COLNE ROBOTICS

The

Colne Robotics

A R M D R O I D

Construction and Operation Manual

Published by

COLNE ROBOTICS LIMITED
1 Station Road
Twickenham
Middlesex TW1 4LL

CONTENTS

INTRODUCTION

The development of Armdroid I arose as a result of a survey of industrial robots. It became apparent that educationalists and hobbyists were starting to show interest in medium and small sized robotic devices. There was however no robot on sale anywhere in the world at a price suitable to these markets. The Armdroid micro-robot now fulfils this role, providing a fascinating new microcomputer peripheral.

Purchase of the robot in kit form enables the assembler to understand its principles and allows for modification, although of course the machine may also be purchased ready assembled.

This manual has been compiled as a guide to the construction and operation of your Armdroid micro-robotic arm, and should be followed carefully. There are separate sections covering both the mechanical and electronic aspects of the robot, as well as the specially written software.

M E C H A N I C S.

MECHANICS

2.1 Description

The ARMDROID consists of five main parts.

The base

The base performs not just its obvious function of supporting the rest of the arm. It also houses the printed circuit boards and the motor that provides the rotation.

The Shoulder

The shoulder, which rotates on the base by way of the main bearing, carries five motors and their reduction gears which mesh with the reduction gears on the upper arm.

The Upper Arm

The lower end of the upper arm carries the gears and pulleys that drive the elbow, wrist and hand. It rotates about a horizontal axis on the shoulder.

The Forearm

The forearm rotates about a horizontal axis on the upper arm and carries the wrist bevel gears.

The Wrist and Hand

The two wrist movements, the rotation about the axis of the hand ("twist") and the rotation of the hand about a horizontal axis ("up and down"), depend on a combination of two independent movements. The twist is accomplished by rotating both bevel gears in opposite directions, while the up and down movement is done by turning the gears in the same direction. Combinations of the two movements can be got by turning one bevel gear more than the other.

The three fingered hand with its rubber fingertips has a straightforward open and shut movement.

2.2 Technical Hints

1. FITTING BELTS ONTO PULLEYS

Fit belt over small pulley first and then work onto unflanged edge of large pulley a little at a time - do not attempt to get belt fully onto pulley until you have got it on by one or two millimetres all round. (Belts can be damaged if they are crimped). When fitted belts should not be drum tight there should be just a little play, or friction will rear its ugly head again.

2. FITTING SWITCHES

On initial fitting do up bolts only enough to hold switches in position. Finally after gears are fitted swing switches so that they clear gears by approximately one millimetre and finally tighten.

3. FITTING PULLEYS TO MOTORS

You will find the motor shafts have end float with a light spring action pulling the shaft in. Do not pull shaft out against this spring when fitting pulley as this will cause friction and loss of effective motorpower.

4. LUBRICATION

Use light oil (three in one or similar), just a drop on all parts that slide or pivot. DELRIN is a self lubricating material but the friction is a lot lower with a drop of oil. We only have limited power from the motors so we want to make the most of it, so work spent on eliminating friction will pay performance dividends. Check all bores and bearings for free running - any tightness is usually caused by burrs or stray bodies in bores. Remove burrs from Delrin with a sharp knife, from metal with a scraper.

Disposable hypodermic is ideal for lubricating - scrounge one from your local friendly GP or Hospital.

REED SWITCH POLICY

Micro-switches are included in the assembled and unassembled Armdroid packages as optional extras. It must be stressed, however, that the machine will function perfectly well without the micro-switches, but a check must be kept on the number of complete revolutions of the base. Any more than 1.5 turns will put a strain on the stepping motor leads where they connect to the printed circuit boards.

To prevent any difficulty in the fitting of reed-switches after the initial assembly the magnets will be inserted during manufacture. This will save the dismantling of the Armdroid in the field. Magnets will be included in all the kits.

There will be a nominal charge of £15 for the inclusion of reed-switches in both the assembled and unassembled Armdroids.

PART NUMBERS INVOLVED: *O9*1O*15*16*18/16*18/12*

2.3 TOOLS LIST INC. Lubricants etc

General and small circlip pliers

7mm spanner	supplied
5.5mm spanner	supplied

Metric steel rule, (part identification)

Hypodermic syringe or small oilcan and 3 in 1 oil

"Superglue" and if possible "Loctite"

Cold vaseline or cycle bearing grease

Tweezers

Allen keys for	M3 grub screws -	supplied
	M4 grub screws -	supplied
	M4 bolts -	supplied

Lightweight hammer (fitting rollpins)

2.4 ASSEMBLY

Description of item	Part No
Base	01
Base Bearing support column	02
Base motor	03b
Base motor short pulley 20 tooth	04b
Base reduction gear spindle	05
Turned thick wide washer 16mm x 2mm	06
Reduction gear	07
Base belt (medium length) 94 teeth	08m
Base switch support 12mm x 11mm	09
Base switch	10
Shoulder pan	11
Shoulder bearing ring	12
Base gear (large internal dim)	13
Bearing adjusting ring	14
Hand motor support bracket	15
Hand motor	03h
Hand switch bracket	16
Motors - Upper arm	03u
Fore arm	03f
Wrist action	03w
Motor pulleys - Upper arm	04u
Fore arm short 14 tooth	04f
Wrist action long 20 tooth	04w
Hand short 20 tooth	04h

DESCRIPTION OF ITEM	Part No	
Shoulder Side Plates	017	
Switch support bar 107mm x M3 at ends	019	
Support bar spacers M3 clearance X	018/16	
	018/12	
Motor support bracket stiffener 107mm x M3 at ends	019	
Support Bar spacers	018/54	
	018/41	
Reduction gears	020	
Reduction gear spindle 96mm x 6mm	021	
Drive belts long = 114 teeth	08/l	Hand
medium = 94 teeth	08/m	Fore/Upper arm
short = 87 teeth	08/s	Wrist action
Upper Arm Drive Gear small internal dim no drum	021	
Upper arm side plates	022	
Upper arm brace	023	
Gears wrist action	024	
hand action	025	
fore arm	026	
Idler pulley	027	
Shoulder pivot 96mm x 8mm spindle	029	
Fore arm side plates	030	
Fore arm brace	031	
Fore arm pulley	032	

DESCRIPTION OF ITEM	Part No.
Elbow Idler pulleys hand	
wrist	033
Elbow spindle 65mm x 6mm	034
Wrist bevel gear carrier	035
Wrist guide pulleys	036
Wrist bevel gears (flanged)	037
Wrist pivots .	038
Hand bevel gear (no flange)	039
Finger support flange	040
Hand pivot	041
Finger tip plates	041
Finger cable clamp	042
Small finger spring	043
Finger tip pivot	044
Middle finger plates	045
Middle finger pivot	046
Large finger spring	047
Finger base	048
Long finger pins 16mm x 3mm	050/l
Short finger pins 13mm x 3mm	050/s
Small finger pulleys	051
Large finger pulleys	052
Large hand sheave pulley	053
Small hand sheave pulley	054
Hand sheave pin	055
Finger tip pads	056
Base pan	057

DESCRIPTION OF ITEM	Part No.
Board Spacers	018/41/54
Spacer bars for boards	058
Rubber feet	059
Cable springs wrist action short	060
Cable springs grip, elbow long	061

PREPARATION AND FIXINGS ETC

DESCRIPTION OF ITEM	Item No.
Magnets	101
Bearing adjustment ring grub screws M4 x 8mm NB + self made plug to protect the fine bearing thread	102
Turned cable clamps 6 x 6mm M3 tapped	103
Cable clamp grub screws M3 x 4 pointed head	104/105
Crimped type cable clamps	
crimped eyelets	106
Gear Cable grub screws M4 x 6mm flat head	107
Bushes 8mm bore long with flange shoulder	108
Shoulder pivot spindle spacer	108a
6mm bore short with flange - elbow	109
8mm bore long with flange wrist	110
8mm bore no flange main gear inserts	111
Gear to sheet metal screws M3 x 6 slot hd CSK	112
Spring pillar and base switch M3 x 10 cheese head	113
Base bearing to shoulder pan M4 x 16 CSK socket head	114

DESCRIPTION ITEM	Item No.
Motor bolts, Base bearing to base M4 x 10 Elbow spindle hex hd	115
Hand to finger, hand to bevel gear M3 x 6 cheese hd	116
Shoulder spindle M5 x 10 hex hd	117
General sheet metal fixing M3 x 6 hex hd	118
M4 Nuts	119
M4 Washers	120
M4 Shakeproofs elbow spindle	121
M5 shakeproofs shoulder spindle	122
M3 Nuts	123
M3 washers - switches	124
6mm steel balls - base bearing	125
Magnetic reed switches	010*(101?)*
Driver board	126
Interface board	127
Edge connector	128
6mm Washers	129
Roll pins	130
4.5mm circlips	131
3mm circlips	132
Elbow spacer	133

2.5 ASSEMBLY

Preparation

Study the parts list, drawings and the parts themselves until you are sure you have identified them all. Assemble the tools suggested in the list of tools (2.3). Read carefully technical hints section. Solder 12 inches of ribbon cable to each motor. Glue magnets (101) into the slots in the reduction gears, noting that the hand gear (25) needs no magnet. Check that the adjusting ring (14) of the main bearing screws easily onto its base. Clean both if necessary. Insert bushes into the arms, if necessary using a vice, but taking care not to distort the sheet metal.

Construction

Fit base bearing support (2) column inside base (1). (M4 bolts, nuts.) NB NUTS INSIDE BASE

Bolt 1 motor (shorter cable) inside base. (M4 hex bolts, washers on motor side - nuts on inside). Fit pulley to spindle base of motor with the grub screw at the top (04b). Fit base reduction gear spindle (07) to base. (Thick turned washer, M4 hex bolt) Fit reduction gear and belt. Place a small drop of oil on the reduction gear spindle before fitting reduction gear.

When fitting belts they should be placed fully on the motor spindle and worked gently onto the reduction gear, a small section of their width at a time. (see general hints on lubrication)

Fit base switch support. (M3 hex bolt) NB DRAWING FOR POSITION. Fit base switch and run wires through adjacent hole. (M3 x 10 cheesehead, washer)

Fit bearing ring (12) (long spigot down) through shoulder base pan (11) from inside. The base gear (13) fits on the lower face of the pan, with the magnet at 2 o'clock as seen from inside the pan with the flange at the top. (M4 countersunk x 16mm bolts, nuts)

(This step and the next are simpler with some help from an assistant). Put shoulder base pan (gear side up) on to 3in supports (books etc,) so that the bearing support column can be inserted. Practise this movement to make sure all is well. Smear vaseline from a fridge, or grease on the bearing track of the flange, and using tweezers to avoid melting the vaseline carefully place 24 ball bearings round the flange, embedding them into grease. There will be a slight gap when all the balls are in place. Invert the base and insert the threaded bearing support column inside the bearing ring taking care not to dislodge any of the balls so that the base pan meshes with the base gear. Keep the two parts level in the same relationship by taping the parts together with a piece of wood or a spanner 5mm thick between the motor pulley and the shoulder base pan.

Large rubber bands can be used instead of tape. An assistant to hold the parts for you will be useful here.

Turn the assembly the other way up (the base is now on the bench with the shoulder base pan above it. Put more grease round the bearing track and embed 24 more ball bearings in it. Gently lower the adjusting ring (14) on to the threaded base and then screw the finger tight, remove with tape, adjust the ring until the base pan moves freely without play then tighten the grub screw, inserting a small wood plug to protect the bearing thread. (M4 grub screws)(102). The bearing may need adjusting after some use as it beds in.

Fit hand motor bracket (15) to shoulder base pan (M3 bolts) then the hand motor O3h(M4) and the hand motor pulley. Then fit the hand reed switch-bracket (M3) and the switch (M3 x 10 cheesehead bolts).

Fit motors to the shoulder side plates (17) and feed the cables through the holes towards the inside. The bolts which are next to the reduction gears should be placed nut out to prevent the reduction gears catching on the end of the bolts. Fit correct pulleys (04u/f/w) to the motor spindles noting which pulleys from the drawing, tighten the grub screws.

Fit the shoulder plates. This is simplified by loosely tightening the end bolts to support the weight. Feed the motor cables down through the main bearing (M3).

Slide switch support (19) bar through spacers (18), switches (101) and motor support bracket (see drawing for correct order of spacers). You will need to be able to adjust the position of the reed switches after the arm is fitted so that they clear the gear wheels ie tangential to shoulder pivot. Fit the motor support stiffener bar and spacers. Leave nuts finger tight. (M3 nuts).

Assemble reduction gear support bar (21), assemble with the correct length drive belts (08s/m/l) over each gear, reduction gears facing in correct direction and the thin metal M6 washers at either end. (see drawing) Slide gently into position and bolt in the support bolts (M4 a 10mm) Fit the belts round the motor pulleys.

Put upper arm drive gear on the outside of the upper arm side plate. The magnet should be at 1 o'clock, viewed from the gear side of the arm. (M3 CSK screws x 6mm) Fit a brace to one upper arm side piece (22), then fit the other side piece to the brace. Fit all bolts and nuts before tightening any of them. Check 8mm shoulder spindle (29) slides freely through accute bushes in upper arm side pieces and through bores of drive gears, pulleys and spacers. Assemble by sliding shaft from one side and threading gears, pulleys and spacers on in the correct order of orientation - use drawing.

Fit pulley (32) to the outside of the forearm side plate (30) (M3x6mm)(countersunk screws). Fit a brace to one forearm side plate, then fit the other side plate to the brace. Check for squareness before finally tightening bolts.

Put elbow pivot through bushes and an 8mm bar through wrist bushes. (M3 bolts, nuts) Check fit before assembly. Assemble the pulleys (33) on the elbow spindle (34), lubricate and fit it to the large arm, and bolt through into spindle. (M4 bolts, washers)

Assemble the wrist bevel gear carrier (35) and wrist pulleys (36) and then tap the roll pins gently home with a small hammer, supporting aluminium gear carrier to prevent distortion.

Fit the wrist gears on the bushes (37) from the outside. Fit bevel gear carrier (35) between the wrist bevel gears (37), line up holes in end of wrist pivot (38) bores with tapped hole in carrier by peering down pivots. If you do not have a screw gripping or magnetic driver use a little piece of masking tape or sellotape to fix M3 cheesehead screw to the end of your screwdriver in such a way that it will pull off after tightening - check gears pivot freely on pivots and that the whole assemble can pivot in oilite bushes (drops of oil on faces of gears and pivots)

Screw the finger support flange (40) to the hand bevel (39). (M3 x 6mm cheesehead screws) Screw the hand pivot (41) to the bevel gear carrier (35). Tighten on a drop of loctite if available, gently by turning a pair of pliers inside it. The bevel gears should be positioned with their grub screws pointing towards the hand when the hand and the forearm are in line (see drawing).

Assemble the fingertip (42) and cable clamp (43) with the small spring (44) on the pivot (45), and clip together with large circlips on the cable clamp. The spring should be positioned so that the "back" of the spring is on the knuckle side of the fingertip, thus tending to open the hand.

Assemble the middle finger (46) and its pivot (47) with the large spring (48). Fix to the finger base (49) with the long pin (50/L) (16mm x 3mm) and two small circlips (see drawing). Fix one circlip to the pin before one begins to assemble.

Join the fingertip to the middle section with the short pin (50/S) (13mm x 3mm) and two small circlips.

Cut off one end of the tip spring about 8mm-10mm beyond its hole. Level with its hole bend the spring through a right-angle to secure it. Repeat at the other end. Trim the inner end of the middle finger spring flush with the end of the finger end and treat the outer end as above.

Fit the small pulley (51) to the finger middle section using a short pin (13mm x 3mm) and two small circlips. Fit the larger pulley (52) to the finger base with a long pin (16mm x 3mm) and two small circlips.

Screw the finger base to the finger support flange. Make sure that the fingers are evenly spaced and do not interfere with each other, and then tighten. (M3 x 6mm cheesehead)

Assemble the large and small hand sheave pulleys using the large circlip on hand sheave pin (55).

CABLE THREADING

Slide arm into shoulder, you will need to align the reduction pulleys between the main drive gears as you lower the arm into place, and assemble using M5 hex head bolts and shakeproof washers. Tighten and check the reduction gears "mesh" correctly and the arm moves freely.

Connect grip action cable tail to shoulder base pan via the spring correctly placed over the pulley and tension using the normal method with the cable clamp.

Glue strips of rubber to finger tips using superglue.

The driver and interface board should be bolted to the base pan using the spacer bars (58) and spacers. Bolt base pan (57) to base (M3 x 6mm hex head).

Hints: Useful tools are:

a) 2 or 3 'bulldog clips' to maintain the tension in the cable over completed sections of each cable while the remainder is threaded. Masking tape can also be used for this purpose,

b) Ends of the cable can be prevented from fraying by placing a drop of 'superglue' on the end of area where it is to be cut. The excess should be wiped off on a piece of paper.

NB. This process also stiffens the end which is useful when threading the cable through the pulleys.

c) Ensure all grub screws are in position but are not obstructing the cable holes. Also check there are no burs remaining from machining blocking the holes.

d) The cables can be threaded before the arm is bolted for the shoulder which eases the problems of access considerably. The 'grip action' cable tail can be taped or clipped to the arm and connected and tensioned with its spring after the arm is fitted to the shoulder,

e) When tensioning the cable, if it is passed through the clamp and back, then connected to the spring adequate tension can be applied by pulling the 'free tail' and then nipping it with the grub screw. A friend will be useful if around, but it is quite possible without. The correct tension can be easily judged, as when completed the coils of the spring should be just separated, though this is not critical.

f) During threading the correct 'route' can be ascertained from the expanding drawings. It is very important these should be followed exactly, especially the position of the grub screws when they are tightened on the cable. If this is wrong it will affect the performance of the arm.

g) Care should be taken to avoid the cable kinking or crossing itself on the drums.

h) Experience has shown that the best order to thread the cables and lengths to use. (Excess can be trimmed easily later but makes tensioning simpler)

First	-	Wrist cables one at a time.	1.47m (each)
Second	-	Elbow cable (set up the spring pillar first - M3 x 10mm cheesehead and 2 M3 hex full nuts) attach crimped cable clamp to forearm first using M3 x 10 cheese head and two nuts as a cable pillar.	0.95m
Third		Single finger cable (fix to the hand sheave pulley using M3 x 6mm cheesehead and crimped cable clamp.	0.18m
Fourth	-	Double finger cable (loop over small hand sheave pulley on grip action pulley and adjust so that G A P is even when pulleys are evenlypositioned).	0.36m
Fifth	-	Grip action cable (start at end fixed in cable drum and stick other end to arm while fitting it to the shoulder then tension with the spring to the shoulder base pan).	1.3 m

i) Ends using the crimped cable eyelets should be threaded through the eyelet and a small thumb knot tied to prevent the cable slipping before crimping the bracket using crimping or ordinary pliers. So not crimp too tight or you may cut through cable, though KEVLAR is very tough,

E L E C T R O N I C S

ELECTRONICS

3.1 Description

The Interface

To enable the Armdroid to function with as wide a range of microprocessor equipment as possible, the interface is designed round a standard 8-bit bidirectional port. This may be latched or non-latched. If non-latched, the interface will normally be used to input data to the micro.

In the output mode the port is configured as follows. The eight lines are defined as four data bits (D8-D5), three address bits (D4-D2) and one bit (Dl) to identify the direction of data travel on the port. Four data lines are provided so that the user can control the stepper motor coils direct from computer.

The address bits are used to channel the step pattern to the selected motor. The three address bits can define eight states, of which 1-6 are used to select one of the motors, while states 0 and 7 are unallocated.

Dl indicates the direction of data travel, to the motors when Dl is low, from the microswitches, if installed, when Dl is high. The transition of Dl from high to low generates a pulse which causes the step pattern to be latched into the addressed output latch.

In the input mode D8 - D3 are used to read the six microswitches on the arm. These reed switches and magnets provide a "zero" point for each of the movements of the arm, which can be used as reference points for resetting the arm in any position before a learning sequence begins.

D2 is spare. It is an input bit which can be buffered and used for an extra input sensor, allowing the user to connect a 'home brew' transducer to the system.

The interface circuitry consists of twelve TTL components which decode the data and route it out to the selected motor driven logic. ICl and IC2 buffer the data out to the decoder and latches. IC6 decodes the three input address bits to provide eight select lines, six of which are for the latches IC7 - IC12.

INTERFACE ONLY

Dl is buffered and fed into a monostable (IC4) to generate a clock pulse. This causes the decoder to provide a latch pulse for approximately 5OOns to the addresses motor control latch. Dl is tied to pull-up resister (Rl) so that the line is high except when are output from the microprocessor. The buffers ICl and IC2 are enabled by the buffered output of bit 1 so that data are fed to the latch inputs only when bit 1 is low. The bit 1 buffer is always enabled because its enable is tied low.

The microswitch inputs are buffered by IC5 which is enabled by the complemented output of bitl, so that when bitl is high IC5 is enabled, and the contents of the microswitches will be input to the microprocessor. This allows the user to operate the arm under bit interupt control, giving instant response to a microswitch change and avoiding having to poll the micro-switches. The six microswitch inputs are pulled up; thus the switches can be connected via only one lead per switch, with the arm chassis acting as ground.

THE MOTOR DRIVERS

the motor drivers are designed so that the arm can be driven from the output of the computer interface circuitry.

The six motor driver stages need two power supplies: 15v at about 3A and 5v at 150 MA.

The four waveforms QA-QD are then fed into IC's 13-16 which are 7 x Darlington Transistor IC's. These provide the high current needed to drive the stepper motor coils, the driving current being about 300 MA at 15v.

INTERFACE DRIVER BOARD

ITEM	VALUE	QUANTITY
Resistors		
Rl	1K0	
R2	1OK	
R3-8	2K2 resitor network	1
R9	1K8	
RIO	1K8	
Rll	1K8	3
R12	15K	1
R13	1OK	2
R14	18ohm 5w	1
R15-R2O	1KO	6
Capacitors		
Cl	lOOp polystyrene	1
C2	l.Ovf Tant	1
C3-C15	lOnf ceramic	13
Semiconductors		
IC1	74LS 125	
IC2	74LS 125	
IC3	74LS 04	
IC4	74LS 123	
IC5	74LS 366	
IC6	74LS 138	
IC7-IC12	74LS 175	
IC13-IC16	ULN2oo3A	
IC17	UA 7805	
ZD1	BZX 13v ZENER	

Miscellaneous

MXJ 10 way edge connector
5 way PCB plug and socket connector
Through Pins
16 pin IC sockets
14 pin IC sockets
4 way modified PCB plug and socket

GENERAL ASSEMBLY SEQUENCE FOR THE PC BOARD

A Fit all of the through pins to the board.

B Fit and screw the 5v regulator to the board.

C Identify and fit the resistors and the 13v zener to the board. The black band v points to the motor connectors (on the zener DIODE).

D Identify and fit all capacitors to the board.

E Solder the 2k2 resistor network, IC sockets, and the 4 and 5 way PCB plugs to the board.

G Solder the 10 way socket to the board.

NOTE:

Refer to the overlay diagram and parts list to ensure that the resistors, capacitors, IC,s and other parts are inserted into the correct locations on the PC Board.

BASIC BOARD CHECKS

A Check the board for dry joints and re-solder any found.

B Hold the board under a strong light source and check the underside to ensure there are no solder bridges between the tracks.

FITTING THE PC BOARD TO THE BASE OF THE ROBOT

The PCB should be fitted to the base plate using the nylon pillars provided.

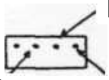
MOTOR CONNECTION

Connect the motors to the 5way sockets, ensuring correct 15v polarity, via the ribbon cable, refering to the diagram provided to ensure correct connection.

POWER CONNECTION

Connect the power to the modified 4way socket ensuring correct polarity as shown below.

Polarising pin



Blue - Pin 1 on I/P connector=Ov

15v = Brown = Pin 2 on I/P connector

NOTE

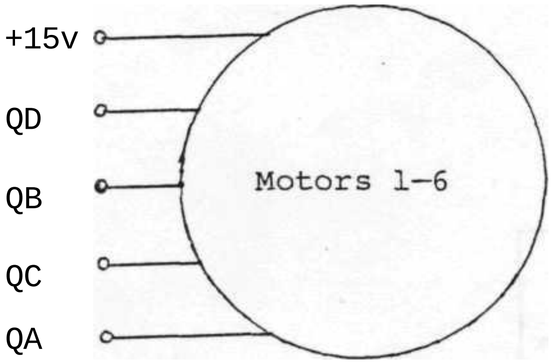
A number of diagrams are given, explaining in detail the intern-connections between the motors and the PCB, if the motors are connected in the manner shown then the software provided will map the keys 1-6 and q,w,e,r,t,y to the motors in the following way

1, q, = GRIPPER. 2, w, = left wrist. 3, e, = right wrist.

4, r, = forearm. 5, t, = shoulder. 6, y, = base.

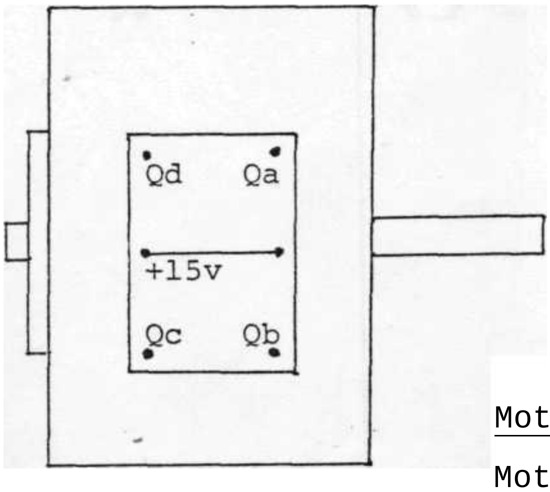
as shown in the diagram, the two middle pins of the stepper motors should be connected together and to 15v.

Motor Connection And Designation Layouts



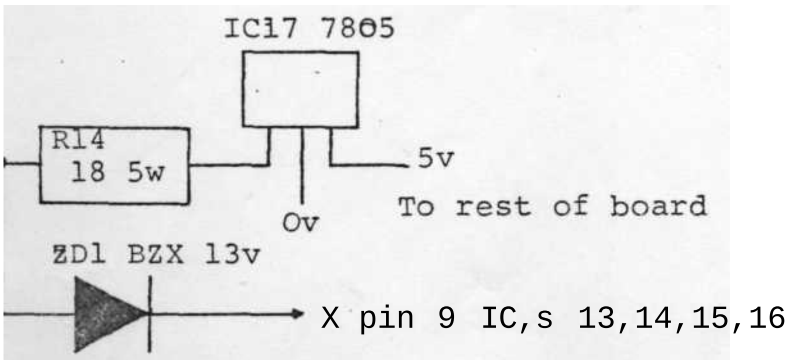
Ribbon Cable To Stepper Motor Connections

Qa Black or Green
Qb Red or Purple
Qc Brown or Blue
Qd Orange or Grey
+15v Yellow or white



Motor Assignments To Functions

Motor 1 = Grip
Motor 2 = Left Wrist
Motor 3 = Right Wrist
Motor 4 = Elbow
Motor 5 = Shoulder
Motor 6 = Base



S O F T W A R E.

4. SOFTWARE

4.1 Introduction

A machine code program, LEARN , to drive the ARMDROID has been specially written. It was designed for the Tandy TRS-8O Model 1 Level 11, and the loading instructions given here apply to that computer. But the program can be easily adapted to any Z80 microprocessor with the necessary port, and versions made available for the leading makes with variations of these instructions where appropriate. But of course users can write their own software in whatever language they choose.

4.2 Loading

When in Basic type SYSTEM, press ENTER, answer the '*' with LEARN and then press ENTER again. The cassette tape will take about 1.5 minutes to load. Answer the next '*' with / 17408 and press ENTER.

4.3 General Description

LEARN is a menu-oriented program for teaching the ARMDROID a sequence of movements which it will then repeat either once or as many times as you like. The program is divided into four sections, one for learning the sequence and for fine-tuning it, one to save the sequence on tape and load it again , one for moving the arm without the learning function, and finally two exit commands.

We suggest that, if this is your first encounter with the program, you should read quickly through the commands without worrying too much about understanding all the details. Then go to Section 4.5 and follow the 'Sequence for Newcomers'. This will give you a good idea of what the program does. After that you can begin to discover some of the subtleties of planning and fine-tuning sequences of movements.

4.4 Explanation

L(EARN)

Stores a sequence of manual movements in memory. The arm is moved using the commands explained under M(ANUAL) . You can exit the command by pressing 0 (zero) , press G(0), and the arm will repeat the movement you have taught it.

On pressing L(EARN) you will be asked whether you want to S(TART) again or C(ONTINUE) from the current position. The first time press S(TART) . The arm is then free to be moved by hand without the motors' torque preventing you. Move it to a suitable starting position, then press the space bar. You will find that you cannot now move the arm by hand.

To add a sequence already in memory press C(ONTINUE) instead of S(TART).

Using the manual commands, move the arm to another position. As it goes the computer is adding up the steps each motor is making, either forward or back, and storing the data in memory. (holding the space bar down during manual control slows the movement)

Exit by pressing 0 (zero).

D (ISPLAY)

Displays the sequence stored in memory. The sequence can be edited with the E(DIT) command.

The six columns of figures correspond to the six motors, and the order is the same as that of the 1-6/Q-Y keys (see M(OVE). The first row (RELPOS) shows the current position. Each row represents a stage of the movement, and the actual figures are the number of steps each motor is to make, positive for forward, negative for reverse. The maximum number of steps stored in a row for one motor is +127 or -128, so if a movement consists of more than this number it is accomodated on several rows.

Movements of the arm can be fine-tuned by editing (see E(DIT)) the figures on display until the arm is positioned exactly.

Scrolling of the display can be halted by pressing 0 (zero). To continue scrolling, press any other key. To display the figures one after the other, keep pressing 0.

E(DIT)

Allows the user to change the figures in the memorised sequence.

Truncate a sequence by pressing R(0W COUNT), then ENTER, then the number of the last row you want performed, and finally ENTER. This clears the memory from the next step onwards, so you should only do this if you do not want the rest of the sequence kept in memory.

By pressing M(OTOR STEP), you can change any of the numbers in any row and column.

S(ET ARM)

Sets the current position of the arm as the 'zero' or starting position.

When pressed from the Menu, it simply zeroes the first row of the display.

S(ET ARM) has another function. During a L(EARN), pressing S(ET ARM) at any moment when the arm is at rest will ensure that the movements before and after are separated from each other instead of being merged. This is the way to make quite sure that the arm passes through a particular point during a sequence. Try the same two movements without pressing S(ET ARM), and note the difference in the display.

It is important to realise that, if a sequence has been memorised and S(ET ARM) is pressed from the Menu when the arm is not in its original starting position, pressing G(0) will take the arm through the sequence but from the new starting point. This can be useful for adjusting the whole of a sequence (perhaps slightly to right or left), but it can lead to the arm running into objects if the new starting point is not selected with care.

W(RITE)

Writes a memorised sequence to cassette tape.

R(EAD)

Reads a previously written sequence from cassette tape into memory.

C(HECK)

Compares a sequence written to cassette tape with the same sequence still in memory, to verify the tape.

G(0)

Moves the arm through a memorised sequence, either once or repeatedly.

It is important to make sure that the starting point in memory is the right one, or the sequence may try to take the arm into impossible positions, (see S(ET ARM)

T(0 START)

Takes the arm back to the zero or starting position.

F(REE)

Removes the motors' torque from the arm, thus allowing it to be moved by hand.

M(ANUAL)

Gives the user control of the movements of the arm direct from the keyboard. It is used (a) for practising manual control before L(EARN)ing, (b) for trying new combinations of separate movements, and (c) for moving the arm to a new starting position before pressing S(ET ARM). Holding the space bar down slows the movement by a factor of about 3.

The motors are controlled with the keys 1-6/Q-Y. The keys operate in pairs, each pair moving a motor forwards and backwards. Any combination of the six motors may be moved together (or of course separately), but pressing both keys of a pair simply cancels any movement on that motor.

The geometry of the arm is designed to give the maximum flexibility combined with maximum practicality. A movement of one joint affects only that joint: with some designs one movement involuntarily produces movement in other joints.

It is a feature of the ARMDROID that it has a so-called 'parallelogram' operation. Starting with the upper arm vertical, the forearm horizontal and the hand pointing directly downwards, the shoulder joint can be rotated in either direction and the forearm and hand retain their orientation. Equally the forearm can be raised and lowered while leaving the hand pointing downwards. Moving the arm outwards and down by rotating both the shoulder joints together still leaves the hand vertical. This is of vital importance for simplifying the picking and placing of objects.

The motors controlled by the keys are:

1/Q: Gripper
2/W: Wrist left
3/E: Wrist right
4/R: Forearm
5/T: Shoulder
6/Y: Base

B(OOT)

Returns the computer to the program start and clears the memories.

Q(UIT)

Returns the computer to TRS8O System level.

ARM TRAINER MK2AL

DIRECT FULL STEP MOTOR CONTROL

FOR TRS80 MODEL 1, LEVEL 11

BY ANDREW LENNARD

*** July 1981 ***

S E C T I O N 1

A S Y S T E M E Q U A T E S

B S Y S T E M V A R I A B L E S

C S Y S T E M C O N S T A N T S

4.5 INTRODUCTORY DEMONSTRATION SEQUENCE

1. After loading the program, the screen shows the menu. Press L to enter L(EARN).

2. Screen: START AGAIN OR C(ONTINUE) FROM PRESENT POSITION, (.) TO EXIT. Press S

3. Screen: " ARM RESET
   ARM NOW FREE TO MOVE
   TYPE SPACE BAR WHEN READY, OR FULL STOP TO EXIT"
   Now move the arm so that both arm and forearm are vertical with the hand horizontal. For coarse movements grasp the forearm or upper arm and move it. For fine adjustments and for movements of the hand, it is better to use the large white gear wheels in the shoulder joint. Press the space bar and the arm will become rigidly fixed.

4. Screen: "*** TORQUE APPLIED ***"
   You can now move the arm using the 1-6/Q-Y keys as explained in the manual section. Try just one movement alone at first. Now press 0 (zero) to exit from L(EARN). The arm will return to the starting position, and the Menu appears on the screen.

5. Screen: Menu. Press D for D(ISPLAY).

6. Screen: Display and Menu. The numbers of steps you applied to each motor have been memorised by the computer, and these steps are now displayed see D(ISPLAY) section for explanation. Press G for G(0).

7. Screen: "DO (F) OREVER OR (O) NCE?. Press O (letter O), and the arm will repeat the movement it has learnt.

8. Screen: "SEQUENCE COMPLETE" and Menu. Press L.

9. Screen: as 2 above. This time press C. Now you can continue the movement from this position, using the 1-6/Q-Y keys as before. Now press 0 (zero). Again the arm returns to its original position.

10. Screen: Menu. Press D

11. Screen: Display and menu. Your new movement has been added to your first. Press G.

12. Screen: as 7 above. This time press F. Each time a sequence is started a full point is added to the row on the screen. To stop press full point.

This is a very simple demonstration of how complex movements can be built up, learnt as a sequence and then repeated endlessly and with great accuracy.

SYSTEM EQUATES

```
PORT   EQU  0 4    ;  ARM PORT NUMBER
FINAD  EQU  O2B2   ;  SYSTEM RESTART
PCHR   EQU  0033H  ;  SYSTEM PRINT CHARACTER
GCHR   EQU  0049H  ;  SYSTEM GET CHARACTER
KBD    EQU  002BH  ;  SCAN KEYBOARD
PUTSTR EQU  28A7H  ;  SYSTEM PRINT STRING
CASON  EQU  0212H  ;  CASSETTE ON
CASOF  EQU  01F8H  ;  CASSETTE OFF
RDHDR  EQU  0296H  ;  READ HEADER ON CASSETTE
READC  EQU  0235H  ;  READ CHARACTER FROM CASSETTE
WRLDR  EQU  0237H  ;  WRITE HEADER TO CASSETTE
WRBYA  EQU  0264H  ;  WRITE CHARACTER TO CASSETTE
MINUS  EQU  '-'    ;  ASCII MINUS
SPAC   EQU  '-'    ;  ASCII SPACE
NL     EQU  0DH    ;  ASCII NEW LINE
NUMBA  EQU  30H    ;  ASCII NUMBER BASE
MAXLE  EQU  10     ;  UPPER BOARD FOR ARST ROW COUNTER

;   ORG     1740 8 ;  =  4400 TRS80 HEX ADDRESS
                   ; FOR  START  OF  PROGRAM
```

VARIABLES USED

```
MIN         DEFB 00     ;  Has value of one if number input negative
MAN         DEFB 00     ;  If MAN = zero then steps are stored
STRFG       DEFS 00     ;  If STRFG non zero then store TBUF array
KEYP        DEFB 00     ;  Set if key pressed in KEYIN Routine
FORFG       DEFB 00     ;  Set if sequence to be done forever

COUNT       DEFB 0000   ;  Number of motor slices stored
CUROW       DEFB 0000   ;  Pointer to next free motor slice
```

ARRAYS

```
NUMAR       DEFS 10     ;  Store used for Binary to ASCII Conversion
                        ;  Routine CTBAS

POSAR       DEFS 12     ;  Each two bytes of this six element array
                        ;  contain one value which is used to
                        ;  keep track of each motor's motion,
                        ;  hence the array can be used to reset
                        ;  the arm, moving it into a defined
                        ;  start position.
                        ;  Each 16 bit value stores a motor's
                        ;  steps in two's complement arithmetic.

CTPCS       DEFS  6     ;  6 Bytes, each relating to a motor.
                        ;  A number from 1-4 is stored in
                        ;  each byte  and this is used to
                        ;  index the FTABL (see constant definition)

TBUF        DEFS  6     ;  When  learning a move sequence the
                        ;  six motors' motions are stored in this
                        ;  six byte array.  Each byte relates
                        ;  to a motor and holds a motor step
                        ;  count in the range -128 to +127
                        ;  If the motor changes direction or a
                        ;  count exceeds the specified range then
                        ;  the whole TBUF array is stored in
                        ;  the ARST array and the TBUF array
                        ;  is cleared.
                        ;  TBUF means temporary buffer.

DRBUF       DEFS  6     ;  Each  byte relates to the previous
                        ;  direction of a motor.

MOTBF       DEFS  6     ;  A six byte array used by DRAMT to
                        ;  tell which motors are being driven, and
                        ;  in which direction.
                        ;  Bit zero set if motor to be driven.
                        ;  Bit  one set if motor in reverse
                        ;  Byte zero if motor should not be driven.

ARST        DEFS N*6    ;  This array holds the sequence that
                        ;  the user teaches the system.  The array
                        ;  consists of N*6 bytes where N is
                        ;  the number of rows needed to store the
                        ;  sequence.
```

CONSTANTS USED

```
FTABL           DEFB 192    ;
                DEFB 144    ;
                DEFB  48    ;
                DEFB  96    ;


                ;  FTABL is a small table which defines the
                ;  order of the steps as they are sent out
                ;  to the arm.  To drive each motor the
                ;  DRAMT  routine adds the motor's offset
                ;  which is obtained from CTPOS and adds
                ;  this to the FTABL start address -1.  This
                ;  will now enable the DRAMT routine to
                ;  fetch the desired element from the FTABL
                ;  array, and this value is then sent to
                ;  the motor via the output port.
```

CONSTANTS AND ARRAYS
STRINGS

```
         SIGON    DEFM                     *** COLNE ROBOTICS ARM CONTROLLER
MK (AL2) ***'
                  DEFW      000DH
         RELYQ    DEFB      0DH
                  DEFM      'REALLY QUIT? (Y/N)'
                  DEFW      00
         SIGOF    DEFW      0D0DH
                  DEFM      'YOU ARE NOW AT TRS80 SYSTEM LEVEL'
                  DEFW      00
         ECOMS    DEFM      'EDIT (M)OTOR STEP, OR (R) OW COUNT?'
                  DEFW      000DH
         COUTS    DEFM      'NEW UPPER ROW BOUND IS?'
                  DEFB      00
         EDSTR    DEFM      'ROW NUMBER?'
                  DEFB      00
         BADMS    DEFM      '*** BAD INPUT VALUE ***'
                  DEFW      000DH
         MOTNS    DEFM      'CHANGE STEPS ON WHICH MOTOR?'
                  DEFB      00
         NVALS    DEFM      'REPLACEMENT STEP VALUE?'
                  DEFB      00
         QUESS    DEFM      'LRN, READ, CHECK,WRITE, GO, DISP, BOOT, MAN,
                            QUIT,  SETA, TOST, EDT, FREE
                  DEFW      000DH
         RORNM    DEFM      'DO (F)OREVER OR (O)NCE?'
                  DEFB      00
         CASRD    DEFM      'TYPE SPACE BAR WHEN READY, OF FULL STOP TO EXIT
                  DEFB      00
         QMESS    DEFM      'PARDON'
                  DEFW      000DH
         BOOTS    DEFB      0DH
                  DEFM      'WANT TO RE-START (Y/N)?'
                  DEFB
         RELNS    DEFM      'START AGAIN OR (C)ONTINUE FROM CURRENT POSITION
                            (.) TO EXIT
                  DEFW      000DH
         DISPS    DEFB      0DH
                  DEFM      ' *** MOVEMENT ARRAY DISPLAY ***  '
                  DEFB      0DH
                  DEFW      000DH
         NODIS    DEFM      '*** NO SEQUENCE IN STORE ***'
                  DEFB      0DH
                  DEFW      000DH
         OVFMS    DEFM      'NO MORE ARM STORE LEFT, DELETE OR SAVE?'
                  DEFW      000DH
         DONMS    DEFB      0DH
                  DEFM      'SEQUENCE COMPLETE'
                  DEFW      000DH
         RDMSG    DEFM      '*** READ ERROR ***'
                  DEFW      000DH
         TAPOK    DEFM      '*** TAPE OK ***'
                  DEFW      000DH
         STRST    DEFM      'ARM RESET'
                  DEFW      000DH
         NOTOR    DEFM      'ARM NOW FREE TO MOVE'
```

```
        DEFB    000DH
TORMS   DEFB    0DH
        DEFM    '*** TORQUE APPLIED ***'
        DEFW    000DH
POSST   DEFM    'RELPOS='
        DEFB    00
```

SECTION 2

COMMAND ROUTINES

COMMAND INDEX

MAIN LOOP

```
; Program start

STARM    CALL  CLRSC   ;  Clear the TRS80 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 menu  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 0 —> 5
        LD      C,A         ;
        LD      B,0         ;  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      0FFH        ;
        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,0          ;  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,0             ;  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,0           ;  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 TRS8O
;  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
        JF      FINAD         ;  return to TRS8O 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

TCSTM     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,l      ;  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 lire
        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      '0'         ; User typed an '0'
        JR      Z,ONECY     ; Do sequence till end
        CP      'F'         ; User typed an 'F'
        JR      NZ,GO       ; No then re-try
        LD      A,l         ; Set forever flag
        LD      (F0RFG),A   ; to 1
ONECY   LD      A,'.'       ; Print a '.'
        CALL    PUTCHR      ; Using PUTCHR
        CALL    DOALL       ; Execute the sequence
        LD      A,(FORFC)   ; Test FORFG, if zero
        OR      A           ; then we do not want
        JP      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,D0NMS    ; Print sequence done
        CALL    PSTR        ;
        JP      QUESl       ; 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           ;
        CR      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      QUESl         ;  return to the main loop
SETBC   LD      EC,000        ;  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
        LC      1X,NUMAR      ;  1X  points to buffer fcr 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
        JP      Z,NUMPO       ;  If bit = 0 then positive step
        LD      H,0FFH        ;  Make B = negative number
        JR      EVAL          ;  Do rest
NUMPO   LD      H,0           ;  Clear H for positive number
EVAL    LD      L,A           ;  Get low order byte into L
        LD      1X,NUMAR      ;  Point to result string
        CALL    CBTAS         ;  Call conversion routine
        LD      PL,NUMAR      ;  HL points to result
        CALL    PSTR          ;  Print resulting conversion
        LD      A,(3810H)     ;  Get keyboard memory location
        BIT     0,A           ;  Test for zero key pressed
        JR      Z,NOSTP       ;  Not pressed, then skip
DOSTF   CALL    GCER          ;  Wait till next character entered
        CP      '.'           ;  Is it a dot?
        JR      NZ,NOSTP      ;  No then carry on
        CALL    PNEWI.        ;  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
```

SECTION 3

SUBROUTINES.

SUBROUTINES INDEX

SUBROUTINES INDEX (continued)

SUBROUTINE DOALL

```
;  This subroutine executes a sequence in store once.
;  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,0006       ;  Motor count of six
          LDIR                  ;  Copy memory slice into TBUF
          PUSH    HL            ;  Save new memory pointer
          CALL    DRIVL         ;  Drive all motors fcr 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 = 0


DRIVL     LD     C,0          ;
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     0            ;  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,l          ;  Set MOTBF = 1 for
          LD     (DE),A       ;  a positive drive
          INC    (BL)         ;  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,l          ;
          LD     (KEYP),A     ;  Set key pressed flag
          CALL   STEPM        ;  Step all motors once if
          DEC    C            ;  any to step
          JF     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,0          ;  Set HL = 0
            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 MOVTO

```
; 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    ID      HL,POSAR        ; HL points to POSAR
        LD      B,12            ; B = count of 12
NRES1   LD      A,(HL)          ; Get POSAR element
        CR      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+10     ; HL points to POSAR
        LP      DE,MOTBF+    5  ; DE points to MOTBF
        LE      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
        JP      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,l             ; 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,l         ;  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,0F0H      ;  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 CTPOS array
NSET1     LD     (HL),1     ;  Set CTPOS(N) to start position = 1
          INC    HL         ;  Increment HL
          DJNZ   NSET1      ;  Do set up next CTPOS 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,l           ;  Reset table offset
NORST     LD      (HL),A        ;  Store in CTPOS (N)
IGMTN     INC     HL            ;  Increment CTPOS pointer
          DEC     DB            ;  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,0           ;
          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
TRY0        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  TRY0        ;  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    0           ; See if positive or negative
        LD    A,(DE)      ; Get identical previous motor slot
        JP    P,PDIR      ; if positive do for positive motor
NDIR    CP    0           ; 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    0           ; 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 slot
;  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     lX,DRBUF+6    ;  1X  = previous direction buffer
            LD     lY,MOTBF+6    ;  1Y = current buffer
            LD     HL,TBUF +6    ;  HL = step buffer
NTMOT       DEC    1Y            ;
            DEC    1X            ;
            DEC    HL            ;  move pointers
            LD     A,(1Y +0)     ;  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,(lX+0)      ;  Get previous direction
            CP     1             ;  Direction change?
            JR     NZ,CFORD      ;  No then advance TBUF(N) step
            CALL   SETST         ;  Set the store flag
            LD     (1Y+0) ,0     ;  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     (lX+0),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,(lX+0)      ;  Get previous direction
            CP     3             ;  Direction reversed?
            JR     NZ,CREV1      ;  No then continue
            CALL   SETST         ;  Else set store TBUF in ARST flag
            LD     (lY+0),0      ;  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     (1X+0),1      ;  Set Direction
            JR     NODRV         ;  Do next motor
SETST       PUSH   AF            ;  Save AF
            LD     A,l           ;  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 0.  It then drives the motors corresponding
;  to the keys pressed.  If in learn mode the
;  sequence is stared.

KEYIN    CALL    CLRMF      ;  Clear MOTBF array
         LD      A,(3840H)  ;  Get TRS80 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,(3810H)  ;
         BIT     0,A        ;  Is the zero key pressed?
         JR      Z,TRYS     ;  No then skip
         JP      NOTNG      ;  Go to do nothing
TRYS     LD      A,(3804H)  ;  See if
         BIT     3,A        ;  'S'  key pressed
         LD      A,(3810H)  ;  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,0       ;  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,0        ;  Clear BC offset for motor 1
            LD .    A,(3804H)   ;  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 motor 2
TRYE        INC     BC          ;  Advance pointer offset
            LD      A,(3801H)   ;  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,(3804H)   ;  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 motor 5 backward
TRYY        LD      A,(3808H)   ;  Is the 'Y' key pressed?
            INC     BC          ;  Advance offset
            BIT     1,A         ;  No key
            JP      Z,SOMEN     ;  'Y' then skip
            CALL    BACMT       ;  Set motor 6 for backward
SOMEN       CALL    SRAMT       ;  Step motors, 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 for forward direction
            JP      SETMT       ;  Do set motor slot in MOTBF
BACMT       LD      E,l         ;  Set for 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                 ;  Restore AF and return
DOMOT       LD      (HL),E      ;  Set byte in MOTBF
            LD      A,l         ;  and set
            LD      (KEYP),A    ;  key pressed flag
            POP     AF          ;  Restore AF
            RET                 ;  exit from routine
```

SUBROUTINE CBTAS

```
;  This subroutine makes a signed binary value in
;  HL  into arm ASCII String and stores the string
;  in the locations pointed to by 1X


CBTAS    PUSH   AF          ;  *
         PUSH   HL          ;  *
         PUSH   DE          ;  *  Save Registers
         PUSH   1X          ;  *
         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     (1X+0),A    ;  Pointed to by 1X
         INC    1X          ;  Advance 1X pointer
         JR     CONUM       ;  Do rest of conversion
POSNO    LD     A,SPAC      ;  Place a space if number positive
         JR     PUTSN       ;  Jump to copy space to memory
CONUM    PUSH   1Y          ;  Save 1Y register
         LD     1Y,BTOAT    ;  Point to subtraction table
NUMLP    LD     A,NUMBA     ;  Get ASCII 0 in A
         LD     E,(1Y+0)    ;
         LD     D,(1Y+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 subtraction
GONEN    ADD    HL,DE       ;  Restore value before last
                            ;  subtraction
         LD     (1X+0),A    ;  Store ASCII Number in memory
         INC    1X          ;  Inc memory pointer
         INC    1Y          ;  Point to next table value
         INC    1Y          ;
         DEC    E           ;  Test if E = 0
         JR     NZ,NUMLP    ;  No then try for next digit
         XOR    A           ;  Clear A and place in store
         LD     (1X+0),A    ;  as EOS = End of string
         POP    1Y          ;  *
         POP    1X          ;  *
         POP    DE          ;  *  Restore all saved registers
         POP    HL          ;  *  and
         POP    AF          ;  *
         RET                ;  Exit
```

```
BTOAT       DEFW   10000   ;  Table of subtraction constants
            DEFW   1000    ;  for conversion routine
            DEFW   100     ;
            DEFW   10
            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(0)
         LD      DE,MOTBF +1   ; Point to MOTBF(1)
         LD      BC,5          ; BC = Count
         LD      (HL),0        ; MOTBF (0) = 0
         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(0)
         LD      DE,TBUF + 1   ; DE points to TBUF(l)
         LD      BC,17         ; BC = Count of 17
         LD      (HL),0        ; Clear first element
         LDIR                  ; Now clear next 17 elements
         POP     HL            ; *
         POP     DE            ; *  Restore Registers
         POF     BC            ; *
         RET                   ; Exit
```

SUBROUTINE GINT

```
;  This subroutine gets a signed 16 bit integer
;  from the TRS80 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,l         ;  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 10
            CP      0           ;
            JR      C,ERRN2     ;  If number less than ASCII 0 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,0         ;  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,l              ;  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      1X,NUMAR   ;  1X points to result string
           CALL    CBTAS      ;  Convert HL and leave in (1X)
           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
           FOP     HL         ;  *
           POP     DE         ;  *
           POP     BC         ;  *  Restore all Registers
           POP     AF         ;  *
           RET                ;  Now return
```

```
; 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 30 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+l0; 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 frcm 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     STPCS        ;   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 dc next element check
        JR    EXIT        ;  All TBUF zero so exit
STOR1   LD    (lX+0),0    ;  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,0006     ;  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    OVRFM       ;  Bad input, try again
REDO    CALL  INIT        ;  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 subroutine clears the POSAR array

```
RESET   PUSH    BC          ;
        PUSH    DE          ;  *  Save Registers
        PUSH    HI.         ;  *
        LD      HL,POSAR    ;  Point to POSAR start
        LD      DE,POSAR+1  ;  Point to next element
        LD      (HL),00     ;  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       ; Restore 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 TRS80
                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,20     ;  A = Space character
                CALL   PUTCHR   ;  Print it
                POP    AF       ;  Restore AF
                RET             ; Done, Exit

; PNEWL  prints a new line to the screen

PNEWL           PUSH   AF       ;  Save AF
                LD     A,0DH    ;  A = Newline character
                CALL   PUTCHR   ;  Print it
                POP    AF       ;  Restore 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 keyboard 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 10 * B + 10 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                 ;  Restore BC
            RET                         ;  Exit
DELS        PUSH     BC                 ;  Save BC
            LD       B,20               ;  Set B for 0.001 sec delay (apx)
            CALL     DELSW              ;  Do delay
            POP      BC                 ;  Restore BC
            RET                         ;  Exit
DELT        PUSH     BC                 ;  Save BC
            LD       B,0                ;  Set B for 0.01 sec delay (apx)
            CALL     DELSW              ;  Dc delay
            POP      BC                 ;  Restore BC
            RET                         ;  Exit
DELLN       PUSH     BC                 ;  Save BC
            LD       B,200              ;  Set B for 1.0 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	0	1	0	1
1	0	0	1	2
0	1	0	1	3
0	1	1	0	4

HALF STEPPING PULSE SEQUENCE

QA	QB	QC	QD	STEP
1	0	1	0	1
1	0	0	0	1.5
1	0	0	1	2
0	0	0	1	2.5
0	1	0	1	3.0
0	1	0	0	3.5
0	1	1	0	4
0	0	1	0	4.5

The documented 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

A P P L I C A T I O N S

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

CONSTRUCTION OF A SUITABLE PORT FOR THE ARMDROID

A circuit diagram is given which describes in particular the construction of an 8 bit bi-directional, non latched port. The circuit as given is for the TRS80 bus, but it should be possible with reasonably simple modifications to alter it for most Z80 type systems.

The circuit described is a non latched port so the output data will appear for only a short period on the 8 data lines.

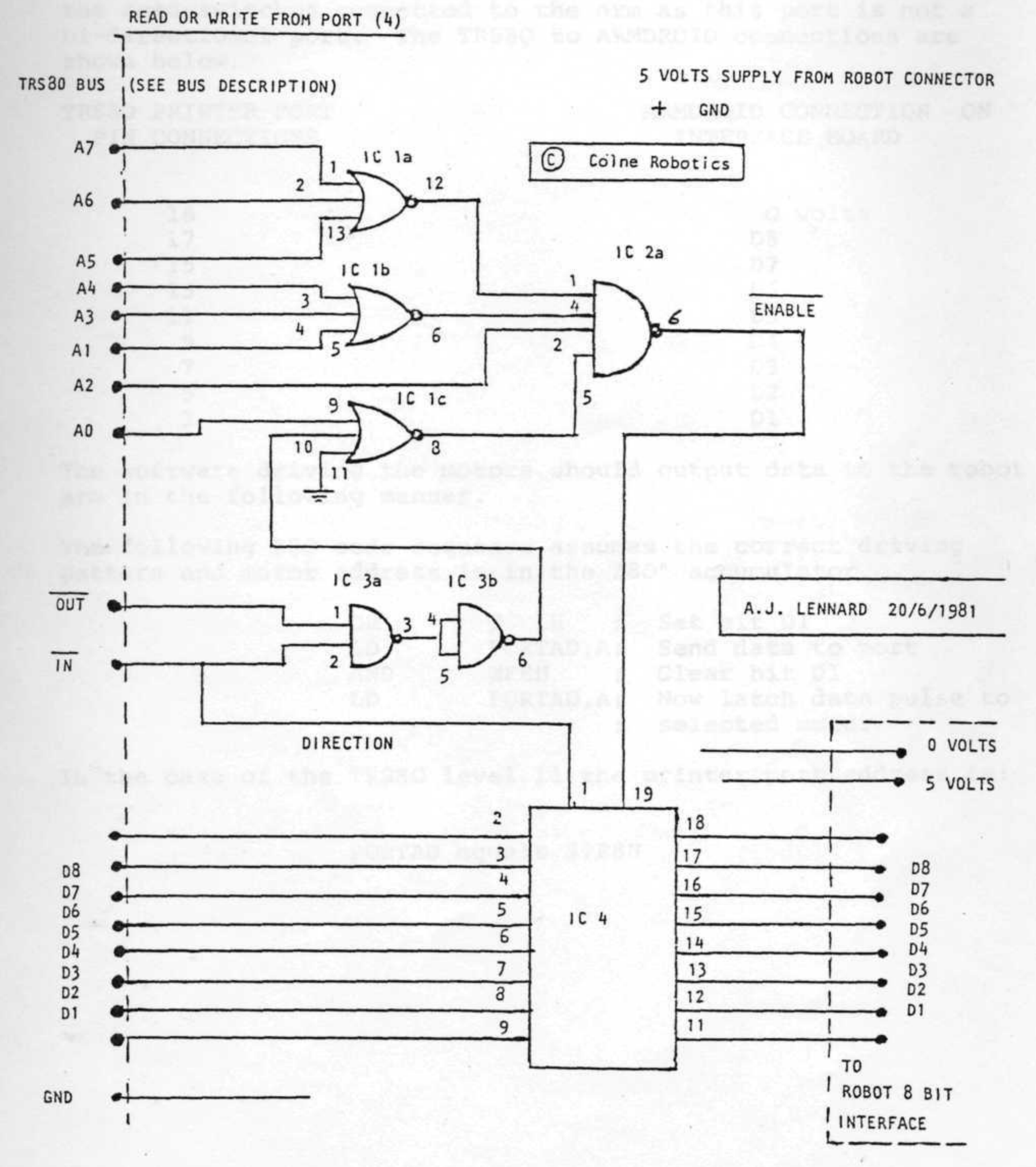
As can be seen from the diagram, the circuit draws its 5 volt power supply from the arm's interface port, and not from the processor it is connected to. The port was constructed this way due to the fact that some commercial microprocessor systems do not have a 5v output supply.

When the above circuit is connected to the arm's interface card the bottom bit is usually pulled high, thus if the user inputs from the port at any time the data presented will mirror the state of the reed switches.

To output data to the arm using this port the user should send the data to the port with the bottom bit cleared. The data will then be latched through to the addressed arm motor latch.

The components for the described port should be easily available from most sources.

TRS80 8 BIT INTERFACE (NON LATCHED BI-DIRECTTONAL)



IC 1:	74LS27	Pin 14: 5 Volts, Pin 7: GND	3*3 INPUT NOR
IC 2:	74LS20	Pin 14: 5 Volts, Pin 7: GND	2*4 INPUT NAND
IC 3:	74LS00	Pin 14: 5 Volts, Pin 7: GND	4*2 INPUT NAND
IC 4:	74LS245	Pin 20: 5 Volts, Pin 10: GND	OCTAL BUS TRANSCEIVER (Tri-state)

CONNECTION OF THE ARMDROID TO THE TRS8O PRINTER PORT

The TRS8O printer port can be used to drive the robot arm, but when using the printer port it will not be possible to read the reed-switches connected to the arm as this port is not a bi-directional port. The TRS8O to ARMDROID connections are shown below.

TRS8O PRINTER PORT PIN CONNECTIONS	ARMDROID CONNECTION ON INTERFACE BOARD
18	0 volts
17	D8
15	D7
13	D6
11	D5
9	D4
7	D3
5	D2
3	Dl

The software driving the motors should output data to the robot arm in the following manner.

The following Z80 code sequence assumes the correct driving pattern and motor address is in the Z80 accumulator.

```
OR       0  1H    ;  Set bit Dl
LD       PORTAD,A;  Send data to port
AND      0FEH     ;  Clear bit Dl
LD       PORTAD,A;  Now latch data pulse to
                  ;  selected motor
```

In the case of the TRS8O level ll the printer port address is:

PORTAD equals 37E8H

CONNECTION OF ARMDROID TO PET/VIC COMPUTERS

PET/VIC USER PORT CONNECTOR

PIN NO	PET/VIC NOTATION	ARMDROID NOTATION
C	PAO	Dl
D	PA1	D2
E	PA2	D3
F	PA3	D4
H	PA4	D5
J	PA5	D6
K	PA6	D7
L	PA7	D8
N	GROUND	GROUND

I/O Register Addresses (User Ports)

VIA Data Direction Control: 37138

PET Data Directional Control Register: 59459

VIC I/O Register Address: 37136

PET Data Register Address: 59471

The data direction registers in the VIA define which bits on the respective user ports are input and which are to be used as output bits. A binary one in any bit position defines an output bit position and a zero defines that bit as an input bit.

SIMPLE BASIC ARM DRIVER FOR VIA (PET/VIC)

```
  5 L  =  37136:  Q = 37138
 10 PRINT "VIC ARMDROID TEST"
 20 PRINT
 30 PRINT "HALF STEP VALUES"
 40 T = 8: C = 2: S = 10: M = 1: I = 1: A$ = "F"
 50 FOR I = 1 TO T:  READ W(I):  PRINT W(I):  NEXT I
 60 POKE Q, 255
 70 INPUT "MOTOR NUMBER (1-6)"; M
 80 IF M<1 OR M>8 THEN 70
 90 INPUT "FORWARD BACKWARD"; A$
100 IF A$ = "F" THEN D = O:  GOTO 130
110 IF A$ = "B" THEN D = 1:  GOTO 130
120 GOTO 90
130 INPUT "STEPS"; S
140 IF S<1 THEN 130
150 O = M + M +1
160 FOR Y = 1 TO S*C
170 F = W(I) + O
180 POKE L,F
190 POKE L,F-1
2OO IF D = 0 THEN 230
210 Y = Y + 1:  IF Y>T THEN Y = 1
220 GOTO 240
230 Y = Y - 1:  IF Y<1 THEN Y = T
240 NEXT Y
250 GOTO 70
260 DATA 192, 128, 144, 16, 48, 32, 96, 64
```

THE VALUES FOR L AND Q FOR THE PET ARE

Q = 59459 = DATA DIRECTION
L = 59471 = I/O

MOTOR STEP RELATIONSHIP PER DEGREE INCREMENT

Below are shown the calculations for each joint to enable the user to calculate the per motor step relationship to actual degree of movement.

These constants are necessary for users wishing to formulate a cartesian frame reference system or a joint related angle reference system.

Base

Motor step angle x ratio 1 x ratio 2

$$7.5 \times \frac{20 \text{ teeth}}{72 \text{ teeth}} \times \frac{12 \text{ teeth}}{108 \text{ teeth}}$$

= 0.2314 degree step or 4.32152 steps per degree.

Shoulder

$$7.5 \times \frac{14 \text{ teeth}}{72 \text{ teeth}} \times \frac{12 \text{ teeth}}{108 \text{ teeth}}$$

= 0.162 degree per step or 6.17284 steps per degree

Elbow

Same as shoulder joint

Wrists

Same as base joint calculations

Hand

$$7.5 \times \frac{20 \text{ teeth}}{72 \text{ teeth}} \times \frac{12 \text{ teeth}}{108 \text{ teeth}} = 0.231 \text{ degree per step}$$

$$\frac{\text{pi} \times d \times 0.231}{360} = (0.0524/2)\text{mm}$$

=0.0262mm = hand pulley motion per step

Total hand open to close pulley movement = 20.0mm

Angletraversedbysinglefinger=50degrees

$$\frac{50°}{20.0 \text{ mm}} \times 0.0262\text{mm}$$

= 0.0655 degrees per step or 15.2672 steps per degree

pi = 3.1415926

d = 26mm = pulley diameter

SOME OVERALL DIMENSIONS

Shoulder	pivot to pivot	= 190mm
Forearm	pivot to pivot	= 190mm
Finger	wrist pivot to fingers closed = 90mm wrist pivot to finger open (90) = 99mm	

Bottom of base to shoulder pivot = 238mm

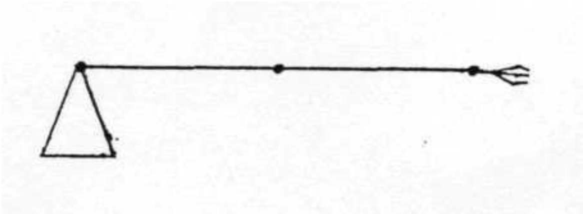
ANGULAR JOINT SPANS

Shoulder	up = 153	,down 45
Forearm	up = 45	,down 150
Wrist	up = 100	,down 100
Base	no limit ,but suggest caution not to overwind cables in base	
Hand	fingers move over 50	

(All above measurements are in degrees)

NOTE

The above measurements were taken with the arm joints held in a horizontal plane:



SOME EXTRA POINTS TO BEAR IN MIND

a) Long Lead of LED goes to NEGATIVE
Short lead of LED goes via 4.7 kohm Resistor
to POSITIVE

b) Due to LED hole being slightly too large a grommet
will first have to be fitted to the LED and its holder
can then be super glued if necessary into the grommet.

c) The Torque available is largely a function of speed
and hence the user can expect performance to deteriorate
as speed is increased. Tables are supplied earlier
in the manual.

FINAL NOTE

BEST WISHES AND GOOD LUCK