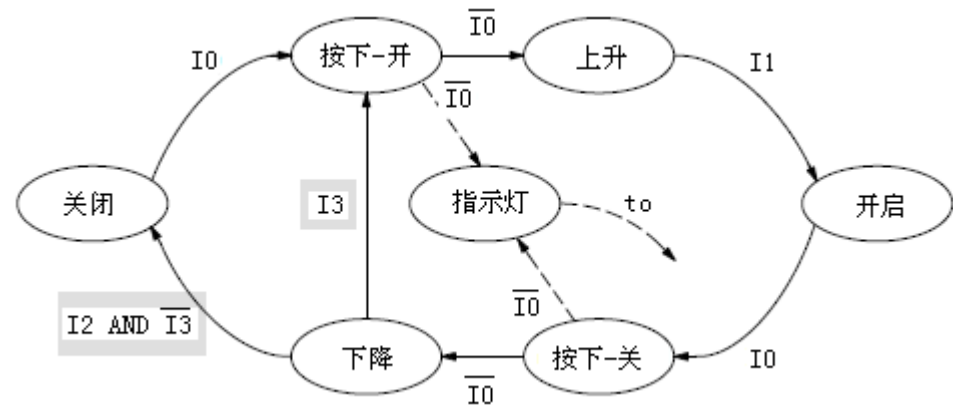
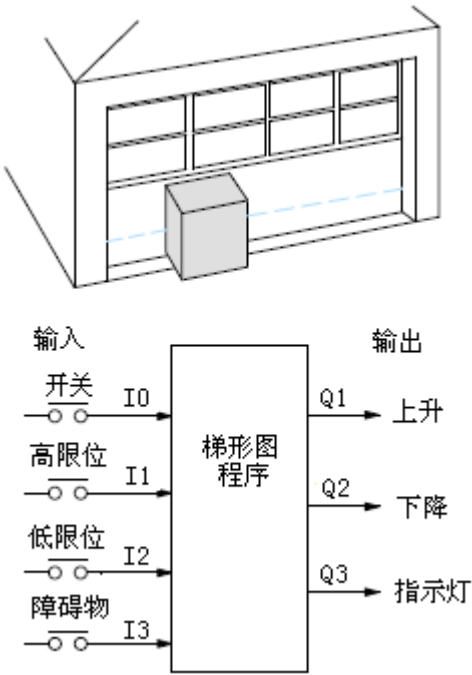
当 S6 为 ON 时，指示灯亮。主路径中的状态转移照常进行，不受 S6 的影响。因此，门可上升、下降，期间指示灯要持续点亮 3 分钟。



添加紧急停止功能

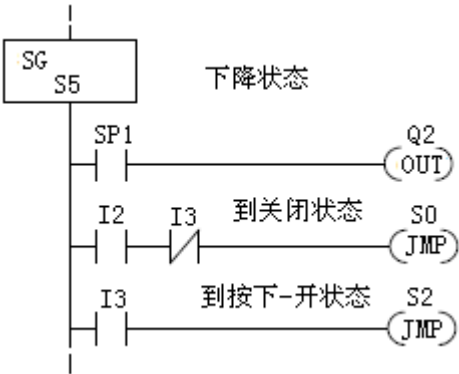
现在有的车库门控制器有紧急停止功能，当检测到门下有物体时，就暂停门的下降动作。一般情况下用光电管（“电子眼”）实现，下降中的门将停止下降并上升。本例中，我们将添加这样的功能。在框图中添加一个光电管的输入 I3，见右图。如果检测到物体在门的运动路径中，I3 将闭合。

下面，在状态转移图中做一个简单的添加，下面的状态转移图中，阴影部分就是添加的部分。注意下降状态，门正在下降时，当检测到有障碍物时（I3 闭合），下降状态将转移到按下-开状态。我们没有直接将状态转移到上升状态，这样在上升输出 Q1 闭合前，给了下降输出 Q2 一个扫描周期的时间来断开。



低限位开关输入（I2）和障碍物输入（I3）同时接通理论上是有可能的。这种情况下，状态将同时转换到按下-开状态和关闭状态，这样做是没什么意义的。

通过将到关闭状态的转移条件改为 I2 AND I3，给障碍物事件较高的优先级，这样就保证了障碍物事件有较高的优先级。级 S5 中的程序修改见右图，第一行的程序不变，第二、第三行的程序按照需要进行修改。注意，程序中使用了 I3 的常开和常闭触点，这就保证了同一时间只能执行一个 JMP 指令。

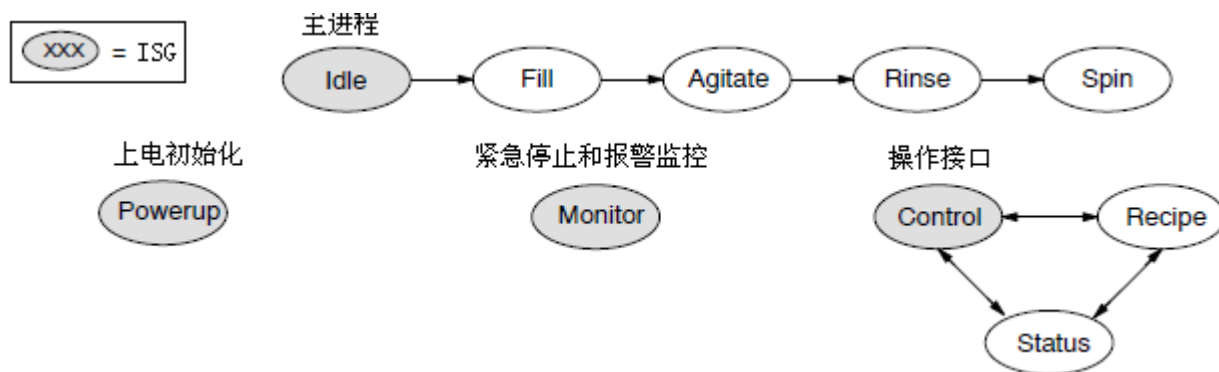


22.7 级式程序设计注意事项

级式程序组织

到目前为止，本章的例子中主进程都是使用一个独立的状态转移图。但是，一个程序中可以有多个进程。初学者可能会错误的认为每次只能一个级为 ON。对于需要每次扫描都执行的程序段，只需将它们放在有一个常 ON 的级中即可。

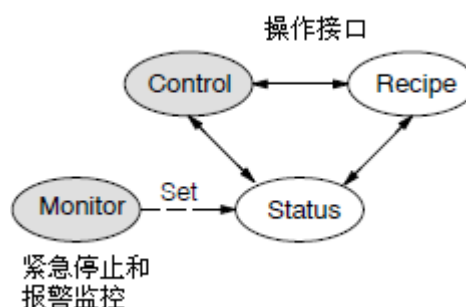
下面的图示是一个典型的应用。操作期间，主进程、上电初始化、紧急停止和报警监控、操作界面都要运行。上电时，下面的四个初始级开始运行。



上面的典型应用中，各独立级的操作顺序如下：

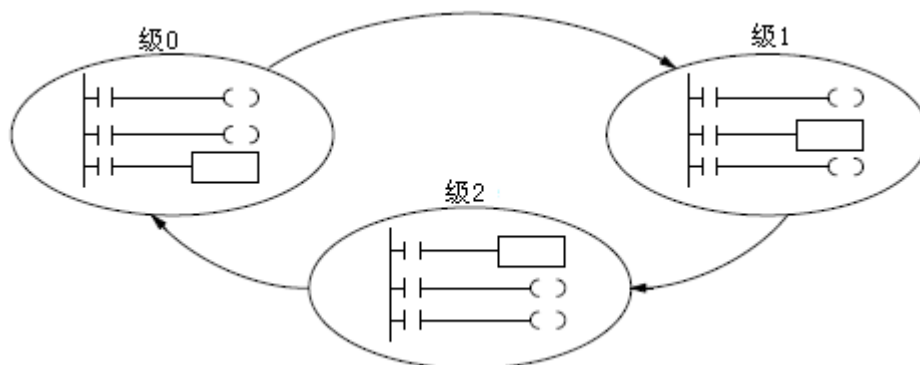
- **上电初始化：**级中的程序仅上电时扫描一次。级中的最后一行程序将级复位，因此级仅激活一个扫描周期。
- **主进程：**级中的程序控制进程或机器的核心部分。主进程的程序全部运行一次代表机器的一次循环，或进程的一次处理。
- **紧急停止和报警监控：**这个级常 ON，因为它要检测需要报警或紧急停车的错误。此级经常将主进程或其它地方的级复位，是为了在出错之后将它们初始化。
- **操作接口：**这是另一个要始终为 ON 的级，因为它要随时准备应对一个操作。它允许操作员界面来改变模式等，独立于当前主进程。

虽然进程是独立的，但是它们之间也有联系。例如，在一个错误的条件下，状态级希望操作接口级自动转换为状态模式，来显示错误信息，见右图。监控级（Monitor）可将状态级（Status）置位，将控制级（Control）和配方级（Recipe）复位。



级中指令的执行

可认为状态或者级将梯形图程序简单的划分成下图中的样子。每个级中包含相应的状态所需要的程序，将级转移的逻辑也包含在这个级中。使用初始级（ISG）可以简单的将级设置为上电时为 ON。



几乎所有的指令在级中和在标准的梯形图中的运行是一样的，可以将级看做状态为 ON 或 OFF 的小型梯形图程序。

输出线圈：在状态为 ON 的级中，输出线圈将相应的输出置为 ON 或 OFF。但是，请注意以下几点：

- 输出正常运行，每个输出（比如“Q3”）仅在一个级中使用。
- 当级状态为 OFF 时，输出线圈自动转换为 OFF，但是 SET 和 RST 指令后的输出不会改变状态。
- 一个输出可用于多个级，但是要求同一时间只能一个级 ON。
- 如果多个级同时控制同一个输出，那么每次扫描中，程序最下方的一个状态为 ON 的级决定了输出的状态。因此，如果想使用多个级对同一个输出进行逻辑或控制，可使用 OUT 指令替代 ZOUT 指令。

一次输出线圈（PD）：如果要在一个级中使用一个正差动线圈，要注意，这个线圈的输入必须经历一个 0→1 的转换。如果级变为 ON 时的第一个扫描周期中线圈已经接通，那么 PD 线圈将不工作，这是因为 0→1 的转换没有发生。

PD 线圈替代：如果想让一个任务只执行一次（在一次扫描周期中执行），可将这个任务放在一个级中，这个级在同一个扫描周期中要转移到下一级。

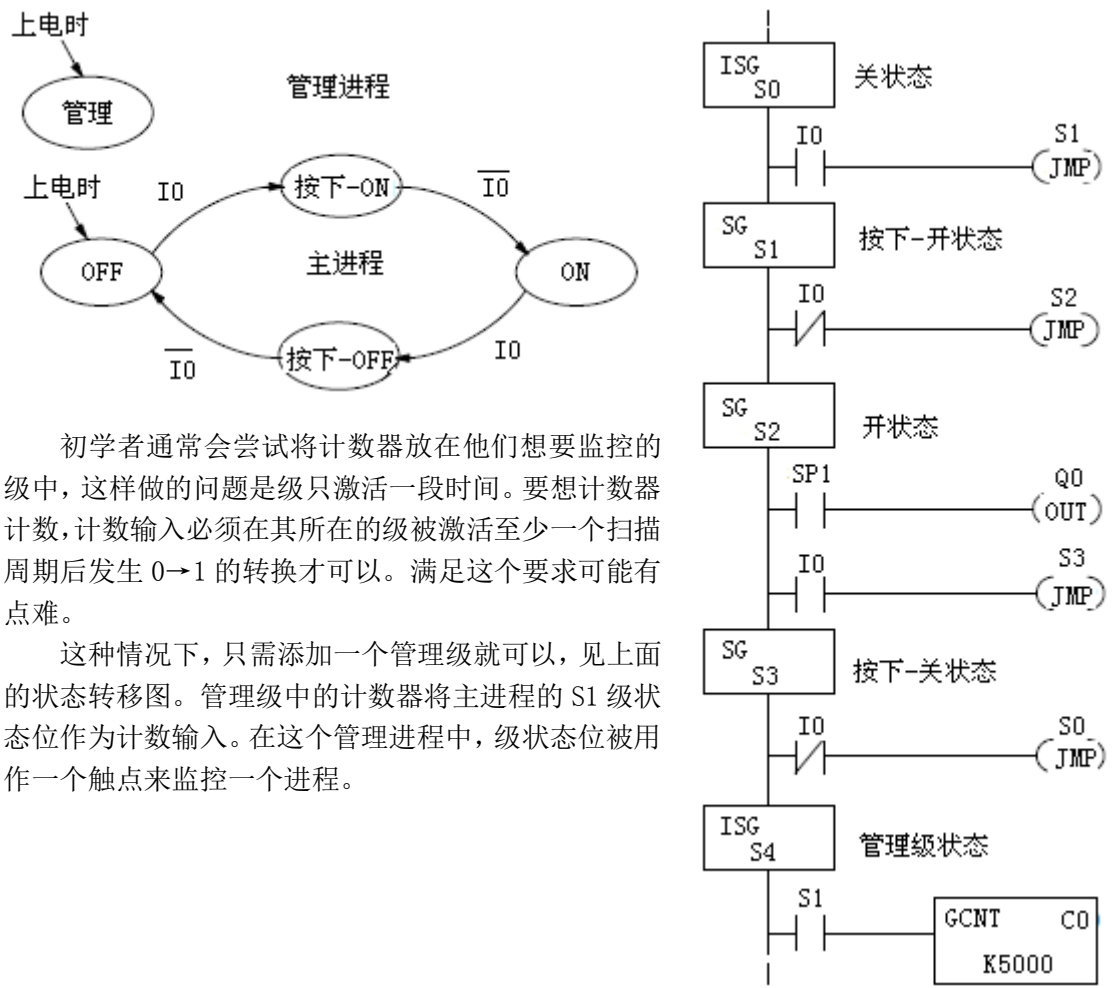
计数器：如果在级中使用了计数器，那么在计数器输入发生一个 0→1 的转换前，级必须被激活了至少一个扫描周期。否则，计数器输入没有转换，计数器将不计数。普通计数器指令在级中使用时有个限制：它的计数值不能被其它级中的 RST 指令复位。但是级计数器指令解决了这个问题。

级计数器：级计数器有个优点，那就是它的计数值可被其它级中的 RSTTC 指位复位。它有一个计数输入，没有复位输入，这是它同普通计数器的唯一区别。

凸轮控制：凸轮控制有它自己的进程，同级式编程的进程不一样。如果需要在级中使用凸轮指令，要确保将其放在初始级中。

将级用作管理级

上面的灯开关控制例子中，假设我们要通过灯 ON-OFF 次数计数来监控灯进程的“生产力”。这个应用需要添加一个简单的计数器，但是问题的关键是在哪里添加计数器。



初学者通常会尝试将计数器放在他们想要监控的级中，这样做的问题是级只激活一段时间。要想计数器计数，计数输入必须在其所在的级被激活至少一个扫描周期后发生 0→1 的转换才可以。满足这个要求可能有点难。

这种情况下，只需添加一个管理级就可以，见上面的状态转移图。管理级中的计数器将主进程的 S1 级状态位作为计数输入。在这个管理进程中，级状态位被用作一个触点来监控一个进程。



注意：管理级和关状态级都是初始级，管理级始终是激活状态。

级计数器

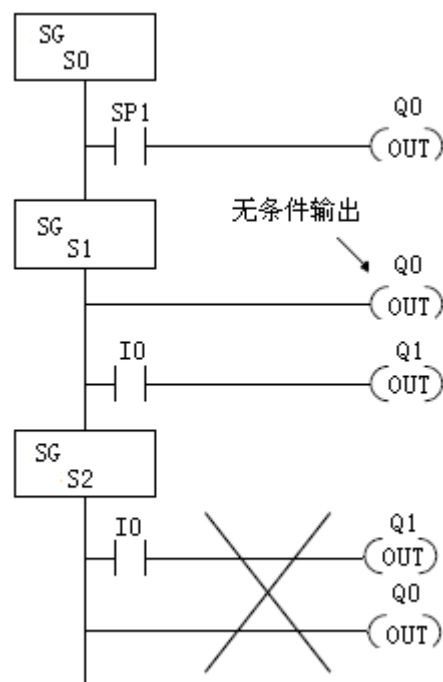
上例中的计数器是一个级计数器。注意到它没有复位输入（不带复位端）。当执行一个复位指令 RSTTC 时，本例中 RSTTC 指令的操作数是 C0，计数复位。级计数器的优点是它的计数值可被其它级中的 RSTTC 指令复位。普通计数器没有这个功能。

无条件的输出

到目前为止本章的大部分例子都如右图的 S0 一样，在输出前添加一个常 ON 触点。这个触点可被省略掉，只需将这个无条件输出放在级的第一行程序中即可，见级 S1。

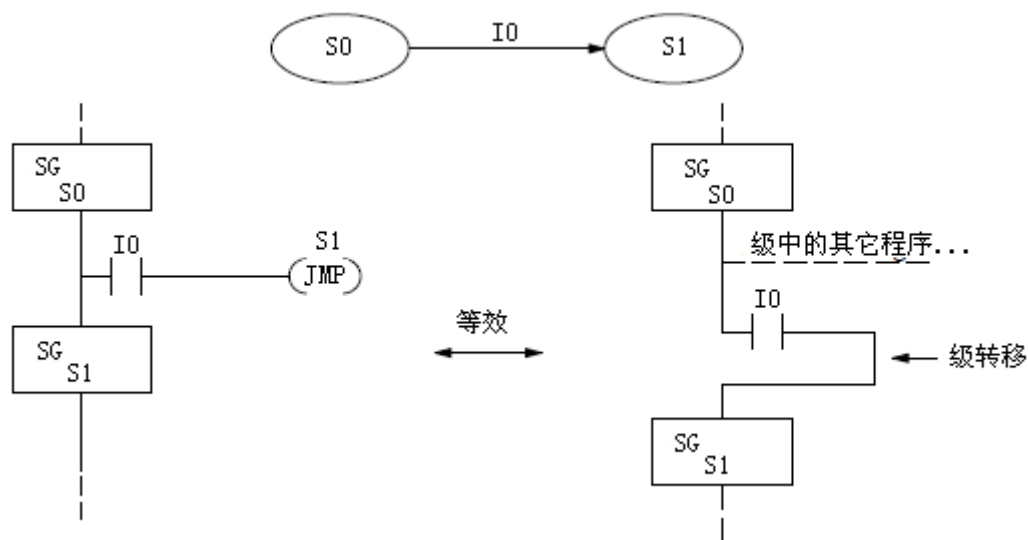


警告：无条件输出如果放在级的其它地方，那么级被激活时，输出不一定为 ON。在右图的级 S2 中，Q0 是一个无条件输出，由于它接通与否取决于上面一行程序，因此，Q0 状态同 Q1 状态一致（这样做是不正确的）。



级转移技术

在“级转移指令的使用方法”一节中已经介绍了级的转移方法。在 KPPsoft 中，可以使用下面的方法来替代级转移方法。主要的要求是当前级要放在下一级（要转到的级）的上方。见下图：



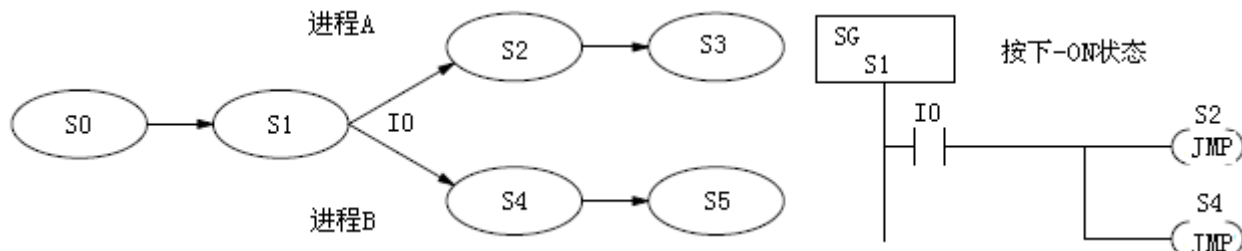
上面讲过，JMP 指令可在当前级的任何地方发生，结果都是一样的。但是，上图中的级转移只能发生在级的最后一行程序中，级中的其它程序都要在它之前。

级转移方法用起来没有 JMP 指令简单，因此，还是建议使用 JMP 指令。

22.8 并行进程的概念

进程并行

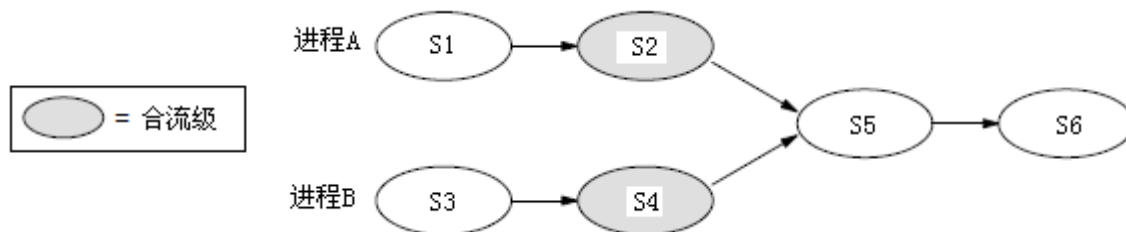
前面介绍的都是一个状态转移到另一个状态，但是有些情况下，需要同时转移到两个或两个以上的进程中，见下图。可以全部使用 JMP 指令实现转移，见下图。也可以使用一个 JMP 指令和一个 SET 指令，但是至少要使用一个 JMP 指令，因为这样可以离开其所属级。注意，级中的所有指令都会执行，甚至当发生级转移后也会执行。



注意，如果想让 S2 和 S4 在同一个扫描周期被激活，这两个级必须同时放在 S1 的上面或下面。总的来说，并行进程处理很简单。

进程合流

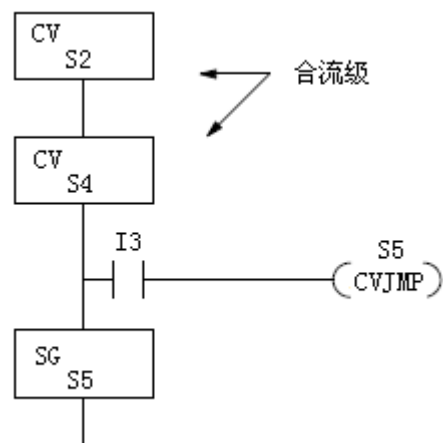
现在考虑相反的情况，就是进程合流。这其实就是停下多个进程，同时开始另一个进程。见下图，当 S2 和 S4 在某个时间点同时转移到 S5 中时，进程 A 和进程 B 合流。因此，S2 和 S4 是合流级。



合流级登记指令 (CV)

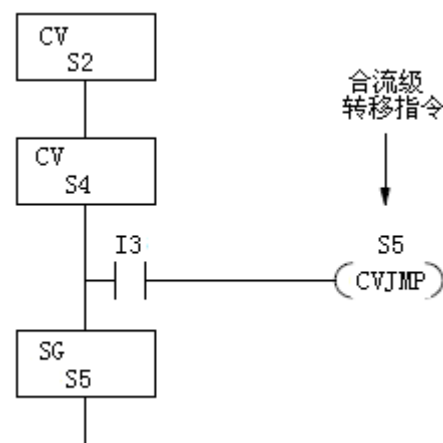
虽然合流原理很简单，但是它也带来新的问题：多个并行进程基本上不会同时完成，我们怎么会知道哪个并行的级（本例中是 S2 和 S4）是最后一个完成的呢。这一点很重要，因为我们要确定怎么转移到级 S5 中。

解决的方法是将合流级作成合流移行条件，我们使用级合流指令 (CV) 来实现。见右边的例子，合流级 S2 和 S4 被放在一起组成一个合流级群，两个合流级之间不能添加任何指令。转换条件（本例中为 I3）必须放在最后一个合流级下方。当合流级群中的所有级被激活后（为 ON），转换条件才能接通。



合流级转移指令（CVJMP）

当级群中所有的级被激活后，转换条件才能接通。合流级转移指令仅同合流级登记指令配合使用。右图中，当转换条件 I3 闭合后，将转移到级 S5，而且 CVJMP 指令将级组中所有级自动复位。因此，CVJMP 指令是一个强大的指令。



下面是级合流指令使用的一些要求和技巧：

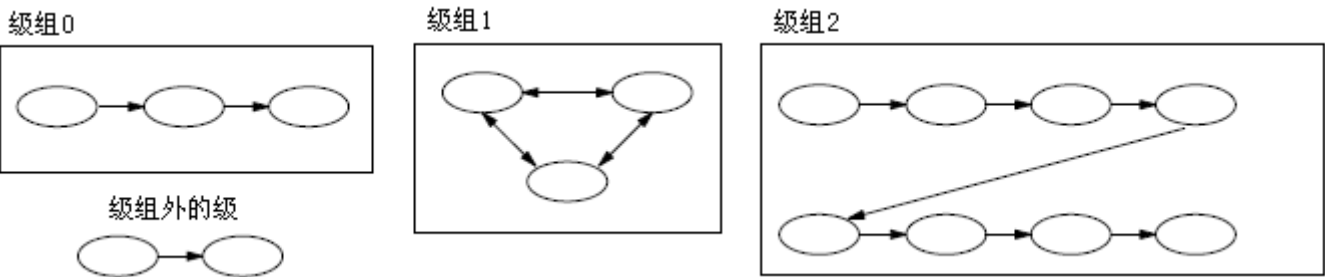
- 合流级用作进程的最后一级，这个进程同其它一个或多个进程并行。合流级转移意味着一个特定进程的结束，也代表一个等待点等待其它所有平行进程全部结束。
- 合流级群中最多可有 17 个级，也就是说，最多可有 17 个级合流到一个级。
- 合流级群中的级在程序中必须放在一起，合流级之间不能添加任何指令。
- 在一个合流级群中，级的激活可以是任何顺序，不用管哪个级是最后激活，因为级群中的所有级都要被激活之后，最后一个合流级才能接通。
- 级群中的最后一个级中可以有逻辑指令，但是，只有级群中所有的级被激活后，这个逻辑才能执行。
- 合流级转移指令（CVJMP）同合流级登记指令配合使用，CVJMP 指令将级群中的所有级复位，将指定的级激活。
- CVJMP 指令只能同合流级登记指令配合使用，同普通级和初始级式指令使用时是无效的。
- CV 和 CVJMP 指令不能用于子程序和中断程序中。

22.9 大型程序的管理

一个级中可以有很多行程序，也可以只有一两行程序。良好的程序设计应确保每个级的平均程序行数比较少。但是，大型的应用程序还是会创建大量的级。这里介绍级组的概念，将相关的级组成一个组，叫级组。

级组指令（BSTART、BEND）

级组是包含级的程序的一部分。下图中，每个级组有一个编号。同级一样，级组状态可为 ON 或 OFF。级组中级如何从一个级转移到另一个级是没有限制的。注意，使用级组不需要程序中的每个级都在级组中，见下图“级组外的级”。

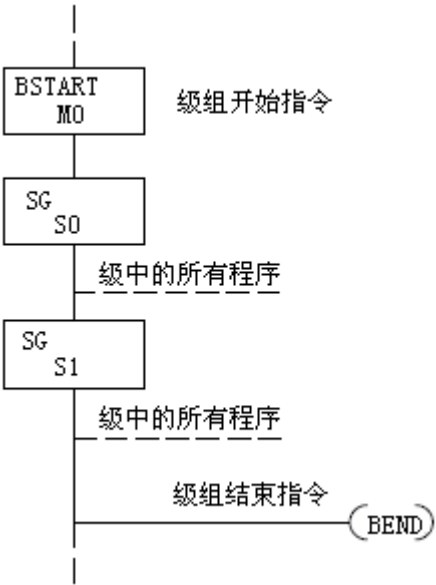


一般认为一个有 20 个级以上的程序已经足够大，可以使用级组了，但是级组的使用不是强制的。使用级组时，级组的数量应为两个以上，因为使用一个级组的优点基本上可以忽略不计。

使用特殊的开始和结束指令将级组与其它程序分隔开。右图中，BSTART 指令标志着级组的开始，BEND 指令标志着级组的结束。级组中的级（右图中是 S0 和 S1）及级中的程序组成了级组。

注意，级组指令的操作数（本例中是 M0），级组指令使用一个内部线圈作为操作数，以使程序的其它部分可以控制一个级组。级组指令使用的任何一个内部线圈在程序中都不可再用作别的输出。

注意，级组中的级必须是普通的级（SG）或合流级（CV），不可使用初始级（ISG）。级组中的级编号可以是任何顺序，且完全独立于级组的编号。



级组请求指令（BREQ）

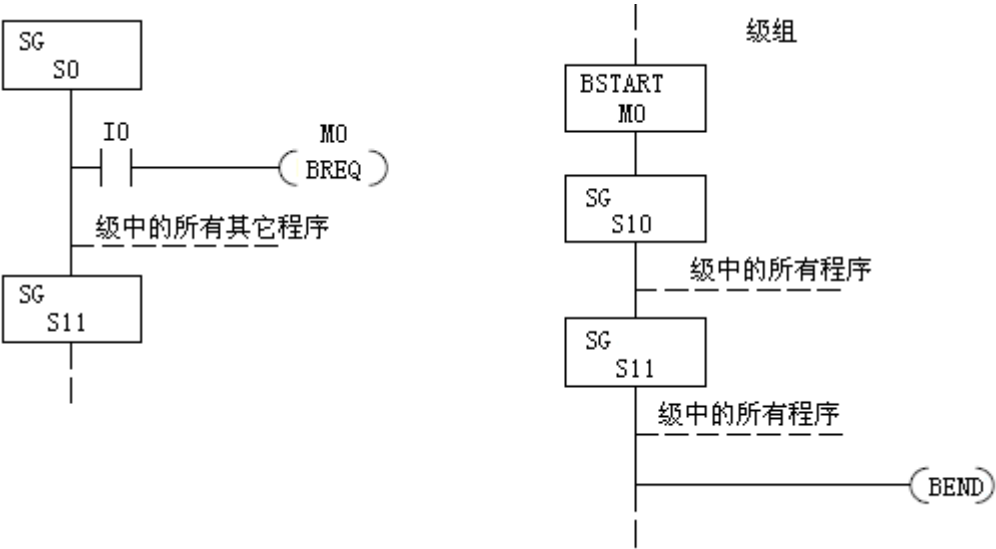
BREQ 指令的作用是激活一个级组。在上电时或是在编程→运行模式转换时，所有的级组及其里面的级都没有激活。BREQ 指令是一种输出线圈，见下图。当 IO 闭合时，BREQ 指令将操作数为 M0 的级组激活。当 BREQ 指令变为 OFF 时，相应的级组及其里面的级均变为 OFF。

不要将级组请求指令的操作同子程序调用指令的操作混淆。BREQ 线圈执行后，程序继续执行下一行程序。只要程序执行到 BREQ 指令指定的级组，级组中的程序就会执行，因为级组是 ON 状态。同样的，不要将 BREQ 指令同级转移指令（JMP）混淆。



当级组状态变为 ON 后，级组中的第一个级的状态在同一个扫描周期中自动变为 ON。级组的第一个级是程序中 BSTART 指令下方紧邻 BSTART 指令的一个级，因此这个级同初始级有类似作用。

BREQ 指令可在程序中多次使用。很明显，BREQ 指令的第一次执行必须在一个级组之外，因为级组的初始状态是 OFF。然后，BREQ 指令可在一行普通的程序中执行，或者它可以在一个激活的级中执行，见下图。注意，不论是将 BREQ 指令状态转换为 OFF 或是将 BREQ 指令所在的级状态转换为 OFF，都会将相应的级组状态转换为 OFF。也可以控制一个 BREQ 指令在另一级组中的级组。



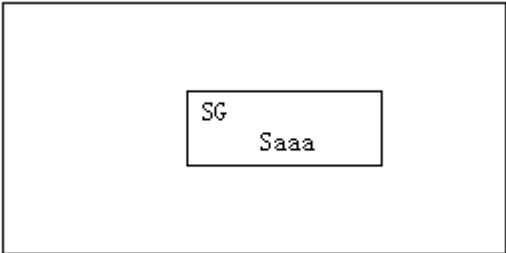
注意：级组可放在 BREQ 指令的之前或之后。

BREQ 指令有多种用法，因此很难说哪种用法最好。只需记住级组的目的是将相关联的级做成一个级组，以帮助用户组织应用程序。注意，初始级必须放在级组的外面。

第 23 章 级式指令

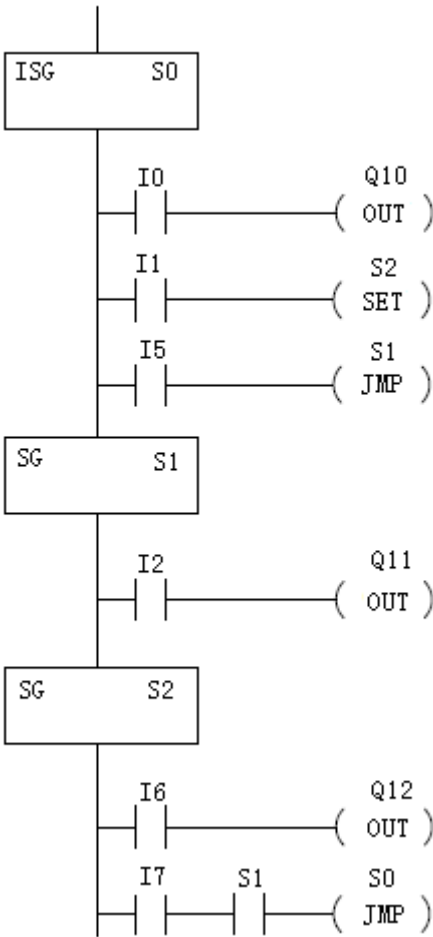
23.1 级登记指令（SG）

输入 SG 指令用于创建级式程序。级是可以
通过转移逻辑（JMP 或 SET）激活的程序段。转
移逻辑（JMP 或 RST）执行后的下一个扫描周期，
级状态变为 OFF。



操作数类型	NK1 范围
	aaa
级 S	0-1777

下面的例子是一个简单的级式程序例子。例中使用了初始级、级和转移指令。

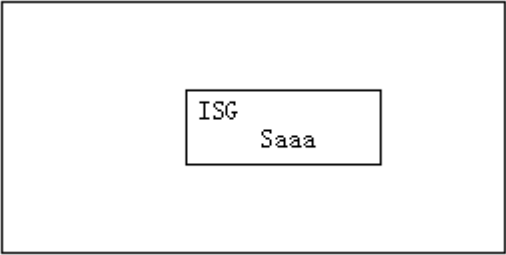


语句视图

```
ISG S0
LD I0
OUT Q10
LD I1
SET S2
JMP S1
SG S1
LD I2
OUT Q11
SG S2
LD I6
OUT Q12
LD I7
AND S1
JMP S0
```

23.2 初始级登记指令（ISG）

输入 初始级登记指令通常用于级式程序的开头。在 CPU 进入运行模式后，初始级就被激活。初始级也可被状态为 ON 的级中的转移逻辑（JMP 和 SET）激活。转移逻辑（JMP 或 RST）执行后的下一个扫描周期，初始级状态变为 OFF。只要级号不重复，ISG 指令可多次使用。



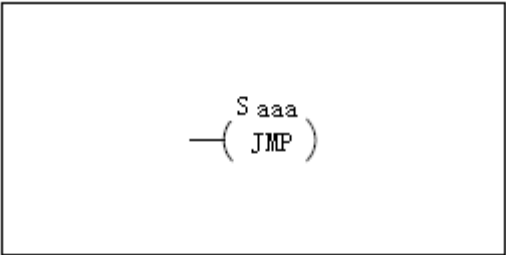
操作数类型		NK1 范围
		aaa
级	S	0-1777



注意：如果初始级被设置了停电保持，那么上电后，初始级在首次扫描时将保持其断电前的状态而不是被置为 ON。

23.3 级转移指令（JMP）

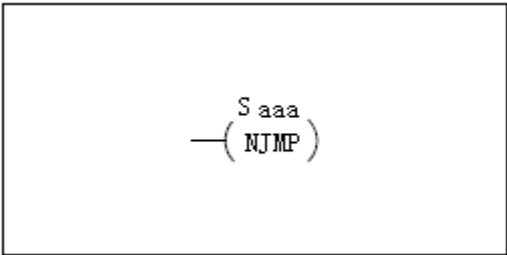
输入 级转移指令是允许程序将所属级的 ON 状态在转移条件成立时向指定的级转移的指令。当级转移的条件成立时，级转移发生。JMP 指令执行后的下一个扫描周期，其所属级状态被置为 OFF。



操作数类型		NK1 范围
		aaa
级	S	0-1777

23.4 条件不成立时级转移指令（NJMP）

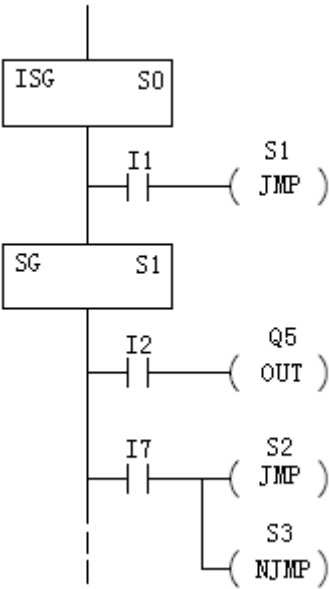
输入 NJMP 指令是允许程序将所属级的 ON 状态在转移条件不成立时向指定的级转移的指令。当级转移的条件不成立时，级转移发生。NJMP 指令执行后的下一个扫描周期，其所属级状态被置为 OFF。



操作数类型		NK1 范围
		aaa
级	S	0-1777

在下面的例子中，仅当 ISG S0 为 ON 时，CPU 开始执行程序。当 I1 为 ON 时，程序执行从初始级 S0 转移到 S1。在级 S1 中，当 I2 为 ON 时，输出 Q5 被置为 ON。如果 I7 为 ON，程序执行将从 S1 转移到 S2；如果 I7 为 OFF，程序执行将从 S1 转移到 S3。

语句视图



```
ISG S0
LD I1
JMP S1
SG S1
LD I2
OUT Q5
LD I7
JMP S2
NJMP S3
```

23.5 合流级登记指令（CV）

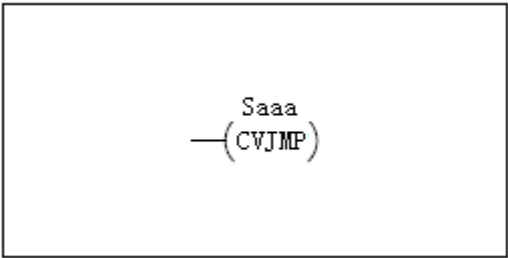
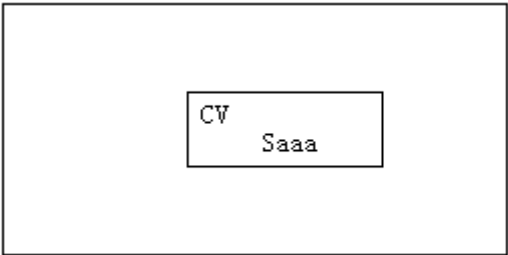
23.6 合流级转移指令（CVJMP）

 合流级登记指令用于将几个级定义为合流级，由 CV 指令登记的级群称为合流级群。

当合流级群中的所有合流级为 ON 时，CVJMP 指令（或者最后一个合流级后的逻辑）将被执行。最后一个合流级中的程序执行前，前面所有合流级的状态必须为 ON。CVJMP 指令执行后的下一个扫描周期，所有合流级状态被置为 OFF。

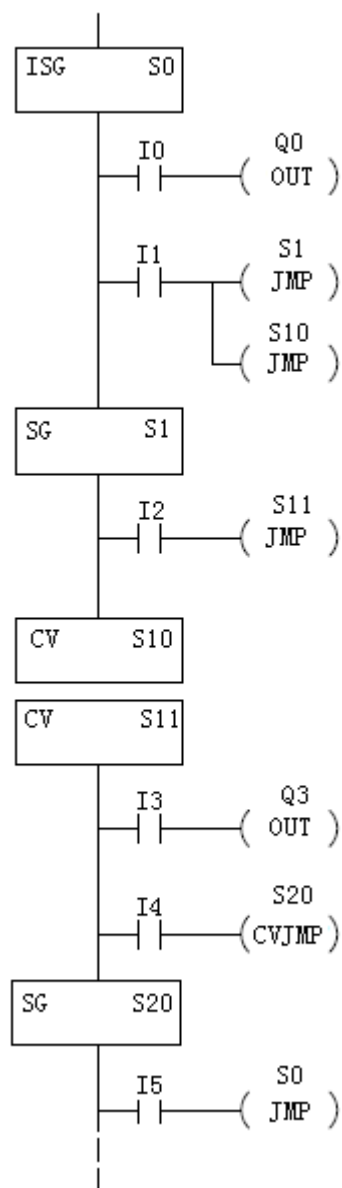
只能在最后一个合流级登记指令和 CVJMP 指令之间添加逻辑指令。允许使用多个 CVJMP 指令。

合流级只能用于主程序中，不可用于子程序或中断程序中。



操作数类型		NK1 范围
		aaa
级	S	0-1777

下面的例子中，当合流级 S10 和 S11 都为 ON，并且 I4 为 ON 时，CVJMP 指令执行。CVJMP 指令将 S10 和 S11 置为 OFF，将 S20 置为 ON。然后，如果 I5 为 ON，程序执行将返回到初始级 S0。



语句视图

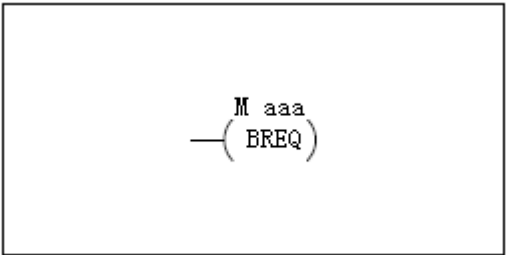
```

ISG S0
LD I0
OUT Q0
LD I1
JMP S1
JMP S10
SG S1
LD I2
JMP S11
CV S10
CV S11
LD I3
OUT Q3
LD I4
CVJMP S20
SG 20
LD I5
JMP S0
  
```

23.7 级组请求指令（BREQ）

 BREQ 指令用于激活一个级组,关于 BREQ 指令需要知道以下几点:

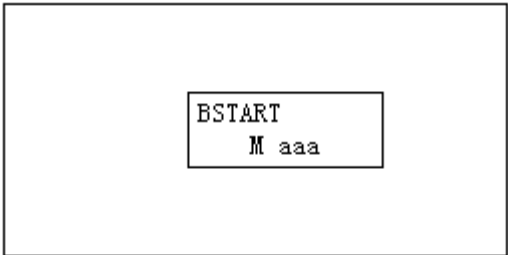
- 使用内部线圈作为操作数，不能将级作为操作数。被使用的内部线圈在程序中不能再用作别的输出。
- 必须保持 ON 状态: BREQ 指令实际上控制 BSTART 与 BEND 指令之间的所有程序，即使在级组程序已经开始执行之后，BREQ 指令也必须保持 ON 状态，否则级组中所有的级状态自动变为 OFF。如果 BREQ 指令或是 BREQ 指令所在的级其中的一个状态变为 OFF，那么所定义的级组中的所有级状态将变为 OFF。
- 激活级组中第一个级: 当 BREQ 指令执行时，自动激活 BSTART 指令后的第一个级。



操作数类型	NK1 范围
	aaa
级	S 0-1777

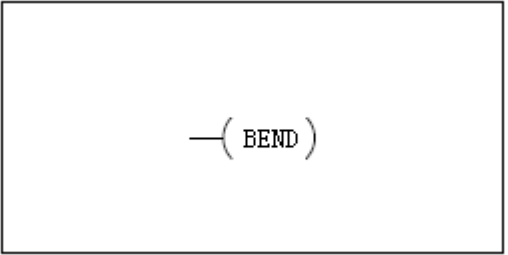
23.8 级组开始指令（BSTART）

 BSTART 指令是一个级组开始的标志，级组中的级状态可被成组置为 ON 或 OFF。BSTART 指令的下一个指令必须是 SG 指令。级组中不能使用初始级。BSTART 指令中使用的内部线圈在程序中不能再用作别的输出。



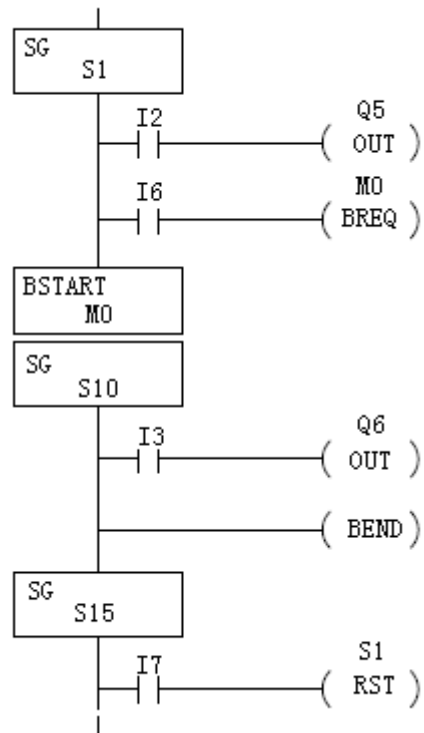
23.9 级组结束指令（BEND）

 BEND 指令是级组结束的标志。这个指令没有操作数。



操作数类型	NK1 范围
	aaa
级	S 0-1777

下面的例子中，当级 S1 为 ON 并且 I6 为 ON 时，BREQ 指令自动激活级 S10。如果 BREQ 指令变为 OFF，或者其所在的级状态变为 OFF，那么 BSTART 与 BEND 指令之间的所有级自动变为 OFF。级 S15 中，I7 可将 S1 复位，因此可将 BREQ 指令置为 OFF，也因此可将级组中的所有级复位。



语句视图

```

SG S1
LD I2
OUT Q5
LD I6
BREQ MO
BSTART MO
SG S10
LD I3
OUT Q6
BEND
SG S15
LD I7
RST S1
  
```


第 24 章 凸轮控制介绍

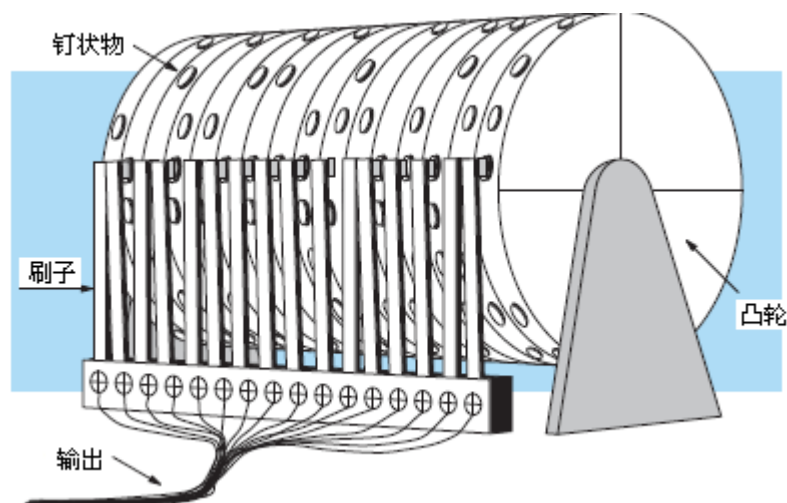
24.1 简介

凸轮控制术语

凸轮控制指令适用于包含一定步骤的重复性过程控制。简单的一条凸轮控制指令，可以完成许多行梯形图程序才完成的工作。因此，凸轮控制指令可以省去很多编程和调试的时间。

这里通过下图的原始机械凸轮，介绍一下凸轮控制指令相关术语。机械凸轮表面通常以特定的模式分布着一些钉状物，代表着一组期望的机械控制动作。电动机或电磁铁在特定的时间对凸轮旋转一个精确的量。在旋转过程中，固定的刷子感应钉状物的存在（存在时=ON，不存在时=OFF）。钉状物与刷子的碰触使电触点进行开合，产生了电信号输出，这个输出信号可用来控制机器的开关。

通常情况下凸轮一周有一定数量的位置，叫做步。每步代表一些进程步骤。上电时，凸轮复位到一个指定的步。凸轮从一步到下一步是基于一个定时器或是一些外部事件。在一些特定的条件下，机器操作者可以使用一个外部事件输入手动转移凸轮的步。固定的刷子和钉状物碰触生成的唯一的 ON/OFF 序列，用于机器的特定控制。因为凸轮是环形的，因此每一圈的序列都是自动重复的。凸轮的应用千差万别，有的应用一秒转一圈，有的应用一星期转一圈。



相比机械凸轮，电子凸轮有更多的优点，例如，它有预设功能，而机械凸轮不可能有这个功能。预设功能可使控制从当前步直接跳到任何其它步。

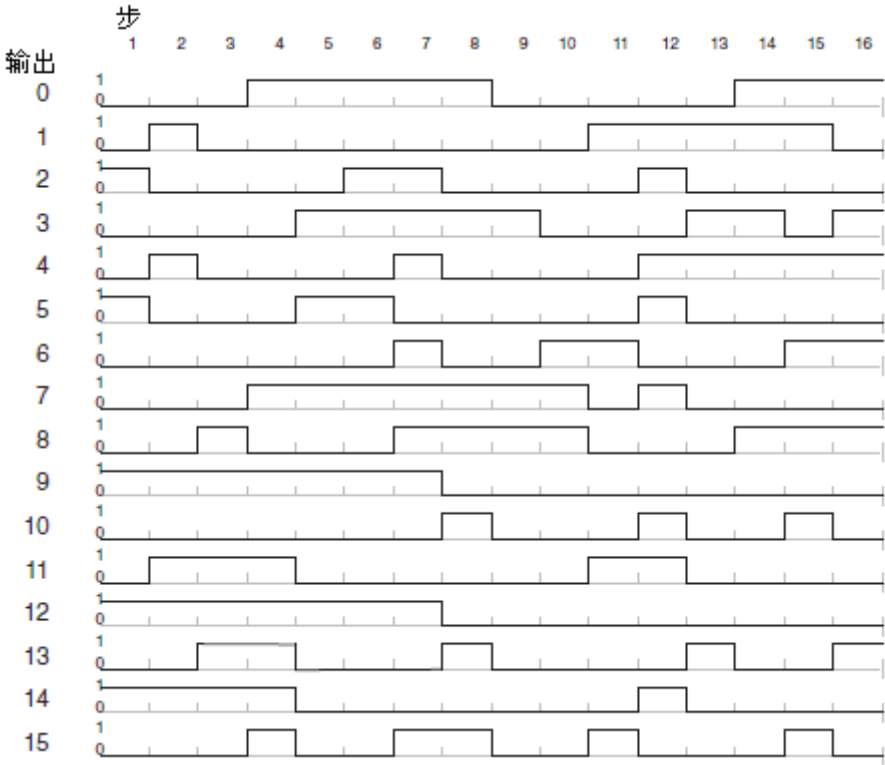
凸轮控制图表

为了编辑的方便，电子凸轮在编程软件和本手册中以图表的形式表现，见下表。每一行代表一步，编号是 1-16。每一列代表一个输出，编号是 0-15。实心点代表刷子碰到钉状物（ON 状态），空心点代表刷子没有碰到钉状物（OFF 状态）。

步骤	输出															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	○	●	○	●	○	○	●	○	○	○	●	○	○	●	○	○
2	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
3	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
4	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
5	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
6	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
7	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
8	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
9	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
10	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
11	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
12	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
13	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
14	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
15	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
16	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

输出顺序

下图是上述的凸轮控制图表的 ON/OFF 输出顺序表，对比二者，你会发现，两者是完全相等的。



24.2 步转移

凸轮控制指令类型

NK1 CPU 中有四种类型的凸轮控制指令：

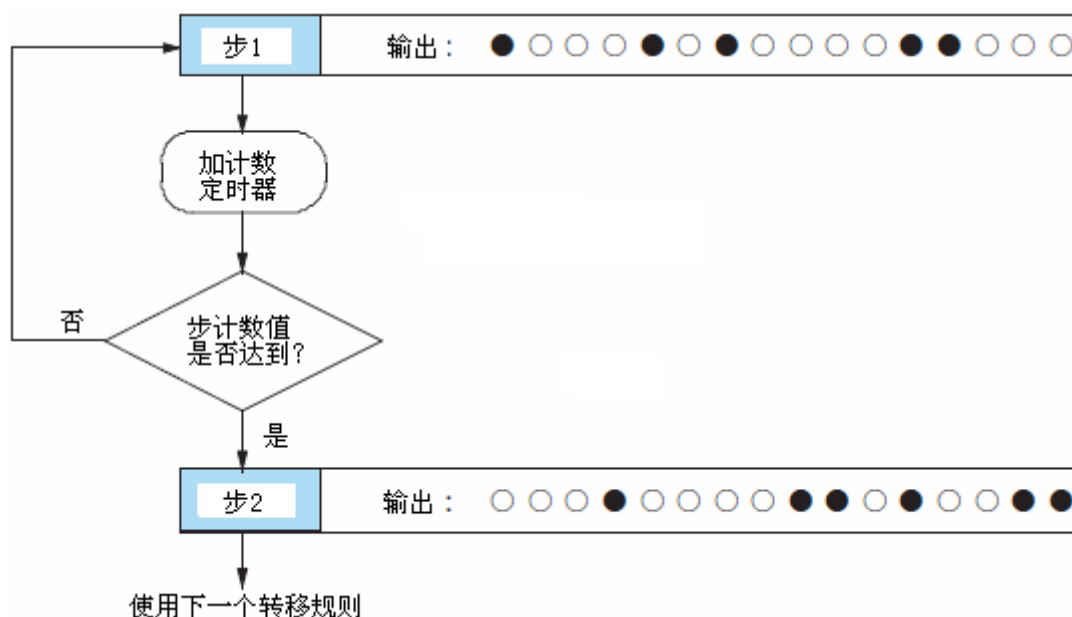
- 时间驱动型凸轮（DRUM）
- 时间/事件型凸轮（EDRUM）
- 带掩膜事件驱动型凸轮（离散点输出）（MDRMD）
- 带掩膜事件驱动型凸轮（字输出）（MDRMW）

这四个凸轮指令都包含了基于时间的步转移，其中三个还包含了基于事件的步转移，以及其它选项包括将输出定义为字或单个位和输出屏蔽（单个输出禁用/启用）。

每个凸轮有 16 步，每步有 16 个输出，输出可以是 Q 或 M 线圈。

仅基于时间步转移

步转移仅基于时间。每步都有各自的转移条件，转移条件在输入凸轮指令时已经设定完成。下图演示了时间驱动型步转移是如何工作的。



凸轮停留在步 1 一个特定的时间（可以编程设定）。定时器的时基可以编程设定，范围是 0.01–99.99s。这样相当于确立了每个“滴答时钟”或停留时间的分辨率。每步使用相同的时基，但是有各自独立的计数值，可以编程设定。凸轮在每步的停留时间由下面的公式算出：

每步的停留时间=时基×每步的计数值

例如，如果步 1 是 5s 的时基和 12 的计数值，则凸轮将在步 1 停留 60s 的时间。每步的最大停留时间由下式算出：

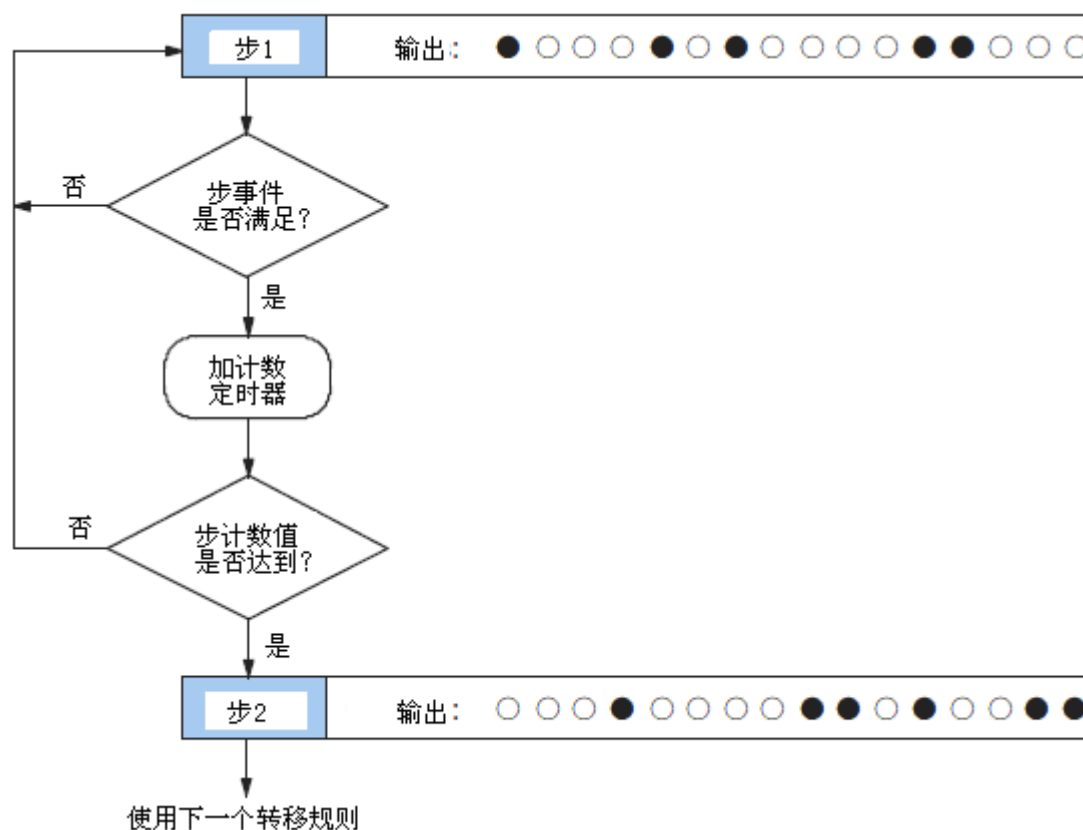
每步最大停留时间=0.01s×9999×9999=999,800 秒=277.7 小时=11.6 天



注意：如果是第一次选择时基，一个好的方法是：使时基=最短步的停留时间×1/10。然后就可以以 10%的增量调整最短步的停留时间了。其它时间较长的步可以以更小的增量优化停留时间（以百分比计算）。还要注意的，凸轮控制指令每个 CPU 扫描周期执行一次，因此，如果时基比 CPU 扫描时间快很多的话是毫无意义的。

时间/事件型步转移

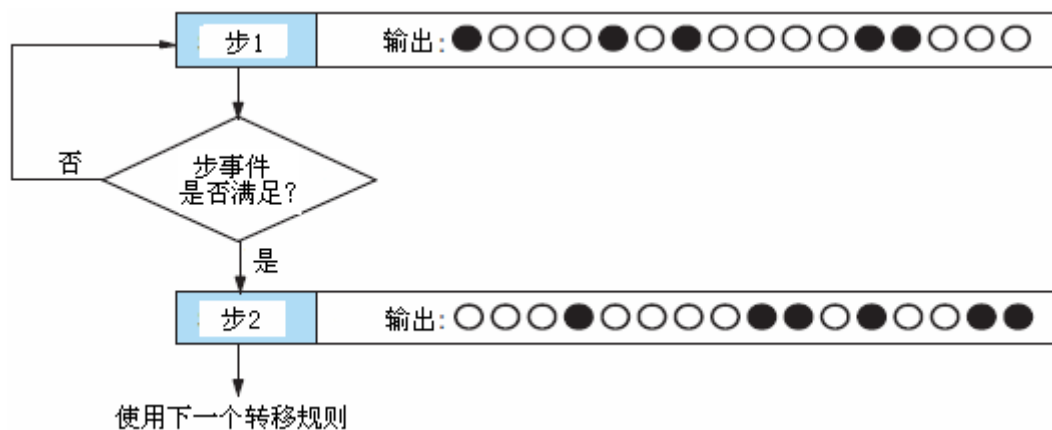
步转移基于时间/事件。下图演示了时间/事件驱动型转移是如何工作的。



当凸轮进入步 1，输出图表如上图所示。然后开始检测此步的步事件是否满足。步事件可以是 I、Q、M 等开关量。假设我们选择 I0 为步 1 的事件。如果 I0 为 OFF，则凸轮停留在步 1。当 I0 为 ON 时，事件条件满足，计数器开始计数。只要 I0 保持为 ON，计数器就计数。当步 1 的计数达到时，凸轮将进入步 2，输出将立即改变为步 2 的输出。

仅基于事件步转移

步转移不一定都需要事件和时间两个条件，可以只使用两个条件的中的其中一个，甚至一个凸轮中的所有步可以混合使用转移规则。例如，步 1 的转移条件可以是一个事件，步 2 的转移条件是时间，步 3 的转移条件是时间和事件。此外，还可以选择使用 16 步中的几步或者 16 个输出中的几个输出。



计数器分配

每个凸轮控制指令要使用四个计数器。对凸轮控制指令进行编程时，选择第一个计数器编号，凸轮指令自动使用接下来的三个计数器。当凸轮完成一个循环时，第一个计数器的计数器位变为 ON；当凸轮被重置时，第一个计数器的计数器位变为 OFF。这些计数器的计数值和计数器位精确地反应了凸轮指令的进程，可以在程序中监控这些数据。

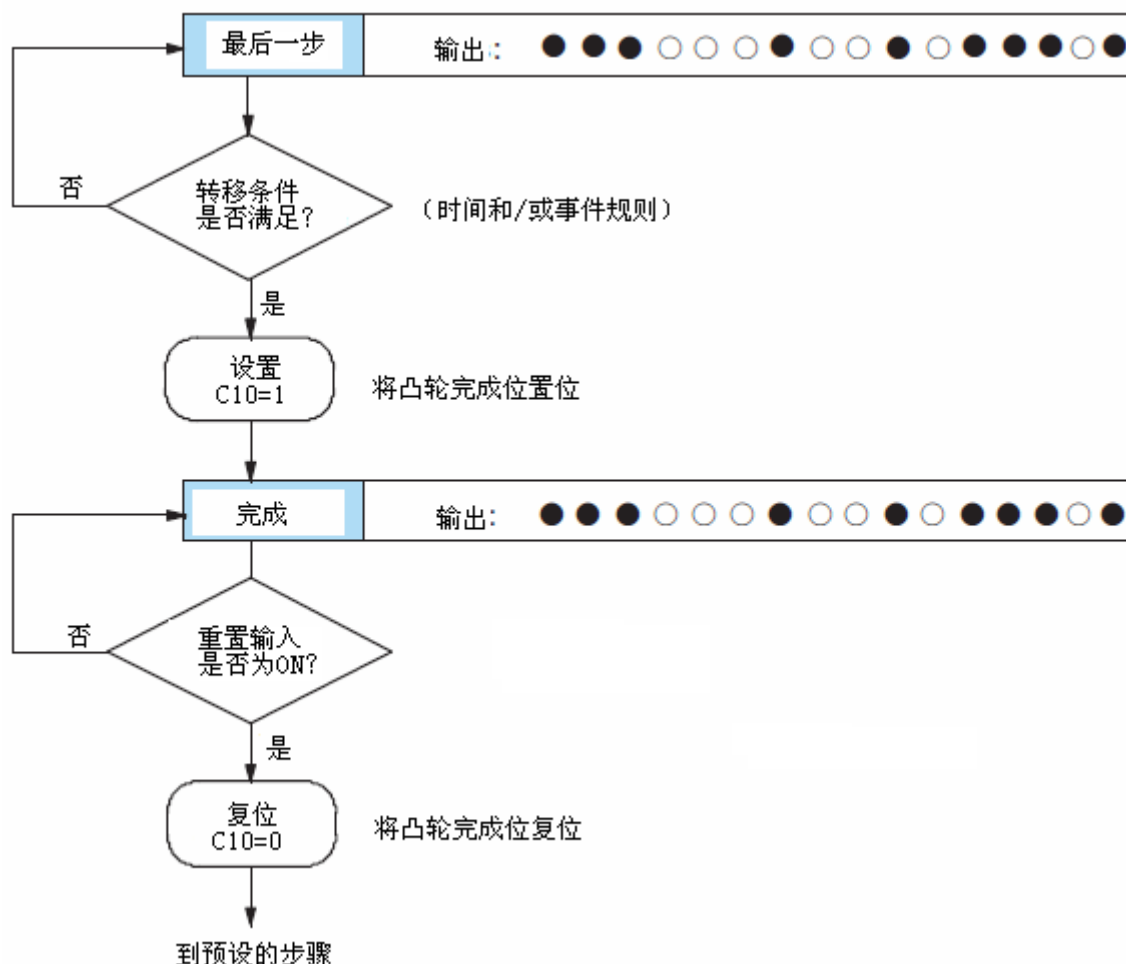
假设编程时使用一个凸轮的 8 个步，选择 C10 为第一个计数器（注意计数器编号是八进制的），计数器的使用见右边的表格。最右边一栏是典型值，下面会进行讲解。

计数器分配			
C10	步的计数	R1010	1528
C11	定时器值	R1011	0200
C12	预设步	R1012	0001
C13	当前步	R1013	0004

C10 的值 1528 表明当前步中的计数值是 1528，当前步是步 4，由 C13 的计数值表示。如果编程时设定步 4 的计数值是 3000，那么此步的计数刚好完成一半。C11 是计数定时器，单位是 0.01s。因此，数值的最小变化是 0.01s。200 表示计数到 1528 所用的时间是 2s (0.01×200)。C12 保持编程时在凸轮控制指令中设定的步预设值。当凸轮的重置输入为 ON 时，凸轮回到步 1。仅当程序向其写入数据或凸轮指令被编辑又或者程序重启时，C12 的值才改变。当凸轮的循环结束时，C10 的计数器位变为 ON，当凸轮被重置时，C10 的计数器位变为 OFF。

最后一步完成

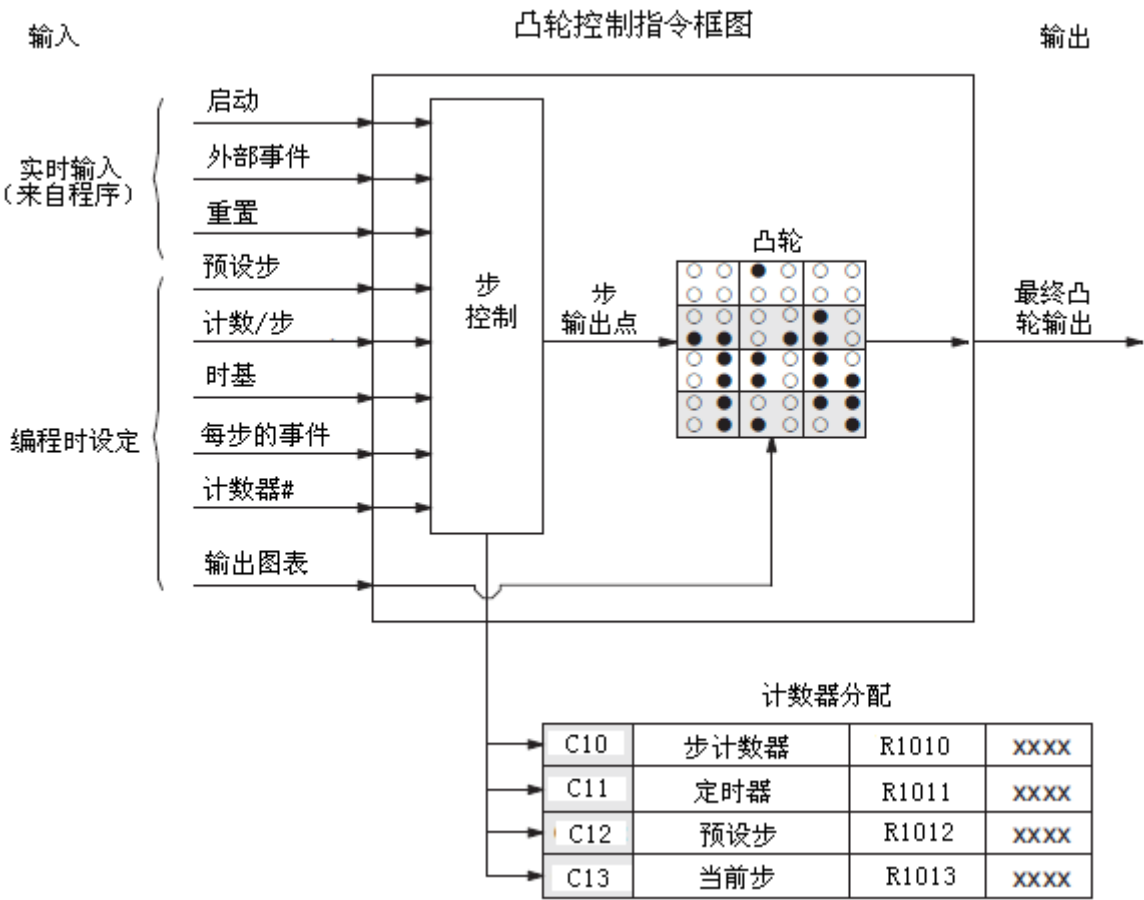
凸轮控制的最后一步可以是任何一步，因为局部凸轮是有效的。见下图，当最后一步的转移条件满足后，凸轮将相应的计数器（本例中为 C10）位置 ON，这个计数器编程时在凸轮控制指令中设定。此后进入到“凸轮完成”的状态，此时的凸轮输出仍保持最后一步的输出。凸轮的一个循环完成时，启动和外部事件输入失效。当重置输入变为 ON 时（或 CPU 完成编程→运行模式的转换时），凸轮离开“凸轮完成”状态。此时，凸轮完成位（本例中为 C10）被复位，然后凸轮直接进入预设的步骤中。



24.3 凸轮操作概述

凸轮控制指令框图

凸轮操作指令的输入和输出，见下图。



凸轮控制指令接受几个输入用于步控制，即凸轮的主要控制。这些输入及其功能如下：

- 启动：仅当重置输入为 OFF 时，启动输入有效。当启动输入为 ON 时，时间驱动型凸轮的定时器开始计时，事件驱动型的凸轮检测事件输入。当启动输入为 OFF 时，凸轮停止在当前的状态（重置输入必须保持为 OFF），凸轮输出保持当前的 ON/OFF 状态。
- 外部事件：仅当重置输入为 OFF 时，外部事件输入有效（启动输入可以是 ON 或是 OFF）。当外部事件输入发生 OFF→ON 的转换时，凸轮转移到下一步（仅 EDRUM 指令支持这个输入）。
- 重置：重置输入的优先级比启动输入高。当重置输入为 ON 时，凸轮转移到预设步。当重置输入为 OFF 时，启动输入可正常操作。
- 预设步：定义预设的步，范围是 1-16（通常情况下预设步为 1）。当重置输入为 ON 或 CPU 首次进入运行模式时，凸轮转移到预设的步。
- 计数/步：凸轮停留在每步的定时计数值。每步都有自己的计数值，计数/步是可选的。
- 时间值：计数/步定时器的当前值。
- 计数器#：凸轮用于步转移控制的四个连续计数器的首个计数器号。通过监控这些计数器可以监控凸轮的控制过程。
- 每步的事件：I, Q, M, S, T, C 或 SP 开关量输入都可以作为步转移输入。每步都有各自的事件，对于时间/事件驱动型凸轮，事件是可选的。



警告：CPU 在 RUN 模式下，凸轮的输出在任何时间都是启用状态。启动输入不必为 ON，并且重置输入不禁用输出。在进入运行模式后，凸轮根据当前步的输出图表自动将输出置 ON 或 OFF。起始步取决于计数器的设置：是停电保持还是非停电保持。

凸轮寄存器的上电状态

选择上电和编程→运行模式时的启动步是很重要的，请参考下面的图表。如果计数器存储器被设置为非停电保持，凸轮在每次上电和编程→运行模式转换时进入的步都是一样的。但是，如果计数器存储器被设置为停电保持，凸轮会保持它先前的状态。

计数器号	功能	上电后的初始状态	
		非停电保持	停电保持
C (n)	步的计数值	初始值=0	先前状态（无改变）
C (n+1)	计数器定时值	初始值=0	先前状态（无改变）
C (n+2)	预设步	初始值=预设步	先前状态（无改变）
C (n+3)	当前步	初始值=预设步	先前状态（无改变）

相对来说凸轮循环时间较短的、上电时需要重置的应用，可选用非停电保持；循环时间较长的、上电时需保持停止时状态的应用，可选用停电保持。缺省设置是停电保持。因此，如果初始化 SPD 寄存器，存储器将是停电保持。

24.4 凸轮控制技术

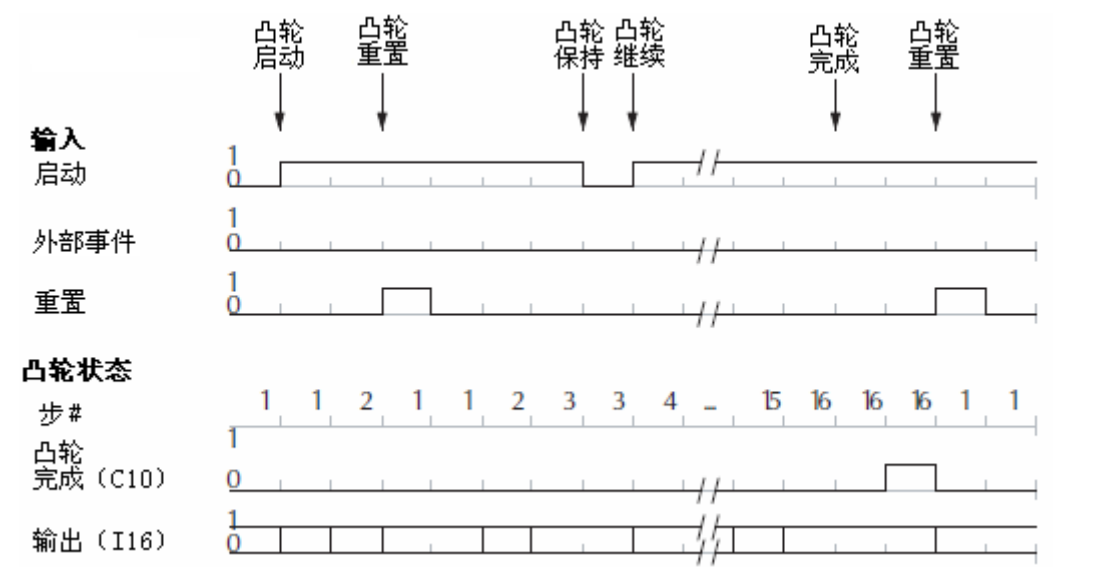
凸轮控制输入

右图是一个简化的通用凸轮控制指令，有三个输入：启动，外部输入和重置输入（只有 EDRUM 指令支持外部事件输入）。第一个计数器的计数器位（例如 C10）为 ON 表明凸轮循环结束。

下面的时序图显示了一个时间驱动型凸轮的输入以及凸轮是如何响应的。当 CPU 进入运行模式，凸轮将步号初始化为预设步号（通常情况下为步 1）。当启动输入变为 ON 时，凸轮开始运行，等待一个事件和/或计数器计时的到来。

凸轮进入步 2 后，重置输入变为 ON，而启动输入仍然为 ON。由于重置信号的优先级高于启动信号的优先级，因此凸轮进入到预设步（步 1）。注意在重置时凸轮保持在预设步，并且那个步没有运行（对时间或定时器的响应）直到重置信号变为 OFF。

凸轮进入步 3 后，启动输入变为 OFF，在启动输入变为 OFF 的瞬间，凸轮的定时器停止计时直到启动信号重新变为 ON。



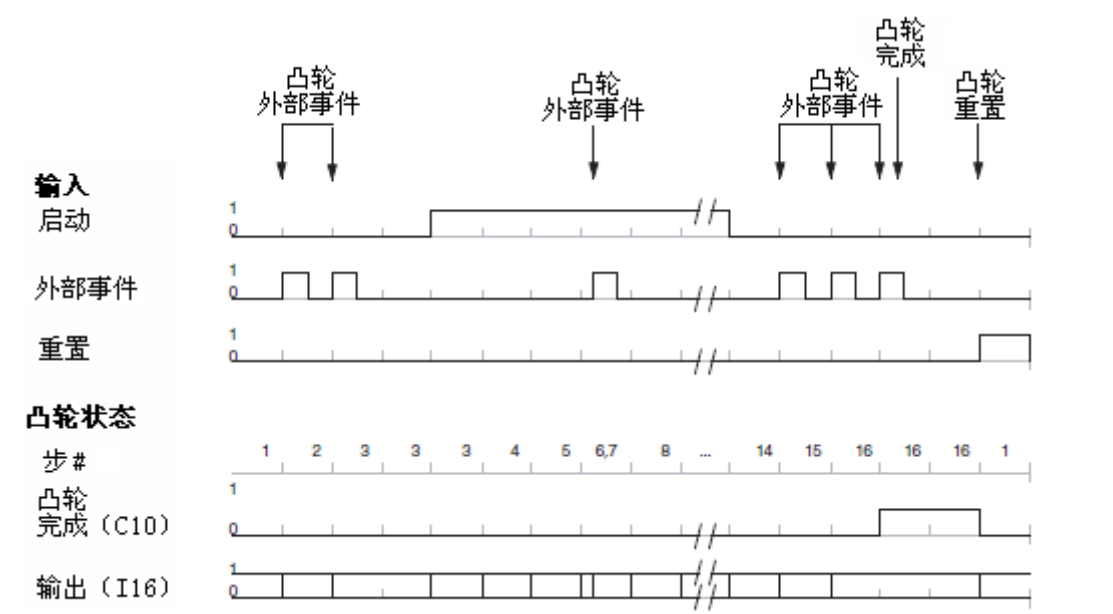
当凸轮完成最后一步后（本例中是步 16），凸轮完成位（C10）变为 ON，此时凸轮仍保持在步 16。当重置输入变为 ON 时，凸轮完成位（C10）变为 OFF，然后凸轮进入预设的步。



注意：上面的时序图中所有步的持续时间都相同，实际应用中，步的持续时间可以相差很大，可以在编程时设定。

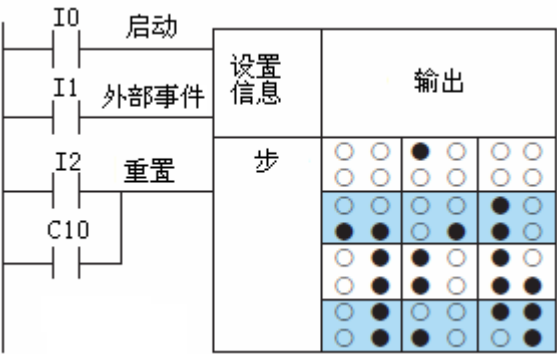
下面的时序图中，把重点放在外部事件输入对事件驱动型凸轮的操控。注意到，当外部事件输入发生 OFF→ON 转换时，凸轮的步就会增加一。启动输入可以是 ON 也可以是 OFF（但是重置输入必须为 OFF）。下图中，两个外部事件输入使凸轮进入了步 3。启动输入变为 ON 后，凸轮开始正常运行。在步 6 中，又一个外部事件输入发生，使凸轮进入了步 7，并将定时器复位到 0。由于启动输入仍然为 ON，凸轮正常运行到下一个步中。

当凸轮进入步 14 后，启动输入变为 OFF。两个或更多的外部事件输入使凸轮进入到步 16。但是注意，需要第三个外部事件输入将凸轮由步 16 转移到“凸轮完成”。最后，重置信号进入，将凸轮转移到预设步并将凸轮完成位置 OFF。



自复位凸轮

实际应用中通常需要凸轮完成一个循环后能自动开始下一个循环,使用凸轮完成位很容易实现这一功能。右图中，凸轮指令使用的计数器是 C10,因此，将 C10 同重置输入逻辑或。当最后一步完成后，凸轮将 C10 置 ON，这样就将凸轮转移到预设的步，同时也将 C10 置 OFF。I2 仍可用作一个手动重置触点。



初始化凸轮输出

只要 CPU 进入运行模式，凸轮输出便被启用。在编程→运行模式的转换后，凸轮马上进入预设步，预设步的输出被激活。如果应用需要上电时所有输出都为 OFF，可将预设步用作“重置步”，将所有输出都置为 OFF。

使用复杂步事件进行步转移

事件型凸轮指令每步只能输入一个事件触点用于步转移,但是这并不意味着用于步转移的事件只能是一个触点。例如，将 M0 触点作为凸轮控制指令其中一步的步转移事件，但是在程序里，可对 M0 进行编程，将其作为一个很多其它“事件”的逻辑输出触点。

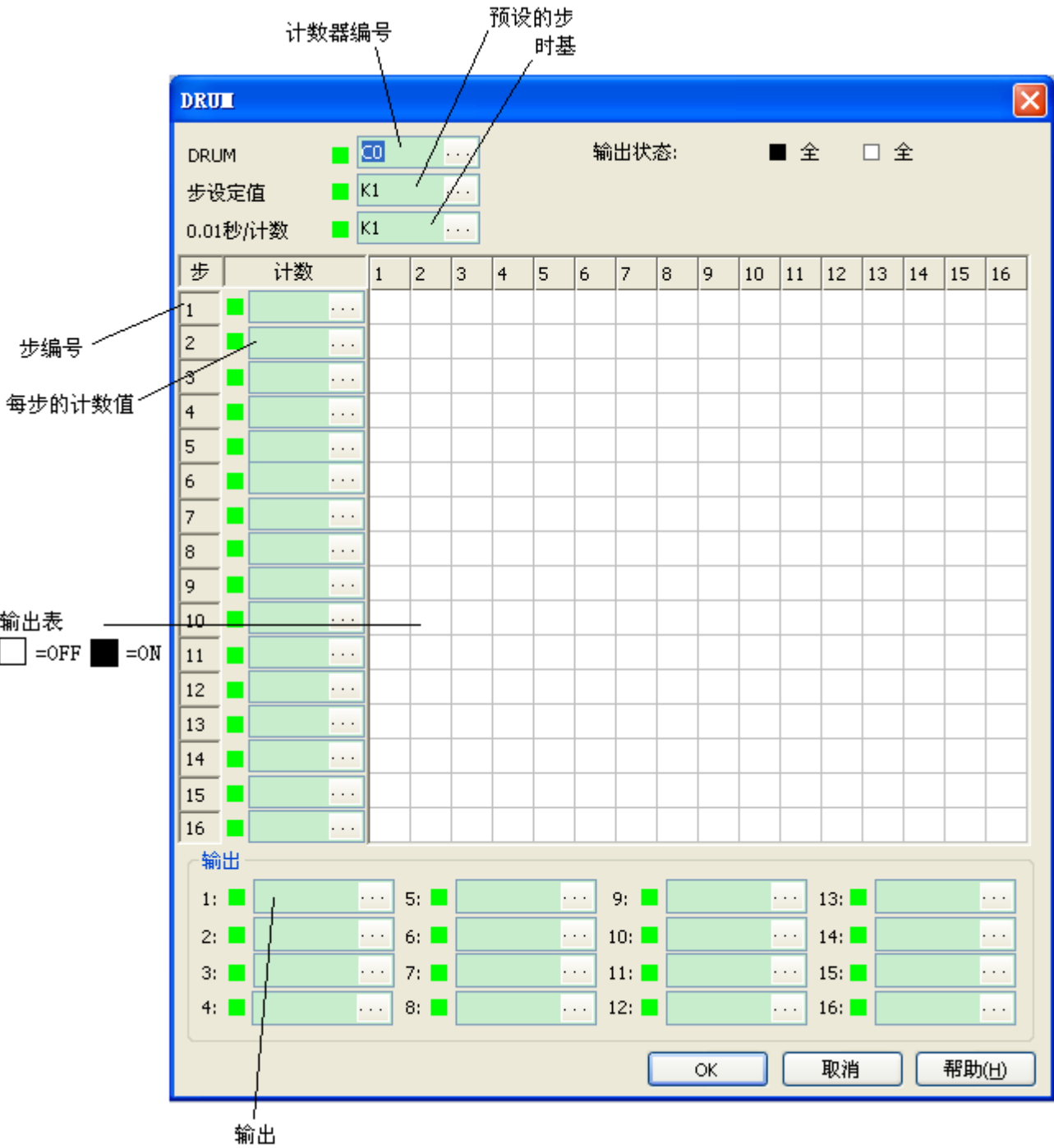
第 25 章 凸轮控制指令

所有凸轮控制指令可以使用编程软件进行编程。

25.1 时间驱动型凸轮控制指令（DRUM）

DRUM 指令是最基本的凸轮控制指令。

DRUM 指令有 16 步和 16 个输出，只基于时间进行步转移，每步的时基相同，计数值可单独设定。不使用的步编程时要设定步计数值=0（缺省值）。开关量输出可以是单独的 I、Q 或 M 输出，也可以不使用。



当启动输入为 ON 时，凸轮的定时器开始计时，最后一步完成后或重置输入为 ON 时计时

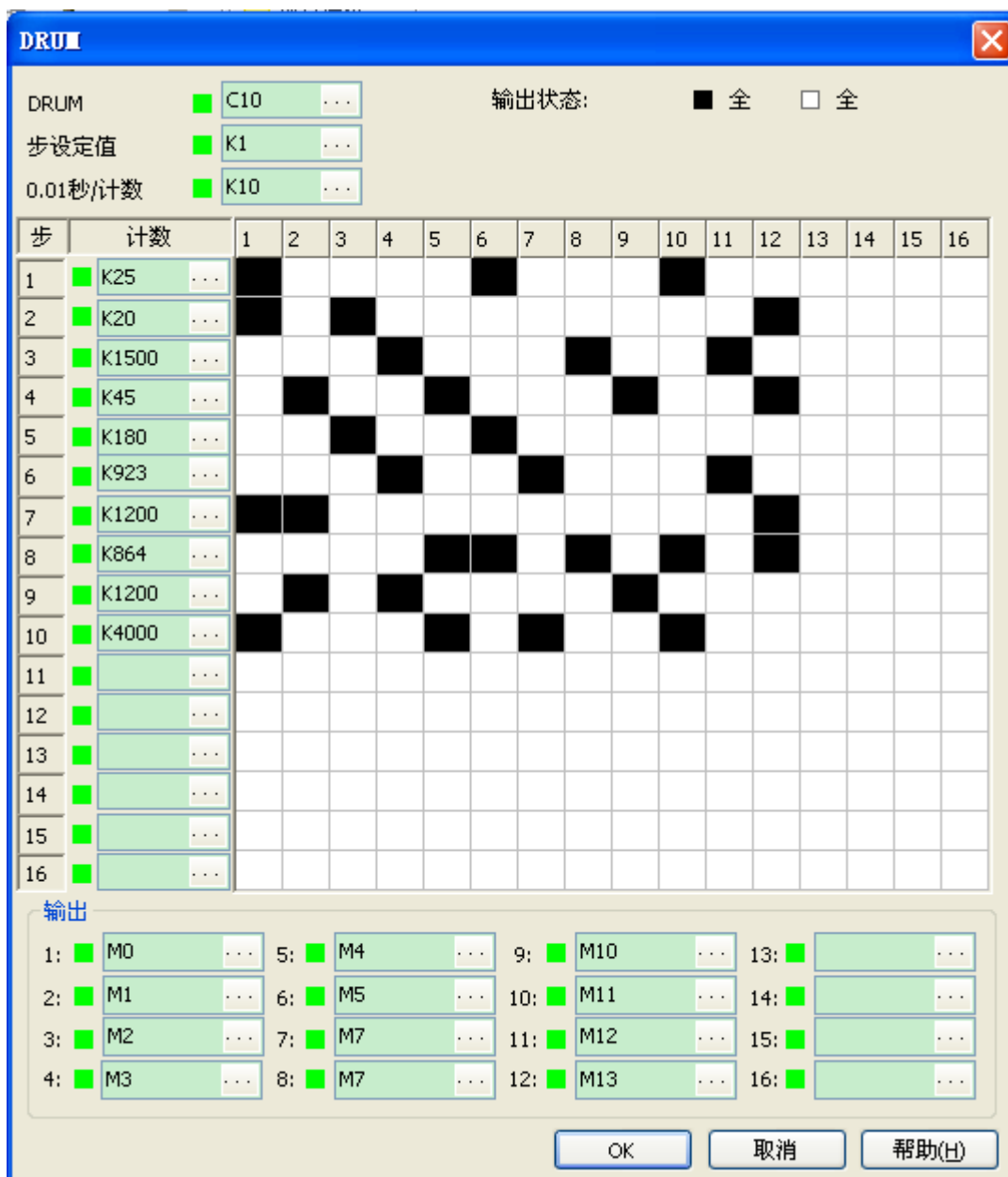
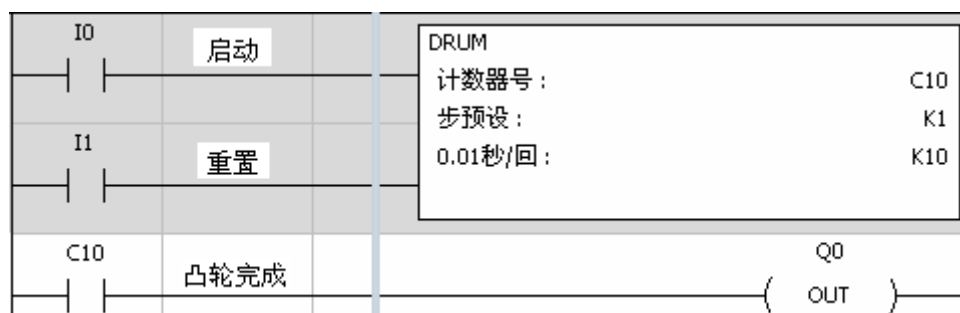
停止。在 CPU 进行编程→运行的模式转换后或重置输入为 ON 时，凸轮进入预设定的步。

凸轮参数	数据类型	范围
计数器编号	C	0-774
预设定的步	K	1-16
时基	K	0-99.99s
每步的计数值	K	0-9999
开关量输出	I, Q, M	所有（附录 1）

凸轮指令使用四个计数器。可读取计数器的值来监控凸轮的状态。在程序中可随时向 CR(n+2) 中写入预设定的步。其他三个计数器仅做监控用。

计数器编号	n 的范围	功能	计数器位功能
C(n)	0-774	每步的计数	C(n)=凸轮完成
C(n+1)	1-775	定时器值	C(n+1) 未使用
C(n+2)	2-776	预设定的步	C(n+2) 未使用
C(n+3)	3-777	当前步	C(n+3) 未使用

下面的程序例中，使用了步 1-10 和 16 个输出中的 12 个输出。预设定的步是步 1。时基是 $K10 \times 0.01 = 0.1s$ 。因此，步 1 的停留时间是 $25 \times 0.1 = 2.5s$ 。最后一行程序，凸轮完成最后一步时计数器位 C10 将 Q0 置位。凸轮重置输入可将 C10 复位。



25.2 时间/事件驱动型凸轮控制指令（EDRUM）

EDRUM 指令是基于时间和事件进行步转移的指令。

EDRUM 指令有 16 步和 16 个输出，基于时间和/或事件进行步转移，外部事件输入发生 OFF→ON 的转换时也会发生步转移。步转移的时间在每步的计数值中设定，事件为开关型触点。不使用的步和事件必须空着。开关量输出点可单独设定。

EDRUM

EDRUM ☒ C0 ...

步设定值 ☒ K1 ...

0.01秒/计数 ☒ K1 ...

输出状态: ☒ 全 ☐ 全

步	计数	事件	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	☒	☒																
2	☒	☒																
3	☒	☒																
4	☒	☒																
5	☒	☒																
6	☒	☒																
7	☒	☒																
8	☒	☒																
9	☒	☒																
10	☒	☒																
11	☒	☒																
12	☒	☒																
13	☒	☒																
14	☒	☒																
15	☒	☒																
16	☒	☒																

输出

1: ☒ ... 5: ☒ ... 9: ☒ ... 13: ☒ ...

2: ☒ ... 6: ☒ ... 10: ☒ ... 14: ☒ ...

3: ☒ ... 7: ☒ ... 11: ☒ ... 15: ☒ ...

4: ☒ ... 8: ☒ ... 12: ☒ ... 16: ☒ ...

OK 取消 帮助(H)

输出表
☐ =OFF ☒ =ON

凸轮参数	数据类型	范围
计数器编号	C	0-774
预设定的步	K	1-16
时基	K	0-99.99s
每步的计数值	K	0-9999
事件	I, Q, M, S, T, C, SP	所有附录 1
开关量输出	I, Q, M	

当启动输入为 ON 时，凸轮定时器被启用。只要当前步的事件为 ON，则当前步的定时器开始计时。当步计数值等于步的设定计数值时，凸轮转移到下一步。当最后一步完成或重置输入为 ON 时，此过程停止。CPU 在编程→运行的模式转换后或是重置输入为 ON 时，凸轮进入预设定的步。

凸轮指令使用四个计数器。可读取计数器的值来监控凸轮的状态。在程序中可随时向 CR(n+2) 中写入预设定的步。其他三个计数器仅做监控用。

计数器编号	n 的范围	功能	计数器位功能
C(n)	0-774	每步的计数	C(n)=凸轮完成
C(n+1)	1-775	定时器值	C(n+1) 未使用
C(n+2)	2-776	预设定的步	C(n+2) 未使用
C(n+3)	3-777	当前步	C(n+3) 未使用

下面的程序例中，使用了步 1-11 和全部 16 个输出。预设定的步是步 1。时基是 $K10 \times 0.01 = 0.1s$ 。因此，步 1 的停留时间是 $1 \times 0.1 = 0.1s$ 。注意步 1 仅基于时间进行步转移（事件是空着的），并且步 1 的所有输出均为 OFF，这是一个典型的上电时的状态。最后一行程序凸轮完成最后一步时完成位 C4 将 Q0 置位。凸轮重置输入可将 C4 复位。



EDRUM

C4

...

输出状态:

全

全

步设定值

K1

...

0.01秒/计数

K10

...

输出

1:

Q1

...

2:

M34

...

3:

M3

...

4:

M7

...

5:

M30

...

6:

M14

...

7:

Q2

...

8:

Q0

...

9:

M2

...

10:

M4

...

11:

M7

...

12:

Q5

...

13:

Q4

...

14:

Q1

...

15:

M10

...

16:

M7

...

OK

取消

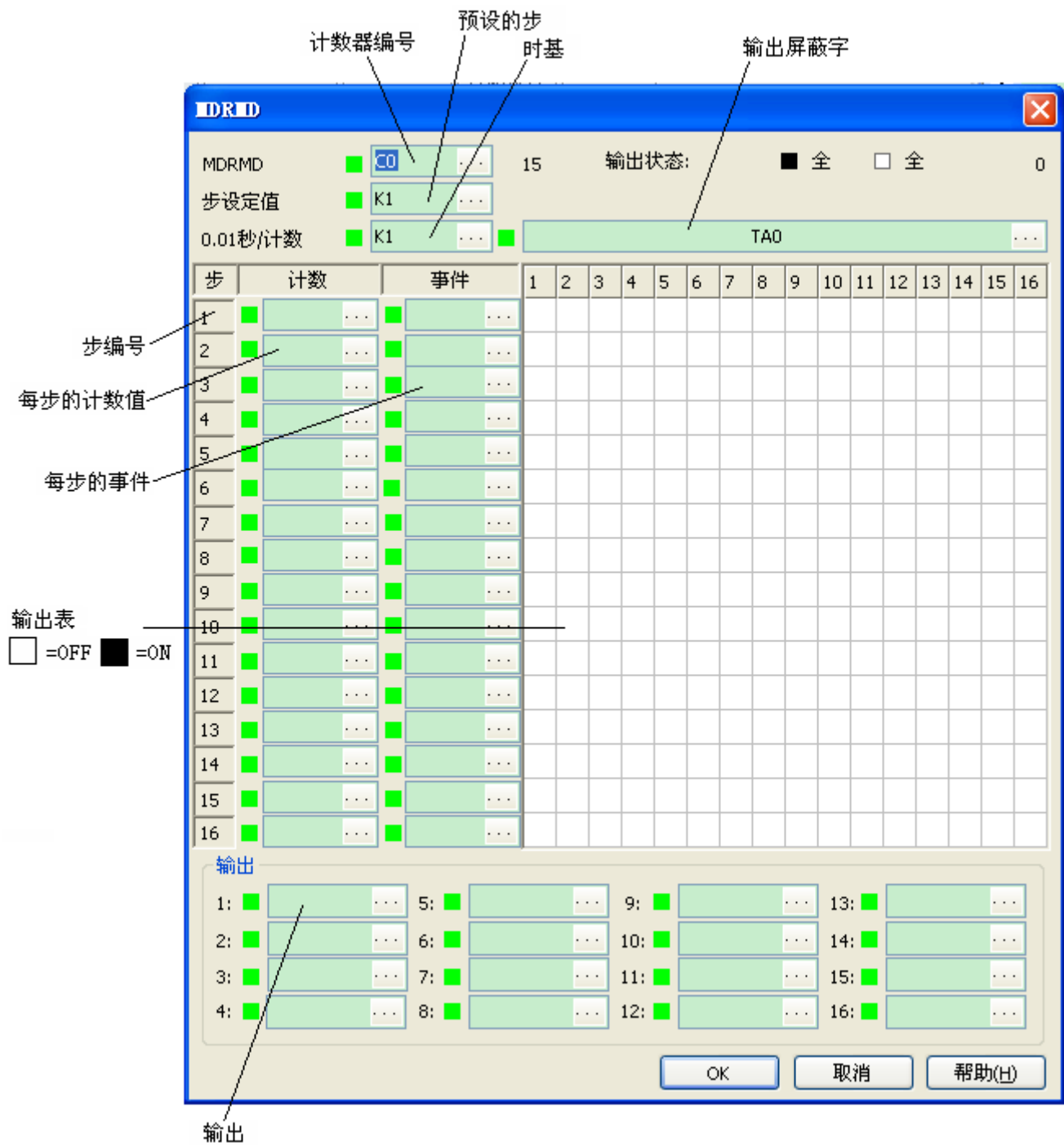
帮助(H)



注意：如果一个事件驱动型凸轮（所有步的计数值均为 0）的所有事件均为 ON，则 PLC 每个扫描周期完成一步；因此，凸轮完成一个循环需要 16 个扫描周期。但是，由于 CPU 处于运行模式时凸轮的输出一直是启用的，因此凸轮的输出将相当于每个扫描周期的脉冲输出。

25.3 带掩膜事件驱动型凸轮控制指令（MDRMD）（离散点输出）

MDRMD 指令不但具有基本事件型凸轮的所有功能，还增加了每步的最终输出控制功能。它根据前面讲的凸轮运行基本规则运行。



MDRMD 指令有 16 步和 16 个输出，每步的凸轮输出同一个输出屏蔽字位逻辑与。输出屏蔽字区域指定了 16 个输出屏蔽字的首个屏蔽字地址。基于时间和/或事件进行步转移，外部事件输入发生 OFF→ON 的转换时也会发生步转移。步转移的时间在每步的计数值中设定，事件为开关型触点。不使用的步和事件必须空着（缺省状态）。当启动输入为 ON 时，凸轮定时器被启用。只要当前步的事件为 ON，则当前步的定时器开始计时。当步计数值等于步的设

定计数值时，凸轮转移到下一步。当最后一步完成或重置输入为 ON 时，此过程停止。CPU 在编程→运行的模式转换后或是重置输入为 ON 时，凸轮进入预设定的步。

凸轮参数	数据类型	范围
计数器编号	C	0-774
预设定的步	K	1-16
时基	K	0-99.99s
每步的计数值	K	0-9999
事件	I, Q, M, S, T, C, SP	所有附录 1
开关量输出	I, Q, M	
输出屏蔽	R	

凸轮指令使用四个计数器。可读取计数器的值来监控凸轮的状态。在程序中可随时向 CR(n+2)中写入预设定的步。其他三个计数器仅做监控用。

计数器编号	n 的范围	功能	计数器位功能
C(n)	0-774	每步的计数	C(n)=凸轮完成
C(n+1)	1-775	定时器值	C(n+1)未使用
C(n+2)	2-776	预设定的步	C(n+2)未使用
C(n+3)	3-777	当前步	C(n+3)未使用

下面的程序例中，使用了步 1-11 和全部 16 个输出。输出屏蔽字从 R2000 开始。凸轮的最终输出是输出表中输出同屏蔽字逻辑与后的结果。如果需要每次上电时凸轮所有输出为 OFF，可在第一个扫描周期向 R2000 中写入 0。程序可随时更新屏蔽字来启用或禁用凸轮的输出。

预设定的步是步 1。时基是 $K10 \times 0.01 = 0.1s$ 。因此，步 1 的停留时间是 $5 \times 0.1 = 0.5s$ 。注意步 1 仅基于时间进行步转移（事件是空着的）。最后一行程序凸轮完成最后一步时完成位 C10 将 Q0 置位。凸轮重置输入可将 C10 复位。





注意：上面的程序例中需要向 R2000-R2012 的 11 个掩码寄存器中写入常数，因为凸轮使用了 11 步。

MDRMD

C10

...

15

输出状态:

全

全

0

步设定值

K1

...

0.01秒/计数

K10

...

R2000

...

步	计数	事件	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	K5 ...	...																
2	K20 ...	Q20 ...																
3	K150 ...	I21 ...																
4	K48 ...	I22 ...																
5	K180 ...	M0 ...																
6	K923 ...	M1 ...																
7	K120 ...	I10 ...																
8	K864 ...	I35 ...																
9	K120 ...	I13 ...																
10	K4000 ...	Q17 ...																
11	...	M20 ...																
12	...	...																
13	...	...																
14	...	...																
15	...	...																
16	...	...																

输出

1:

Q1

...

5:

M30

...

9:

M2

...

13:

Q12

...

2:

M34

...

6:

M14

...

10:

M4

...

14:

Q13

...

3:

Q2

...

7:

Q4

...

11:

Q6

...

15:

M10

...

4:

Q7

...

8:

Q10

...

12:

Q5

...

16:

M7

...

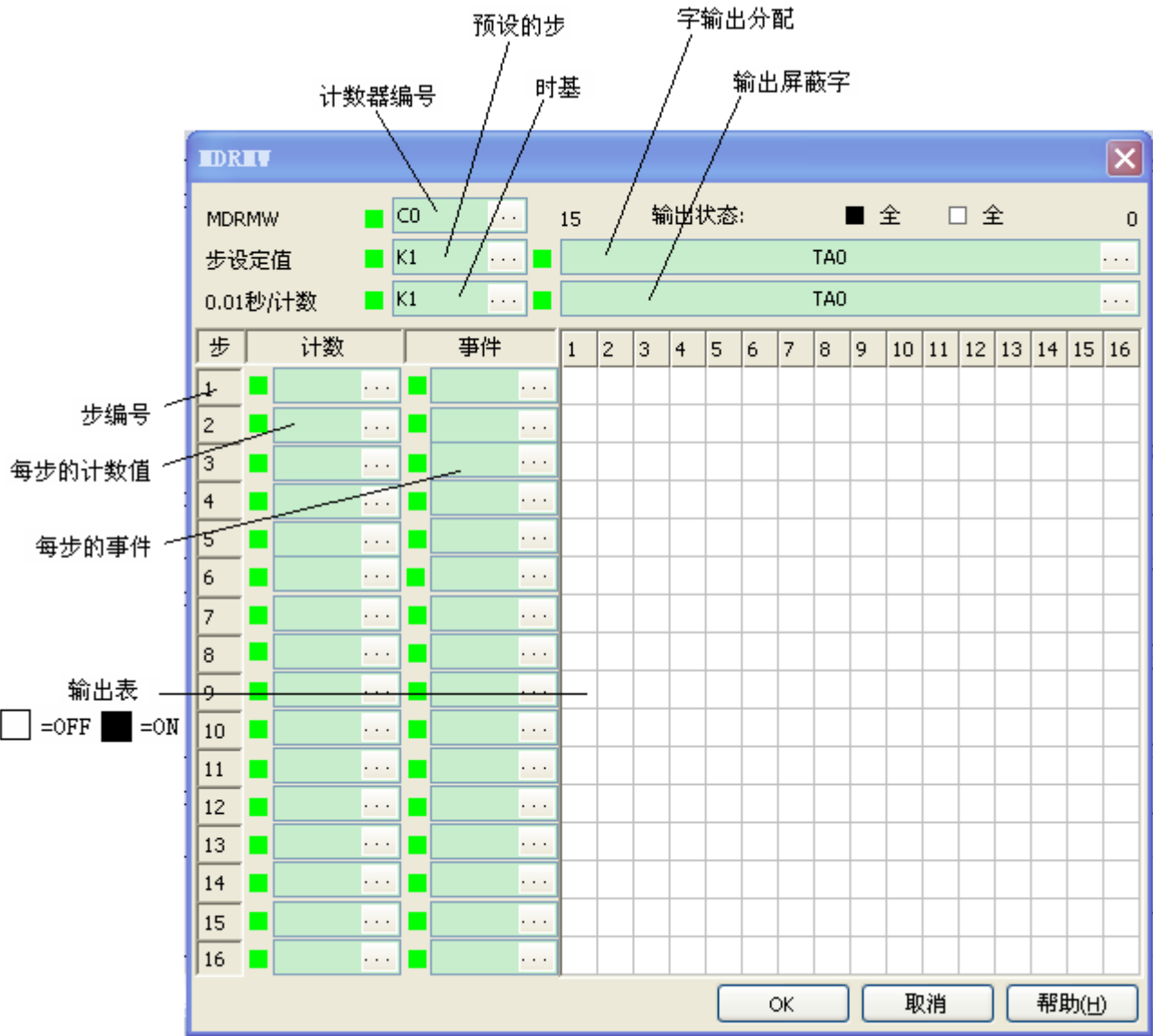
OK

取消

帮助(H)

25.4 带掩膜事件驱动型凸轮控制指令（MDRMW）（字输出）

MDRMW 指令与 MDRMD 相比，输出的是一个字，而不是离散点。它根据前面讲的凸轮运行基本规则运行。



MDRMW 指令有 16 步和 16 个输出，每步的凸轮输出同一个输出屏蔽字位逻辑与。输出屏蔽字区域指定了 16 个输出屏蔽字的首个屏蔽字地址。最终的输出在“字输出分配”区域中指定。基于时间和/或事件进行步转移，外部事件输入发生 OFF→ON 的转换时也会发生步转移。步转移的时间在每步的计数值中设定，事件为开关型触点。不使用的步和事件可以空着（缺省状态）。当启动输入为 ON 时，凸轮定时器被启用。只要当前步的事件为 ON，则当前步的定时器开始计时。当步计数值等于步的设定计数值时，凸轮转移到下一步。当最后一步完成或重置输入为 ON 时，此过程停止。CPU 在编程→运行的模式转换后或是重置输入为 ON 时，凸轮进入预设定的步。

凸轮指令使用四个计数器。可读取计数器的值来监控凸轮的状态。在程序中可随时向 CR(n+2) 中写入预设定的步。其他三个计数器仅做监控用。

凸轮参数	数据类型	范围
计数器编号	C	0-774
预设定的步	K	1-16
时基	K	0-99.99s
每步的计数值	K	0-9999
事件	I, Q, M, S, T, C, SP	所有（附录 1）
字输出	R	
输出屏蔽	R	

计数器编号	n 的范围	功能	计数器位功能
C(n)	0-774	每步的计数	C(n)=凸轮完成
C(n+1)	1-775	定时器值	C(n+1) 未使用
C(n+2)	2-776	预设定的步	C(n+2) 未使用
C(n+3)	3-777	当前步	C(n+3) 未使用

下面的程序例中，使用了步 1-11 和全部 16 个输出。输出屏蔽字从 R2000 开始。凸轮的最终输出在 R2001 中。输出表中输出同 R2000 中的屏蔽字逻辑与，生成的结果存放在 R2001 中。如果需要每次上电时凸轮所有输出为 OFF，可在第一个扫描周期向 R2000 中写入 0。程序可随时更新屏蔽字来启用或禁用凸轮的输出。

预设定的步是步 1。时基是 $K50 \times 0.01 = 0.5s$ 。因此，步 1 的停留时间是 $5 \times 0.5 = 2.5s$ 。注意步 1 仅基于时间进行步转移（事件是空着的）。最后一行程序凸轮完成最后一步时完成位 C14 将 Q0 置位。凸轮重置输入可将 C14 复位。



注意：上面的程序例中需要向 R2000-R2012 的 11 个掩码寄存器中写入常数，因为凸轮使用了 11 步。

第 26 章 IBOX 指令

26.1 16 位复制指令（MOVEW）（IB-200）

输入 MOVEW 指令将一个字直接或间接（通过指针）传送（复制）到一个存储器地址中。或者说，将一个十六进制常数，从一个地址直接或间接的传送（复制）到另一个地址。

MOVEW (IB-200)	
16位复制	
移动源：	TA0
移动到：	TA0

26.1.1 MOVEW 指令的参数

- 移动源：指定将被传送的字。
- 移动到：指定字被传送的目标地址。

参数		相应 CPU 范围
移动源	R, P, K	K0-FFFF;或见附录相应 CPU 功能存储器表-用户寄存器
移动到	R, P	见附录相应 CPU 功能存储器表-用户寄存器

26.1.2 MOVEW 指令举例

下面的例子中，MOVEW 指令在 M100 变为 ON 时，将存放在 R2000 中的 16 位数据传送（复制）到 R3000 中。

	MOVEW (IB-200)	
	16位复制	
	移动源：	R2000
	移动到：	R3000

26.2 32 位复制指令（MOVED）（IB-201）

输入 MOVED 指令将一个双字直接或间接（通过指针）传送（复制）到两个连续的存储器地址中。或者说，将一个双字十六进制常数，从一个地址直接或间接的传送（复制）到另一个地址。

MOVED (IB-201)	
32位复制	
移动源：	TA0 - TA1
移动到：	TA0 - TA1

26.2.1 MOVED 指令的参数

- 移动源：指定将被传送的双字。
- 移动到：指定双字被传送的目标地址。

参数		相应 CPU 范围
移动源	R, P, K	K0-FFFFFFFF;或见附录相应 CPU 功能存储器表-用户寄存器
移动到	R, P	见附录相应 CPU 功能存储器表-用户寄存器

26.2.2 MOVED 指令举例

下面的例子中，MOVEW 指令在 M100 变为 ON 时，将存放在 R2000 和 R2001 中的 32 位数据传送（复制）到 R3000 和 R3001 中。

	MOVED (IB-201)	
	32位复制	
	移动源：	R2000 - R2001
	移动到：	R3000 - R3001

26.3 反转开关指令（PONOFF）（IB-300）

输入 PONOFF 指令在其执行条件发生 OFF 到 ON 的转换时，改变一个输出线圈的状态。

需要一个额外的线圈位来处理数据状态信息。这个位不能再用于程序的其他地方。这也被称为“触发器电路”。

PONOFF (IB-300)	
反转开关	
开关量输入：	M0
开关量输出：	M0
内部状态：	M0

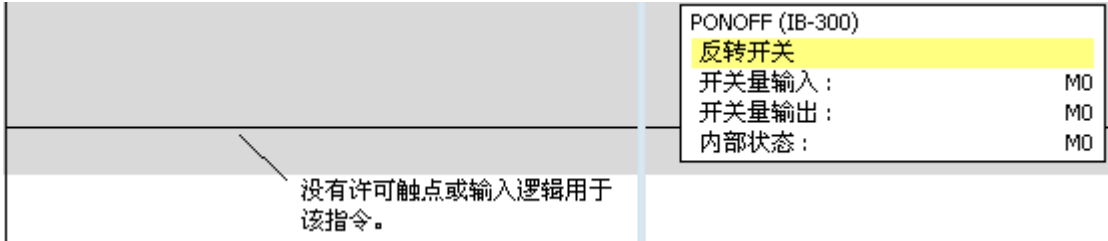
26.3.1 PONOFF 指令的参数

- 开关量输入：指定用于开关指定输出的输入位。
- 开关量输出：指定被开关的输出。
- 内部状态：指定用于 PONOFF 指令的内部状态位。

参数		相应 CPU 范围
开关量输入	I, Q, M, S, T, C, GI, GQ, SP, B, P	见附录相应 CPU 功能存储器表
开关量输出	I, Q, M, GI, GQ, B	见附录相应 CPU 功能存储器表
内部状态	I, Q, M	见附录相应 CPU 功能存储器表

26.3.2 PONOFF 指令举例

下面的例子中，PONOFF 指令使用输入 M10 控制输出 M20 的 ON 和 OFF 状态。当 M10 被按下时，M20 被置为 ON；当 M10 再次被按下时，M20 被置为 OFF。M100 是此 IBox 指令的内部状态位。



注意：该 IBox 回路中没有许可触点或逻辑输入。

26.4 ON 延迟定时器指令（ONDTMR）（IB-301）

输入 ONDTMR 指令按照预先设定的定时器的值（最大 99.99 秒）来延迟打开指定的输出。

当 IBox 指令失去执行条件时，输出位将立即变为 OFF。指定长度的时间一到，输出将变为 OFF。如果在延迟时间结束之前此 IBox 指令失去执行条件，那么定时器将被复位，输出位将不会为 ON，因此在输出变为 ON 之前，必须要保证此 IBox 指令上有执行条件的时间要大于定时器的设定时间。

此 IBox 指令中使用了一个定时器，这个定时器不能再用于程序中的其他地方。

ONDTMR (IB-301)	
ON延迟定时器	
定时器号：	T0
ON延迟定时器(0.01秒)：	TA0
输出：	M0

26.4.1 ONDTMR 指令参数

- 定时器号：指定用于 ONDTMR 指令的定时器（HTMR）号。
- On 延迟定时器（0.01 秒）：指定 ONDTMR 指令得到执行条件后，输出位将延长多久的时间把输出置为 ON（最长 99.99s）。
- 输出：指定被延迟变为 ON 的输出位。

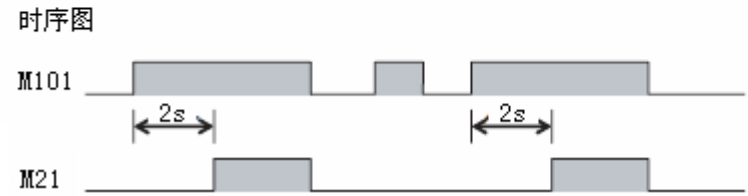
参数		相应 CPU 范围
定时器号	T	T0-377
On 延迟定时器（0.01 秒）	R, K	K0-9999；或见附录相应 CPU 功能存储器表-用户寄存器
输出	I, Q, M, GI, GQ, B	见附录相应 CPU 功能存储器表

26.4.2 ONDTMR 指令举例

下面的例子中，ONDTMR 指令用于延迟输出 M21 变为 ON 的时间。定时器 T1 被设置为 2 秒，即为延迟的时间。

当 M101 变为 ON 时，M21 延迟 2 秒后变为 ON。当 M101 变为 OFF 时，M21 立即变为 OFF。

	ONDTMR (IB-301)
	ON延迟定时器
	定时器号：T1
	ON延迟定时器(0.01秒)：K200
	输出：M21



26.5 OFF 延迟定时器指令（OFFDTMR）（IB-302）

输入 OFFDTMR 指令按照预先设定的定时器的值（最大 99.99 秒）来延迟关闭指定的输出。

此 IBox 指令满足执行条件时，输出位将立即变为 ON；此时，如果 IBox 指令失去执行条件，输出位将保持 ON 指定长度的时间。指定长度的时间一到，输出将变为 OFF。如果在延迟时间结束之前此 IBox 指令的执行条件再次得到满足，那么定时器将被复位，输出位将继续为 ON，因此在输出变为 OFF 之前，必须要保证此 IBox 指令执行条件不具备的时间大于定时器的设定时间。

此 IBox 指令中使用了一个定时器，这个定时器不能再用于程序中的其他地方。

OFFDTMR (IB-302)	
OFF延迟定时器	
定时器号：	T0
OFF延迟定时器(0.01秒)：	TA0
输出：	M0

26.5.1 OFFDTMR 指令参数

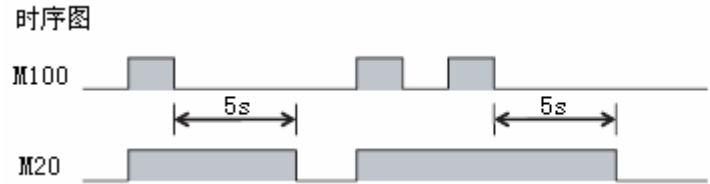
- 定时器号：指定用于 OFFDTMR 指令的定时器（HTMR）号。
- OFF 延迟定时器（0.01 秒）：指定 OFFDTMR 指令失去执行条件后，输出位将持续 ON 的时间（最长 99.99s）。
- 输出：指定被延迟变为 OFF 的输出位。

参数		相应 CPU 范围
定时器号	T	T0-377
OFF 延迟定时器（0.01 秒）	R, K	K0-9999；或见附录相应 CPU 功能存储器表-用户寄存器
输出	I, Q, M, GI, GQ, B	见附录相应 CPU 功能存储器表

26.5.2 OFFDTMR 指令举例

下面的例子中，OFFDTMR 指令用于延迟输出 M20 变为 OFF 的时间。定时器 T2 被设置为 5 秒，即为延迟的时间。

当 M100 变为 ON 时，M20 立即变为 ON。当 M100 变为 OFF 时，M20 将继续 ON 5 秒的时间，然后变为 OFF。



26.6 1 周期输出指令（ONESHOT）（IB-303）

输入 ONESHOT 指令在执行条件发生 OFF 到 ON 的转换时，将输出位置 ON 一个扫描周期的时间。此 IBox 指令同 PD 指令功能相同，仅名字不一样。

ONESHOT (IB-303)

1周期输出

开关量输出：M0

26.6.1 ONESHOT 指令的参数

- 开关量输出：指定将被置 ON 一个扫描周期的输出位。

参数	相应 CPU 范围
开关量输出 I, Q, M	见附录相应 CPU 功能存储器表

26.6.2 ONESHOT 指令举例

下面的例子中，ONESHOT 指令用于在 M0 发生一个 OFF 到 ON 的转换后，将 M100 置 ON 一个扫描周期的时间。要想 ONESHOT 指令执行，输入逻辑上必须发生一个 OFF 到 ON 的转换。

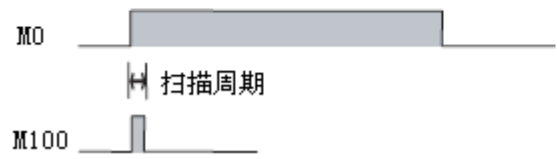


ONESHOT (IB-303)

1周期输出

开关量输出：M100

时序图



26.7 高/低报警-BIN 指令 (HILOALB) (IB-401)

输入 HILOALB 指令监控一个存放在 R 寄存器中的二进制（十进制）值，并设置四个可能的报警状态，高-高，高，低，低-低。输入的报警值可以是 0-65535 的十进制常数，也可以是存放 BIN 值的 R 寄存器。

输入的报警值必须有效，要满足条件 HH(高-高) $\geq$ H(高) $\geq$ L(低) $\geq$ LL(低低)。注意，当 HH 或 LL 条件满足时，H 或 L 的条件也满足。这就意味着如果只需要一个 H 报警，则可以将 HH 和 H 的报警值和报警输出位设置成一样。还要注意，边界条件是包括在内的，也就是说，如果低限是 K50，低-低限是 K10，那么当监控值等于 10 时，低报警和低-低报警输出位都为 ON。如果此指令没有被执行，则不管监控值是多少，所有的报警输出位都为 OFF。

HILOALB (IB-401)	
高/低报警-BIN	
监测值(BIN) :	TA0
高-高限制 :	TA0
高-高报警 :	MO
高限制 :	TA0
高报警 :	MO
低限制 :	TA0
低报警 :	MO
低-低限制 :	TA0
低-低报警 :	MO

26.7.1 HILOALB 指令的参数

- 监测值 (BIN) : 指定存放被监控 BIN 值的 R 寄存器地址。
- 高-高限制: 高-高限制值，可以是常数也可以是 R 寄存器地址。
- 高-高报警: 高-高报警输出位，到达高-高报警值时 ON。
- 高限制: 高限制值，可以是常数也可以是 R 寄存器地址。
- 高报警: 高报警输出位，到达高报警值时 ON。
- 低限制: 低限制值，可以是常数也可以是 R 寄存器地址。
- 低报警: 低报警输出位，到达低报警值时 ON。
- 低-低限制: 低-低限制值，可以是常数也可以是 R 寄存器地址。
- 低-低报警: 低-低报警输出位，到达低-低报警时 ON。

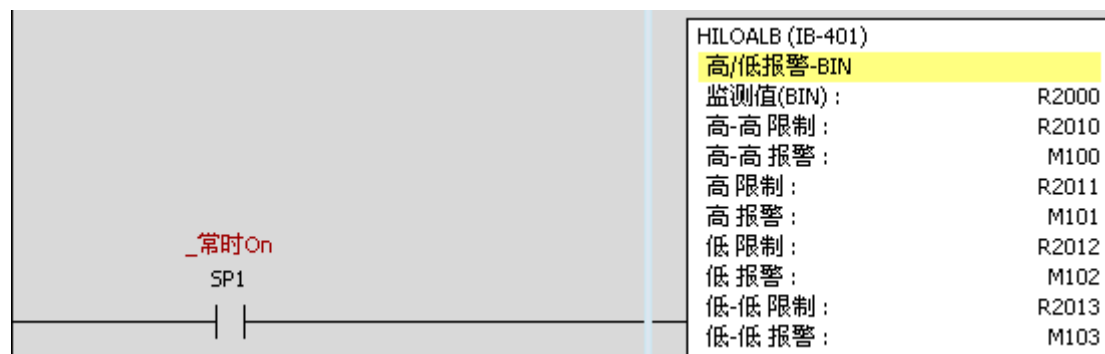
参数	相应 CPU 范围
监测值 (BIN) R	见附录相应 CPU 功能存储器表-用户寄存器
高-高限制 R, K	K0-65535; 或见附录相应 CPU 功能存储器表-用户寄存器
高-高报警 I, Q, M, GI, GQ, B	见附录相应 CPU 功能存储器表
高限制 R, K	K0-65535; 或见附录相应 CPU 功能存储器表-用户寄存器
高报警 I, Q, M, GI, GQ, B	见附录相应 CPU 功能存储器表
低限制 R, K	K0-65535; 或见附录相应 CPU 功能存储器表-用户寄存器
低报警 I, Q, M, GI, GQ, B	见附录相应 CPU 功能存储器表
低-低限制 R, K	K0-65535; 或见附录相应 CPU 功能存储器表-用户寄存器
低-低报警 I, Q, M, GI, GQ, B	见附录相应 CPU 功能存储器表

26.7.2 HILOALB 指令举例

下面的例子中，HILOALB 指令用于监控一个存放在 R2000 中的二进制值。如果这个二进制值到达或超过存放在 R2011 中的高限制值，M101 将被置为 ON。如果这个值继续增加到达或超过存放在 R2010 中的高-高限制值，M100 将被置为 ON。此时，两个报警输出位皆为 ON。如果只需要一个高报警，则可以将高-高和高的限制值和报警输出位设置成一样。

如果这个二进制值到达或低于存放在 R2012 中的低限制值，M102 将被置为 ON。如果这个值继续减少到达或低于存放在 R2013 中的低-低限制值，M103 将被置于 ON。此时，两个报警输出位皆为 ON。如果只需要一个低报警，则可以将低-低和低的限制值和报警输出位设置

成一样。



26.8 过滤超时时间 (BIN) 指令 (FILTERB) (IB-402)

输入 FILTERB 指令以规定的时间间隔对原始数据值进行首次过滤。

计算算式是：

$$\text{New} = \text{Old} + [(\text{Raw} - \text{Old}) / \text{FDC}]$$

New: 新的过滤值

Old: 老的过滤值

Raw: 原值

FDC: 过滤因子

过滤因子是一个整数，取值范围是 K1-K100，如果过滤因子是 K1，则不过滤。

过滤计算完成的速率以 0.01 秒为单位进行设定。注意，本指令中嵌入使用的定时器指令不可再用于程序的其它地方，因为它是控制过滤计算开始的许可条件。如果定时器为 OFF，过滤值将不更新。在模式由编程转换成运行后的首个扫描周期，过滤值被初始化为 0。

FILTERB (IB-402)	
过滤超时时间(BIN)	
过滤周期定时器：	T0
过滤周期时间(0.01秒)：	K0
原值(BIN)：	TA0
过滤因子(1-100)：	K1
过滤值(BIN)：	TA0

26.8.1 FILTERB 指令的参数

- 过滤周期定时器：指定用于 FILTERB 指令的定时器。
- 过滤周期时间（0.01 秒）：指定计算完成的速率。
- 原值（BIN）：指定二进制格式的原始未过滤值存放的 R 寄存器地址。
- 过滤因子（1-100）：此常数用于控制过滤效果，较大的值将增加过滤的平滑效果，如果值是 1 将不过滤。
- 过滤值（BIN）：指定过滤完成的数据存放的 R 寄存器地址。

参数		相应 CPU 范围
过滤周期定时器	T	T0-377
过滤周期时间（0.01 秒）	K	K0-9999
原值（BIN）	R	见附录相应 CPU 功能存储器表-用户寄存器
过滤因子（1-100）	K	K1-K100
过滤值（BIN）	R	见附录相应 CPU 功能存储器表-用户寄存器

26.8.2 FILTERB 指令举例

下面的例子中，FILTERB 指令用于过滤存放于 R2000 中的二进制数据。定时器 T1 设定是 0.5s，是过滤器计算执行的速率。过滤因子为 3。较大的过滤因子可增加过滤的平滑效果，1 表示不过滤。过滤后的值存放于 R2100 中。

_常时On

SP1

FILTERB (IB-402)	
过滤超时时间(BIN)	
过滤周期时间：	T1
过滤周期时间(0.01秒)：	K50
原值(BIN)：	R2000
过滤因子(1-100)：	K3
过滤值(BIN)：	R2100

26.9 12 位模拟量二进制变换指令（ANSCLB）（IB-403）

输入 ANSCLB 指令将一个 12 位二进制模拟量原始值（0-4095 BCD）换算成二进制（十进制）格式的工程量值。

在此指令中可以指定工程量的上限值（模拟量值原始值上限是 4095 时）和下限值（模拟量原始值下限是 0 时），也可以指定存放换算完成的工程量值的寄存器地址。工程量值的数据格式是二进制，范围是 0-65535（如果模拟量原始值是 BCD 格式，请使用 ANSCL 指令）。

注意，此 IBox 指令仅用于单极性无符号的模拟量原始值，不能用于双极性模拟量原始值或有符号二进制补码原码值。

ANSCLB (IB-403)	
12位模拟量二进制变换	
原值(12位二进制)：	TA0
上限值：	K0
下限值：	K0
工程值(二进制)：	TA0

26.9.1 ANSCLB 指令的参数

- 原值（12 位二进制）：指定单极无符号的模拟量原始值（12 位二进制=0-4095 十进制）存放的 R 寄存器地址。
- 上限值：当模拟量原始值上限是 4095 时指定工程量值的上限。
- 下限值：当模拟量原始值下限是 0 时指定工程量值的下限。
- 工程值（二进制）：指定换算好的工程量值存放的 R 寄存器地址。

参数		相应 CPU 范围
原值（12 位二进制）	R, P	见附录相应 CPU 功能存储器表-用户寄存器
上限值	K	K0-65535
下限值	K	K0-65535
工程值（二进制）	R, P	见附录相应 CPU 功能存储器表-用户寄存器

26.9.2 ANSCLB 指令举例

下面的例子中，ANSCLB 指令用于换算 R2000 中的原始值（0-4095 二进制）。工程单位值范围是 0-1000。换算完成的工程单位值以二进制格式存放于 R2100 中。

	ANSCLB (IB-403)
	12位模拟量二进制变换
	原值(12位二进制)：R2000
	上限值：K1000
	下限值：K0
	工程值(二进制)：R2100

26.10 高/低报警设定（BCD）指令（HILOAL）（IB-421）

输入 HILOAL 指令监控一个存放在 R 寄存器中的 BCD 值，并设置四个可能的报警状态，高-高，高，低，低-低。输入的限制值可以是 0-9999 的 BCD 值，也可以是存放 BCD 值的 R 寄存器。

输入的限制值必须有效，要满足条件 HH(高-高) $\geq$ H(高) $>$ L(低) $\geq$ LL(低低)。注意，当 HH 或 LL 条件满足时，H 或 L 的条件也满足。这就意味着如果只需要一个 H 报警，则可以将 HH 和 H 的限制值和报警输出位设置成一样。还要注意，边界条件是包括在内的，也就是说，如果低限是 K50，低-低限是 K10，那么当监控值等于 10 时，低报警和低-低报警输出位都为 ON。如果此指令没有被接通，则不管监控值是多少，所有的报警输出位都为 OFF。

HILOAL (IB-421)	
高/低报警设定(BCD)	
监测值(BCD) :	TA0
高-高限制 :	TA0
高-高报警 :	MO
高限制 :	TA0
高报警 :	MO
低限制 :	TA0
低报警 :	MO
低-低限制 :	TA0
低-低报警 :	MO

26.10.1 HILOAL 指令的参数

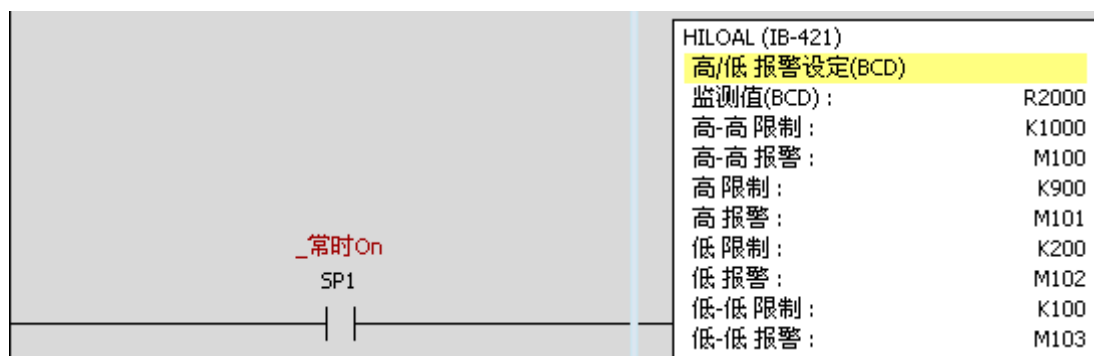
- 监测值（BCD）：指定存放被监控 BCD 值的 R 寄存器地址。
- 高-高限制：高-高限制值，可以是常数也可以是 R 寄存器地址。
- 高-高报警：高-高报警输出位，到达高-高报警值时 ON。
- 高限制：高限制值，可以是常数也可以是 R 寄存器地址。
- 高报警：高报警输出位，到达高报警值时 ON。
- 低限制：低限制值，可以是常数也可以是 R 寄存器地址。
- 低报警：低报警输出位，到达低报警值时 ON。
- 低-低限制：低-低限制值，可以是常数也可以是 R 寄存器地址。
- 低-低报警：低-低报警输出位，到达低-低报警时 ON。

参数	相应 CPU 范围
监测值（BCD） R	见附录相应 CPU 功能存储器表-用户寄存器
高-高限制 R, K	K0-9999; 或见附录相应 CPU 功能存储器表-用户寄存器
高-高报警 I, Q, M, GI, GQ, B	见附录相应 CPU 功能存储器表
高限制 R, K	K0-9999; 或见附录相应 CPU 功能存储器表-用户寄存器
高报警 I, Q, M, GI, GQ, B	见附录相应 CPU 功能存储器表
低限制 R, K	K0-9999; 或见附录相应 CPU 功能存储器表-用户寄存器
低报警 I, Q, M, GI, GQ, B	见附录相应 CPU 功能存储器表
低-低限制 R, K	K0-9999; 或见附录相应 CPU 功能存储器表-用户寄存器
低-低报警 I, Q, M, GI, GQ, B	见附录相应 CPU 功能存储器表

26.10.2 HILOAL 指令举例

下面的例子中，HILOAL 指令用于监控一个存放在 R2000 中的 BCD 值。如果这个 BCD 值到达或超过高限制值 K900，M101 将被置为 ON。如果这个值继续增加到达或超过高-高限制值 K1000，M100 将被置为 ON。此时，两个报警输出位皆为 ON。如果只需要一个高报警，则可以将高-高和高的限制值和报警输出位设置成一样。

如果这个 BCD 值到达或低于低限制值 K200，M102 将被置为 ON。如果这个值继续减少到达或低于低-低限制值 K100，M103 将被置于 ON。此时，两个报警输出位皆为 ON。如果只需要一个低报警，则可以将低-低和低的限制值和报警输出位设置成一样。



26.11 过滤超时时间（BCD）指令（FILTER）（IB-422）

输入 FILTER 指令以规定的时间间隔对原始数据值进行首次过滤。

计算算式是：

$$\text{New} = \text{Old} + [(\text{Raw} - \text{Old}) / \text{FDC}]$$

New：新的过滤值

Old：老的过滤值

Raw：原值

FDC：过滤因子

过滤因子是一个整数，取值范围是 K1-K100，如果过滤因子是 K1，则不过滤。

过滤计算完成的速率以 0.01 秒为单位进行设定。注意，本指令中嵌入使用的定时器指令不可再用于程序的其它地方，因为它是控制过滤计算开始的许可条件。如果定时器为 OFF，过滤值将不更新。在模式由编程转换成运行后的首个扫描周期，过滤值被初始化为 0。

FILTER (IB-422)	
过滤超时时间(BCD)	
过滤周期定时器：	T0
过滤周期时间(0.01秒)：	K0
原值(BCD)：	TA0
过滤因子(1-100)：	K1
过滤值(BCD)：	TA0

26.11.1 FILTER 指令的参数

- 过滤周期定时器：指定用于 FILTER 指令的定时器。
- 过滤周期时间（0.01 秒）：指定计算完成的速率。
- 原值（BCD）：指定 BCD 格式的原始未过滤值存放的 R 寄存器地址。
- 过滤因子（1-100）：此常数用于控制过滤效果，较大的值将增加过滤的平滑效果，如果值是 1 将不过滤。
- 过滤值（BCD）：指定过滤完成的 BCD 数据存放的 R 寄存器地址。

参数		相应 CPU 范围
过滤周期定时器	T	T0-377
过滤周期时间（0.01 秒）	K	K0-9999
原值（BCD）	R	见附录相应 CPU 功能存储器表-用户寄存器
过滤因子（1-100）	K	K1-K100
过滤值（BCD）	R	见附录相应 CPU 功能存储器表-用户寄存器

26.11.2 FILTER 指令举例

下面的例子中，FILTER 指令用于过滤存放于 R2000 中的 BCD 数据。定时器 T0 设定是 0.5s，是过滤器计算执行的速率。过滤因子为 2。较大的过滤因子可增加过滤的平滑效果，1 表示不过滤。过滤后的值存放于 R2100 中。

_ 常时 On

SP1

FILTER (IB-422)	
过滤超时时间(BCD)	
过滤周期定时器：	T0
过滤周期时间(0.01秒)：	K50
原值(BCD)：	R2000
过滤因子(1-100)：	K2
过滤值(BCD)：	R2100

26.12 12 位模拟量 BCD 转换指令（ANSCL）（IB-423）

输入

ANSCL 指令将一个 12 位 BCD 模拟量原始值（0-4095 BCD）换算成 BCD 格式的工程量值。

在此指令中可以指定工程量的上限值（模拟量原始值上限是 4095 时）和下限值（模拟量原始值下限是 0 时），也可以指定存放换算完成的工程量值的寄存器地址。工程量值的数据格式是 BCD，范围是 0-9999（如果模拟量原始值是二进制格式，请使用 ANSCLB 指令）。

注意，此 IBox 指令仅用于单极性无符号的模拟量原始值，不能用于双极性模拟量原始值或有符号的原码值。

ANSCL (IB-423)

12位模拟量BCD转换

原值(0-4095 BCD) :TA0

上限值 :K0

下限值 :K0

工程值(BCD) :TA0

26.12.1 ANSCL 指令的参数

- 原值（0-4095 BCD）:指定单极无符号的模拟量原始值（0-4095）存放的 R 寄存器地址。
- 上限值：当模拟量原始值上限是 4095 时指定工程量值的上限。
- 下限值：当模拟量原始值下限是 0 时指定工程量值的下限。
- 工程值（BCD）:指定换算好的工程量值存放的 R 寄存器地址。

参数		相应 CPU 范围
原值（0-4095 BCD）	R, P	见附录相应 CPU 功能存储器表-用户寄存器
上限值	K	K0-9999
下限值	K	K0-9999
工程值（BCD）	R, P	见附录相应 CPU 功能存储器表-用户寄存器

26.12.2 ANSCL 指令举例

下面的例子中，ANSCL 指令用于换算 R2000 中的原始值（0-4095 BCD）。工程量值范围是 0-100。换算完成的工程量值以 BCD 格式存放于 R2100 中。

常时On

SP1

ANSCL (IB-423)

12位模拟量BCD转换

原值(0-4095 BCD) :R2000

上限值 :K100

下限值 :K0

工程值(BCD) :R2100

26.13 BIN 算术运算指令（MATHBIN（IB-501）/MATHBIND（IB-551））

输入 MATHBIN/MATHBIND 指令用于计算复杂的数学表达式，和在 Visual Basic，Excel 或 C++中的计算类似，括号嵌套最多是 4 级。除了+ - * /，还可取模，左移（<<）和右移（>>），按位与（&）按位或（|）按位异或（^）以及一些 BIN 功能：BCD 变换（BCD），BIN 变换（BIN），译码（DECO），编码（ENCO），取反（INV），HEX 显示成 7 段译码（SEG），求和。

MATHBIN (IB-501)

BIN算术运算

结果(字): R3000

"R2000*K10"

MATHBIND (IB-551)

Math-Binary DWORD

结果(双字): R2000

"R3000*K10"

例如：((R2000+R2001)/(R2003-K100))*SUM(R3000&K001F)

每个使用的 R 寄存器中的数据必须是一个单字二进制数。对于 MATHBIND 指令，其中间值和结果都为 32 位。对于 MATHBIN 指令，计算中的值可以是 32 位，但是最终结果是 16 位 BIN 数，乘除运算就是典型的例子，例如 (R2000*K1000)/K4095，计算的中间值可能超过 65535，但是不超过 32 位，除去 4095 后，最终结果始终不超过 16 位。

BIN 函数可以使用 R 寄存器，但不可以使用表达式。例如：BIN(R2000)是允许的，可将 R2000 中的数据格式由 BCD 变换成二进制，但 BIN(R2000 + R3000)是不允许的。

此外，对于 MATHBIN 指令最终结果是 16 位二进制数（单字），可以采用 BCD 函数将其转换为 BCD 数。对于 MATHBIND 指令，最终结果是 32 位二进制数（双字）。

26.13.1 MATHBIN/MATHBIND 指令的参数

- 结果（字/双字）：指定数学表达式计算结果（BIN）存放的地址（MATHBIN 指令结果必须不超过 16 位；MATHBIND 指令结果为 32 位双字）。
- 表达式：列出要计算的数学表达式，数学表达式中用到的每一个 R 寄存器中的数据都必须是 BIN 格式。K 常数指定为 10 进制数。

参数	相应 CPU 范围
结果（字/双字）R	见附录一相应 CPU 功能存储器表-用户寄存器
表达式	文本

26.13.2 MATHBIN 指令举例

下面的例子中，当 M100 为 ON 时，MATHBIN 指令计算数学表达式 (R1200*K1000)/K4095，并将计算结果存放在 R2000 中。

M100

MATHBIN (IB-501)

BIN算术运算

结果(字): R2000

表达式: "(R1200*K1000)/K4095"

26. 14 BIN 数求和指令（SUMBIN）（IB-502）

输入 SUMBIN 指令对一组连续地址的 16 位二进制数求和，结果是一个 32 位二进制数。

指定要求和的寄存器组的起始和结束地址。
SUMBIN 指令执行时，将寄存器组中所有数据相加求和（指令前需添加允许触点。）。

SUMBIN 指令可用于计算平均值的第一步。

SUMBIN (IB-502)	
BIN数求和	
开始地址：	R2000
结束地址(包含)：	R2000
结果(双字 二进制)：	R2000 - R2001

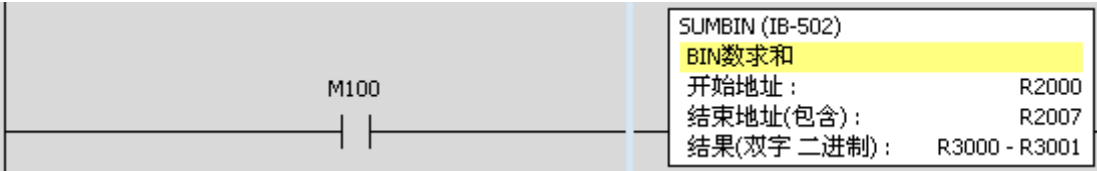
26. 14. 1 SUMBIN 指令的参数

- 开始地址：指定寄存器组的起始寄存器地址。
- 结束地址（含计算）：指定寄存器组的结束寄存器地址。
- 结果（双字 二进制）：指定求和结果的存放地址。

参数		相应 CPU 范围
开始地址	R	见附录相应 CPU 功能存储器表-用户寄存器
结束地址（含计算）	R	见附录相应 CPU 功能存储器表-用户寄存器
结果（双字 二进制）	R	见附录相应 CPU 功能存储器表-用户寄存器

26. 14. 2 SUMBIN 指令举例

下面的例子中，当 M100 为 ON 时，SUMBIN 指令将 R2000-R2007 中的二进制数求和，将结果以 32 位二进制格式存放于 R3000 和 R3001 中。



26. 15 BIN 平方指令 (SQUAREB) (IB-503)

 SQUAREB 指令对一个给定的 16 位二进制数求平方，结果保存为一个 32 位二进制数。

SQUAREB (IB-503)	
BIN平方	
值(字 二进制) :	TA0
结果(双字 二进制) :	R2000 - R2001

26. 15. 1 SQUAREB 指令的参数

- 值（字 二进制）:指定将被求平方的二进制数或其存放地址。
- 结果（双字 二进制）: 指定结果的存放地址。

参数		相应 CPU 范围
值（字 二进制）	R, P, K	K0-65535; 见附录相应 CPU 功能存储器表-用户寄存器
结果（双字 二进制）	R	见附录相应 CPU 功能存储器表-用户寄存器

26. 15. 2 SQUAREB 指令举例

下面的例子中，当 M100 为 ON 时，SQUAREB 指令对一个存放在 R2000 中的 16 位二进制数求取平方，结果是一个 32 位二进制数，存放在 R3000 和 R3001 中。

	SQUAREB (IB-503)					
	<table><tr><td colspan="2">BIN平方</td></tr><tr><td>值(字 二进制) :</td><td>R2000</td></tr><tr><td>结果(双字 二进制) :</td><td>R3000 - R3001</td></tr></table>	BIN平方		值(字 二进制) :	R2000	结果(双字 二进制) :
BIN平方						
值(字 二进制) :	R2000					
结果(双字 二进制) :	R3000 - R3001					

26.16 BCD 算术运算指令（MATHBCD (IB-521) /MATHBCDD (IB-550)

输入 MATHBCD/MATHBCDD 指令用于计算复杂的数学表达式，和在 Visual Basic, Excel 或 C++中的计算类似，括号嵌套最多是 4 级。除了+ - * /，还可取模（又名余数），按位与（&）按位或（|）按位异或（^）以及一些 BCD 功能：BCD 变换（BCD），BIN 变换（BIN），BCD 补码（BCDCPL），格雷码转换（GRAY），取反（INV），BCD/HEX 显示成 7 段译码（SEG）。

MATHBCD (IB-521)

BCD算术运算

结果(字): R2000

"R3000*R3010*K20"

MATHBCDD (IB-550)

Math-BCD DWORD

结果(双字): R2000

"R3000*R3010*K30"

例如：((R2000+R2001)/(R2003-K100))*GRAY(R3000&K001F)

每个使用的 R 寄存器中的数据必须是一个单字 BCD 数。计算中的值可以是 32 位，对于 MATHBCDD 指令的最终结果是 32 位。对于 MATHBCD 指令，只要最终结果是 16 位 BCD 数，这个结果就有效，乘除运算就是典型的例子，例如 (R2000*K1000)/K4095，计算的中间值可能超过 9999，但是不超过 32 位，除去 4095 后，最终结果始终不超过 16 位。

BCD 函数可以使用 R 寄存器，但不可以使用表达式。例如：BCD(R2000)是允许的，可将 R2000 中的数据格式由二进制变换成 BCD，但 BCD(R2000 + R3000)是不允许的。

MATHBCD 指令最终结果是 16 位 BCD 数，可以采用 BIN 函数将其转换为二进制数。

MATHBCDD 指令最终结果是 32 位 BCD 数。

26.16.1 MATHBCD/MATHBCDD 指令的参数

- 结果（字/双字）：指定数学表达式计算结果（BCD）存放的地址（MATHBCD 指令结果必须不超过 16 位；MATHBCDD 指令结果为一个双字）。
- 表达式：列出要计算的数学表达式，数学表达式中用到的每一个 R 寄存器中的数据都必须是 BCD 格式。

参数	相应 CPU 范围
结果(字/双字)	R 见附录一相应 CPU 功能存储器表-用户寄存器
表达式	文本

26.16.2 MATHBCD 指令举例

下面的例子中，当 M100 为 ON 时，MATHBCD 指令计算数学表达式 (R1200*K1000)/K4095，并将计算结果存放在 R2000 中。

M100

MATHBCD (IB-521)

BCD算术运算

结果(字): R2000

表达式: "(R1200*K1000)/K4095"

26.17 BCD 数求和指令（SUMBCD）（IB-522）

输入 SUMBCD 指令对一组连续地址的 4 位数 BCD 数求和，结果是一个 8 位数 BCD 数。

指定要求和的寄存器组的起始和结束地址。
SUMBCD 指令执行时，将寄存器组中所有数据相加求和（指令前需添加允许触点。）。

SUMBCD 指令可用于计算平均值的第一步。

SUMBCD (IB-522)	
BCD数求和	
开始地址：	R2000
结束地址(包含)：	R2000
结果(双字BCD)：	R2000 - R2001

26.17.1 SUMBCD 指令的参数

- 开始地址：指定寄存器组的起始寄存器地址。
- 结束地址（包含）：指定寄存器组的结束寄存器地址。
- 结果（双字 BCD）：指定求和结果的存放地址。

参数		相应 CPU 范围
开始地址	R	见附录相应 CPU 功能存储器表-用户寄存器
结束地址（包含）	R	见附录相应 CPU 功能存储器表-用户寄存器
结果（双字 BCD）	R	见附录相应 CPU 功能存储器表-用户寄存器

26.17.2 SUMBCD 指令举例

下面的例子中，当 M100 为 ON 时，SUMBCD 指令将 R2000-R2007 中的 BCD 数求和，将结果以 8 位数 BCD 格式存放于 R3000 和 R3001 中。

	SUMBCD (IB-522)
	BCD数求和
	开始地址：R2000
	结束地址(包含)：R2007
	结果(双字BCD)：R3000 - R3001

26.18 BCD 平方指令 (SQUARE) (IB-523)

输入 SQUARE 指令对一个给定的 4 位 BCD 数求平方，结果保存为一个 8 位 BCD 数。

SQUARE (IB-523)	
BCD平方	
值(字 BCD) :	TA0
结果(双字BCD) :	R2000 - R2001

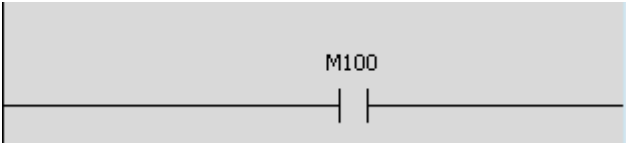
26.18.1 SQUARE 指令的参数

- 值（字 BCD）:指定将被求平方的 BCD 常数或其存放地址。
- 结果（双字 BCD）: 指定结果的存放地址。

参数		相应 CPU 范围
值（字 BCD）	R, P, K	K0-9999; 见附录相应 CPU 功能存储器表-用户寄存器
结果（双字 BCD）	R	见附录相应 CPU 功能存储器表-用户寄存器

26.18.2 SQUARE 指令举例

下面的例子中，当 M100 为 ON 时，SQUARE 指令对一个存放在 R2000 中的 4 位 BCD 数求平方，结果是一个 8 位 BCD 数，存放在 R3000 和 R3001 中。

	SQUARE (IB-523)
	BCD平方 值(字 BCD) : R2000 结果(双字BCD) : R3000 - R3001

26.19 实数算术运算指令（MATHR）（IB-541）

输入 MATHR 指令用于计算复杂的数学表达式，和在 Visual Basic, Excel 或 C++ 中的计算类似，括号嵌套最多是 4 级。除了 + - * /，还可以按位与（&）按位或（|）按位异或（^）。DL260/265 还支持几种实数功能：反余弦（RACOS），反正弦（RASIN），反正切（RATAN），余弦（RCOS），正弦（RSIN），正切（RATAN），度变换（RDEG），弧度变换（RRAD），取反（INV），HEX 显示成 7 段译码（SEG），平方根（SQRT）。

MATHR (IB-541)

实数算术运算

结果(双字): R2000

表达式:
"(R1200*F10.5)/F2.7"

例如：((R2000+R2002) / (R2004-F2.5))*RSIN (RRAD (R3000/F10.0))
每个使用的 R 寄存器中的数据必须是一个双字实数。

26.19.1 MATHR 指令的参数

- 结果（双字）：指定数学表达式计算结果（实数）存放的地址（结果必须是双字实数）。
- 表达式：列出要计算的数学表达式，数学表达式中用到的每一个 R 寄存器中的数据都必须是实数格式。

参数		相应 CPU 范围
结果（双字）	R	见附录相应 CPU 功能存储器表-用户寄存器
表达式		文本

26.19.2 MATHR 指令举例

下面的例子中，当 M100 为 ON 时，MATHR 指令计算数学表达式 (R1200*F10.5) /F2.7，并将计算结果存放在 R2000 和 R2001 中。

M100

MATHR (IB-541)

实数算术运算

结果(双字): R2000

表达式:
"(R1200*F10.5)/F2.7"

26. 20 实数求和指令（SUMR）（IB-542）

输入 SUMR 指令对一组连续地址的实数求和，结果是一个实数。

指定要求和的寄存器组的起始和结束地址。

注意，实数是双字，每个实数占用两个连续的 R 寄存器，因此，实数的个数是寄存器个数的一半。

注意，寄存器组的结束地址可以是最后一个实数占用的两个寄存器的任何一个地址。例如，对存放在 R2000-R2007（R2000、R2002、R2004 和 R2006）中的 4 个实数求和，参数中的结束寄存器地址可以是 R2006，也可以是 R2007。

SUMR 指令执行时，将寄存器组中所有数据相加求和（指令前需添加允许触点。）。

SUMR 指令可用于计算平均值的第一步。

SUMR (IB-542)	
实数求和	
开始地址(双字):	R2000 - R2001
结束地址(包含 双字):	R2000
结果(双字):	R2000 - R2001

26. 20. 1 SUMR 指令的参数

- 开始地址（双字）：指定寄存器组的起始寄存器地址。
- 结束地址（包含 双字）：指定寄存器组的结束寄存器地址。
- 结果（双字）：指定求和结果的存放地址。

参数		相应 CPU 范围
开始地址（双字）	R	见附录相应 CPU 功能存储器表-用户寄存器
结束地址（包含 双字）	R	见附录相应 CPU 功能存储器表-用户寄存器
结果（双字）	R	见附录相应 CPU 功能存储器表-用户寄存器

26. 20. 2 SUMR 指令举例

下面的例子中，当 M100 为 ON 时，SUMR 指令将 R2000-R2007 中的 32 位实数求和，将结果以 32 位实数格式存放于 R3000 和 R3001 中。

	SUMR (IB-542)	
	实数求和	
	开始地址(双字):	R2000 - R2001
	结束地址(包含 双字):	R2007
	结果(双字):	R3000 - R3001

26.21 实数平方指令（SQUARER）（IB-543）

输入 SQUARER 指令对一个给定的双字实数求平方，结果保存为一个双字实数。

SQUARER (IB-543)	
实数平方	
值(双字 实数)：	TA0 - TA1
结果(双字 实数)：	R2000 - R2001

26.21.1 SQUARER 指令的参数

- 值（双字 数）：指定将被求平方的双字实数常数或其存放地址。
- 结果（双字 数）：指定结果的存放地址。

参数	相应 CPU 范围
值（双字 数） R, P, F	实数常数；见附录相应 CPU 功能存储器表-用户寄存器
结果（双字 数） R	见附录相应 CPU 功能存储器表-用户寄存器

26.21.2 SQUARER 指令举例

下面的例子中，当 M100 为 ON 时，SQUARER 指令对一个存放在 R2000 和 R2001 中的实数求取平方，结果是实数，存放在 R3000 和 R3001 中。

	SQUARER (IB-543)
	实数平方
	值(双字 实数)：R2000 - R2001
	结果(双字 实数)：R3000 - R3001

26.22 16 位 BCD-带小数点实数转换指令（BCDTOR）（IB-560）

输入 BCDTOR 指令将一个给定的 16 位 BCD 数转换成一个实数，小数位数是 K0-K4。

例如，将 K1234 转换，如果小数位数是 K1，则转换结果是 R123.4。

BCDTOR (IB-560)

16位BCD-带小数点实数转换

值(字 BCD):TA0

小数点位数:K0

结果(双字 实数):R2000 - R2001

26.22.1 BCDTOR 指令的参数

- 值（字 BCD）：指定将要被转换为实数的字或常数。
- 小数点位数：指定转换结果的小数位数。
- 结果（双字 实数）：指定存放转换结果的寄存器地址。

参数	相应 CPU 范围
值（字 BCD）R, P, K	K0-9999;或见附录相应 CPU 功能存储器表-用户寄存器
小数点位数K	K0-4
结果（双字 实数）R	见附录相应 CPU 功能存储器表-用户寄存器

26.22.2 BCDTOR 指令举例

下面的例子中，BCDTOR 指令用于将存放在 R2000 中的一个 16 位数据从 BCD 数转换为一个 32 位实数，并将其存放在 R3000 和 R3001 中。

K2 表示转换生成的实数有 2 位小数。

M100

BCDTOR (IB-560)

16位BCD-带小数点实数转换

值(字 BCD):R2000

小数点位数:K2

结果(双字 实数):R3000 - R3001

26.23 实数-16 位 BCD 转换指令（RTOBCD）（IB-561）

输入 RTOBCD 指令将一个给定的实数转换为一个 16 位 BCD 数，根据结果中隐含的小数位数（K0-K4）进行转换，并完成四舍五入。

例如，将 F56.74 进行转换，小数位数为 K1，转换结果是 BCD 数 567。如果小数位数为 K0，则转换结果是 BCD 数 57（注意进行了四舍五入）。如果实数是负数，则结果是正的，是它的绝对值。

RTOBCD (IB-561)	
实数-16位BCD转换	
值(双字 实数):	TA0 - TA1
小数点位数:	K0
结果(字 BCD):	R2000

26.23.1 RTOBCD 指令的参数

- 值（双字 实数）：指定要被转换的实数常数或其存放地址。
- 小数点位数：指定转换结果中隐含的小数位数。
- 结果（字 BCD）：指定转换结果的存放地址。

参数		相应 CPU 范围
值（双字 实数）	R, P, F	F; 见附录相应 CPU 功能存储器表-用户寄存器
小数点位数	K	K0-4
结果（字 BCD）	R	见附录相应 CPU 功能存储器表-用户寄存器

26.23.2 RTOBCD 指令举例

下面的例子中，当 M100 为 ON 时，RTOBCD 指令将一个存放在 R3000 和 R3001 中的 32 位实数转换为一个 16 位 BCD 数，将结果存放于 R2000 中。

K2 表示结果中隐含的小数位数是 2 位。

	RTOBCD (IB-561)	
	实数-16位BCD转换	
	值(双字 实数):	R3000 - R3001
	小数点位数:	K2
	结果(字 BCD):	R2000

26. 24 32 位 BCD-带小数点实数转换指令（BCDTORD）（IB-562）

输入 BCDTORD 指令将一个给定的 32 位 BCD 数转换成一个实数，小数位数是 K0-K8。

例如，将 K12345678 转换，如果小数位数是 K5，则转换结果是 R123. 45678。

BCDTORD (IB-562)	
32位BCD-带小数点实数转换	
值(双字 BCD)：	TA0 - TA1
小数点位数：	K0
结果(双字 实数)：	R2000 - R2001

26. 24. 1 BCDTORD 指令的参数

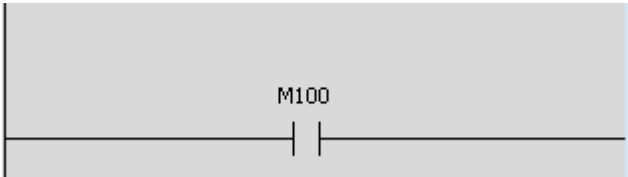
- 值（双字 BCD）：指定将要被转换为实数的双字或常数。
- 小数点位数：指定转换结果的小数位数。
- 结果（双字 实数）：指定存放转换结果的寄存器地址。

参数		相应 CPU 范围
值（双字 BCD）	R, P, K	K0-99999999;或见附录相应 CPU 功能存储器表-用户寄存器
小数点位数	K	K0-8
结果（双字 实数）	R	见附录相应 CPU 功能存储器表-用户寄存器

26. 24. 2 BCDTORD 指令举例

下面的例子中，BCDTORD 指令用于将存放在 R2000 和 R2001 中的一个 32 位数据从 32 位数 BCD 数转换为一个 32 位实数，并将其存放在 R3000 和 R3001 中。

K2 表示转换生成的实数有 2 位小数。

	BCDTORD (IB-562)
	32位BCD-带小数点实数转换
	值(双字 BCD)：R2000 - R2001
	小数点位数：K2
	结果(双字 实数)：R2000 - R2001

26. 25 实数-32 位 BCD 转换指令（RTOBCDD）（IB-563）

输入 RTOBCDD 指令将一个给定的实数转换为一个 32 位 BCD 数，根据结果中隐含的小数位数（K0-K8）进行转换，并完成四舍五入。

例如，将 F38156.74 进行转换，小数位数为 K1，转换结果是 BCD 数 381567。如果小数位数为 K0，则转换结果是 BCD 数 38157（注意进行了四舍五入）。

如果实数是负数，则结果是正的，是它的绝对值。

RTOBCDD (IB-563)	
实数-32位BCD转换	
值(双字 实数):	TA0 - TA1
小数点位数:	K0
结果(双字BCD):	R2000 - R2001

26. 25. 1 RTOBCDD 指令的参数

- 值（双字 实数）：指定要被转换的实数常数或其存放地址。
- 小数点位数：指定转换结果中隐含的小数位数。
- 结果（双字 BCD）：指定转换结果的存放地址。

参数		相应 CPU 范围
值（双字 实数）	R, P, F	实数常数；见附录相应 CPU 功能存储器表-用户寄存器
小数点位数	K	K0-8
结果（双字 BCD）	R	见附录相应 CPU 功能存储器表-用户寄存器

26. 25. 2 RTOBCDD 指令举例

下面的例子中，当 M100 为 ON 时，RTOBCDD 指令将一个存放在 R3000 和 R3001 中的 32 位实数转换为一个 32 位 BCD 数，将结果存放于 R2000 和 R2001 中。

K2 表示结果中隐含的小数位数是 2 位。

	RTOBCDD (IB-563)	
	实数-32位BCD转换	
	值(双字 实数):	R3000 - R3001
	小数点位数:	K2
	结果(双字BCD):	R2000 - R2001

26.26 CRC 校验码运算指令（CRCXOR）（IB-552）

输入 CRCXOR 指令把一组给定寄存器中的数据，按字节单位（最大 256 字节）进行异或（XOR）运算，运算结果存放到指定寄存器（结果字）的低位字节中，结果字高位字节为 0。

当指定的字节数为奇数时，指定进行异或运算的寄存器组的最后一个寄存器，仅对其低位字节进行异或运算。

CRCXOR (IB-552)	
CRCXOR	
Start Register Number :	R2000
Number of Bytes (1-256) :	TA10
结果(字) :	R3000

26.26.1 CRCXOR 指令的参数

- 开始寄存器：指定要进行异或运算寄存器组的开始寄存器地址。
- 异或字节数：指定进行异或运算的字节数，最大 256 字节。
- 结果（字）：指定异或运算结果存放地址。

参数		相应 CPU 范围
开始寄存器	R, P	见附录一相应 CPU 功能存储器表-用户寄存器
异或字节数	R, K	K1-256, R 见附录一相应 CPU 功能存储器表-用户寄存器
结果（字）	R	见附录一相应 CPU 功能存储器表-用户寄存器

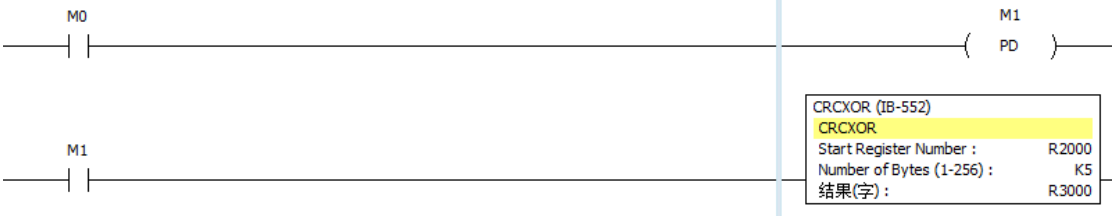
注意：设定指令的 3 个参数时，所使用的寄存器不要有重叠。

26.26.2 CRCXOR 指令举例

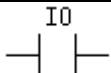
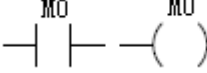
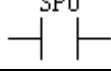
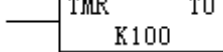
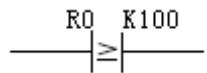
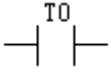
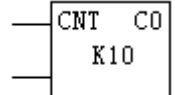
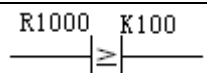
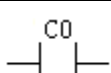
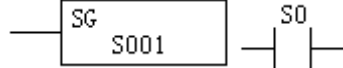
下面的例子中，每当 M0 接通一次，就对 R2000 开始的 3 个寄存器中的 5 个字节（R2002 仅对低位字节进行运算）进行异或运算，结果存放在 R3000 的低位字节，R3000 高位字节为 0。

Rn	High Byte	Low Byte
R2000	12h	34h
R2001	56h	78h
R2002	ABh	CDh
R3000	00h	结果字节

$R3000 = 34h (XOR) 12h (XOR) 78h (XOR) 56h (XOR) CDh$



附录 1 NK1 功能存储器表

功能存储器 类型	功能号 (八进制)	寄存器地址 (八进制)	数量 (十进制)	图标
输入	I0-I1777	R40400-R40477	1024	
输出	Q0-Q1777	R40500-R40577	1024	
内部线圈	M0-M3777	R40600-R40777	2048	
特殊线圈	SP0-SP1777	R41200-R41277	1024	
定时器	T0-T777	无	512	
定时器经过 值	无	R0-R777 P0-P777	512W	
定时器状态 位	T0-T777	R41100-R41137	32W	
计数器	C0-C777	无	512	
计数器经过 值	无	R1000-R1777 P1000-P1777	512W	
计数器状态 位	C0-C777	R41140-R41177	32W	
用户寄存器	无	R2000-R7377 P2000-P7377 R10000-R36777 P10000-P36777	14592W	没有特定的图标，同许多指令一起使用
特殊寄存器	无	R7400-R7777 P7400-P7777	256W	系统定义好的特殊用途寄存器
级	S0-S1777	R41000-R41077	1024	
特殊寄存器	无	R37000-R37777 P37000-P37777	512W	扩展槽中插入 AD/智能模块时使用，可指定停电保持
通讯 I/O	GI0-3777 GQ0-3777	R40000-R40177 R40200-R40377	2048 2048	Link 模块不使用的時候，可以用作内部线圈用

注：每段寄存器地址的最后一个寄存器不能作为 32 位指令的操作数。

附录 2 ASCII 转换表

DEC→HEX→ASCII 转换											
DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII
0	00	NUL	32	20	空格	64	40	@	96	60	
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	“	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	\$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	TAB	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

光洋电子(无锡)有限公司

Koyo ELECTRONICS (WUXI) CO., LTD.

地址：江苏省无锡市滨湖区建筑西路 599 号 1 栋 21 层

邮编：214072

电话：0510-85167888

传真：0510-85161393

[http: //www. koyoele. com. cn](http://www.koyoele.com.cn)

KEW-M2512D

2020 年 5 月