JMON PROGRAMMER'S UTILITIES

JMON PROGRAMMER'S UTILITIES

Below is a description of the use of all the JMON utilities routines. Before you can understand the operation of the utilities, I must explain the SOFTWARE STACK. The SOFTWARE STACK, as its name suggests, is a stack created by software. Its purpose is to store the address that JMON is currently viewing. Addresses may also be recovered from the SOFTWARE STACK on a last-on-first-off bases (just like the real stack). The utilities ROM contains numerous routines that make it possible for the programmer to manually follow through a program the same way the Z80 does. For example: If you are following through a program and you encounter a call you can go to the new address and view the sub-routine. When you encounter the RET instruction you can come back to where you called from.

Up to 16 addresses can be stored on the SOFTWARE STACK thus giving you 16 levels of nested calls. This will be enough for most requirements.

The SOFTWARE STACK has been located at 0FFF and works down to 0FE0. This is because JMON has used most of the first RAM page already. If this area conflicts with your program then the SOFTWARE STACK must not be used. It will not corrupt anything in this address range if it is not used. The utilities that you can safely use in this case are:

The code relocation routine, the block relocation routine, the byte insert or delete routines, search/replace routine and address jump.

All the other routines may effect your program.

Below is a discussion of each individual utility routine:

CODE RELOCATION ROUTINE:

ADDRESS, GO, 1

This very clever routine shifts a program from one spot in memory to another and changes all the absolute jumps and calls. Memory pointers are also altered if the memory pointers are loaded into any of the following registers: BC, DE or HL (sorry, not IX or IY) and point to a location between the start and end address of the program being relocated.

I.E. 01 xx xx , 11 xx xx or 21 xx xx where xx xx is an address between the start and end of the program being relocated.

In addition, to these direct loadings, any indirect loading of HL is also altered if it is in three byte format.

IE. 2A xx xx or 22 xx xx is altered if xx xx is an address that is between the start and end address, while ED 63 xx xx or ED 6B xx xx is not altered even though they are the same instructions as above.

The program MUST BE IN ITS CORRECT EXECUTING LOCATION BEFORE THIS ROUTINE IS USED.

Any reference to a location outside the start and end range, is not altered.

The variables for this routine are:

THE START and THE END of the program to shift, and the DESTINATION or the new start address.

These variables are loaded by using the PERIMETER HANDLER and the RELOCATION ROUTINE is executed from the PERIMETER HANDLER by hitting "GO".

BYTE DELETE and INSERT ROUTINES:

DELETE: - ADDRESS, GO, 2
INSERT: - ADDRESS, GO, 3.

The byte delete and insert routines removes or adds a byte from memory, and then alters ALL effected jumps, calls and address pointers as described above.

A two byte displacement can be added so that a routine that is not currently in its correct executing memory can be modified. This feature is useful if you wish to modify JMON (by relocating it to 1000H performing the changes with an offset as described below, and then replacing it a 0000 again).

The variables for this routine are:

The START and END ADDRESSES of the program to have a byte added or deleted, the OFFSET (if any, this valve is zeroed by default), and the TARGET ADDRESS POINTER.

The target address pointer is automatically entered into the PERIMETER HANDLER as a copy of the current display address and therefore does not need to be entered if the target address is the same as the address on the LED display.

The offset is the difference between the actual location the program is in and the real executing location.

Eg. Program runs at 0000 but is at 1000, offset = 1000-0000. Offset = 1000.

The offset could also be a negative number (greater than 7FFF).

Eg. The program runs at 3800 but is currently at 1000. The offset is then:

1000 - 3800 = F800

BLOCK SHIFT ROUTINE:

ADDRESS, GO, 4

This routine is the simplest of all the utilities. The action of this routine is to move a block from one address to another. None of the bytes in the routine are altered in any way. To use this routine call it up by using ADDR, GO, 4. Now enter the start address, the end address and the destination address (it may be between the start and end addresses). When you have done this then hit "GO" and your block will be shifted for you.

REL JUMP ROUTINE:

ADDRESS, GO, 5

The REL JUMP ROUTINE saves the current address on the software stack and then displays the address that the current data byte lands on if it was a two's compliment displacement for a jump relative.

After using this routine, you can return back by using the SOFTWARE POP routine described below.

You will find this routine a god-sent!

ADDRESS JUMP ROUTINE:

ADDRESS, GO, 9

This routine first saves the current address on the software stack and then displays the address pointed to by the data byte in the display (low order byte) and the next byte (high order byte). This is the normal Z80 format for addresses.

Use the SOFTWARE POP (ADDRESS, GO, 6) to return.

STACK CURRENT ADDRESS:

ADDRESS, GO, 7

This routine saves the current address on the software stack and returns to JMON as it left it. This routine is used in conjunction with the following two RELATIVE DISPLACEMENT CALCULATOR routines or can be used as a "note pad" to remember an important location.

RETRO REL:

ADDRESS, GO, 8

RETRO REL is a routine that will calculate and enter the TWO'S COMPLIMENT displacement between the current

display address and the address on the top of the software stack AT THE ADDRESS ON THE SOFTWARE STACK. The top address on the software stack is then removed. The address on the software stack is incremented before the calculation to allow for the fact that the Z80's program counter is incremented before the displacement is added to it.

This is how to use it:

When entering a program and you come to enter a FORWARD RELATIVE JUMP DISPLACEMENT, stack the address of the displacement (use ADDRESS, GO, 7). Continue to enter the code until you come to the LANDING ADDRESS for the REL JUMP. Now invoke the RETRO REL ROUTINE by pushing ADDRESS, GO, 8. The correct displacement has been retrospectively entered at the address you put on the stack and the address is removed from the stack. Eg.

```
START:  18 XX   JR LAND
```

XX is a displacement you don't know. With JMON pointing to the address of XX, Use ADDRESS, GO, 7. This will put its address on the SOFTWARE STACK. Now, enter the remainder of the code:

```
        00      NOP
        00      NOP
        00      NOP
LAND:   3E 44   LD A,44
```

When you come to LAND, Use ADDRESS, GO, 8.
The right displacement has been placed in the JP REL instruction. Try it!

RETRO LAND:

ADDRESS, GO, B

This is the compliment to the RETRO REL routine. The action of this routine is to calculate the displacement between the current display address and the address on the software stack as described above. The difference here is that the actual landing address is the address on the software stack and the address of where you want the displacement is the current display address. This arrangement is for when you come to the LANDING ADDRESS BEFORE the BACKWARD REL JUMP.

To use it, you stack the landing address as you come to it and enter the rest of the code until you come to the actual address of the DISPLACEMENT. When at this address use ADDRESS, GO, B. The required displacement is entered before your eyes and the landing address is removed from the stack.

SOFTWARE POP:

ADDRESS, GO, 6

This routine returns the address on top of the software stack to the display address buffer of JMON and then removes the return address from the stack. When JMON is re-entered it displays the software popped address. This routine is useful for returning from a software REL JUMP.

SEARCH OR SEARCH/REPLACE:

ADDRESS, GO, A

This routine will search for TWO bytes and optionally replace them. If the optional replace function is not required, then JMON is re-entered and is pointing to the first found occurrence of the two bytes. If the optional replace is enabled, the two bytes are replaced with the new two provided by you in the PERIMETER HANDLER.

This routine uses the PERIMETER HANDLER to enter FOUR variables:

The START and END of the search field, the TWO BYTE VALUE to look for and the OPTIONAL REPLACEMENT BYTES.

The address to look for and the optional replacement is entered so that the high order byte shifts to the left side of the address display.

I.E. Search for 12 34 and replace with 56 78. This will be entered as 34 12 under the tB (target Bytes) heading and 78 56 under the rP (RePlace) heading.

If you do not require an optional replace value, then enter FFFF under the rP heading (you must enter it yourself as it is NOT set to FFFF by default).

This utility can be use to change port numbers

```
Eg. to change       OUT (07),A    D3 07
to:                 OUT (06),A    D3 06
```

Enter 07D3 in the tB window and 06D3 in the rP window and run the utility.

ADDRESS CALL

ADDRESS, GO, C

This utility is similar to the address jump. The difference is that address call puts the current address on the SOFTWARE STACK so you can return to where you called from. Its operation is just like that of a normal call instruction except that its is been simulated by software.

The ADDRESS CALL is designed to allow you to follow the path of a call instruction to its sub-routine when tracing through a program. The Keystrokes are easy to remember for ADDRESS CALL, just think of C for CALL.

STRATEGIES FOR USING THE UTILITIES

The operating condition of the utilities should be considered when writing programs. One particular thing to watch is the use of the HL, DE and BC register pairs. The contents of these registers are subject to being altered by the insert/delete and the code relocation routine.

If one of these registers is being loaded with a value that is not an address pointer within the program, it must be taken into consideration that this value will be altered by the above mentioned routines if it fall within the start and end address range of the program.

I fell into this trap when writing JMON. The loop counters in the tape software were altered and as a result the first 16 JMONs had a faulty high speed tape save routine.

To avoid such problem a good strategy is to load the register pair one byte at a time.

Eg. Instead of this:

```
21 00 02        LD HL,0200
```

Use:

```
26 02           LD H,02
2E 00           LD L,00
```

For the sake of one extra byte you leave the program open for easy editing.

Some times the reverse happens. An address in HL is in fact an address pointer that indexes data within the program block. This value in HL may have been generated by two separate 8 bit values being brought together. When this happens the relocation routine has no way of knowing how to alter the 8 bit values and as a result the address in HL is left unaltered and therefore incorrect.

The way around this problem is to avoid generating address pointers from 8 bit values. If this is not possible then the next best thing to do is to carefully document the offending area of code so that it can be manually altered later.

Another thing to watch is that indirect loading of BC, DE and HL (in the 4 byte form of the instruction) from memory are not altered. These instructions should be carefully documented and changed manually later.

The search and replace utility routine will be handy for this operation.

JMON PROGRAMMERS UTILITIES DISASSEMBLY

```
3800  C3 00 3B          JP 3B00           jump to the reset routine
```

Below is the jump table for the ADDR, GO, (Data key) routine selection of JMON

```
3820  FF FF             unused
3822  F7 39             relocation routine
3824  50 38             byte delete
3826  55 38             byte insert
3828  D4 3A             block shift
382A  07 3B             rel jump
382C  0D 3B             soft pop
382E  57 3B             soft stack
3830  5C 3B             retro rel
3832  76 3B             addr jump
3834  82 3B             search/replace
3836  D3 3B             retro land
3838  73 3B             addr call
```

Start of byte insert/delete set-up routine

```
3850  21 61 38          LD HL,3861        HL = delete routine address
3853  18 03             JR 3858
3855  21 66 38          LD HL,3866        HL = insert routine address
3858  22 88 08          LD (0888),HL      store PH jump on "GO" addr
385B  CD BC 3A          CALL 3ABC         call PH command string set-up
385E  C3 44 00          JP 0044           jump to JMON perimeter handler
```

Delete routine start: The delete routine is called and then a common jump/call address corrector is jump to

```
3861  CD 6B 38          CALL 386B         call delete
3864  18 3C             JR 38A2           jump to corrector
```

Insert routine start:

```
3866  CD 87 38          CALL 3887         call insert
3869  18 37             JR 38A2           jump to common corrector
```

Delete block shift routine. This routine calculates the count and then moves the block above the pointer down one location using the LDDR instruction.

```
386B  2A 9A 08          LD HL,(089A)      LD HL with end address
386E  ED 5B 9E 08       LD DE,(089E)      DE with current pointer
3872  B7                OR A              clear carry
3873  ED 52             SBC HL,DE         is end less that pointer?
3875  DA 4A 00          JP C 004A         jump to JMON err-in if so
3878  E5                PUSH HL           else result = count+1
3879  C1                POP BC            put count +1 into BC
387A  0B                DEC BC            correct count
387B  D5                PUSH DE           save current pointer
387C  E1                POP HL            put it in HL
387D  23                INC HL            increase by source pointer one
387E  ED B0             LDIR              perform block increment shift
3880  ED 53 9A 08       LD (089A),DE      save new end addr
3884  3E FF             LD A,FF           set ACCUM to FF to flag delete
3886  C9                RET               function and return
```

Insert block shift routine. This routine calculates the count and then moves the block using the LDIR instruction.

```
3887  2A 9A 08          LD HL,(089A)      put end in HL
388A  ED 4B 9E 08       LD BC,(089E)      pointer in BC
388E  E5                PUSH HL           save end
388F  E5                PUSH HL           twice on stack
3890  D1                POP DE            put end in DE
3891  B7                OR A              clear carry
3892  ED 42             SBC HL,BC         sub end, pointer to get count-1
3894  E5                PUSH HL           save count-1
3895  C1                POP BC            put count-1 in BC
3896  E1                POP HL            recover end in HL
3897  13                INC DE            point DE to pointer+1
3898  03                INC BC            increase BC to real count
3899  ED 53 9A 08       LD (089A),DE      save new end
389D  ED B8             LDDR              do block decrement shift
389F  AF                XOR A             clear ACCUM to flag insert function
38A0  12                LD (DE),A         and clear new byte in memory
```

```
38A1  C9                  RET             done
```

Below is the common corrector. The corrector uses the byte in the ACCUM to know if the operation was a insert or delete. The value in the ACCUM was placed there by the insert or delete routine. This byte is stored at 08A4 for future reference

```
38A2  32 A4 08            LD (08A4),A     put insert/delete flag in buffer
38A5  2A 98 08            LD HL,(0898)    put start in "working" start
38A8  22 A0 08            LD (08A0),HL    buffer and
38AB  2A 9A 08            LD HL,(089A)    end in "corrector end"
38AE  22 A2 08            LD (08A2),HL    buffer
38B1  CD B5 38            CALL 38B5       call main corrector routine
38B4  C9                  RET             and return
```

Main corrector routine

```
38B5  2A A0 08            LD HL,(08A0)    get first addr from working start
38B8  7E                  LD A,(HL)       get op-code
38B9  CD 6B 39            CALL 396B       call length to find how many
38BC  F5                  PUSH AF         bytes in instr: save flags
38BD  79                  LD A,C          put byte count in ACCUM
38BE  FE 03               CP 03           is it a 3 byte instruction?
38C0  28 06               JR Z 38C8       jump if it is to 3 byte handler
38C2  F1                  POP AF          else recover flags
38C3  79                  LD A,C          get length from C again
```

The first jump below is executed if the instruction is one that may modify execution sequence E.g a RET, JR, JP(HL) etc: the carry was set in the length routine

```
38C4  38 59               JR C 391F       jump to exception handler
38C6  18 51               JR 3919         jump from here if just normal instr
```

3 byte instruction handler. First test that the instruction is not an IX or IY reference, if not then must be absolute address reference so correct address.

```
38C8  F1                  POP AF          clean up stack
38C9  00                  NOP             fixed
38CA  00                  NOP             some errors
38CB  00                  NOP             here
38CC  00                  NOP
38CD  00                  NOP
38CE  FE DD               CP DD           test for IX instruction
38D0  28 47               JR Z 3919       jump if it is IX instruction
38D2  FE FD               CP FD           else test for IY instruction
38D4  28 43               JR Z 3919       jump if so
38D6  23                  INC HL          else must be a 3 byte jump or
38D7  5E                  LD E,(HL)       memory pointer: put target addr
38D8  23                  INC HL          into DE
38D9  56                  LD D,(HL)
38DA  ED 4B 9E 08         LD BC,(089E)    put pointer in BC
38DE  1B                  DEC DE          temporary sub 1 from target address
38DF  2A 9C 08            LD HL,(089C)    get user provided offset
38E2  19                  ADD HL,DE       add target addr and offset
                                          to form target addr to match
                                          new area: put new target-1 in DE
38E3  EB                  EX DE,HL        and old target addr-1 in HL
38E4  CD 60 39            CALL 3960       call to see if landing target lower than
38E7  2A A0 08            LD HL,(08A0)    pointer: put instruction addr in HL: jump
38EA  38 1D               JR C 3909       if landing below pointer (no change required)
38EC  E5                  PUSH HL         save instruction pointer
38ED  ED 4B A2 08         LD BC,(08A2)    put end in BC
38F1  CD 60 39            CALL 3960       call to see if targ lower than end
38F4  E1                  POP HL          put current instr pointer in HL
38F5  30 12               JR NC 3909      jump if targ above end (no alt)
38F7  23                  INC HL          else get actual targ addr
38F8  5E                  LD E,(HL)       in DE
38F9  23                  INC HL          as we are going to correct the address
38FA  56                  LD D,(HL)
38FB  3A A4 08            LD A,(08A4)     test for insert or delete
38FE  B7                  OR A            if A=0 then insert
38FF  20 02               JR NZ 3903      jump if delete
3901  13                  INC DE          else increment target addr
3902  13                  INC DE          twice
3903  1B                  DEC DE          decrement target addr
3904  72                  LD (HL),D       store
3905  2B                  DEC HL          new
3906  73                  LD (HL),E       targ addr
```

```
3907  18 01          JR 390A          jump to up-date the pointer to next instruction
3909  23             INC HL           various sections jump around here to set
390A  23             INC HL           HL to point to the next instruction
390B  23             INC HL           depending on length of current inst
390C  22 A0 08       LD (08A0),HL     store new instruction pointer
390F  ED 5B A2 08    LD DE,(08A2)     test if instruction pointer
3913  B7             OR A             is equal to end pointer
3914  ED 52          SBC HL,DE
3916  38 9D          JR C 38B5        jump for more if not
3918  C9             RET              else all done, go home
```

Normal instruction processed here

```
3919  23             INC HL           HL is incremented
391A  0D             DEC C            once for each byte in the
391B  20 FC          JR NZ 3919       instruction
391D  18 ED          JR 390C          jump to check for end
391F  FE 02          CP 02            test here for a jump relative
3921  20 F6          JR NZ 3919       jump if not 2 bytes, else must be rel jump
3923  23             INC HL           else get displacement
3924  5E             LD E,(HL)        in e and inc HL to simulate PC
3925  23             INC HL           being incremented before jump
```

Below the display segment is sign extended, that is turned into a 16 bit two's complement value in DE

```
3926  AF             XOR A            clear accum
3927  CB 7B          BIT 7,E          is displacement
3929  28 01          JR Z 392C        negative: jump if not
392B  2F             CPL              else set all a bits high
392C  57             LD D,A           put in D
392D  19             ADD HL,DE        add displacement and pointer+2
392E  ED 5B A0 08    LD DE,(08A0)     HL now = landing addr: put pointer into DE
3932  ED 4B 9E 08    LD BC,(089E)     DE and target addr in BC
3936  2B             DEC HL           set HL to
3937  2B             DEC HL           landing-2
3938  CD 55 39       CALL 3955        call maths
```

Carry is clear if the both the landing addr and addr of jump is greater than targ addr or if both the land addr and jump addr is below the targ addr. in other words The jump does not cross the target addr and the displacement doesn't need to be altered.

```
393B  30 29          JR NC 3966       jump if no alt required
393D  D5             PUSH DE          put pointer in HL
393E  E1             POP HL
393F  23             INC HL           point to displacement
3940  3A A4 08       LD A,(08A4)      get insert/delete flag
3943  B7             OR A
3944  20 0B          JR NZ 3951       jump if delete
3946  7E             LD A,(HL)        get displacement in a
3947  CB 7F          BIT 7,A          test for backward jump
3949  20 03          JR NZ 394E       jump if so
394B  34             INC (HL)         else increment displacement
394C  18 BD          JR 390B          jump to store and continue
394E  35             DEC (HL)         decrement displacement
394F  18 BA          JR 390B          jump to store
3951  7E             LD A,(HL)        delete corrector: get displacement
3952  2F             CPL              in ACCUM: toggle bits to use above
3953  18 F2          JR 3947          corrector and jump to correct
```

General purpose maths section

```
3955  B7             OR A             clear carry
3956  ED 42          SBC HL,BC        subtract BC from HL
3958  30 06          JR NC 3960       jump if HL = or BC
395A  C5             PUSH BC          put BC
395B  E1             POP HL           into HL
395C  B7             OR A             clear carry
395D  ED 52          SBC HL,DE        subtract DE from HL
395F  C9             RET              done
3960  D5             PUSH DE          put DE
3961  E1             POP HL           into HL
3962  B7             OR A             clear carry
3963  ED 42          SBC HL,BC        sub BC from HL
3965  C9             RET              done
```

Routine jumps here if displacement not required to be altered

3966	2A A0 08	LD HL,(08A0)	get pointer
3969	18 9F	JR 390A	jump to up-date pointer and cont

Length routine

396B	0E 04	LD C,04	length routine
396D	7E	LD A,(HL)	this routine works out the length
396E	23	INC HL	of each instruction and returns
396F	46	LD B,(HL)	with it in the C register.
3970	2B	DEC HL	as well as the length, this
3971	E6 DF	AND DF	routine checks to see if the
3973	FE DD	CP DD	instruction may break the normal
3975	20 11	JR NZ 3988	sequence of execution Eg a ret
3977	78	LD A,B	jump or call and sets the carry
3978	30 06	JR NC 3980	if so
397A	C8	RET Z	because its operation is
397B	FE 36	CP 36	straight forward and obvious,
397D	C8	RET Z	comments for each instruction are
397E	FE 21	CP 21	unnecessary
3980	C8	RET Z	
3981	E6 F7	AND F7	
3983	FE 22	CP 22	
3985	C8	RET Z	
3986	18 0F	JR 3997	
3988	7E	LD A,(HL)	
3989	FE ED	CP ED	
398B	20 1F	JR NZ 39AC	
398D	78	LD A,B	
398E	E6 C7	AND C7	
3990	FE 43	CP 43	
3992	C8	RET Z	
3993	B7	OR A	
3994	0D	DEC C	
3995	0D	DEC C	
3996	C9	RET	
3997	0D	DEC C	
3998	78	LD A,B	
3999	E6 B8	AND B8	
399B	FE 30	CP 30	
399D	C8	RET Z	
399E	78	LD A,B	
399F	E6 06	AND 06	
39A1	FE 06	CP 06	
39A3	C8	RET Z	
39A4	0D	DEC C	
39A5	78	LD A,B	
39A6	FE E9	CP E9	
39A8	37	SCF	
39A9	C8	RET Z	
39AA	3F	CCF	
39AB	C9	RET	
39AC	0D	DEC C	
39AD	7E	LD A,(HL)	
39AE	E6 CF	AND CF	
39B0	FE 01	CP 01	
39B2	C8	RET Z	
39B3	7E	LD A,(HL)	
39B4	E6 E7	AND E7	
39B6	FE 22	CP 22	
39B8	C8	RET Z	
39B9	7E	LD A,(HL)	
39BA	FE C3	CP C3	
39BC	37	SCF	
39BD	C8	RET Z	
39BE	FE CD	CP CD	
39C0	37	SCF	
39C1	C8	RET Z	
39C2	E6 C7	AND C7	
39C4	FE C2	CP C2	

```
39C6  37          SCF
39C7  C8          RET Z
39C8  FE C4       CP C4
39CA  37          SCF
39CB  C8          RET Z
39CC  0D          DEC C
39CD  FE 06       CP 06
39CF  C8          RET Z
39D0  FE C6       CP C6
39D2  C8          RET Z
39D3  0D          DEC C
39D4  37          SCF
39D5  C8          RET Z
39D6  7E          LD A,(HL)
39D7  E6 F7       AND F7
39D9  C8          RET Z
39DA  0C          INC C
39DB  7E          LD A,(HL)
39DC  E6 E7       AND E7
39DE  FE C3       CP C3
39E0  C8          RET Z
39E1  E6 C7       AND C7
39E3  37          SCF
39E4  C8          RET Z
39E5  0D          DEC C
39E6  7E          LD A,(HL)
39E7  FE E9       CP E9
39E9  37          SCF
39EA  C8          RET Z
39EB  FE C9       CP C9
39ED  37          SCF
39EE  C8          RET Z
39EF  E6 C1       AND C1
39F1  FE C0       CP C0
39F3  37          SCF
39F4  C8          RET Z
39F5  3F          CCF
39F6  C9          RET
```

Set-up for the code relocation routine

```
39F7  21 06 3A    LD HL,3A06     load HL with routine start addr
39FA  22 88 08    LD (0888),HL   save in perimeter go addr buffer
39FD  21 EE 3A    LD HL,3AEE     point HL to command string
3A00  CD BF 3A    CALL 3ABF      shift command sting to ram
3A03  C3 44 00    JP 0044        jump to perimeter handler
```

Code relocate routine re-starts here after perimeter handler.

```
3A06  CD 0C 3A    CALL 3A0C      call block shift
3A09  C3 45 3A    JP 3A45        and jump to corrector
```

Block shift starts here

```
3A0C  2A 98 08    LD HL,(0898)   put start in HL
3A0F  ED 4B 9C 08 LD BC,(089C)   destination in BC
3A13  ED 5B 9A 08 LD DE,(089A)   and end in DE
3A17  E5          PUSH HL        save start
3A18  B7          OR A           clear carry
3A19  ED 42       SBC HL,BC      get offset between start and dest
3A1B  30 06       JR NC 3A23     jump if dest below start
3A1D  C5          PUSH BC        else put dest in HL
3A1E  E1          POP HL
3A1F  13          INC DE         inc end
3A20  B7          OR A           clear carry
3A21  ED 52       SBC HL,DE      dest - end
3A23  E1          POP HL         put start into HL again
3A24  F5          PUSH AF        save flags
3A25  E5          PUSH HL        save start
3A26  EB          EX DE,HL       DE=start HL=end
3A27  B7          OR A           clear carry
3A28  ED 52       SBC HL,DE      end-start
3A2A  EB          EX DE,HL       DE=no of bytes
```

```
3A2B  E1          POP HL          recover start
3A2C  30 04       JR NC 3A32      jump if end greater than start
3A2E  F1          POP AF          else clean up stack
3A2F  C3 4A 00    JP 004A         jump to display err-in
3A32  F1          POP AF          recover flags
3A33  D5          PUSH DE         swap
3A34  C5          PUSH BC         DE
3A35  D1          POP DE          and
3A36  C1          POP BC          BC
3A37  30 08       JR NC 3A41      jump if dest is between start and
3A39  EB          EX DE,HL        end: else swap HL and DE
3A3A  09          ADD HL,BC       calculate end of new block
3A3B  EB          EX DE,HL        put start in HL dest in DE
3A3C  09          ADD HL,BC       calculate end of original block
3A3D  03          INC BC          increase count to true count
3A3E  ED B8       LDDR            block shift from end first
3A40  C9          RET             done
3A41  03          INC BC          increase BC to real count
3A42  ED B0       LDIR            block shift from the start first
3A44  C9          RET             done
```

The jump/call corrector routine for the code relocater starts here

```
3A45  2A 9C 08    LD HL,(089C)    put dest in HL
3A48  ED 5B 98 08 LD DE,(0898)    put start in DE
3A4C  B7          OR A            clear carry
3A4D  ED 52       SBC HL,DE       get offset between dest and start
3A4F  22 A4 08    LD (08A4),HL    store in correction factor buffer
3A52  2A 9A 08    LD HL,(089A)    get end in HL
3A55  ED 5B 98 08 LD DE,(0898)    put start in DE
3A59  B7          OR A            clear carry
3A5A  ED 52       SBC HL,DE       sub start from end
3A5C  23          INC HL          correct HL to real count
3A5D  ED 5B 9C 08 LD DE,(089C)    put dest in DE
3A61  19          ADD HL,DE       find end of dest block
3A62  22 A2 08    LD (08A2),HL    save it
3A65  2A 9E 08    LD HL,(089E)    get new block start (the destination)
3A68  22 A0 08    LD (08A0),HL    put in working buffer
3A6B  00          NOP             idea
3A6C  00          NOP             scraped
3A6D  00          NOP             to lazy to remove nops!
3A6E  00          NOP
3A6F  2A A0 08    LD HL,(08A0)    get pointer
3A72  7E          LD A,(HL)       get instruction
3A73  CD 6B 39    CALL 396B       find length
3A76  79          LD A,C          put length in a
3A77  FE 03       CP 03           is it a 3 byte instruction?
3A79  20 3B       JR NZ 3AB6      jump if not
3A7B  FE DD       CP DD           is it a
3A7D  28 33       JR Z 3AB2       IX or
3A7F  FE FD       CP FD           IY
3A81  28 2F       JR Z 3AB2       instruction: jump if so
3A83  E5          PUSH HL         else must be 3 byte pointer
3A84  D1          POP DE          put pointer in DE
3A85  23          INC HL          put addr
3A86  4E          LD C,(HL)       in
3A87  23          INC HL          BC
3A88  46          LD B,(HL)
3A89  2A 98 08    LD HL,(0898)    start in HL
3A8C  2B          DEC HL
3A8D  B7          OR A            clear carry
3A8E  ED 42       SBC HL,BC       sub target from start-1
3A90  30 11       JR NC 3AA3      jump if target start
3A92  2A 9A 08    LD HL,(089A)    put end in HL
3A95  B7          OR A            clear carry
3A96  ED 42       SBC HL,BC       sub target from end
3A98  38 09       JR C 3AA3       jump if target higher than end
3A9A  2A A4 08    LD HL,(08A4)    get correction factor
3A9D  09          ADD HL,BC       add to jump/call/pointer address
3A9E  EB          EX DE,HL        put new addr in DE
3A9F  23          INC HL          and store
```

```
3AA0 73          LD (HL),E       it
3AA1 23          INC HL          back
3AA2 72          LD (HL),D       to jump/call etc instruction
3AA3 2A A0 08    LD HL,(08A0)    get pointer
3AA6 23          INC HL          increase to next instruction
3AA7 23          INC HL
3AA8 23          INC HL
3AA9 22 A0 08    LD (08A0),HL    store it
3AAC ED 5B A2 08 LD DE,(08A2)    get end
3AB0 B7          OR A
3AB1 ED 52       SBC HL,DE       test for finish
3AB3 38 BA       JR C 3A6F       jump if not finished
3AB5 C9          RET             done
3AB6 23          INC HL          routine comes here if not 3 byte
3AB7 0D          DEC C           instruction: HL is incremented
3AB8 20 FC       JR NZ 3AB6      to point to the next instruction
3ABA 18 ED       JR 3AA9         jump to end test
```

Perimeter set-up for insert delete routines (See the TECPACK for an explaination of the PERIMETER HANDLER set-up values)

```
3ABC 21 DC 3A    LD HL,3ADC      point HL to start of command
3ABF 11 80 08    LD DE,0880      string and DE to ram area
3AC2 01 08 00    LD BC,0008      set for 8 bytes (no jump vector)
3AC5 ED B0       LDIR            shift variables
3AC7 21 00 00    LD HL,0000      clear optional
3ACA 22 9C 08    LD (089C),HL    offset buffer
3ACD 2A 2E 08    LD HL,(082E)    get current pointer
3AD0 22 9E 08    LD (089E),HL    put it in working buffer
3AD3 C9          RET             done
```

perimeter set-up for block shift

```
3AD4 21 0C 3A    LD HL,3A0C
3AD7 C3 FA 39    JP 39FA

3ADC FF FF                       unused
3ADE E4 3A                       data displays address
3AE0 99 08                       ram buffer+1
3AE2 00                          number of first window
3AE3 03                          number of allowable windows-1
```

Display codes for block shift

```
3AE4 04 A7                       -S
3AE6 04 C7                       -E
3AE8 04 EB                       -0
3AEA 04 4F                       -P
```

Command string for code relocation routine.

```
3AEE FF FF F6 3A 99 08 00 02
```

Display codes for code relocation routine.

```
3AF6 04 A7                       -s
3AF8 04 C7                       -e
3AFA 04 EC                       -d

3B00 21 FF 0F    LD HL,0FFF      utilities reset routine
3B03 22 FC 08    LD (08FC),HL    set soft stack at 0FFF
3B06 C9          RET             done
```

Rel jump routine

```
3B07 CD 10 3B    CALL 3B10       call stack routine
3B0A C3 29 3B    JP 3B29         jump to rel calculator
```

Soft pop start

```
3B0D C3 3E 3B    JP 3B3E         jump to soft pop routine
```

Stack routine

```
3B10 ED 5B D0 0F LD DE,(0FD0)    get soft stack pointer
3B14 7B          LD A,E
3B15 FE DF       CP DF           test for end
3B17 CA 3C 08    JP Z 083C       jump to sound error bell
3B1A 21 2F 08    LD HL,082F      point to current display addr
```

```
3B1D 7E                 LD A,(HL)           put it in
3B1E 12                 LD (DE),A           soft stack
3B1F 1B                 DEC DE
3B20 2B                 DEC HL
3B21 7E                 LD A,(HL)
3B22 12                 LD (DE),A
3B23 1B                 DEC DE
3B24 ED 53 FC 08        LD (08FC),DE        store new soft stack value
3B28 C9                 RET                 done
```

Rel jump calculator

```
3B29 2A 2E 08           LD HL,(082E)        get current display pointer
3B2C 5E                 LD E,(HL)           sign extend
3B2D AF                 XOR A               displacement
3B2E CB 7B              BIT 7,E             in
3B30 28 01              JR Z 3B33           DE
3B32 2F                 CPL
3B33 57                 LD D,A
3B34 23                 INC HL
3B35 19                 ADD HL,DE           add displacement and pointer
3B36 22 2E 08           LD (082E),HL        store new pointer
3B39 AF                 XOR A               set JMON to data mode
3B3A 32 2B 08           LD (082B),A
3B3D C9                 RET                 done
```

Unstack stack routine

```
3B3E ED 5B FC 08        LD DE,(08FC)        get soft stack pointer
3B42 7B                 LD A,E              test for
3B43 FE FF              CP FF               last location
3B45 28 F2              JR Z 3B39           go if it is
3B47 21 2E 08           LD HL,082E          else
3B4A 13                 INC DE              get
3B4B 1A                 LD A,(DE)           low byte
3B4C 77                 LD (HL),A           put in display buffer
3B4D 13                 INC DE              do
3B4E 23                 INC HL              for
3B4F 1A                 LD A,(DE)           high
3B50 77                 LD (HL),A           byte
3B51 ED 53 FC 08        LD (08FC),DE        save new soft stack pointer
3B55 18 E2              JR 3B39             jump to set data mode
```

Addr, go, 7 routine (stack current location)

```
3B57 CD 10 3B           CALL 3B10           call soft stacker
3B5A 18 DD              JR 3B39             jump to set data mode
```

Retro rel

```
3B5C 2A 2E 08           LD HL,(082E)        save the current display
3B5F E5                 PUSH HL             pointer on the stack
3B60 CD 3E 3B           CALL 3B3E           call soft pop
3B63 ED 5B 2E 08        LD DE,(082E)        put poped addr in DE
3B67 E1                 POP HL              recover current disp pointer
3B68 22 2E 08           LD (082E),HL        restore in buffer
3B6B 13                 INC DE              inc DE as PC is incremented
3B6C B7                 OR A                before rel jump: clear a
3B6D ED 52              SBC HL,DE           get displacement
3B6F 7D                 LD A,L              from L
3B70 1B                 DEC DE              point DE to displacement addr
3B71 12                 LD (DE),A           store displacement
3B72 C9                 RET                 done
```

Address jump

```
3B73 CD 10 3B           CALL 3B10           stack current disp addr
3B76 2A 2E 08           LD HL,(082E)        get current disp addr
3B79 5E                 LD E,(HL)           put 16 bit contents
3B7A 23                 INC HL              into DE
3B7B 56                 LD D,(HL)
3B7C ED 53 2E 08        LD (082E),DE        store DE as new current disp addr
3B80 18 B7              JR 3B39             jump to set data mode
```

Search/replace perimeter handler set-up

```
3B82 21 C1 3B           LD HL,3BC1          point HL to command string
```

```
3B85  11 80 08      LD DE,0880      DE to ram area
3B88  01 0A 00      LD BC,000A      BC for 10 bytes
3B8B  ED B0         LDIR            move variables
3B8D  C3 44 00      JP 0044         jump to PH
```

Search/replace routine

```
3B90  2A 98 08      LD HL,(0898)    put start in HL
3B93  ED 4B 9A 08   LD BC,(089A)    end in BC
3B97  ED 5B 9C 08   LD DE,(089C)    target addr in DE
3B9B  7B            LD A,E          test low order byte
3B9C  BE            CP (HL)
3B9D  23            INC HL          point to high order byte
3B9E  20 13         JR NZ 3BB3      jump if low byte not the same
3BA0  7A            LD A,D          test
3BA1  BE            CP (HL)         high byte
3BA2  20 0F         JR NZ 3BB3      jump if not the same
3BA4  ED 5B 9E 08   LD DE,(089E)    get optional replace addr
3BA8  13            INC DE          test
3BA9  7A            LD A,D          for
3BAA  B3            OR E            FFFF
3BAB  1B            DEC DE
3BAC  28 0D         JR Z 3BBB       jump if FFFF as no replacement required
3BAE  2B            DEC HL          else
3BAF  73            LD (HL),E       replace low byte
3BB0  23            INC HL
3BB1  72            LD (HL),D       and then high byte
3BB2  23            INC HL          next byte
3BB3  E5            PUSH HL         save pointer
3BB4  B7            OR A            test for end
3BB5  ED 42         SBC HL,BC
3BB7  E1            POP HL
3BB8  38 DD         JR C 3B97       jump if more
3BBA  C9            RET             done
```

The routine comes here when the addr found but no replacement is wanted

```
3BBB  2B            DEC HL          correct HL
3BBC  22 2E 08      LD (082E),HL    store in current display buff
3BBF  18 99         JR 3B5A         jump to set data mode
```

Search/replace command string

```
3BC1  FF FF
3BC3  CB 3B                         data display address
3BC5  99 08                         ram buffer+1
3BC7  00                            number of first window
3BC8  03                            number of allowable windows-1
3BC9  90 3B                         jump address
```

search/replace displays

```
3BCB  04 A7                         -s
3BCD  04 C7                         -e
3BCF  C6 E6                         tb
3BD1  44 4F                         rp
```

Retro land

```
3BD3  2A 2E 08      LD HL,(082E)    save current pointer
3BD6  E5            PUSH HL
3BD7  CD 3E 3B      CALL 3B3E       get addr on soft stack
3BDA  ED 5B 2E 08   LD DE,(082E)    put it in DE
3BDE  E1            POP HL          restore current pointer
3BDF  22 2E 08      LD (082E),HL
3BE2  EB            EX DE,HL        put current pointer in DE and
3BE3  13            INC DE          landing address in HL
3BE4  B7            OR A            inc DE as PC is inc before jump
3BE5  ED 52         SBC HL,DE       find offset
3BE7  7D            LD A,L          put 8 bit offset into ACCUM
3BE8  1B            DEC DE          point DE to instruction displacement
3BE9  12            LD (DE),A       store displacement in jump REL
3BEA  18 D3         JR 3BBF         jump to set JMON data mode
```