SECTION 2

COMMAND ROUTINES

COMMAND INDEX

```
MAIN  LOOP

;  Program start

STARM           CALL   CLRSC     ;  Clear the TRS8Ø Screen
                LD     HL,SIGON  ;  Point to sign on message
                CALL   PSTR      ;  Print it
                CALL   PNEWL     ;  Print a new line
                CALL   INIT      ;  Set up system
QUES1           CALL   DELT      ;  Small delay
                LD     HL,QUESS  ;  Point to menue string
                CALL   PSTR      ;  Print it
                CALL   GCHRA     ;  Get response and print it
                CALL   PNEWL     ;  Print new line
                CP     NL        ;  Is response a newline
                JR     Z,QUES1   ;  Yes then ignore
                CP     'L'       ;  Is response an 'L'
                JP     Z,LEARN   ;  Yes do learn section
                CP     'E'       ;  Is it an 'E'
                JP     Z,EDIT    ;  Yes do edit
                CP     'R'       ;  Is it an 'R'
                JP     Z,READ    ;  Yes then do read command
                CP     'W'       ;  Is it a 'W'
                JP     Z,WRITE   ;  Yes do write command
                CP     'C'       ;  Is it a 'C'
                JP     Z,CHECK   ;  Yes do check routine
                CP     'S'       ;  Is it an 'S'
                JP     Z,SETAM   ;  Yes then do arm set
                CP     'T'       ;  a 'T'
                JP     Z,TOSTM   ;  Yes then move arm to start
                CP     'G'       ;  a 'G'
                JP     Z,GO      ;  Do execute movements stored
                CP     'D'       ;  a 'D'
                JP     Z,DISP    ;  Yes then display ARST array
                CP     'B'       ;  a 'B'
                JP     Z,BOOT    ;  Yes then restart system
                CP     'M'       ;  an 'M'
                JP     Z,MANU    ;  Yes the Manual control of arm
                CP     'F'       ;  a 'F'
                JP     Z,FREARM  ;  Yes then clear all motors
                CP     'Q'       ;  a 'Q'
                JP     Z,FINSH   ;  Yes then quit program
                LD     HL,QMESS  ;  Point to 'PARDON' message
                CALL   PSTR      ;  Print it
                JP     QUES1     ;  Try for next command
```

THE LEARN ROUTINE

```
;  This section deals with the recording
;  of an arm sequence

LEARN       LD     HL,RELNS   ;  Point to learn message
            CALL   PSTR       ;  Print the message
            CALL   GCHRA      ;  Get response and print it
            CALL   PNEWL      ;  Print a new line
            CP     '.'        ;  Response a '.'
            JP     Z,QUES1    ;  Back to main loop is uder types a '.
            CP     'S'        ;  Response an 'S'
            JR     Z,WAIT1    ;  Learn sequence from start
            CP     'C'        ;  a 'C'
            JR     Z,NOINT    ;  Continue learning from end of
                              ;  sequence
            CALL   PNEWL      ;  output a new line
            JR     LEARN      ;  Bad answer so try again
WAIT1       CALL   MOVTO      ;  Move arm to start position
            CALL   INIT       ;  Clear variables
WAIT2       LD     HL,CASRD   ;  Point to waiting message
            CALL   PSTR       ;  Print it
            CALL   GCHRA      ;  Get response and print it
            CALL   PNEWL      ;  Print new line character
            CP     '.'        ;  Response a '.'
            JP     QUES1      ;  Exit to main loop if so
            CP     SPAC       ;  Is it a space?
            JR     NZ,WAIT2   ;  If not then bad input, try again
            CALL   TORQUE     ;  Switch motors on
            JR     STLRN      ;  Do rest of learn
NOINT       LD     HL,(COUNT) ;  Get current count
            LD     A,L        ;
            OR     H          ;  Is it zero?
            JR     Z,NOSTR    ;  Yes then can't add to nothing
STLRN       XOR    A          ;  Clear manual flag
            LD     (MAN)A     ;  Because we are in learn mode
CONLN       CALL   KEYIN      ;  Drive motors and store sequence
            OR     A          ;  Zero key pressed
            JR     NZ,CONLN   ;  No then continue
            CALL   MOVTO      ;  Move arm to start position
            JP     QUES1      ;  Back to main loop
```

EDIT FUNCTION

```
EDIT    LD      HL,(COUNT)      ;  Get row count
        LD      A,L             ;
        OR      H               ;  Test for zero
        JP      Z,NOSTR         ;  Yes then nothing in store
EDSRT   LD      HL,ECOMS        ;  Print edit message
        CALL    PSTR            ;
        CALL    GCHRA           ;  Get response
        CALL    PNEWL           ;  Print a new line
        CP      'M'             ;  Is response an 'M'
        JR      Z,EDMOT         ;  Yes then edit motor
        CP      'R'             ;  Is response an 'R'
        JR      NZ,EDSRT        ;  No then try again
        LD      HL,COUTS        ;  HL = New row count message
        CALL    PSTR            ;  Print it
        CALL    GINT            ;  Get 16 bit signed integer
        JP      NZ,BADC         ;  Non zero return means bad input
        LD      A,H             ;  Test top bit of HC
        BIT     7,A             ;
        JP      NZ,BADC         ;  If negative then bad input
        LD      BC,(COUNT)      ;  Get count value
        PUSH    HL              ;  Save response
        OR      A               ;  Clear carry flag
        SBC     HL,BC           ;  See if response < current count
        POP     HL              ;  Restore response
        JR      NC,BADC         ;
        LD      (COUNT),HL      ;  Replace count with response
        JP      QUES1           ;  Back to main loop
EDMOT   LD      HL,EDSTR        ;
        CALL    PSTR            ;  Print 'row number'
        CALL    GINT            ;  Get integer response
        JR      NZ,BADC         ;  Bad answer
        LD      A,H             ;
        BIT     7,A             ;  No negative row count
        JR      NZ,BADC         ;  allowed
        LD      A,H             ;
        OR      L               ;  or zero row count
        JR      Z,BADC          ;
        LD      BC,(COUNT)      ;  Get row count into BC
        INC     BC              ;  Move count up one
        PUSH    HL              ;  Clear carry flag
        SBC     HL,BC           ;  Subtract count from response
        POP     HL              ;  Restore response
        JR      NC,BADC         ;  If greater than allowed error
EDOK    DEC     HL              ;  Move response down one
        ADD     HL,HL           ;  Double HL
        PUSH    HL              ;  Save it
        ADD     HL,HL           ;  Row count x 4
        POP     BC              ;  BC = row count x 2
```

```
        ADD     HL,BC       ;  HL = Row count x 6
        LD      BC,ARST     ;  Get store start address
        ADD     HL,BC       ;  Add row offset
        PUSH    HL          ;  Save resulting pointer
        LD      HL,MOTNS    ;  Print
        CALL    PSTR        ;  Motor number string
        CALL    GINT        ;  Get Answer
        JR      NZ,BADNM    ;  Bad answer
        LD      A,H         ;
        OR      A           ;
        JR      NZ,BADNM    ;  Response too large
        LD      A,L         ;
        CP      1           ;
        JR      C,BADUM     ;  No motor number < 1
        CP      7           ;
        JR      NC,BADNM    ;  No motor number > 6
        POP     HL          ;  Restore = Memory pointer
        DEC     A           ;  Motor offset Ø → 5
        LD      C,A         ;
        LD      B,Ø         ;  Add to memory pointer
        ADD     HL,BC       ;  Now we point to motor in store
        PUSH    HL          ;  Save pointer
        LD      HL,NVALS    ;
        CALL    PSTR        ;  Print new step value
        CALL    GINT        ;  Get response
        JR      NZ,BADNM    ;  Bad answer
        LD      A,H         ;
        CP      ØFFH        ;
        JR      NZ,PEDIT    ;  We have a positive response
        BIT     7,L         ;  New negative step value too
        JR      Z,BADNM     ;  large
        JR      MOTAS       ;  Step value OK
PEDIT   OR      A           ;  New positive step value too
        JR      NZ,BADNM    ;  large
        BIT     7,L         ;  so exit
        JR      NZ,BADNM    ;  else  ok
MOTAS   LD      A,L         ;  Get step value
        POP     HL          ;  Restore memory pointer
        LD      (HL),A      ;  Place step value in store
        JP      QUES1       ;  Go do next operation
BADNM   POP     HL          ;
BADC    LD      HL,BADMS    ;  Print error message and
        CALL    PSTR        ;
        JP      QUES1       ;  return to main loop
```

READ ROUTINE

```
;  Reads stored sequence from cassette
;  into memory

READ    LD      HL,CASRD    ;  Point to wait message
        CALL    PSTR        ;  Print it
        CALL    GCHRA       ;  Get response
        CALL    PNEWL       ;  Print new line
        CP      '.'         ;  Is response a dot?
        JP      Z,QUES1     ;  Yes then exit
        CP      SPAC        ;  Is it a space?
        JR      NZ,READ     ;  No then try again
        XOR     A           ;  Clear A=Drive zero
        CALL    CASON       ;  Switch on drive zero
        CALL    DELS        ;  Short delay
        CALL    RDHDR       ;  Read header from tape
        CALL    READC       ;  Read first character
        LD      B,A         ;  Put in B
        CALL    READC       ;  Read second character
        LD      C,A         ;  Place in C
        OR      B           ;  BC now equals count
        JP      Z,NOSTR     ;  Count zero, so exit
        LD      (COUNT),BC  ;  Set count = read count
        LD      HL,ARST     ;  Point to start of store
ROWNR   PUSH    BC          ;  Same count
        LD      E,Ø         ;  E = Check sum for a row
        LD      B,6         ;  B = Column Count
RDBYT   CALL    READC       ;  Read a row element
        LD      (HL),A      ;  Store it
        ADD     A,E         ;  Add it to check sum
        LD      E,A         ;  Store in check sum
        INC     HL          ;  Inc memory pointer
        DJNZ    RDBYT       ;  Do next element
        POP     BC          ;  Restore row count
        CALL    READC       ;  Read check digit
        CP      E           ;  Same as calculated?
        JR      NZ,RDERR    ;  No then error
        DEC     BC          ;  Decrement row count
        LD      A,B         ;  See if row count
        OR      C           ;  is zero
        JR      NZ,ROWNR    ;  No then read next row
        CALL    CASOF       ;  Switch cassette off
        JP      TAPEF       ;  exit
RDERR   LD      HL,RDMSG    ;  Error message for tape
        CALL    PSTR        ;  Print it
        JP      QUES1       ;  Go to main loop
```

WRITE ROUTINE

```
;  Writes a stored sequence to tape

WRITE       LD      BC,(COUNT)      ;  Get row count
            LD      A,B             ;
            OR      C               ;
BADWI       JP      Z,NOSTR         ;  If zero exit
            LD      HL,CASRD        ;  print message
            CALL    PSTR            ;
            CALL    GCHRA           ;  Get answer
            CALL    PNEWL           ;  Print new line
            CP      '.'             ;  Is answer a dot
            JP      Z,QUES1         ;  Yes then exit
            CP      SPAC            ;  Is answer a space
            JR      NZ,BADWI        ;  No then try again
            XOR     A               ;  Clear drive number
            CALL    CASON           ;  Switch on drive zero
            CALL    DELT            ;  delay
            CALL    WRLDR           ;  Write Leader
            CALL    DELT            ;  delay
            LD      BC,(COUNT)      ;  Get count into BC
            LD      A,B             ;
            CALL    WRBYA           ;  Write higher byte
            LD      A,C             ;  Get lower byte of count into A
            CALL    DELT            ;  delay
            CALL    WRBYA           ;  Write lower byte
            LD      HL,ARST         ;  Point to start of sequence of store
ROWNW       PUSH    BC              ;  Save row count
            LD      E,Ø             ;  Clear check sum
            LD      B,6             ;  Six motor slots per row
WRBYT       LD      A,(HL)          ;  Get motor  slot N
            CALL    DELS            ;  delay
            CALL    WRBYA           ;  Write it
            CALL    DELS            ;  delay
            ADD     A,E             ;  add to check sum
            LD      E,A             ;
            INC     HL              ;  Inc memory pointer
            DJNZ    WRBYT           ;  Do for all six motors
            CALL    WRBYA           ;  Write check sum
            POP     BC              ;  Restore row count
            DEC     BC              ;  Decrement row count
            LD      A,B             ;
            OR      C               ;  Test if zero
            JR      NZ,ROWNW        ;  No then try again
            CALL    CASOF           ;  Switch cassette off
            JP      QUES1           ;  Back to main loop
```

CHECK ROUTINE

```
;   Checks tape with sequence in store

CHECK       LD      BC,(COUNT)      ;  Get row count
            LD      A,B             ;
            OR      C               ;
            JP      Z,NOSTR         ;  If zero exit
BADCI       LD      HL,CASRD        ;  Print wait message
            CALL    PSTR            ;
            CALL    GCHRA           ;  Get answer
            CALL    PNEWL           ;  Print new line
            CP      '.'             ;  Is response a '.'
            JP      Z,QUES1         ;  Yes then go to main loop
            CP      SPAC            ;  Is it a space
            JR      NZ,BADCI        ;  No then try again
            XOR     A               ;  Clear cassette number
            CALL    CASON           ;  Switch drive zero on
            CALL    RDHDR           ;  Read header from tape
            LD      BC,(COUNT)      ;  Get row count
            CALL    READC           ;  Read first section
            CP      B               ;  Same?
            JR      NZ,RDERR        ;  No then error
            CALL    READC           ;  Read lower byte of count
            CP      C               ;  Same?
            JR      NZ,RDERR        ;  No then error
            OR      B               ;  Zero count from tape
            JP      Z,NOSTR         ;  So exit
            LD      HL,ARST         ;  Point to start of memory
ROWNC       PUSH    BC              ;  Save count
            LD      E,Ø             ;  Check sum is zero
            LD      B,6             ;  Count is 6
CKBYT       CALL    READC           ;  Read a motor step element
            CP      (HL)            ;  Same as in store?
            JP      NZ,RDERR        ;  Not the same so error
            ADD     A,E             ;
            LD      E,A             ;  Add to check sum
            INC     HL              ;  Advance memory pointer
            DJNZ    CKBYT           ;  Do next row element
            POP     BC              ;  Restore row count
            CALL    READC           ;  Read check sum
            CP      E               ;  Same as check sum calculated
            JP      NZ,RDERR        ;  No then error
            DEC     BC              ;  Decrement count
            LD      A,B             ;
            OR      C               ;  Is count zero?
            JP      NZ,ROWNC        ;  No then do next row
            CALL    CASOF           ;  Switch cassette off
TAPEF       LD      HL,TAPOK        ;  Print tape off message
            CALL    PSTR            ;
            JP      QUES1           ;  and back to main loop
```

BOOT AND FINISH COMMANDS

```
;  This routine restarts the program


BOOT    LD      HL,BOOTS        ;  Print "DO YOU REALLY
        CALL    PSTR            ;  WANT TO RESTART?"
        CALL    GCHRA           ;  Get answer
        CP      'Y'             ;  User typed 'Y'?
        JP      Z,STARM         ;  Yes then restart program
        CP      'N'             ;  No 'N'?
        JR      NZ,BOOT         ;  Then try again
        CALL    PNEWL           ;  else print new line and
        JP      QUES1           ;  back to main loop

;  This is the exit from program Section to TRS80
;  system level

FINSH   LD      HL,RELYQ        ;  Print "REALLY QUIT"
        CALL    PSTR            ;
        CALL    GCHRA           ;  Get answer
        CP      'Y'             ;  User typed a 'Y'
        JR      NZ,TRYNO        ;  No then try 'N'
        LD      HL,SIGOF        ;  Print ending message
        CALL    PSTR            ;  and then
        JP      FINAD           ;  return to TRS80 System
TRYNO   CP      'N'             ;  User typed an 'N'
        JR      NZ,FINSH        ;  No then try again
        CALL    PNEWL           ;  Print a new line
        JP      QUES1           ;  Back to main loop
```

OTHER SHORT COMMANDS

```
;  SETAM  clears arm position array

SETAM     CALL    RESET   ;  Clear Arm array  (POSAR)
          JP      QUES1   ;  Back to main loop

;  TOSTM  moves the arm back to its start position

TOSTM     CALL    MOVTO   ;  Steps motors till POSAR elements
          JP      QUES1   ;  are zero then back to main loop

;  FREARM  frees all motors for user to move arm
;  by hand

FREARM    CALL    CLRMT   ;  Output all ones to motors
          JP      QUES1   ;  and now to main loop

;  MANU allows the user to move the arm using
;  the 1-6 keys and the 'Q' 'W' 'E' 'R' 'T' 'Y' keys
;  The movements made are not stored.

MANU      LD      A,1     ;  Set in manual mode for the
          LD      (MAN),A ;  keyin routine
MANUA     CALL    KEYIN   ;  Now get keys and move motors
          JP      NZ,MANUA;  If non zero then move to be done
          XOR     A       ;  Clear manual flag
          LD      (MAN),A ;
          JP      QUES1   ;  Back to main loop
```

THE GO COMMAND

```
;  This command causes the computer to step
;  through a stored sequence and makes the arm
;  follow the steps stored, if the sequence is to
;  be done forever then the arm resets itself at
;  the end of each cycle.

GO      CALL    PNEWL       ;  Print a new line
        CALL    MOVTO       ;  Move arm to start
        XOR     A           ;  Clear
        LD      (FORFG),A   ;  Forever Flag FORFG
        LD      HL,AORNM    ;  Print "DO ONCE OR FOREVER
        CALL    PSTR        ;  Message
        CALL    GCHRA       ;  Get answer and print it
        CALL    PNEWL       ;  Print a new line
        CP      'O'         ;  User typed an 'O'
        JR      Z,ONECY     ;  Do sequence till end
        CP      'F'         ;  User typed an 'F'
        JR      NZ,GO       ;  No then re-try
        LD      A,1         ;  Set forever flag
        LD      (FORFG),A   ;  to 1
ONECY   LD      A,'.'       ;  Print a '.'
        CALL    PUTCHR      ;  Using PUTCHR
        CALL    DOALL       ;  Execute the sequence
        LD      A,(FORFG)   ;  Test FORFG, if zero
        OR      A           ;  then we do not want
        JR      Z,NORET     ;  to carry on so exit
        CALL    DELT        ;  delay
        CALL    MOVTO       ;  Move arm to start
        CALL    DELLN       ;  Delay approx 1 second
        JR      ONECY       ;  Do next sequence
NORET   LD      HL,DONMS    ;  Print sequence done
        CALL    PSTR        ;
        JP      QUES1       ;  and go to main loop
```

THE DISPLAY COMMAND

```
;  This command allows the user to display
;  the motor sequence so that he can then
;  alter the contents of a sequence by using
;  the edit command


DISP     LD      HL,DISPS      ;  Point to header string
         CALL    PSTR          ;  and display it
         CALL    POSDS         ;  Print out the relative position
         LD      HL,ARST       ;  Point to sequence start
         LD      BC,(COUNT)    ;  BC = how many rows to print
         LD      A,B           ;
         OR      C             ;  Test if count is zero
         JP      NZ,SETBC      ;  No then jump to rest of
NOSTR    LD      HL,NODIS      ;  display else print message
         CALL    PSTR          ;  telling user no display and
         JP      QUES1         ;  return to the main loop
SETBC    LD      BC,ØØØ        ;  Clear BC for row count
DOROW    PUSH    BC            ;  Save it
         PUSH    HL            ;  Save memory position
         LD      H,B           ;
         LD      L,C           ;  HL = row count
         INC     HL            ;  Now row count =N+1
         LD      IX,NUMAR      ;  IX  points to buffer for ASCII String
         CALL    CBTAS         ;  Convert HL to ASCII
         LD      HL,NUMAR      ;  Point to ASCII string
         CALL    PSTR          ;  now print it
         LD      A,'.'         :
         CALL    PUTCHR        ;  Print a '.'
         POP     HL            ;  Restore memory pointer
         LD      B,6           ;  Motor count to B (6 motors)
NEXTE    LD      A,(HL)        ;  Get step value
         PUSH    HL            ;  Save memory pointer
         PUSH    BC            ;  Save motor count
         BIT     7,A           ;  Test bit 7 of A for sign
         JR      Z,NUMPO       ;  If bit = Ø then positive step
         LD      H,ØFFH        ;  Make H = negative number
         JR      EVAL          ;  Do rest
NUMPO    LD      H,Ø           ;  Clear H for positive number
EVAL     LD      L,A           ;  Get low order byte into L
         LD      IX,NUMAR      ;  Point to result string
         CALL    CBTAS         ;  Call conversion routine
         LD      HL,NUMAR      ;  HL points to result
         CALL    PSTR          ;  Print resulting conversion
         LD      A,(381ØH)     ;  Get keyboard memory location
         BIT     Ø,A           ;  Test for zero key pressed
         JR      Z,NOSTP       ;  Not pressed, then skip
DOSTP    CALL    GCHR          ;  Wait till next character entered
         CP      '.'           ;  Is it a dot?
         JR      NZ,NOSTP      ;  No then carry on
         CALL    PNEWL         ;  else print a new line
         POP     BC            ;  and restore all the registers
         POP     HL            ;  and the stack level
```

```
                POP     BC          ;
                JP      QUES1       ; Jump back to main loop
NOSTP           POP     BC          ; Restore column count
                POP     HL          ; Restore memory pointer
                INC     HL          ; Increment memory pointer
                CALL    PSPAC       ; Print a space between
                                    ; numbers
                DJNZ    NEXTE       ; Do for six motors
                CALL    PNEWL       ; Print a new line
                POP     BC          ; Restore row count
                INC     BC          ; Increment row count
                LD      A,(COUNT)   ; Get lower count byte
                CP      C           ; Is it the same
                JR      NZ,DOROW    ; No then do next row
                LD      A,(COUNT+1) ; Get higher order count byte
                CP      B           ; Same?
                JR      NZ,DOROW    ; No then do next row else
                CALL    PNEWL       ; print a new line and then
                JP      QUES1       ; back to main loop
```

SUBROUTINES INDEX

DOALL.....Execute a stored sequence once

DRIVL.....Drives all motors directed by TBUF

INIT.....Set up system

MOVTO.....Use POSAR to rest system arm

TORQUE.....Turn on off motors

CLRMT.....Turn off all motors

SETDT.....Reset CTPOS elements to one

DRAMT.....Drive directed motors

STEPM.....Step motors via DRAMT

DNEWD.....Delay on direction change

SPAMT.....Update TBUF array during learn

KEYIN.....Scan keyboard and build up motors to move

CBTAS.....Convert 16 bit 2's complement number to ASCII

CLRMF.....Clear MOTBF array

CTBUF.....Clear TBUF, DRBUF & MOTBF arrays

GINT.....Get 16 bit signed value from keyboard

POSDS.....Display relative position array elements

POSIC..... .....Increment relative position array elements

STORE.....Copy TBUF to current ARST slice

RESET.....Clear POSAR array

PUTCHR.....Print a character

PSTR.....Print a string

PSPAC.....Print a space

PNEWL.....Print a carriage return

SECTION 3

S

U

B

R

O

U

T

I

N

E

S.

SUBROUTINES INDEX (continued)

SCKBD.....Scan the keyboard

GCHRA.....Get a character and print it

CLRSC.....Clear the Screen

DELSW.....Delay on value in B

DELS.....Delay approx Ø.ØØ1 sec

DELT.....Delay approx Ø. Ø1 sec

DELLN.....Dealy approx 1. Ø sec

SUBRCUTINE DOALL

```
;  This subroutine executes a sequence in store cnce.
;  Forever flag FORFG is cleared if user types a '.'

DOALL     LD      BC,(COUNT)     ;  Get sequence row count
          LD      A,B            ;
          OR      C              ;  If count zero then
          JR      Z,RET2         ;  exit
          LD      HL,ARST        ;  HL points to memory start
NMOTS     LD      DE,TBUF        ;  DE points to temporary buffer
          PUSH    BC             ;  Save count
          LD      BC,ØØØ6        ;  Motor count of six
          LDIR                   ;  Copy memory slice into TBUF
          PUSH    HL             ;  Save new memory pointer
          CALL    DRIVL          ;  Drive all motors for this slice
          CALL    SCKBD          ;  See if keyboard input
          POP     HL             ;  Restore memory pointer
          POP     BC             ;  Restore row count
          CALL    DNEWD          ;
          CP      '.'            ;  User typed a '.'
          JR      NZ,CARON       ;  No then continue
RET2      XOR     A              ;  Clear A
          LD      (FORFG),A      ;  Clear flag to halt routine above
          RET                    ;  exit
CARON     DEC     BC             ;  Decrement count
          LD      A,B            ;
          OR      C              ;  Test for zero
          JR      NZ,NMOTS       ;  No then carry on else
          RET                    ;  return
```

```
SUBROUTINE DRIVL

;  This routine is given TBUF, it then drives all
;  the motors that need to be driven, till TBUF = Ø

DRIVL       LD      C,Ø             ;
SCANW       LD      B,6             ;  Set BC = motor count
            LD      HL,TBUF         ;  Point to TBUF
TBZER       LD      A,(HL)          ;  Get step value from TBUF
            OR      A               ;  Is it zero?
            JR      NZ,TBNZR        ;  No then continue
            INC     HL              ;  Point to next TBUF location
            DJNZ    TBZER           ;  Do next motor check
            RET                     ;  If no motor to step, then return
TBNZR       LD      DE,MOTBF + 5    ;  DE points to last direction array
            LD      HL,TBUF + 5     ;  HL points to TBUF
            LD      B,6             ;  B = motor count
DOAGN       LD      A,(HL)          ;  Get motor step value
            CP      Ø               ;  Is it zero?
            JR      Z,NOEL          ;  Yes then skip
            JP      M,SNEG          ;  Is it negative ie, reverse
SPOS        LD      A,3             ;  No positive, so load MOTBF (N)
            LD      (DE),A          ;  With 3
            DEC     (HL)            ;  Decrement motor count in TBUF
            JR      NOFIL           ;  Complete the MOTBF array
SNEG        LD      A,1             ;  Set MOTBF = 1 for
            LD      (DE),A          ;  a positive drive
            INC     (HL)            ;  Decrement negative count
            JR      NOFIL           ;  Do rest of MOTBF
NOEL        XOR     A               ;  Clear MOTBF (N)
            LD      (DE),A          ;
NOFIL       DEC     DE              ;  Move to next MOTBF element
            DEC     HL              ;  Move to next TBUF element
            DJNZ    DOAGN           ;  Do for all six motors
            LD      A,1             ;
            LD      (KEYP),A        ;  Set key pressed flag
            CALL    STEPM           ;  Step all motors once if
            DEC     C               ;  any to step
            JP      NZ,SCANW        ;  Do for maximum of 128 cycles
            RET                     ;  then return
```

SUBROUTINE INIT

```
;  INIT clears the row count (COUNT), resets the
;  MAN flag, clears the TBUF, DRBUF, & MOTBF arrays
;  The CUROW pointer is reset  to the start of the ARST,
;  position array is cleared.


INIT        LD      HL,Ø          ;  Set HL = Ø
            LD      (COUNT),HL    ;  and clear the  row count
            XOR     A             ;  Clear A
            LD      (MAN),A       ;  Now clear MAN
            LD      HL,ARST       ;  HL  =  start of arm store
            LD      (CURCW),HL    ;  CUROW = start of arm store
            CALL    CTBUF         ;  Clear TBUF, DRBUF & MOTBF
            CALL    RESET         ;  Clear the POSAR array
            CALL    CLRMT         ;  Free all motors
            RET                   ;  EXIT
```

```
SUBROUTINE MOVTC

;  This routine takes the POSAR array and uses it to drive
;  all the motors until the ARM is in its defined start position

MOVTO     PUSH     AF              ;  *
          PUSH     BC              ;  *
          PUSH     DE              ;  *  Save registers
          PUSH     HL              ;  *
RES1      LD       HL,POSAR        ;  HL points to POSAR
          LD       B,12            ;  B = count of 12
NRES1     LD       A,(HL)          ;  Get POSAR element
          OR       A               ;  Is it zero?
          JR       NZ,MTSA         ;  No then continue
          INC      HL              ;  Point to next POSAR element
          DJNZ     NRES1           ;  See if all zero
          JR       ENDSC           ;  All zero so end!
MTSA      LD       HL,POSAR+1Ø     ;  HL points to POSAR
          LD       DE,MOTBF+ 5     ;  DE points to MOTBF
          LD       B,6             ;  B  = count
RSCAN     PUSH     BC              ;  Save count
          LD       C,(HL)          ;  Get lower byte
          INC      HL              ;  Advance HL pointer
          LD       B,(HL)          ;  Get high byte of POSAR element
          LD       A,C             ;  Get low byte into A
          OR       B               ;  See if POSAR(N) is zero
          JR       NZ,DOMPL        ;  no skip
          LD       (DE),A          ;  Zero MOTBF (N)
          DEC      HL              ;  advance POSAR pointer
          JR       NMDR            ;  Do next motor
DOMPL     LD       A,B             ;  See direction to move in
          BIT      7,A             ;
          JR       Z,RMOT1         ;  Go in reverse
          INC      BC              ;  Go  forward
          LD       A,1             ;  A = forward
          JR       DOIT1           ;  Do rest
RMOT1     DEC      BC              ;  Dec count for reverse
          LD       A,3             ;  Set reverse in A
DOIT1     LD       (DE),A          ;  Store reverse in MOTBF (N)
          LD       (HL),B          ;  Store updated POSAR count
          DEC      HL              ;  in POSAR (N)
          LD       (HL),C          ;  Store lower byte
NMDR      DEC      HL              ;
          DEC      HL              ;  point to next POSAR element
          DEC      DE              ;  Move to next MOTBF element
          POP      BC              ;  Restore motor count
          DJNZ     RSCAN           ;  Do for next motor
          CALL     DRAMT           ;  Drive all motors to be driven
          JR       RES1            ;  Do till all POSAR slots zero
ENDSC     POP      HL              ;  *
          POP      DE              ;  *
          POP      BC              ;  *  Restore all registers
          POP      AF              ;  *
          RET                      ;  Return
```

SUBROUTINES TORQUE, CLRMT AND SETDT

```
;  TORQUE switches  of motors on and sets CTPOS(N)'s
;  CLRMT  turns all motors off and sets CTPOS(1-6)
;  SETDT  sets all CTPOS elements to start offset
;  position which equals 1.


TORQUE      PUSH     AF          ;  *  Set clear motor-
            PUSH     BC          ;  *
            PUSH     DE          ;  *  Save Registers
            PUSH     HL          ;  *
            LD       HL,TORMS    ;  Print TORQUE ON message
            CALL     PSTR        ;
            LD       DE,CTPOS    ;  Point to FTABL offset array
            LD       HL,MOTBF    ;  Point to last drive table
            LD       B,6         ;  B = motor count
TORQ1       LD       A,(HL)      ;  Get motor value
            OR       A           ;  Is it zero?
            JR       NZ,TORQ2    ;  No then skip
            LD       A,1         ;  Reset CTPOS(N) to position 1
            LD       (DE),A      ;  in FTABL
            LD       A,B         ;  Get motor address in A
            SLA      A           ;  Shift it left for interface defn
            OR       192         ;  or in FTABL pulse
            OUT      (PORT),A    ;  Output it to selected motor
TORQ2       INC      DE          ;  Advance points to next
            INC      HL          ;  motors
            DJNZ     TORQ1       ;  Do next motor
            JR       TOQCL       ;  Exit with register restoration
CLRMT       PUSH     AF          ;  *  clear all motors torque
            PUSH     BC          ;  *
            PUSH     DE          ;  *  Save Registers
            PUSH     HL          ;  *
            LD       HL,NOTOR    ;  Print "NO TORQUE" message
            CALL     PSTR        ;
            LD       D,ØFØH      ;  Pattern for motors off
OTMT        LD       B,6         ;  B  = Motor count
CLNT        LD       A,B         ;  Get motor address in A
            SLA      A           ;  Shift into correct bit position
            OR       D           ;  Combine with coils off pattern
            OUT      (PORT),A    ;  Output to selected motor
            DJNZ     CLMT        ;  Do next motor
            CALL     SETDT       ;  Clear CTPOS array to value of 1
TOQCL       POP      HL          ;  *
            POP      DE          ;  *
            POP      BC          ;  *  Restore Registers
            POP      AF          ;  *
            RET                  ;  Done, exit
```

```
SETDT       PUSH    BC          ;  *  Set CTPOS elements to start
            PUSH    DE          ;  *  Save used registers
            PUSH    HL          ;  *
            LD      B,6         ;  Motor count to B
            LD      HL,CTPOS    ;  HL points to CTPCS array
NSET1       LD      (HL),1      ;  Set CTPOS(N) to start position = 1
            INC     HL          ;  Increment HL
            DJNZ    NSET1       ;  Do set up next CTPCS element
            POP     HL          ;  *
            POP     DE          ;  *  Restore used registers
            POP     BC          ;  *
            RET                 ;
```

```
SUBROUTINE DRAMT

;  DRAMT drives all six motors directly and uses
;  FTABL to output the correct pulse patterns.
;  For half stepping the pattern must be changed in FTABL
;  and the bounds in DRAMT


DRAMT       PUSH    AF              ;  *
            PUSH    BC              ;  *
            PUSH    DE              ;  *  Save Registers
            PUSH    HL              ;  *
            LD      B,6             ;  B = motor count
            LD      DE,MOTBF +5     ;  Point  to MOTBF array
            LD      HL,CTPOS        ;  HL points to FTABL offset array
NMTDT       LD      A,(DE)          ;  Get MOTBF(N)
            OR      A               ;  Is it zero?
            JR      Z,IGMTN         ;  If zero, then skip
            BIT     1,A             ;  Test direction
            CALL    OUTAM           ;  Step motor
            JR      Z,REVMT         ;  If direction negative then jump
            INC     A               ;  Increment table counter
            CP      5               ;  Upper bound?
            JR      C,NORST         ;  No then continue
            LD      A,1             ;  Reset table offset
NORST       LD      (HL),A          ;  Store in CTPOS (N)
IGMTN       INC     HL              ;  Increment CTPOS pointer
            DEC     DE              ;  Decrement MOTBF pointer
            DJNZ    NMTDT           ;  Do for next motor
            CALL    DELT            ;  Delay after all pulses out
            CALL    DELS            ;  *
            POP     HL              ;  *
            POP     DE              ;  *
            POP     BC              ;  *  Restore Registers
            POP     AF              ;  *
            RET                     ;  Exit
REVMT       DEC     A               ;  Move table pointer on
            CP      1               ;  Compare with lower bound
            JR      NC,NORST        ;  If no overflow then continue
            LD      A,4             ;  Reset table offset
            JR      NORST           ;  Do next motor
OUTAM       LD      A,(HL)          ;  Get table offset 1-4
            PUSH    AF              ;  *
            PUSH    DE              ;  *  Save Registers
            PUSH    HL              ;  *
            LD      HL,FTABL-1      ;  Get table start
            LD      D,Ø             ;
            LD      E,A             ;  DE now equals 1-4
            ADD     HL,DE           ;  Add to FTABL -1 to get address
            LD      A,(HL)          ;  Get motor pulse pattern
            LD      C,B             ;  Get address field in C and
            SLA     C               ;  shift it one to the left
            OR      C               ;  or in the pulse pattern
            OUT     (PORT),A        ;  Output to interface circuitry
            POP     HL              ;  *
            POP     DE              ;  *  Restore Registers
            POP     AF              ;  *
            RET                     ;  Return
```

```
SUBROUTINE STEPM

;  This routine causes all motors that should be
;  stepped to be so, and updates the motors relative
;  positions from their start positions.

STEPM       PUSH   AF           ;  *
            PUSH   HL           ;  *  Save Register
            PUSH   BC           ;  *
            LD     HL,MOTBF     ;  HL points to motor buffer
            LD     B,6          ;  B  =  Count
TRYØ        LD     A,(HL)       ;  Get motor value 3 or 1
            OR     A            ;  Zero?
            JR     NZ,CONTA     ;  No then continue
CONT        INC    HL           ;  Point to next motor
            DJNZ   TRYØ         ;  Do next motor
            POP    BC           ;  *
            POP    HL           ;  *  Restore Registers
            POP    AF           ;  *
            RET                 ;  Exit
CONTA       POP    BC           ;  *
            POP    HL           ;  *  Restore registers
            CALL   DRAMT        ;  Drive motors
            CALL   POSIC        ;  Increment relative position
            POP    AF           ;  *  Restore AF
            RET                 ;  Exit
```

```
SUBROUTINE  DNEWD
;   This subroutine checks to see if any motors are
;   changing direction , if so a delay is inserted
;   into the sequence.


DNEWD   PUSH   AF          ; *
        PUSH   BC          ; *
        PUSH   DE          ; * save used registers
        PUSH   HL          ; *
        LD     BC,6        ; Load BC with count
        OR     A           ; Clear carry
        SBC    HL,BC       ; HC points to previous motor slice
        LD     D,H         ;
        LD     E,L         ; Move HL to DE
        POP    HL          ; Restore current row pointer
        PUSH   HL          ; Save again
        LD     B,C         ;
NCOMP   LD     A,(HL)      ; Get contents of this row
        CP     Ø           ; See if positive or negative
        LD     A,(DE)      ; Get identical previous motor slot
        JP     P,PDIR      ; if positive do for positive motor
NDIR    CP     Ø           ; Compare if both in same
        JP     M,NXTCK     ; direction then skip else
CDDEL   CALL   DELLN       ; delay  and
NCDSG   POP    HL          ; *
        POP    DE          ; *
        POP    BC          ; * Restore registers
        POP    AF          ; *
        RET                ; Now return
PDIR    CP     Ø           ; If previous motor is negative
        JP     P,NXTCK     ; then delay, else do for next
        JR     CDDEL       ; motor slot
NXTCK   INC    HL          ; increment current row pointer
        INC    DE          ; increment lost row pointer
        DJNZ   NCOMP       ; do for next motor
        JR     NCDSG       ; Return with no large (1 sec) delay
```

```
SUBROUTINE SRAMT

;  SRAMT is responsible for updating the TBUF
;  elements and for setting the STRFG if a situation
;  exists where the TBUF array should be stored in the
;  current ARST slot.  This will occur if any motor changes
;  direction or a motor exceeds the allowed slct
;  boundary of -128 to 127.

SRAMT       LD      A,(MAN)      ;  Get manual flag
            OR      A            ;  Is it zero?
            JP      NZ,STEPM     ;  Yes then just step motors
            LD      (STRFG),A    ;  Clear the store flag
            LD      B,6          ;  B  =  motor count
            LD      IX,DRBUF+6   ;  IX = previous direction buffer
            LD      IY,MOTBF+6   ;  IY = current buffer
            LD      HL,TBUF +6   ;  HL = step buffer
NTMOT       DEC     IY           ;
            DEC     IX           ;
            DEC     HL           ;  move pointers
            LD      A,(IY +Ø)    ;  Get current motor direction
            OR      A            ;  No work to do
            JR      Z,NODRV      ;  skip, if so
            CP      1            ;  Reverse
            JR      Z,REVDR      ;  Yes then skip
FORDR       LD      A,(IX+Ø)     ;  Get previous direction
            CP      1            ;  Direction change?
            JR      NZ,CFORD     ;  No then advance TBUF(N) step
            CALL    SETST        ;  Set the store flag
            LD      (IY+Ø) ,Ø    ;  Clear MOTBF element.
            JR      NODRV        ;  Do next motor
CFORD       INC     (HL)         ;  Increment motor step in TBUF
            LD      A,(HL)       ;  Get new value
            CP      127          ;  Check against upper board
            CALL    SETST        ;  Limit reached then store flag
            LD      (IX+Ø),3     ;  Set previous direction
NODRV       DJNZ    NTMOT        ;  Do next motor
            CALL    STEPM        ;  Step motors to be driven
            LD      A,(STRFG)    ;  Examine store flag
            OR      A            ;  Zero?
            JP      NZ,STORE     ;  No then do store operation
            RET                  ;  Exit
REVDR       LD      A,(IX+Ø)     ;  Get previous direction
            CP      3            ;  Direction reversed?
            JR      NZ,CREV1     ;  No then continue
            CALL    SETST        ;  Else set store TBUF in ARST flag
            LD      (IY+Ø),Ø     ;  clear MOTBF element
            JR      NODRV        ;  Do next motor
CREV1       DEC     (HL)         ;  Advance step count in TBUF (N)
            LD      A,(HL)       ;  Get element
            CP      -128         ;  Compare with upper negative bound
            CALL    Z,SETST      ;  Limit reached so set store flag
CREVD       LD      (IX+Ø),1     ;  Set Direction
            JR      NODRV        ;  Do next motor
SETST       PUSH    AF           ;  Save AF
            LD      A,1          ;  Set store flag STRFG
SETSC       LD      (STRFG),A    ;  to one
            POP     AF           ;  Restore AF
            RET                  ;  Continue
```

SUBROUTINE KEYIN

```
;  This routine scans the keyboard checking for
;  the keys '1-6' and 'Q''W''E''R''T''Y' and 'S'
;  and Ø.  It then drives the motors corresponding
;  to the keys pressed.  If in learn mode the
;  sequence is stored.

KEYIN     CALL     CLRMF      ;  Clear MOTBF array
          LD       A,(384ØH)  ;  Get TRS8Ø keyboard byte
          BIT      7,A        ;  See if
          JR       Z,IGDEL    ;  No space key so skip
          CALL     DELT       ;  *
          CALL     DELT       ;  *  Slow motor driving
IGDEL     XOR      A          ;  Clear KEY PRESSED flag
          LD       (KEYP),A   ;
          LD       A,(381ØH)  ;
          BIT      Ø,A        ;  Is the zero key pressed?
          JR       Z,TRYS     ;  No then skip
          JP       NOTNG      ;  Go to do nothing
TRYS      LD       A,(38Ø4H)  ;  See if
          BIT      3,A        ;  'S'  key pressed
          LD       A,(381ØH)  ;  Restore memory value
          JR       Z,TRYN1    ;  No then skip
          LD       A,(MAN)    ;  See if in manual mode
          CR       A          ;
          CALL     Z,STORE    ;  No then store TBUF
          OR       1          ;  Set not finished flag
          RET                 ;  and exit to caller
TRYN1     LD       BC,Ø       ;  Clear MOTBF offset in BC
          BIT      1,A        ;  See if '1' key is pressed
          JP       Z,TRYN2    ;  No then skip else
          CALL     FORMT      ;  Set up motor 1 position in MOTBF
TRYN2     INC      BC         ;  Increment MOTBF offset
          BIT      2,A        ;  See if '2' key pressed
          JP       Z,TRYN3    ;  No skip
          CALL     FORMT      ;  Set second motor forward
TRYN3     INC      BC         ;  Advance offset
          BIT      3,A        ;
          JP       Z,TRYN4    ;  See if '3' key pressed, No skip
          CALL     FORMT      ;  Set forward direction on Motor 3
TRYN4     INC      BC         ;  Increment offset in BC
          BIT      4,A        ;  See if key '4' is pressed
          JP       Z,TRYN5    ;  No then test key '5'
          CALL     FORMT      ;  Do forward direction for Motor 4
TRYN5     INC      BC         ;  Advance offset
          BIT      5,A        ;  Key '5' pressed
          JP       Z,TRYN6    ;  No skip
          CALL     FORMT      ;  Do set up for motor 5
TRYN6     INC      BC         ;  Advance offset
          BIT      6,A        ;  Key '6' pressed
          JP       Z,TRYQT    ;  No then try 'Q'
          CALL     FORMT      ;  Do for motor 6
```

```
TRYQT     LD    BC,Ø          ;  Clear BC offset for motor 1
          LD    A,(38Ø4H)     ;  See if 'Q' key pressed
TRYQ      BIT   1,A           ;
          JP    Z,TRYW        ;  No then skip
          CALL  BACMT         ;  Set motor 1 for backward
TRYW      INC   BC            ;  Advance pointer
          BIT   7,A           ;  See if 'W' key pressed
          JP    Z,TYRE        ;  No skip
          CALL  BACMT         ;  Do backward for motcr 2
TRYE      INC   BC            ;  Advance pointer offset
          LD    A,(38Ø1H)     ;  See if
          BIT   5,A           ;  'E' key pressed
          JR    Z,TRYR        ;  No skip
          CALL  BACMT         ;  Set motor 3 for backward
TRYR      INC   BC            ;  Advance pointer offset
          LD    A,(38Ø4H)     ;  See if
          BIT   2,A           ;  Key 'R' is pressed
          JP    TRYT          ;  No skip
          CALL  BACMT         ;  Set motor 4 backward
TRYT      INC   BC            ;  Advance offset
          BIT   4,A           ;  Is key 'T' pressed?
          JP    Z,TRYY        ;  No skip
          CALL  BACMT         ;  Set mctor 5 backward
TRYY      LD    A,(38Ø8H)     ;  Is the 'Y' key pressed?
          INC   BC            ;  Avarce offset
          BIT   1,A           ;  No key
          JP    Z,SOMEN       ;  'Y' then skip
          CALL  BACMT         ;  Set mctcr 6 for backward
SOMEN     CALL  SRAMT         ;  Step mctcrs, maybe store.
          OR    1             ;  Set zero key not pressed flag
          RET                 ;  Return to caller
NOTNG     LD    A,(MAN)       ;  Zero was pressed so see
          OR    A             ;  if in learn mode
          CALL  Z,STORE       ;  Yes then store
          XOR   A             ;  Set zero flag and
          RET                 ;  Return to caller
FORMT     LD    E,3           ;  Set fcr forward directicn
          JR    SETMT         ;  Do set motor slot in MOTBF
BACMT     LD    E,1           ;  Set fcr reverse direction
SETMT     LD    HL,MOTBF      ;  Point to MOTBF
          ADD   HL,BC         ;  Add in motor offset
          PUSH  AF            ;  Save AF
          LD    A,(HL)        ;  Get byte
          OR    A             ;  See if zero
          JR    Z,DOMOT       ;  Yes then set byte
          XOR   A             ;  Clear
          LD    (HL),A        ;  byte in MOTBF user wants both
          POP   AF            ;  directions clear byte
          RET                 ;  Restcre AF and return
DOMOT     LD    (HL),E        ;  Set byte in MOTBF
          LD    A,1           ;  and set
          LD    (KEYP),A      ;  key pressed flag
          POP   AF            ;  Restcre AF
          RET                 ;  exit from routine
```

```
SUBROUTINE CBTAS

;  This subroutine makes a signed binary value in
;  HL  into arm ASCII String and stcres the string
;  in the lccations pointed tc by lX

CBTAS       PUSH    AF          ;  *
            PUSH    HL          ;  *
            PUSH    DE          ;  *  Save Registers
            PUSH    lX          ;  *
            BIT     7,H         ;  Test sign of number
            JR      Z,POSNO     ;  If zero then positive number
            LD      A,H         ;
            CPL                 ;  Complement number if negative
            LD      H,A         ;
            LD      A,L         ;
            CPL                 ;
            LD      L,A         ;
            INC     HL          ;  Now 2's complement negative
            LD      A,MINUS     ;  Place minus sign in string
PUTSN       LD      (lX+Ø),A    ;  Pointed to by lX
            INC     lX          ;  Advance lX pointer
            JR      CONUM       ;  Do rest cf conversion
POSNO       LD      A,SPAC      ;  Place a space if number positive
            JR      PUTSN       ;  Jump to copy space to memory
CONUM       PUSH    lY          ;  Save lY register
            LD      lY,BTOAT    ;  Point to subtraction table
NUMLP       LD      A,NUMBA     ;  Get ASCII Ø in A
            LD      E,(lY+Ø)    ;
            LD      D,(lY+1)    ;  Get table value
SUBBA       OR      A           ;  Clear carry bit
            SBC     HL,DE       ;  Subtract table value from value
                                ;  input
            JP      C,GONEN     ;  If carry then do for next digit
            INC     A           ;  Inc count (ASCII in A)
            JR      SUBBA       ;  Do next subtracticn
GONEN       ADD     HL,DE       ;  Restore value before last
                                ;  subtraction
            LD      (lX+Ø),A    ;  Store ASCII Number in memory
            INC     lX          ;  Inc memory pointer
            INC     lY          ;  Point to next table value
            INC     lY          ;
            DEC     E           ;  Test if E = Ø
            JR      NZ,NUMLP    ;  No then try for next digit
            XOR     A           ;  Clear A and place in store
            LD      (lX+Ø),A    ;  as EOS = End of string
            POP     lY          ;  *
            POP     lX          ;  *
            POP     DE          ;  *  Restore all saved registers
            POP     HL          ;  *  and
            POP     AF          ;  *
            RET                 ;  Exit
```

```
BTOAT       DEFW    1ØØØØ    ;  Table of subtraction constants
            DEFW    1ØØØ     ;  for conversion routine
            DEFW    1ØØ      ;
            DEFW    1Ø
            DEFW    1.
```

CLEARING AND RESETTING ROUTINES

```
;  CLRMF  clears the MOTBF array

CLRMF     PUSH     BC           ;  *
          PUSH     DE           ;  *  Save Registers used
          POP      HL           ;  *
          LD       HL,MOTBF     ;  Point to MOTBF(Ø)
          LD       DE,MOTBF +1  ;  Point to MOTBF(1)
          LD       BC,5         ;  BC = Count
          LD       (HL),Ø       ;  MOTBF (Ø) = Ø
          LDIR                  ;  Copy through complete array
          POP      HL           ;  *
          POP      DE           ;  *  Restore Registers used
          POP      BC           ;  *
          RET                   ;  Exit

;  CTBUF clears TBUF, DRBUF and MOTBF
;  Note all must be in order

CTBUF     PUSH     BC           ;  *
          PUSH     DE           ;  *  Save Registers
          PUSH     HL           ;  *
          LD       HL,TBUF      ;  HL points to TBUF(Ø)
          LD       DE,TBUF + 1  ;  DE points to TBUF(1)
          LD       BC,17        ;  BC = Count of 17
          LD       (HL),Ø       ;  Clear first element
          LDIR                  ;  Now clear next 17 elements
          POP      HL           ;  *
          POP      DE           ;  *  Restore Registers
          POP      BC           ;  *
          RET                   ;  Exit
```

```
SUBROUTINE GINT

;  This subroutine gets a signed 16 bit integer
;  from the TRS8Ø Keyboard.
;  If a bad number istyped it returns with the
;  Status flag - non zero.
;  The 2's complement number is returned in HL

GINT        PUSH    BC          ;  *
            PUSH    DE          ;  *  Save Registers
            XOR     A           ;  Clear A and carry
            SBC     HL,HL       ;  Zero HL
            LD      B,5         ;  Maximum of 5 characters
            LD      (MIN),A     ;  Clear MIN=Minus Flag
GINT1       CALL    GCHRA       ;  Get a character and display it
            CP      SPAC        ;  Is it a space?
            JR      Z,GINT1     ;  Yes then skip
            CP      NL          ;  Is it a newline?
            JP      Z,PRET1     ;  Done if new line, return zero
            CP      MINUS       ;  A minus number ?
            JR      NZ,POSON    ;  No then see if positive
            LD      A,1         ;  Set minus flag
            LD      (MIN),A     ;
            JR      GINT2       ;  Get rest of number
PCSON       CP      '+'         ;  Is number a positive number
            JR      NZ,NUM1     ;  See if numeric
GINT2       CALL    GCHRA       ;  Get next character
NUM1        CP      NL          ;  Newline?
            JR      Z,NUMET     ;  Yes then exit
            ADD     HL,HL       ;  Double number
            PUSH    HL          ;  Save X 2
            ADD     HL,HL       ;  X 4
            ADD     HL,HL       ;  X 8
            POP     DE          ;  Restore X 2
            ADD     HL,DE       ;  Now add to get X 1Ø
            CP      Ø           ;
            JR      C,ERRN2     ;  If number less than ASCII Ø ERR
            CP      '9' + 1     ;  If number greater than ASCII
            JR      NC,ERRN2    ;  9 then error
            SUB     NUMBA       ;  Number input OK, so make into
            LD      E,A         ;  Binary and
            LD      D,Ø         ;  load into DE
            ADD     HL,DE       ;  Now add to total
            DJNZ    GINT2       ;  Do for next digit
            CALL    PNEWL       ;  Print a new line
NUMET       LD      A,(MIN)     ;  Is number negative?
            OR      A           ;
            JR      Z,PRET1     ;  No then finish off
            LD      A,L         ;  else complement
            CPL                 ;  The value in HL
            LD      L,A         ;
            LD      A,H         ;  (2's Complement)
```

```
                CPL                     ;
                LD      H,A             ;
                INC     HL              ;
PRET1           XOR     A               ;  Clear A and flags
                POP     DE              ;  *  Restore Registers
                POP     BC              ;  *
                RET                     ;  and return
ERRN2           CALL    PNEWL           ;  Print a newline
                LD      A,1             ;  Set A to 1
                OR      A               ;  Clear carry flag
                SBC     HL,HL           ;  Clear HL
                OR      A               ;  Clear carry flag
                JR      PRET2           ;  Return with ERROR CODE
```

```
SUBROUTINE POSDS

;  This routine displays the POSAR array for the
;  user to see how far the arm is from its
;  "Home position"

POSDS       PUSH    AF          ;  *
            PUSH    BC          ;  *
            PUSH    DE          ;  *  Save all registers
            PUSH    HL          ;  *
            LD      HL,POSST    ;  Print "RELPOS="
            CALL    PSTR        ;  String
            LD      B,6         ;  Motor count into B
            LD      DE,POSAR    ;  Point to array containing offsets
NPOSA       LD      A,(DE)      ;  Get lower order byte into
            LD      L,A         ;  L
            INC     DE          ;  Increment memory pointer
            LD      A,(DE)      ;  Get higher order byte into
            LD      H,A         ;  H
            INC     DE          ;  Increment to next number
            LD      IX,NUMAR    ;  IX points to result string
            CALL    CBTAS       ;  Convert HL and leave in (IX)
            LD      HL,NUMAR    ;  Point to result string
            CALL    PSTR        ;  Print it
            CALL    PSPAC       ;  Print a space
            DJNZ    NPOSA       ;  Do for next motor
            CALL    PNEWL       ;  Print a new line, all done
            POP     HL          ;  *
            POP     DE          ;  *
            POP     BC          ;  *  Restore all Registers
            POP     AF          ;  *
            RET                 ;  Now return
```

```
SUBROUTINE POSIC

;  POSIC increments the signed 2's complement 16 bit
;  motor step offset counts.  It does not check for overflow,
;  but this is very unlikely.  The base would need to
;  be rotated about 3Ø times to cause such an event.

POSIC     PUSH   AF          ;  *
          PUSH   BC          ;  *
          PUSH   DE          ;  *  Save registers
          PUSH   HL          ;  *
          LD     B,6         ;  B = motor count
          LD     DE,MOTBF+5  ;  Point to MOTBF
          LD     HL,POSAR+1Ø;  Point to POSAR (relative position)
NPOS1     PUSH   BC          ;  Save motor count
          LD     C,(HL)      ;  Get lower POSAR byte in C
          INC    HL          ;  Point to Higher byte
          LD     B,(HL)      ;  Get higher byte in B
          LD     A,(DE)      ;  Get direction byte from MOTBF
          AND    3           ;  Clear all higher bits from D7-D3
          OR     A           ;  Is it zero?
          JR     NZ,NONZM    ;  No skip
          DEC    HL          ;  Yes then move POSAR pointer back
          JR     NPOS2       ;  and continue with next motor
NONZM     BIT    1,A         ;  Test direction bit
          JR     NZ,RDPOS    ;  Do for reverse direction
          INC    BC          ;  Advance element
          JR     STPOS       ;  Restore 16 bit POSAR element
RDPOS     DEC    BC          ;  Advance negative POSAR element
STPOS     LD     (HL),B      ;  Store higher byte
          DEC    HL          ;  Move pointer to lower byte
          LD     (HL),C      ;  Store lower byte
NPOS2     DEC    HL          ;  Back up POSAR pointer to
          DEC    HL          ;  next motor position slot
          DEC    DE          ;  Backup MOTBF pointer to next slot
          POP    BC          ;  Restore Motor count
          DJNZ   NPOS1       ;  Do next motor
          POP    HL          ;  *
          POP    DE          ;  *  Restore used Registers
          POP    BC          ;  *
          POP    AF          ;  *
          RET                ;  Done, Exit
```

SUBROUTINE STORE

```
;  STORE copies the TBUF array into the locations pointed to
;  by CUROW.  If the TBUF array is completely empty then the
;  copy is not done.  The COUNT and the CUROW variables
;  are both updated, and a check is made to ensure that
;  a store overflow is caught and the user told.

STORE     PUSH    BC          ;  *
          PUSH    HL          ;  *  Save registers
          LD      HL,TBUF     ;  Point to TBUF
          LD      B,6         ;  B = motor count
STEST     LD      A,(HL)      ;  Get TBUF (N)
          OR      A           ;  Is TBUF element zero
          JR      NZ,STOR1    ;  No then do store
          INC     HL          ;  Point to next element
          DJNZ    STEST       ;  Go do next element check
          JR      EXIT        ;  All TBUF zero so exit
STOR1     LD      (IX+Ø),Ø    ;  Clear DRBUF element
          LD      HL,(COUNT)  ;  Get current count value
          INC     HL          ;  Advance it
          LD      A,H         ;  See if over or at 512 bytes
          CP      1           ;
          JP      NC,OVRFW    ;  Yes then overflow
          LD      (COUNT),HL  ;  Put back advanced count
          LD      DE,(CUROW)  ;  Get current row pointer in DE
          LD      HL,TBUF     ;  Get TBUF pointer in HL
          LD      BC,ØØØ6     ;  Count for six motors
          LDIR                ;  Copy TBUF to ARST(1)
          LD      (CUROW),DE  ;  Replace updated row pointer CUROW
          CALL    CTBUF       ;  Clear buffers
EXIT      POP     HL          ;  *
          POP     BC          ;  *  Restore Registers
          RET                 ;  Now return to caller
OVRFW     LD      HL,OVFMS    ;  Print overflow situation
          CALL    PSTR        ;  Message
          CALL    GCHRA       ;  Get response
          CALL    PNEWL       ;  Print a new line
          CP      'D'         ;  User typed a 'D'
          JP      Z,REDO      ;  Yes then clear all
          CP      'S'         ;  User typed an 'S'
          JR      Z,EXIT2     ;  Yes exit with sequence saved
          JR      OVRFW       ;  Bad input,  try again
REDO      CALL    JNIT        ;  Clear all arrays etc
EXIT2     POP     HL          ;  *
          POP     BC          ;  *  Restore Registers
          POP     BC          ;  Throw away return address
          JP      QUES1       ;  Back to main loop
```

```
SUBROUTINE RESET

;  This subrcutine clears the POSAR array


RESET     PUSH    BC            ;  *
          PUSH    DE            ;  *  Save Registers
          PUSH    HL            ;  *
          LD      HL,POSAR      ;  Point to POSAR start
          LD      DE,POSAR+1    ;  Point to next element
          LD      (HL),ØØ       ;  Clear first POSAR element
          LD      BC,11         ;  Eleven more row counts to clear
          LDIR                  ;  Clear POSAR array
          LD      HL,STRST      ;  Print "ARM RESET" message
          CALL    PSTR          ;  and
          POP     HL            ;  *
          POP     DE            ;  *  Restore Registers and
          POP     BC            ;  *
          RET                   ;  Return to caller
```

INPUT/OUTPUT ROUTINES

```
;  PUTCHR prints a character in A

PUTCHR          PUSH    AF          ;  Save AF
                PUSH    DE          ;  Save DE
                CALL    PCHR        ;  Print character in A
                POP     DE          ;  Restcre DE
                POP     AF          ;  Restore AF
                RET                 ;  Done, Exit

;  PSTR prints a string pointed to by HL

PSTR            PUSH    BC          ;  *  Save registers that are
                PUSH    DE          ;  *  corrupted by the TRS8Ø
                CALL    PUTSTR      ;  Print the string
                POP     DE          ;  *  Restore Registers
                POP     BC          ;
                RET                 ;  Done, Exit

:  PSPAC prints a space character

PSPAC           PUSH    AF          ;  Save AF
                LD      A,2Ø        ;  A = Space character
                CALL    PUTCHR      ;  Print it
                POP     AF          ;  Restcre AF
                RET                 ;  Done, Exit

;  PNEWL prints a new line to the screen

PNEWL           PUSH    AF          ;  Save AF
                LD      A,ØDH       ;  A = Newline character
                CALL    PUTCHR      ;  Print it
                POP     AF          ;  Restcre AF
                RET                 ;  Done, Exit

:  SCKBD  Scans the keyboard once and returns, non
;  zero if character found

SCKBD           PUSH    DE          ;  Save DE
                CALL    KBD         ;  See if character is there
                POP     DE          ;  Restore
                RET                 ;  Done, Exit

;  GCHRA gets a character from keybcard and displays it

GCHRA           CALL    GCHR        ;  Get a character
                CALL    PUTCHR      ;  Print it
                RET                 ;  Done, Exit
```

```
CLEAR SCREEN ROUTINE

;     Simple scrolling type screen clear

CLRSC           PUSH      BC        ;  Save used register
                LD        B,16      ;  Get screen row count
UP1RW           CALL      PNEWL     ;  Print a new line
                DJNZ      UP1RW     ;  Do 16 times
                POP       BC        ;  Restore Register
                RET                 ;  Exit
```

```
DELAY ROUTINES

                                        ;  Delay for 1Ø * B + 1Ø M cycles
DELSW           PUSH    BC              ;  Save BC
DELS1           PUSH    BC              ;  Delay for 11 T state
                NOP                     ;  4 T state delay
                NOP                     ;  4 T state delay
                POP     BC              ;  Delay for 11 T states
                DJNZ    DELS1           ;  Do delay times value in B
                POP     BC              ;  Restcre BC
                RET                     ;  Exit
DELS            PUSH    BC              ;  Save BC
                LD      B,2Ø            ;  Set B for Ø.ØØ1 sec delay (apx)
                CALL    DELSW           ;  Do delay
                POP     BC              ;  Restcre EC
                RET                     ;  Exit
DELT            PUSH    BC              ;  Save BC
                LD      B,Ø             ;  Set B for Ø.Ø1 sec delay (apx)
                CALL    DELSW           ;  Dc delay
                POP     BC              ;  Restore BC
                RET                     ;  Exit
DELLN           PUSH    EC              ;  Save BC
                LD      B,2ØØ           ;  Set B for 1.Ø sec delay (apx)
DDDD            CALL    DELSW           ;  Do delay
                DJNZ    DDDD            ;  Do next delay section
                POP     BC              ;  Restore BC
                RET                     ;  Exit
```

FULL STEPPING AND HALF STEPPING THE MOTORS

Two tables are shown below, the first indicates the sequence for full stepping the motors and the second table shows the pulse pattern for half stepping the motors.

FULL STEPPING SEQUENCE

QA	QB	QC	QD	STEP
1	Ø	1	Ø	1
1	Ø	Ø	1	2
Ø	1	Ø	1	3
Ø	1	1	Ø	4

HALF STEPPING PULSE SEQUENCE

QA	QB	QC	QD	STEP
1	Ø	1	Ø	1
1	Ø	Ø	Ø	1.5
1	Ø	Ø	1	2
Ø	Ø	Ø	1	2.5
Ø	1	Ø	1	3.Ø
Ø	1	Ø	Ø	3.5
Ø	1	1	Ø	4
Ø	Ø	1	Ø	4.5

The documental program contains a table FTABL which is shown below. This table contains the step sequence for full stepping also shown below is the new table FTABLH which contains the sequence for half stepping. To use this table (FTABLH) in the program it will be necessary to alter a few lines of code in the DRAMT routine. The comparison with 5 CPI 5 should be changed to a comparison with 9 and the program line LD A,4 should be changed to LD A,8. The table FTABL should now be changed so it appears as FTABLH

FULL STEP TABLE

```
                                        Step number
FTABL           DEFB        192             1
                DEFB        144             2
                DEFB         48             3
                DEFB         96             4
```

HALF STEP TABLE

```
                                        Step number
FTABLH          DEFB        192             1
                DEFB        128             1.5
                DEFB        144             2
                DEFB         16             2.5
                DEFB         48             3
                DEFB         32             3.5
                DEFB         96             4
                DEFB         64             4.5
```

If you compare the table values with the tables on the previous page you will note a difference, this is because QB and QC are exchanged in the above table due to the hardware switching these two lines.

NOTE

REMEMBER WHEN WRITING PROGRAMS DIRECTLY DRIVE THE ARM SO THAT THE QB AND QC OUTPUT BITS SHOULD BE REVERSED, SO THAT THE TOP FOUR BITS ARE:-

D8 = QA
D7 = QC
D6 = QB
D5 = QD