SPECIFICATIONS
Configuration 5 Axes of rotation
Gripper 3 Finger type
Drive 6 Stepper motors
with open loop control
Controller Any micro computer
with an 8 bit parallel port
Power Requirement 15 volts 5 to 6 amps
Weight 3.5 Kg. without power pack
Size 150mm x 230mm x 310mm high

PERFORMANCE
Resolution 4mm
Load Capacity 300gms
Gripping Force 20 Newtons
Reach 430mm

THE DEVICE
The ArmNoid represents an important step forward in automation and handling. The device has five axes of rotation and is a continuous path machine. In other words it is able to use several joints at once and to perform a programmed move sequence under computer control. The robot comes either as a kit or in assembled form. This low cost robotic development tool can be used in the home, school, factory or research laboratory as an educational device. It is available with two distinct modes of control — computer control or manual control.

COMPUTER CONTROL AND SOFTWARE
The ArmNoid can be driven by most micro computers and can be used as a handling device or alternatively as a computer peripheral. All the well known names will operate the machine such as Pet, Apple, TRS 80, ZX 81, RML 380Z, Acorn, BBC Computer and many more. We now have software available for many of these computers. Programs are memory orientated and have a learning capability so that a robot is able to repeat a sequence which has been taught to it as many times as required.

MANUAL CONTROL
A hand held control box using separate centre-off switches to operate each of the six motors is available to special order.

THE ELECTRONICS
The computer controlled robot has an interface board for an 8 bit bi-directional parallel port. Micro switches to aid position sensing are optional. A separate interface board is used for manual control and this is now interchangeable with the computer board. Power packs are available for both 220/40v and 110v supplies.

THE HANDBOOK
A set of instructions for both construction and operation is a part of the kit and it contains detailed mechanical drawings, electronics schematics, software listings and description.
ZEAKER INTRODUCTORY PRICE LIST AND ORDER FORM

LINE ROBOTICS CO LTD
AUFORT ROAD
F RICHMOND ROAD
ST TWICKENHAM
DDX TW1 2PH

TELEPHONE 892 8197 or 8241
TELEX 8814066

<table>
<thead>
<tr>
<th>ITEM</th>
<th>DESCRIPTION</th>
<th>PRICE EACH</th>
<th>QTY</th>
<th>TOTAL PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Zeker Mobile Robot, Control Station and connecting leads between control station, Robot and Micro Computer. Manual</td>
<td>In Kit Form</td>
<td>52.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ready Assembled</td>
<td>69.50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software listing for your Micro - See Appendix for Catalogue No</td>
<td>Catalogue Number</td>
<td>Free Of Charge</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interface for ZX81 Computer</td>
<td></td>
<td>13.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interface for Spectrum Computer</td>
<td></td>
<td>25.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cassette of Software for your Micro - See Appendix for Catalogue No</td>
<td>Catalogue Number</td>
<td>6.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SUB TOTAL</td>
</tr>
<tr>
<td></td>
<td>Plus Packing, Postage, and Insurance</td>
<td></td>
<td>3.00</td>
<td></td>
</tr>
</tbody>
</table>

Less otherwise arranged
Remittance is due before delivery (except for Educational, Institutional and Large Commercial chasers where payment is due after delivery)

IL 1983
SPECIFICATIONS
Configuration  5 Axes of rotation
Gripper       3 Finger type
Drive         6 Stepper motors
              with open loop control
Controller    Any micro computer
              with an 8 bit parallel port
Power Requirement  15 volts 5 to 6 amps
Weight        3.5 Kg. without power pack
Size          150mm x 230mm x 310mm high

PERFORMANCE
Resolution     4mm
Load Capacity  300gms
Gripping Force 20 Newtons
Reach          430mm

THE DEVICE
The ArmDroid represents an important step forward in automation and handling. The device has five axes of rotation and is a continuous path machine. In other words it is able to use several joints at once and to perform a programmed move sequence under computer control. The robot comes either as a kit or in assembled form. This low cost robotic development tool can be used in the home, school, factory or research laboratory as an educational device. It is available with two distinct modes of control — computer control or manual control.

COMPUTER CONTROL AND SOFTWARE
The ArmDroid can be driven by most micro computers and can be used as a handling device or alternatively as a computer peripheral. All the well known names will operate the machine such as Pet, Apple, TRS 80, ZX 81, RML 380Z, Acorn, BBC Computer and many more. We now have software available for many of these computers. Programs are memory orientated and have a learning capability so that a robot is able to repeat a sequence which has been taught to it as many times as required.

MANUAL CONTROL
A hand held control box using separate centre-off switches to operate each of the six motors is available to special order.

THE ELECTRONICS
The computer controlled robot has an interface board for an 8 bit bi-directional parallel port. Micro switches to aid position sensing are optional. A separate interface board is used for manual control and this is now interchangeable with the computer board. Power packs are available for both 220/40v and 110v supplies.

THE HANDBOOK
A set of instructions for both construction and operation is a part of the kit and it contains detailed mechanical drawings, electronics schematics, software listings and description.
THE ZEAKER MICRO-TURTLE

The world's first low-cost mobile robot for micro-computers.

WHAT IS IT? Zeaker is a small mobile robot (5" x 5" x 2") with two DC motor drive, four touch sensors, a two-tone horn, direction-indicating LED's, a power supply, 2m umbilical ribbon cable, manual and software.

WHAT DOES IT DO? The Zeaker can be driven from any micro-computer which has an 8 bit bi-directional port (in the case of ZX81 a special interface board is required - see below). Software provides a learning program, control of pen and response of Zeaker to contact with its sensors.

I am interested in purchasing the units indicated below, I understand you will inform me when they are ready for despatch and will then ask me to forward cheque/PO in payment.

___ Kit @ £59.00 (incl VAT)
___ Assembled Unit @ £79.00 (incl VAT)
___ ZX81 Interface Board @ £15.00 (incl VAT)

Please tear off and send to above address.
THE ZEAKER MICRO-MOBILE — a low cost mobile robot for micro-computers

A new product shortly to become available from Colne Robotics will be a 2-wheeled mobile robot known as the Zeaker Micro-Mobile. Its movements can be controlled by a micro-computer, via a connecting umbilical ribbon cable. Software is provided which permits the movements to be memorized and reproduced — that is to say Zeaker has a learning capability. With further appropriate software it is capable of drawing Turtle and Logo graphics.

Sensors indicate when the robot touches an obstacle and the computer instructs it to find an alternative path. Stimulation of the sensors produces one of two notes on a horn, according to the direction of Zeaker's movements.

An additional feature is the built-in, retractable pen beneath the unit, which can trace the path taken across a surface. The pen itself is controlled by the computer, and its position (lowered or retracted) is indicated by a light on the top of the robot. Two further lights change according to the direction of movement.

The Zeaker Micro-Mobile is aimed at the educational market, in which a growing number of schools wish to extend their computer teaching syllabus to cover control systems, through the use of micro-computers. It is also aimed at the rapidly expanding computer hobby market. To keep in line with the fall in micro-computer prices, the units have been produced at very low cost: £59.95 for the kit version and £79.95 for the assembled robot. (INTRODUCTORY OFFER).

Zeaker comes complete with interface, power supply, operation manual and software. It can be driven by any micro-computer which has an 8-bit bi-directional port, as well as by the ZX81 for which a special interface is available from Colne Robotics. We plan to produce interfaces for other popular micro-computers too.

Look out for Zeaker on the front cover of the May '83 issue of "Practical Electronics", available from 8th April.
Patch into 5681

JP EPATCH

EPATCH
ADD HL, HL  Double count
PUSH HL     Save it
ADD HL, HL  Count x 4
POP BC      Restore count x 2
ADD HL, BC  Count x 6
LD BC, ARST Get buffer pointer
ADD HL, BC  Get new CUROW pointer
LD (CUROW), HL Save it
JP QUE81

This patch is designed to overcome the logic error that row edit does not modify CUROW, the pointer to next available row.

Investigate where FTABLE is accessed and determine the feasibility of choosing all or half stepping.
1000 RC = 2.82808
1010 EL = PEEK(ZH) * RC; 'POSAR + 3 ELBOC
1020 SH = PEEK(ZH) * RC; 'POSAR + 5 SHOULDER
1100 GO SUB 1400; BB = BB; 'CALCULATE BC
1110 EL = EL + 1
1120 GO SUB 1400
1130 IF ABS(BC - BB) < 1 THEN 1230
1135 IF CN > 4 THEN 1230
1140 IF BC - BB < 0 THEN 1190
1150 SH = SH + 1; CN = CN + 1
1160 GO SUB 1400
1170 IF ABS(BC - BB) < 1 THEN 1230
1180 GO TO 1135
1190 SH = SH - 1; CN = CN + 1
1200 GO SUB 1400
1210 IF ABS(BC - BB) < 1 THEN 1230
1220 GO TO 1135
1230 RETURN

1400 H = 380 * SIN(EL/2)
1410 BB = SIN(4.71239 - (SH + 0.785398 - EL/2)) * H
1420 RETURN

4044 Start of Basic. 6 B00
690 AR = 2900%
700 CN = 1
710 EL = PEEK(2521H) \times 2.82808
720 SH = PEEK(2521D) \times 2.82808
730 INPUT "HOW MANY STEPS" ; N
740 FOR I = 1 TO N
750 GOSUB 1100
760 NEXT I
770 MERGE %CD, %578D
790 DELETE 5000 \%800 SYSTEM 690 =
1100 GOSUB 1400
1110 BC = 33
1120 EL = EL + 1
1130 GOSUB 1400
1140 IF ABS(BC - BB) < 1 THEN 1280
1150 IF CN > 4 THEN
1160 IF BC - BB < 0 THEN 1720
1170 SH = SH + 1
1180 CN = CN + 1
1190 GOSUB 1400
1200 IF ABS(BC - BB) < 1 THEN 1280
1210 GO TO 1150
1220 SH = SH - 1
1230 CN = CN + 1
1250 GOSUB 1400
1260 IF ABS(BC - BB) < 1 THEN 1280
1270 GO TO 11505
1280 POKE AR + 5, SH
1290 POKE AR + 4, EL
1300 AR = AR + 6
1310 RETURN
1400 H = 380 \times \sin(E1/2)
1410 BB = \sin(4.71239 - (SH + 0.785398 - EL) / 2) \times H
1420 RETURN
100  GO SUB 530
110  OPEN "I", 1, FL$
120  E+
130  LINE INPUT #1, A$
140  FOR I = 1 TO 36
150    AA$ = MID$(A$, I, 1)
160    POKE % 5214 + I, VAL(A$)
170  NEXT I
180  AA$ = RIGHT$(A$, 1)
190  CN = VAL(A$)
200  POKEW % 5205, CN
210  I = 1
220  LINE INPUT #1, A$
230  IF EOF(1) THEN 280
240  FOR J = 1 TO LEN(A$)
250    AA$ = MID$(A$, J, 1)
260    POKE % 8FF + I, VAL(AA$)
270  NEXT J
280  CLOSE 1
290  MERGE %C3, % SS97
300  CN = PEEKW(% 5205)
310  GO SUB 530, "A$ = "$
320  OPEN "O", 1, FL$
330  CN = PEEKW(% 5205)
340  FOR I = 1 TO 36
350    AA$ = CHR$(PEEK(% 5214 + I))
360  IF LEN(A$) <= 25 THEN
370  PRINT #1, A$ + "$
380  A$ = "$
390  GO TO

360 NEXT I
370 A$ = A$ + CHR$(CN)
380 PRINT #1, A$;
390 FOR I = 1 TO CN
400 FOR J = 0 TO 5
410 IJ = (I-1) * 6 + J
420 A$ = ""
430 FOR I = 1 TO CN
440 FOR J = 0 TO 5
450 IJ = (I-1) * 6 + J
460 A$ = A$ + CHR$(PEEK(%9000 + IJ))
470 A$ = A$ + A$
480 IF CEN(A$) < 253 THEN 480
490 PRINT #1, A$;
500 A$ = ""
510 NEXT J
520 NEXT I
530 PRINT
540 IF A$ <> "" THEN PRINT #1, A$;
550 CLOSE 1
560 MERGE %C3, %S597
570 INPUT "Enter file name (without extension)"; FLS
580 IF FLS = "" THEN 580
590 IF DNF$ < "4" OR DNF$ > "3" THEN 550
600 FLS = FLS + "/ARM:" + DNF$
610 RETURN
MERGE CP $0 SET A
TEMP NZ, 4409
PUSH AF SET 7, A
CALL NZ, 4409
POP AF RES 7, A
JP NZ, 5597
$CB, $FF
$CF, $FF
%FE, $F, %F5, %C4, %4409, %F1, %C2, %5597
When using modules which are compiled
with ZBASIC 2XCDM48

Amendments to LEARN/CMD
RDWRT EQU (xxxx) Address of sec entry ptr
56E5H  CDC9 01 CALL CLRSRC (delete this)
replace with CALL RDWRT
56E5H  21E753 LD HL, CASRD (delete this)
replace with C3 xxxx JP (Read entry)
3740 H  ED430552 LD BC (Count) (delete this)
replace with C3 xxxx JP (Write entry)

or delete 56E5H to 5808H and add
the following equides
READ EQU xxxx (Entry to READ)
WRITE EQU xxxx (Entry to Write)
Move 5374H - 5394H up to 536B and
fill with 20H

Amendments to READWRIT/CMD
Look at 1st 3 bytes - JP => secondary entry
At secondary entry module
Replace rel bytes 11 - 13 with
CALL CLSRNC CD C9 01
and bytes 14 - 16 with 00 00 00
At relative byte 33 is a JP xxxx
replace this with JP 5571H (i.e. back to LEA)
Look for proprietary notice (just before string storing
which starts with 1CH, 1FH and ends with
70H, 79H, 0D (72 bytes in all) and replace with
00's

Look for unused text at end. Seems to start
with a redundant C9H and then has 20 asterisks
(2AH)
Similar amendments will be required to other
modules which should be added to READWRIT
L(earn)  
ext by pressing $\emptyset$ (?)

S(tart) - new sequence.
C(ontinue) - from current position.

After S or C move arm to start point.
When satisfied press space bar. (arm locks up)

D(isplay)
Scrolling halted by pressing $\emptyset$.
To continue press any other key.
To step scroll keep pressing $\emptyset$.

E(dit)

R(ow count) truncates sequence then $\langle$ENTER$\rangle$.
Then number of last row to be performed $\langle$ENTER$\rangle$.

M(otor step) allows changes to any row or column.

S(eft Arm)
Sets current position of arm as new start point.

During L(earn) sets a point to which the arm must go before executing another sequence.
W(rite) writes sequence to tape.
R(ead) reads sequence from tape.
C(heck) verifies tape sequence.
G(o) moves the arm through sequence.
T(o Start) takes arm back to start position.
F(release) removes motor torque allowing movement by hand.
M(annual) allows control of movement by pressing keys.

<table>
<thead>
<tr>
<th>Motor</th>
<th>Forward</th>
<th>Reverse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gripper</td>
<td>1</td>
<td>G</td>
</tr>
<tr>
<td>Wrist left</td>
<td>2</td>
<td>W</td>
</tr>
<tr>
<td>Wrist right</td>
<td>3</td>
<td>E</td>
</tr>
<tr>
<td>Forearm</td>
<td>4</td>
<td>R</td>
</tr>
<tr>
<td>Shoulder</td>
<td>5</td>
<td>T</td>
</tr>
<tr>
<td>Base</td>
<td>6</td>
<td>Y</td>
</tr>
</tbody>
</table>

B(oot) clears sequence & restarts program.
Q(uit) returns to DOS.
Colne Robotics Armdroid
The Small-Systems Robot

Steven W. Lehninger
5402 Summit Ridge Trail
Arlington, TX 76017

If you think you've explored all the possible hardware options for your small-computer system and are looking for some excitement, you might be interested in Armdroid, a new computer-controlled robot arm. The bright orange mechanical arm is available from Colne Robotics in kit or assembled form, complete with power supply and interface electronics. The kit form, besides being less expensive, "enables the person assembling the device to understand the principles of the robot," according to the manufacturer. The robot can be used for a variety of experimental and educational applications. It has 6 degrees of motion and a lift capacity of 10 ounces. I received both a kit and an assembled Armdroid for my evaluation, along with a "preliminary" manual.

Mechanical Description
The Armdroid has five major mechanical components: the base, the shoulder, the upper arm, the forearm, and the wrist and hand assembly. Each section is connected to its neighbor by a pivoting or rotating joint. The stationary base sits on the tabletop and provides support for the rest of the arm. The base, which also serves as the enclosure for the stepper-motor-drive electronics, contains the motor which rotates the arm about a vertical axis through the base.

About the Author
Steven W. Lehninger was the design engineer for the original Radio Shack TRS-80 Model I microcomputer. He is now an independent computer consultant.

At a Glance

Name
Armdroid

Use
Robotic arm

Manufacturer
Colne Robotics
207 NE 33rd St.
Fort Lauderdale, FL 33334

Dimensions
At shoulder: 18 by 18 by 29 cm [7 by 7 by 11.5 in]
Shoulder pivot height: 25 cm [10 in]
Arm length at maximum extension from shoulder pivot to finger tip: 48 cm [19 in]

Price
Kit: $595
Assembled: $695

Features
6 degrees of motion; menu-driven control software; 10-ounce load capacity

Additional Hardware Needed
TRS-80 Model I Level II (other microcomputers will be supported in the future)

Additional Software Needed
Learn, an interactive menu-driven control program (included)

Hardware Option
Zero-position sense switches

Documentation
Construction and Operation Manual, 87 pages

Audience
Experimenters, students, and professionals interested in robotics
The shoulder rotates on the main bearing, a fairly heavy-duty ball-bearing assembly at the top of the base. Five stepper motors and associated reduction gears and drive belts are mounted on the shoulder and provide motion control to the arm, wrist, and hand.

The upper arm connects to the shoulder with a horizontal pivot and is rotated on that pivot by one of the stepper motors in the shoulder. If you move the upper arm vertically, the hand is raised and brought closer to the base. Cable-driving gears transmit motion to the forearm and the hand and wrist assembly; these are mounted in the shoulder end of the upper arm.

The forearm fastens to the upper arm with a horizontal pivot and is rotated about that point with one of the motors in the shoulder. The primary response to pivoting the forearm is the raising or lowering of the hand with respect to the tabletop.

The hand and wrist assembly attaches to the end of the forearm with a combination horizontal pivot and bevel gear assembly. The operator uses two motors in the shoulder to either rotate the hand about the pivot (an up-and-down motion) or twist the hand about its axis. The remaining motor in the shoulder opens and closes the hand's three rubber-tipped metal fingers.

You can move any section independently without affecting the orientation of the other sections because of the Armadroid's parallelogram-type construction. This independence of control permits the angle of the hand to remain constant with respect to the workbench while the rest of the arm is manipulated to position the hand in the desired location.

Interface Electronics

The Armadroid I tested came with an I/O (input/output) adapter for the Radio Shack TRS-80 Model I. This adapter, a nonlatched parallel port, plugs into the expansion port on the TRS-80. A cable from the adapter plugs into the base of the Armadroid.

Colne Robotics has mounted two printed-circuit card within the base of the Armadroid: the interface board at the motor-drive board. The interface board accepts signals from the TRS-80, conditions them, and converts them to pulses of the duration and shape suitable for controlling the arm's motors. The motor-drive board amplifies the signals to provide the voltage and current levels required to drive the motors' coils.

You can set the Armadroid's internal electronics for external computer control or operation via manual switches by making the selection on the two printed-circuit boards inside the Armadroid's base.

Building the Kit

Being a disciple of Erector Set and Heathkit, I had no fears about venturing out into the frontiers of robot building. To get a feel for the scope of the project, I laid all the parts out and familiarized myself with the construction section of the manual.

The manual I received was a preliminary version. The entire mechanical assembly instructions were on just six pages! Undaunted, I forged ahead. About halfway through the first paragraph, I was instructed to glue magnets onto some of the gears. Apparently, the magnets are optional (at least they weren't included in the kit), but no mention was made of that fact. The system uses the magnets and their respective reed switches to sense the home position of the gears.

The instructions rambled on, sometimes with several steps in a sentence. The manual specified part numbers (usually) but didn't refer to the drawing numbers.

I knew the next part was going to be tricky because the instructions said that an assistant would be helpful. The task at hand was to assemble a dual-race ball-bearing assembly from scratch. Using refrigerated petroleum jelly
as per the instructions, I greased the bearing track and imbedded 24 ball bearings in the goo. After carefully inserting the base-support column into the bearing and turning the assembly upright, I attempted to repeat the job on the upper bearing track.

Darn. While tightening the adjusting ring, three balls hopped out of the lower bearing and huddled in a mound of petroleum jelly. Back to the beginning; twice more the same thing happened. Arghh! Finally, success! But wait, why was the shoulder pan rubbing on the shoulder-drive gear? And, wasn't that ball-bearing assembly just a little bit off parallel? At this point, I decided to cheat and look at the factory-assembled ArmDraft. It appeared that the bearing-support column was too short. I described my problem to the gentlemen at Colne Robotics over the phone and was told that I probably had the bearing ring—almost but not quite symmetrical part—on upside down.

I tried it again: I disassembled the bearing, inverted the bearing ring, and carefully placed the steel balls in the petroleum-jelly-coated track (I'm pretty good at this by now). Continuing as before, I installed the adjusting ring and beheld a smoothly operating shoulder bearing.

The instructions continued: put this motor here, put these gears there, and see the drawing. Well, I looked at the drawing. (The drawings are good up to a point, but they lack fine detail or close-ups in some areas.) I cheated a couple more times by looking at the assembled arm to verify my understanding of the drawings and text.

Assembly continued on the upper arm and forearm. The wrist posed no major problems. Then disaster struck! The fingers are held together with a large number of "cirkips" (split rings that fit around the outside of a shaft). The cirkips allow you to slide a rod through a hole, then prevent the rod from sliding back again. A special pair of cirkip pliers is an absolute necessity to proceed beyond this point. I tried to make do with what I had (needle-nose pliers, screwdrivers, etc.) and realized I definitely needed the proper tools. It would have been nice if the appropriate pliers came in the kit or were at least available as an option.

The final assembly of the hand progressed easily after I purchased the cirkip pliers. The instructions said to connect the arm assembly to the shoulder and base assembly. The cable threading came next. In the helpful hints section, the instructions said that this operation is greatly simplified by threading the arm before attaching it to the shoulder. So I started over again.

The actual cable threading progressed well, except for a clearance problem on one of the wrist cables. After checking the preassembled arm, I decided that cable clearance in the wrist is an assembly problem that Colne Robotics had experienced and corrected but had not updated in the manual. Ten minutes later, the offending cable had been restrung and worked smoothly.

The two printed-circuit boards went together just about as well as one would expect. No part numbers or reference designators were silk-screened on the boards, so I had to rely on the drawings in the manual for parts placement. Mounting the interface and motor-driver printed-circuit boards into the base of the ArmDraft and connecting the stepper-motor wires to the driver board completed the assembly operation.

Using the ArmDraft
A machine-language cassette for the TRS-80 Model I Level II microcomputer comes with the ArmDraft. The menu-driven program, named Learn, allows you to familiarize yourself with the operation of the robot arm and to create, modify, and save motion sequences.

The manual suggests reading through the software description quickly and proceeding to the "Introductory Demonstration Sequence" section, which tells you to load Learn and enter the learn mode by typing an "L".
The master printer interface at a very low cost

For the first time ever a truly affordable Apple interface offers all the most sophisticated text and graphics capabilities on Epson®, Okidata®, Centronics®, and IDS® printers. With the easy to use PKASO Interface, you simply slip it into your Apple Computer and attach the cable to your printer, and enjoy all these features:

- Broadest range of text printing using your software
- HiRes graphics with up to 40 creative options
- LoRes and Half Tone graphics in 16 levels of gray
- SuperRes plotting with up to 2160 x 960 points per inch
- User created or software defined characters and symbols
- Full text and graphics dump of absolutely any screen image.

Gray scale printing
Snapshot screen dump
Apple III compatibility

At Interactive Structures we've built our reputation on innovation, quality and service, and we're doing it again with the new PKASO series. The PKASO Interface will bring out the best in your Apple Computer, your data printer and your program. It will perform with all popular languages such as BASIC and ASSEMBLER. It will print both text and graphics with PASCAL. And it's the first and only Apple interface to offer all this plus support for the Apple Z-80 CP/M System and for full Apple III operation.

Don't settle for less. And don't pay more. Call us now for the name of the PKASO dealer near you. Circle 210 on inquiry card.

Interactive Structures, Inc.
112 Bala Avenue
P.O. Box 404
Bala Cynwyd, PA 19004
(215) 667-1713

Photo 5: The hand and wrist assembly has three finger fingers are opened and closed in unison under program c. The wrist allows both rotation and up-and-down motion.

This mode lets you manually operate the robot while gramming it to follow the same motions automatically.

The program asks you if you want to start again with the present position, or exit the program. Type "S" to clear the memory and free the arm. Then type "F" for free when no torque is applied to the stepper motor. This allows you to initialize the Armroid's position using the large gears in the shoulder. When you are satisfied with the starting position, press the space bar.

The program applies torque to the arm, effecting fine tuning and locking the arm in place.

You can now move the arm using the Q, W, E, R, and 1 through 6 keys to manually control the movements of the different parts of the arm. If you're like me, I take a couple of tries to predictably move the arm, then the wrist, and open and close the hand under manual control. Type "O" to get out of the learn mode.

Now the miracle of life! Press "G" for go, and the Armroid takes the shortest path to your initial starting point. The program then asks "O" (once) or "F" (for free). Forever seems like a long time for something you have tried yet, so type "O".

Wow! The arm is doing just what you taught it to do! And without the long pauses for head scratching and taking! You are returned to the menu.

To look at the sequence of commands that were sent to the stepper motors, type "D" for display. A table appears on the screen showing the stepper increment values stored in memory.

To extend the sequence of movements, simply repeat the learn mode, and type "C" for continue. You can add additional motions by using the manual control feature.

Once again, you must type "O" to return to the main menu.

After testing the new sequence, you may decide some of the motions need to be fine tuned. This can be done using the edit mode.
Three cassette-tape commands allow you to save your Armadroid sequence for a rainy day. "W" (write) saves the sequence in memory on the tape, "R" (read) retrieves the sequence from tape, and "C" (check) verifies that the data on the tape is the same as that in memory.

Colne Robotics has graciously included the source listing for the Armadroid control software in the manual. The Z80 assembly-language source is well documented and should prove to be a valuable learning tool for the student of robot technology. The source code is also useful to those who wish to modify the control software for a specific application.

I understand that Colne Robotics is developing similar software for other microcomputers, such as the Commodore PET, the Apple II, and the Sinclair ZX81. Watch their advertisements for further details.

Documentation
The 87-page manual is broken down into four sections. The introduction section is nine pages long and stands alone from the purpose of an experimental robot arm. Discussions on the economic and social impact of industrial robots, complete with tables and formulas, seem more like padding than useful information.

The second section deals with the mechanical assembly of the Armadroid. As noted above, some deficiencies and inaccuracies in the instructions exist. A hand-held, step-by-step approach would benefit the novice builder.

The next section details the electronics of the Armadroid. This section was not too bad, but again a step-by-step approach would be helpful.

The final section describes the software package included with the arm. This chapter of the manual was the easiest to use, due in part to the quality of the Learn program itself. And I applaud the inclusion of the program listing as an aid to understanding the ins and outs of microprocessor-controlled robotics.

It should be noted that my review is based on a "preliminary" manual for the Armadroid. I have been assured that the manual will be revised to eliminate some of the limitations that I have noted above.

Conclusions
• The Armadroid is a low-cost manipulator with good dexterity and maneuverability.
• The software delivered with the arm is easy to use and serves as a powerful tool in understanding robot operation.
• The Armadroid kit is not for the inexperienced builder unless the manual is improved.
• I feel I have learned a lot about the mechanics, electronics, and software of robots, thanks to the people at Colne Robotics.

---

Apple Logo
by Harold Abelson

The name Logo describes not only the evolving family of computer languages detailed in this book, but also a philosophy of education that makes full and innovative use of the teaching potential of modern computers. Apple Logo presents the Apple II user with a complete guide to the applications of this unique system and also includes a description of TI Logo for users of the Texas Instruments 99/4 computer.

The designers' vision of an unlimited educational tool becomes a reality for the Apple II user who begins to work with this procedural language. Logo enables even young children to control the computer in self-directed ways (rather than merely responding to it), yet it also offers sophisticated users a general programming system of considerable power.

Apple Logo actually teaches programming techniques through "Turtle Geometry"—fascinating exercises involving both Logo programming and geometric concepts. Later chapters illustrate more advanced projects such as an "INSTANT" program for preschool children and the famous "DOCTOR" program with its simulated "psychotherapist."

ISBN 0-07-00425-0
240 Pages
Softcover, spiral-bound
$14.95
Call Toll-Free 800/258-6420
BYTE Books, 70 Main Street
Peterborough, N.H. 03458

85
COLNE ROBOTICS

The

Colne Robotics

ARMODIO

Construction and Operation Manual

Published by
COLNE ROBOTICS LIMITED
1 Station Road
Twickenham
Middlesex TW1 4LL
(C) Copyright 1981
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>Introduction</strong></td>
<td><em>1-1</em></td>
</tr>
<tr>
<td>2. <strong>Mechanics</strong></td>
<td>*2-1*</td>
</tr>
<tr>
<td>2.1 Description</td>
<td>*2-2*</td>
</tr>
<tr>
<td>2.2 Technical Hints</td>
<td>*2-3*</td>
</tr>
<tr>
<td>2.3 Tools</td>
<td>*2-4* - *2-8*</td>
</tr>
<tr>
<td>2.4 Mechanical Parts</td>
<td>*2-9* - *2-14*</td>
</tr>
<tr>
<td>2.5 Assembly</td>
<td>*3-1* - *3-3*</td>
</tr>
<tr>
<td>3. <strong>Electronics</strong></td>
<td>*3-3* - *3-8*</td>
</tr>
<tr>
<td>3.1 Description</td>
<td>*3-4* - *3-5*</td>
</tr>
<tr>
<td>3.2 Component List</td>
<td>*4-1*</td>
</tr>
<tr>
<td>3.3 Assembly</td>
<td>*4-2* - *4-4*</td>
</tr>
<tr>
<td>4. <strong>Software</strong></td>
<td>*4-5*</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>*4-6* - *4-48*</td>
</tr>
<tr>
<td>4.2 Loading</td>
<td>*4-48* - *4-58*</td>
</tr>
<tr>
<td>4.3 General Description</td>
<td>*4-1*</td>
</tr>
<tr>
<td>4.4 Command Explanation</td>
<td>*4-5*</td>
</tr>
<tr>
<td>4.5 Introductory Demonstration Sequence</td>
<td>*4-48*</td>
</tr>
<tr>
<td>4.6 Detailed Software Description</td>
<td>*4-58*</td>
</tr>
</tbody>
</table>
INTRODUCTION

The development of Armadroid 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 Armadroid 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 Armadroid 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.
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 - 1*
### 2.4 ASSEMBLY

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

*2 - 4*
DESCRIPTION OF ITEM

Shoulder Side Plates
Switch support bar 107mm x M3 at ends
Support bar spacers M3 clearance X

Motor support bracket stiffener
107mm x M3 at ends
Support Bar spacers

Reduction gears
Reduction gear spindle 96mm x 6mm
Drive belts long = 114 teeth
 medium = 94 teeth
 short = 87 teeth

Upper Arm Drive Gear
small internal dim no drum
Upper arm side plates
Upper arm brace
Gears wrist action
 hand action
 fore arm
Idler pulley
Shoulder pivot 96mm x 8mm spindle
Fore arm side plates
Fore arm brace
Fore arm pulley

Part No
017
019
018/16
018/12
019
018/54
018/41
020
021
08/1 Hand
08/m Fore/Upp
08/s Wrist ac

*2 - 5*
<table>
<thead>
<tr>
<th>DESCRIPTION OF ITEM</th>
<th>Part No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elbow Idler pulleys hand wrist</td>
<td>033</td>
</tr>
<tr>
<td>Elbow spindle 65mm x 6mm</td>
<td>034</td>
</tr>
<tr>
<td>Wrist bevel gear carrier</td>
<td>035</td>
</tr>
<tr>
<td>Wrist guide pulleys</td>
<td>036</td>
</tr>
<tr>
<td>Wrist bevel gears (flanged)</td>
<td>037</td>
</tr>
<tr>
<td>Wrist pivots</td>
<td>038</td>
</tr>
<tr>
<td>Hand bevel gear (no flange)</td>
<td>039</td>
</tr>
<tr>
<td>Finger support flange</td>
<td>040</td>
</tr>
<tr>
<td>Hand pivot</td>
<td>041</td>
</tr>
<tr>
<td>Finger tip plates</td>
<td>041</td>
</tr>
<tr>
<td>Finger cable clamp</td>
<td>042</td>
</tr>
<tr>
<td>Small finger spring</td>
<td>043</td>
</tr>
<tr>
<td>Finger tip pivot</td>
<td>044</td>
</tr>
<tr>
<td>Middle finger plates</td>
<td>045</td>
</tr>
<tr>
<td>Middle finger pivot</td>
<td>046</td>
</tr>
<tr>
<td>Large finger spring</td>
<td>047</td>
</tr>
<tr>
<td>Finger base</td>
<td>048</td>
</tr>
<tr>
<td>Long finger pins 16mm x 3mm</td>
<td>050/1</td>
</tr>
<tr>
<td>Short finger pins 13mm x 3mm</td>
<td>050/s</td>
</tr>
<tr>
<td>Small finger pulleys</td>
<td>051</td>
</tr>
<tr>
<td>Large finger pulleys</td>
<td>052</td>
</tr>
<tr>
<td>Large hand sheave pulley</td>
<td>053</td>
</tr>
<tr>
<td>Small hand sheave pulley</td>
<td>054</td>
</tr>
<tr>
<td>Hand sheave pin</td>
<td>055</td>
</tr>
<tr>
<td>Finger tip pads</td>
<td>056</td>
</tr>
<tr>
<td>Base pan</td>
<td>057</td>
</tr>
</tbody>
</table>
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 ............................................... 102
NB + self made plug to protect the  
fine bearing thread ....................................

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

Bushers 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

*2 - 7*
<table>
<thead>
<tr>
<th>DESCRIPTION ITEM</th>
<th>Item No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor bolts, Base bearing to base</td>
<td>115</td>
</tr>
<tr>
<td>M4 x 10 Elbow spindle hex hd</td>
<td></td>
</tr>
<tr>
<td>Hand to finger, hand to bevel gear</td>
<td>116</td>
</tr>
<tr>
<td>M3 x 6 cheese hd</td>
<td></td>
</tr>
<tr>
<td>Shoulder spindle</td>
<td>117</td>
</tr>
<tr>
<td>M5 x 10 hex hd</td>
<td></td>
</tr>
<tr>
<td>General sheet metal fixing</td>
<td>118</td>
</tr>
<tr>
<td>M3 x 6 hex hd</td>
<td></td>
</tr>
<tr>
<td>M4 Nuts</td>
<td>119</td>
</tr>
<tr>
<td>M4 Washers</td>
<td>120</td>
</tr>
<tr>
<td>M4 Shakeproofs elbow spindle</td>
<td>121</td>
</tr>
<tr>
<td>M5 shakeproofs shoulder spindle</td>
<td>122</td>
</tr>
<tr>
<td>M3 Nuts</td>
<td>123</td>
</tr>
<tr>
<td>M3 washers - switches</td>
<td>124</td>
</tr>
<tr>
<td>6mm steel balls - base bearing</td>
<td>125</td>
</tr>
<tr>
<td>Magnetic reed switches</td>
<td>010</td>
</tr>
<tr>
<td>Driver board</td>
<td>126</td>
</tr>
<tr>
<td>Interface board</td>
<td>127</td>
</tr>
<tr>
<td>Edge connector</td>
<td>128</td>
</tr>
<tr>
<td>6mm Washers</td>
<td>129</td>
</tr>
<tr>
<td>Roll pins</td>
<td>130</td>
</tr>
<tr>
<td>4.5mm circlips</td>
<td>131</td>
</tr>
<tr>
<td>3mm circlips</td>
<td>132</td>
</tr>
<tr>
<td>Elbow spacer</td>
<td>133</td>
</tr>
</tbody>
</table>
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 in o 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 (046). 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 with the magnet at 2o'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 support (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 tweezerers to avoid melting the vaseline carefully place 24 bal 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 inser 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 tapping the parts together with a piece of wood 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 03h (M4) and the hand motor pulley. Then fit the hand reed switch bracket (M3) and the switch (M3 x 10 cheeshead 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 (O4u/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 (O8s/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 accurate 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) (M3 x 6mm) (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 pulley (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 grip 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 assembly 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 be 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 with 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 knuckleside 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 (50/16mm 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 (50/513mm 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 meth 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 a grub screw. A frined 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 effect 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 evenly positioned)  
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 light or you may cut through cable, though KEVLAR is very tough.
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 (D1) 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.

D1 indicates the direction of data travel, to the motors when D1 is low, from the microswitches, if installed, when D1 is high. The transition of D1 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. IC1 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

D1 is buffered and fed into a monostable (IC4) to generate a clock pulse. This causes the decoder to provide a latch pulse for approximately 500ns to the addresses motor control latch. D1 is tied to pull-up resistor (R1) so that the line is high except when are output from the microprocessor. The buffers IC1 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 bit1, so that when bit1 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 interrupt control, giving instant response to a microswitch change and avoiding having to poll the microswitches. 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

<table>
<thead>
<tr>
<th>ITEM</th>
<th>VALUE</th>
<th>QUANTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resistors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>1K0</td>
<td>3</td>
</tr>
<tr>
<td>R2</td>
<td>10K</td>
<td></td>
</tr>
<tr>
<td>R3-8</td>
<td>2K2 resitor</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>network</td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td>1K8</td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td>1K8</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td>1K8</td>
<td>3</td>
</tr>
<tr>
<td>R12</td>
<td>15K</td>
<td>1</td>
</tr>
<tr>
<td>R13</td>
<td>10K</td>
<td>2</td>
</tr>
<tr>
<td>R14</td>
<td>18ohm 5w</td>
<td>1</td>
</tr>
<tr>
<td>R15-R20</td>
<td>1K0</td>
<td>6</td>
</tr>
<tr>
<td>Capacitors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>100p polystyrene</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>1.0uf Tant</td>
<td>1</td>
</tr>
<tr>
<td>C3-C15</td>
<td>10nf ceramic</td>
<td>13</td>
</tr>
<tr>
<td>Semiconductors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IC1</td>
<td>74LS 125</td>
<td>2</td>
</tr>
<tr>
<td>IC2</td>
<td>74LS 125</td>
<td>1</td>
</tr>
<tr>
<td>IC3</td>
<td>74LS 04</td>
<td>1</td>
</tr>
<tr>
<td>IC4</td>
<td>74LS 123</td>
<td>1</td>
</tr>
<tr>
<td>IC5</td>
<td>74LS 366</td>
<td>1</td>
</tr>
<tr>
<td>IC6</td>
<td>74LS 138</td>
<td>1</td>
</tr>
<tr>
<td>IC7-IC12</td>
<td>74LS 175</td>
<td>6</td>
</tr>
<tr>
<td>IC13-IC16</td>
<td>ULN2003A</td>
<td>4</td>
</tr>
<tr>
<td>IC17</td>
<td>UA 7805</td>
<td>1</td>
</tr>
<tr>
<td>ZD1</td>
<td>BZX 13v ZENER</td>
<td>1</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MXJ 10 way edge connector</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 way PCB plug and socket connector</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Through Pins</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 pin IC sockets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 pin IC sockets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 way modified PCB plug and socket</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*3 - 3*
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.
F  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

ue = Pin 1 on I/P connector=0v   15v = Brown = Pin 2 on I/P connector

NOTE

A number of diagrams are given, explaining in detail the interconnections 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.

*3 - 4*
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

IC17 7805

15v
R14 18 5w
OV
5V
To rest of board
ED1 BZX 13v
X pin 9 IC's 13, 14, 15, 16.

* 3 - 5 *
COLNE ROBOTICS CO. LTD.

1 µF c2
10k r 13
15k r 12
1k8 r9
1k8 r10
1k8 r11
10K r2
10kΩ c1
1k r1
11 other Cap 10nf

14 DIL
10 way connector
16 DIL
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14
IC 15
IC 16
7805

ARMDROID CABLE

D7
D8
D5
D6
D3
D4
D1
D2
OV
5V
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-80 Model I Level II, 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 wherever 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 1/2 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 the 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 O (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 stores in a row for one motor is +127 or -128, so if a movement consists of more than this number it is accommodated 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 O (zero). To continue scrolling, press any other key. To display the figures one after the other, keep pressing O.

E(DIT)

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

Truncate a sequence by pressing R(OW 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.

!ET ARM!

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

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

(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 articular 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(O) 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)

Writers 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(O)

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 from the arm, thus allowing it to be moved by hand.

M(A)NUAL

Gives the user control of the movements of the arm direct from the keyboard. It is used (a) for practising manual control before L(EARNING), (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 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 TRS80 System level.
ARM TRAINER MK2AL

DIRECT FULL STEP MOTOR CONTROL

FOR TRS80 MODEL 1, LEVEL 11

BY ANDREW LENNARD

*** July 1981 ***
SECTION 1

A SYSTEM EQUATES

B SYSTEM VARIABLES

C SYSTEM CONSTANTS
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 L-6/Q-Y keys as explained in the manual section. Try just one movement alone at first. Now press O (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(O).

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 L-6/Q-Y keys as before. Now press O (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.
STEM EQUATES

RT EQU $F; ARM PORT NUMBER (37 E8H)
RSCN EQU 01C9H; SYSTEM RESTART (402DH)
NAD EQU 02B2H; SYSTEM PRINT CHARACTER
HR EQU 0333H; SYSTEM GET CHARACTER
HR EQU 0349H; SCAN KEYBOARD
D EQU 02BH; SYSTEM PRINT STRING
TSTR EQU 28A7H; CASSETTE ON
.SON EQU 0212H; CASSETTE OFF
.SOF EQU 01F8H; READ HEADER ON CASSETTE
HDR EQU 0296H; READ CHARACTER FROM CASSETTE
.STR EQU 0235H; WRITE HEADER TO CASSETTE
.STR EQU 0287H; WRITE CHARACTER TO CASSETTE
.STR EQU 0264H; ASCII MINUS
.INUS EQU ' - '; ASCII SPACE
.PAC EQU ' - '; ASCII NEW LINE
.D EQU 0DH; ASCII NUMBER BASE
.AXLE EQU 39H; UPPER BOARD FOR ARST ROW COUNTER
.LRSC EQU 01C4H; CLEAR SCREEN

ORG 1740H; = 4480 TRS80 HEX ADDRESS
; FOR START OF PROGRAM

*4 - 6*
VARIABLES USED

HIIN  DEFB $00  ; Has value of one if number input negative
HIN   DEFB $00  ; If HIN = zero then steps are stored
IAN   DEFB $00  ; If STRFG non zero then store TBUF array
STRFG DEFB $00  ; Set if key pressed in KEYIN Routine
KEYF  DEFB $00  ; Set if sequence to be done forever
FORFG DEFB $00  ; Number of motor slices stored
COUNT DEFB $0000  ; Pointer to next free motor slice

ARRAYS N  EQU $11

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

POSAR DEFS 12  ; Each two bytes of this six element array
              ; contain on value which is used to keep track of each
              ; motors 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 steps in two's
              ; complement arithemtic.

CTPOS 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.

MOTRF DEFS 6  ; A six byte array used by DRMF 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.
FTABL

DEFE 192 ; 12.8
DEFB 146 ; 16
DEFB 48 ; 32.
DEFE 96 ; 64

; FTABL is a small table which defines the
; order of the steps as they are sent out
; to the arr. To drive each motor the
; DRAMT routine adds the motors 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.
SECTION 2

COMMAND ROUTINES
CONSTANTS AND ARRAYS

*** COLNE ROBOTICS ARM CONTROLLER ***

(AL2) SIGON

DEFM 'REALLY QUIT? (Y/N)' DEFW 000DH

RELYQ DEFW 0DH

SIGOF DEFW 000DH DEFM 'YOU ARE NOW AT TRS80 SYSTEM LEVEL'

ECONS DEFW 000DH DEFM 'EDIT (M)OTOR STEP, OR (R)_OW COUNT?'

COUTS DEFM 'NEW UPPER ROW BOUND IS?' DEFW 000DH DEFB 0

EDSTR DEFM 'ROW NUMBER?' DEFB 0

BADMS DEFM '*** BAD INPUT VALUE ***' DEFW 000DH

MOTNS DEFM 'CHANGE STEPS ON WHICH MOTOR?' DEFB 0

NVALS DEFM 'REPLACEMENT STEP VALUE?' DEFB 0

QUESS DEFM 'LRN, READ, CHECK, WRITE, GO, DISP, BOOT, MAN, QUIT, SETA, TOST, EDIT, FREE' DEFW 000DH

RONRM DEFM 'DO (F)OREVER OR (O)nce?' DEFB 0

CASRD DEFM 'TYPE SPACE BAR WHEN READY, OF FULL STOP TO EXIT' DEFB 0

QMSS DEFM 'PARDON' DEFW 000DH

BOOTS DEFB 0DH DEFM 'WANT TO RE-START (Y/N)?'

RELNS DEFM 'START AGAIN OR (C)ONTINUE FROM CURRENT POSITION (. ) TO EXIT' DEFW 000DH

DISPS DEFB 0DH DEFM ' *** MOVEMENT ARRAY DISPLAY *** ' DEFW 000DH

NODIS DEFM ' *** NO SEQUENCE IN STORE *** ' DEFB 0DH DEFW 000DH

OVFMS DEFW 000DH DEFM 'NO MORE ARM STORE LEFT, DELETE OR SAVE?'

DONMS DEFB 0DH DEFM 'SEQUENCE COMPLETE' DEFW 000DH

RDMSG DEFW 000DH DEFM '*** READ ERROR ***'

TAPOK DEFW 000DH DEFM '*** TAPE OK ***'

STRST DEFW 000DH DEFM 'ARM RESET'

NOTOR DEFW 000DH DEFM 'ARM NOW FREE TO MOVE'
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TORMS</td>
<td>DEFB</td>
</tr>
<tr>
<td></td>
<td>DEFB</td>
</tr>
<tr>
<td></td>
<td>DEFM</td>
</tr>
<tr>
<td></td>
<td>DEFW</td>
</tr>
<tr>
<td>POSST</td>
<td>DEFM</td>
</tr>
<tr>
<td></td>
<td>DEFB</td>
</tr>
</tbody>
</table>
COMMAND INDEX

STARM........4-12. Program entry point
LEARN........4-13.... Learn a sequence command
EDIT...........4-14.... Edit a sequence command
READ...........4-16.... Read in sequence from tape command
WRITE...........4-17.... Write sequence to tape command
CHECK...........4-18.... Check stored sequence command
BOOT...........4-19.... Re-start system command
FINISH...........4-19.... Exit from system command
SETARM.........4-20.... Set start position command
TOSTM.........4-20.... Move arm to start position command
FREARM.........4-20.... Free all arm joints
MANU...........4-20.... Go into manual mode
GO...............4-21.... Execute stored sequence command
DISPLAY.........4-21.... Display stored Sequence command

*4 - 11*
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
CALL DELT ; Small delay
LD HL,QUESS ; Point to menue string
CALL PSTR ; Print it
CALL GCHRA ; Get response and print it
CALL PNEWL ; Print new line
CP NL ; Is response a newline
JR Z,QUESS1 ; 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,SECTAM ; 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,PREARM ; Yes then clear all motors
CP 'Q' ; a 'Q'
JP Z,FINISH ; Yes then quit program
LD HL,QMESS ; Point to 'PARDON' message
CALL PSTR ; Print it
JP QUES1 ; Try for next command

*4 -12*
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 GCRA ; Get response and print it
CALL PNEWL ; Print a new line
CP ' ' ; Response a ' .'
JP Z, QUES1 ; Back to main loop if user 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 GCRA ; Get response and print it
CALL PNEWL ; Print new line character
CP ' ' ; Response a ' .'
JP Z , QUES1 ; Exit to main loop if so
CP ' ' ; Is it a space?
JR NZ , WAIT2 ; If not then bad input, try again
CALL TORQUE ; Switch motors on
JR STLNR ; Do rest of learn

NOINT
LD HL,(COUNT) ; Get current count
LD A,L ; Is it zero?
OR H ;
JR Z , NOSTR ; Yes then can't add to nothing

STLNR
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

*4 - 13*
EDIT FUNCTION

EDIT   LD    HL,(COUNT)  ; Get row count
       LD    A,L
       OR    H
       JP    Z,NOSTR  ; Yes then nothing in store
EDSRT  LD    HL,ECONS  ; Print edit message
       CALL   PSTR
       CALL   GCHRA
       CALL   PNEWL
       CP     'M'
       JR     Z,EDMOT  ; Yes then edit motor
       CP     'R'
       JR     NZ,EDSRT
       LD    HL,COUTS
       CALL   PSTR
       CALL   GINT
       JP     NZ,BADC
       LD    A,H
       BIT    7,A
       JP     NZ,BADC
       LD    BC,(COUNT)  ; Get count value
       PUSH   HL
       OR     A
       SBC    HL,BC
       POP    HL
       JR     NC,BADC
       LD    (COUNT),HL
       JP     QUESL
       PUSH   HL
       OR     A
       SBC    HL,BC
       POP    HL
       JR     NC,BADC
       LD    A,H
       BIT    7,A
       JP     NZ,BADC
       LD    A,H
       OR     L
       JR     Z,BADC
       LD    BC,(COUNT)
       INC    BC
       PUSH   HL
       SBC    HL,BC
       POP    HL
       JR     NC,BADC
       DEC    HL
       ADD    HL,HL
       PUSH   HL
       ADD    HL,HL
       POP    BC

EDMOT   LD    HL,EDSTR
       CALL   PSTR
       CALL   GINT
       JR     NZ,BADC
       LD    A,H
       BIT    7,A
       JR     NZ,BADC
       LD    A,H
       OR     L
       JR     Z,BADC
       LD    BC,(COUNT)
       INC    BC
       PUSH   HL
       SBC    HL,BC
       POP    HL
       JR     NC,BADC
       DEC    HL
       ADD    HL,HL
       PUSH   HL
       ADD    HL,HL

EDOK

*4 - 14*
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
LD A,L
CP 1
JR C,BADUM ; Response too large
CP 7
JR NC,BADNM ; No motor number < 1
POP HL ; No motor number > 6
DEC A ; Restore = Memory pointer
LD C,A ; Motor offset $ \rightarrow 5$
LD B,\$ ; 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 $\$FFH$
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
JR PEDIT ; New positive step value too
BIT 7,L ; large
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 ; Print error message and
BADC 
LD HL,BADMS ; return to main loop
CALL PSTR 
JP QUES1
READ ROUTINE

; Reads stored sequence from cassette
; into memory

READ
LD  HL,CASRD  ; Point to wait message
CALL PSTR     ; Print it
CALL GCHRA     ; Get response
CALL PNEWL     ; Print new line
CP  ' '        ; Is response a dot?
JP Z, QUES1    ; Yes then exit
CP  SPAC       ; Is it a space?
JR NZ, READ    ; No then try again
XOR A          ; Clear A=Drive zero
CALL CASON     ; Switch on drive zero
CALL DELS      ; Short delay
CALL RDHDR     ; Read header from tape
CALL READC     ; Read first character
LD  B,A        ; Put in B
CALL READC     ; Read second character
LD  C,A        ; Place in C
OR  B          ; BC now equals count
JP Z, NOSTR    ; Count zero, so exit
LD  (COUNT),BC ; Set count = read count
LD  HL, ARST   ; Point to start of store
ROWNR PUSH BC   ; Same count
LD  E, Ø       ; E = Check sum for a row
LD  B, 6       ; B = Column Count
RDBYT CALL READC  ; Read a row element
LD  (HL),A     ; Store it
ADD A, E       ; Add it to check sum
LD  E, A       ; Store in check sum
INC HL         ; Inc memory pointer
DJNZ RDBYT     ; Do next element
POP BC         ; Restore row count
CALL READC     ; Read check digit
CP  E          ; Same as calculated?
JR NZ, RDERR   ; No then error
DEC BC         ; Decrement row count
LD  A,B        ; See if row count
OR  C          ; is zero
JR NZ, ROWNR   ; No then read next row
CALL CASOF     ; Switch cassette off
JP TPEEF       ; exit
RDERR LD  HL, RDMGS ; 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,QUESl ; Yes then exit
CP SPAC ; Is answer a space
JR NZ,BADWI ; No then try again
XOR A ; Clear drive number
CALL CASON ; Switch on drive zero
CALL DELT ; delay
CALL WRLDR ; Write Leader
CALL DELT ; delay
LD BC,(COUNT) ; Get count into BC
LD A,B ;
CALL WRBYA ; Write higher byte
LD A,C ; Get lower byte of count into A
CALL DELT ; delay
CALL WRBYA ; Write lower byte
LD HL,ARST ; Point to start of sequence of store
ROWNW
PUSH BC ; Save row count
LD E,Ø ; Clear check sum
LD B,6 ; Six motor slots per row
WRBYT
LD A,(HL) ; Get motor slot N
CALL DELS ; delay
CALL WRBYA ; Write it
CALL DELS ; delay
ADD A,E ; add to check sum
LD E,A ;
INC HL ; Inc memory pointer
DJNZ WRBYT ; Do for all six motors
CALL WRBYA ; Write check sum
POP BC ; Restore row count
DEC BC ; Decrement row count
LD A,B ;
OR C ; Test if zero
JR NZ,ROWNW ; No then try again
CALL CASOF ; Switch cassette off
JP QUESl ; Back to main loop

*4 - 17*
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
CALL PSTR ; Print wait message
CALL GCHRA
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
PUSH BC ; Save count
LD E, $0 ; Check sum is zero
LD B, 6 ; Count is 6
CALL READC ; Read a motor step element
CP (HL) ; Same as in store?
ADD A, E ; Not the same so error
LD E, A
INC HL
DJNZ CKBYT ; Advance memory pointer
POP BC ; Do next row element
CALL READC ; Restore row count
CP E ; Read check sum
JP NZ, RDERR ; Same as check sum calculated
DEC BC ; No then error
LD A, B ; Decrement count
OR C
JP NZ, ROWNC ; Is count zero?
CALL CASOF ; No then do next row
CALL TAPOK ; Switch cassette off
LD HL, TAPOK ; Print tape off message
CALL PSTR ; and back to main loop
JP QUES1

*4 - 18*
BOOT AND FINISH COMMANDS

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

; This is the exit from program Section to TRS80
; system level
FINSH LD HL,RELYQ ; Print "REALLY QUIT"
CALL PSTR ;
CALL GCHRA ; Get answer
CP 'Y' ; User typed a 'Y'
JR NZ,TRYNO ; No then try 'N'
LD HL,SIGOF ; Print ending message
CALL PSTR ; and then
JP FINAD ; return to TRS80 System
TRYNO CP 'N' ; User typed an 'N'
JR NZ,FINSH ; No then try again
CALL PNEWL ; Print a new line
JP QUES1 ; Back to main loop
OTHER SHORT COMMANDS

; SETAM clears arm position array

SETAM CALL PESET ; 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,1 ; Set in manual mode for the
    LD (MAN),A ; keyin routine

MANUA CALL KEYIN ; Now get keys and move motors
    JP NZ,MANUA; If non zero then move to be done
    XOR A ; Clear manual flag
    LD (MAN),A ;
    JP QUES1 ; Back to main loop

*4 - 20*
THE GO COMMAND

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

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

*4 - 21*
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
CALL PSTR  ; Point to header string
CALL POSDS  ; and display it
LD HL,ARST
LD BC,(COUNT)
LD A,B
CR C
JP NZ,SETBC
CALL PSTR  ; Print out the relative position
CALL NODIS  ; Point to sequence start
JP QUES1
LD HL,NODIS
CALL PSTR  ; BC = how many rows to print
CALL SETBC
LD EC,000
JP QUES1
DOROW
CALL EC  ; Test if count is zero
POP HL
LD H,B
LD L,C
TNC HL
LD 1X,NUMAR
CALL CBTAS
CALL HL,NUMAR
CALL PSTR  ; Display else print message
LD A,'.'
CALL PUTCCHR  ; telling user nc display and
POP HL
LD B,6
LD A,(HL)
CALL PUTCCHR ; return to the main loop
POP HL
LD EC,000
CALL PSTR  ; HL = row count
CALL CBTAS
CALL HL,NUMAR
CALL PSTR  ; Convert HL to ASCII
LD A,'.'
CALL PUTCCHR  ; Point to ASCII string
POP HL
LD B,6
LD A,(HL)
CALL PUTCCHR ; now print it
POP HL
LD EC,000
CALL PSTR  ; Print a '.
POP HL
LD B,6
LD A,(HL)
CALL PUTCCHR ; Restore memory pointer
POP HL
LD 1X,NUMAR
CALL CBTAS
CALL HL,NUMAR
CALL PSTR  ; Motor count to B (6 motors)
CALL PSTR  ; Get step value
CALL HL,NUMAR
CALL EVAL  ; Save memory pointer
CALL HL,NUMAR
CALL EVAL  ; Save motor count
CALL HL,NUMAR
CALL EVAL  ; Test bit 7 of A for sign
CALL HL,NUMAR
CALL EVAL  ; If bit = 0 then positive step
CALL HL,NUMAR
CALL EVAL  ; Make H = negative number
CALL HL,NUMAR
CALL EVAL  ; Do rest
CALL HL,NUMAR
CALL EVAL  ; Clear H for positive number
CALL HL,NUMAR
CALL EVAL  ; Get low order byte into L
CALL HL,NUMAR
CALL EVAL  ; Point to result string
CALL HL,NUMAR
CALL EVAL  ; Call conversion routine
CALL HL,NUMAR
CALL EVAL  ; HL points to result
CALL HL,NUMAR
CALL EVAL  ; Print resulting conversion
CALL HL,NUMAR
CALL EVAL  ; Get keyboard memory location
CALL HL,NUMAR
CALL EVAL  ; Test for zero key pressed
CALL HL,NUMAR
CALL EVAL  ; Not pressed, then skip
CALL HL,NUMAR
CALL EVAL  ; Wait till next character entered
CALL HL,NUMAR
CALL EVAL  ; Is it a dot?
CALL HL,NUMAR
CALL EVAL  ; No then, carry on
CALL HL,NUMAR
CALL EVAL  ; else print a new line
CALL HL,NUMAR
CALL EVAL  ; and restore all the registers
CALL HL,NUMAR
CALL EVAL  ; and the stack level

*4 - 22*
NOSTP

POP BC ; Jump back to main loop
JP QUES1 ; Restore column count
POP BC ; Restore memory pointer
POP HL ; Increment memory pointer
INC HL ; Print a space between numbers
CALL PSPAC

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

*4 - 23*
SUBROUTINES INDEX

DOALL........4-26 ........Execute a stored sequence once
DRIVL........4-27 ........Drives all motors directed by TBUF
INIT........4-28 ........Set up system
MOVTO........4-28a .........Use POSAR to rest system arm
TORQUE........4-29 .........Turn on off motors
CLRMT........4-29 .........Turn off all motors
SETDT........4-30 .........Reset CTPOS elements to one
DRAMT........4-31 .........Drive directed motors
STEPM........4-32 .........Step motors via DRAMT
DNEWD........4-33 .........Delay on direction change
SPANT........4-34 .........Update TBUF array during learn
KEYIN........4-35 .........Scan keyboard and build up motors to move
CBTAS........4-37 .........Convert 16 bit 2's complement number to ASCII
CLRMF........4-39 .........Clear MOTBF array
CTBUF........4-39 .........Clear TBUF, DRBUF & MOTBF arrays
GINT........4-40 .........Get 16 bit signed value from keyboard
POSDS........4-42 .........Display relative position array elements
POSIĆ........4-42 .........Increment relative position array elements
STORE........4-44 .........Copy TBUF to current ARST slice
RESET........4-45 .........Clear POSAR array
PUTCHR........4-46 .........Print a character
PSTR........4-46 .........Print a string
PSPAC........4-46 .........Print a space
PNEWL........4-46 .........Print a carriage return
SUBROUTINES INDEX (continued)

SCKBD..................Scan the keyboard
G CHRN..................Get a character and print it
CLRSC..................Clear the Screen
DELSW.................Delay on value in B
DELS..................Delay approx 0.001 sec
DELT.................Delay approx 0.01 sec
DELLN..................Dealay approx 1.0 sec
SUERCUTINE 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
         PCSH   BC ; Save count
         LD    BC, $006 ; Motor count of six
         LDIR   HL ; Copy memory slice into TBUF
         PUSH   HL ; Save new memory pointer
         CALL   DRIVL ; Drive all motors for this slice
         CALL   SRKBD ; See if keyboard input
         POP    HL ; Restore memory pointer
         POP    BC ; Restore row count
         CALL   DNEWD
         CP    '.' ; User typed a '.'
RET2     XGR    A ; No then continue
         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
SUFRoutine DrivL

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

DRIVL    LD    C,Ø         ; Set BC = motor count
SCANW     LI    E,6        ; 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    TBNZR          ; Do next motor check
RET             ; If no motor to step, then ret;
TBNZR     LD    DE,MOTBF + 5 ; DE points to last direction a:
        LD    HL,TBUF + 5 ; HL points to TBUF
        LD    B,6        ; B = motor count
DGAGN     LD    A,(BL)     ; Get motor step value
        CP    Ø          ; Is it zero?
JR      Z,NOEL        ; Yes then skip
JP      M,SNEG        ; Is it negative ie, reverse
SFCS      LD    A,3        ; No positive, so load MOTBF (N
        LD    (DE),A     ; With 3
DEC     (HL)          ; Decrement motor count in TBUF
JR      NCFIL         ; Complete the MOTBF array
SNEG      LD    A,1        ; Set MOTBF = 1 for
        LD    (DE),A     ; a positive drive
INC     (HL)          ; Decrement negative count
JR      NCFIL         ; Do rest of MOTBF
NOEL      XOR    A        ; Clear MOTBF (N
        LD    (DE),A     ;
NOFIL     EBC   DE        ; Move to next MOTBF element
        DEC   HL         ; Move to next TBUF element
        DJNZ  DOAGN      ; Do for all six motors
        LD    A,1        ;
        LD    (KEYP),A   ; Set key pressed flag
CALL    STEP4         ; Step all motors once if
DEC     C              ; any to step
JF      NZ,SCANW      ; Do for maximum of 128 cycles
        RET

* 4 - 27 *
SUBROUTINE INIT

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

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

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

MOVTO    PUSH    AF         ; *
        PUSH    BC         ; *
        PUSH    CE         ; * Save registers
        PUSH    HL         ; *
RESL1     LD      HL,POSAR  ; HL points to POSAR
        LD      B,12      ; B = count of 12
NRESL1    LD      A,(HL)    ; Get POSAR element
        OR      A         ; Is it zero?
        JR      NZ,MTSA   ; No then continue
        INC     HL        ; Point to next POSAR element
        DJNZ     NRESL1    ; See if all zero
        JR      ENDSCE     ; All zero so end!
MTSA      LD      HL,POSAR+10 ; HL points to POSAR
        LD      DE,MOTBF+5 ; DE points to MOTBF
        LD      B,6       ; B = count
RSCAN     PUSH    BC         ; Save count
        LD      C,(HL)    ; Get lower byte
        INC     HL        ; Advance HL pointer
        LD      B,(HL)    ; Get high byte of POSAR element
        LD      A,C       ; Get low byte into A
        OR      B         ; See if POSAR(N) is zero
        JR      Z,DOMPL   ; No skip
        LD      (DE),A    ; Zero MOTBF (N)
        INC     HL        ; advance POSAR pointer
        JR      NMDR      ; Do next motor
DOMFL     LD      A,B       ; See direction to move in
        BIT      7,A       ;
        JR      Z,RMOT1   ; Go in reverse
        INC     BC         ; Go forward
        LD      A,1       ; A = forward
        JR      DOIT1     ; Do rest
RMOT1     DEC     EC         ; Dec count for reverse
        LD      A,3       ; Set reverse in A
        LD      (DE),A    ; Store reverse in MOTBF (N)
        DEC     HL         ; Store updated POSAR count
        LD      (HL),B    ; in POSAR (N)
        DEC     HL         ; Store lower byte
        LD      (HL),C    ; point to next POSAR element
        DEC     DE         ; Move to next MOTBF element
        POP     BC         ; Restore motor count
        POP     DE         ; Do for next motor
        POP     EC         ; Drive all motors to be drive
        POP     AF         ; Do till all POSAR slots zero
        JR      RESL1     ; *
        POP     HL         ; *
        POP     DE         ; * Restore all registers
        POP     BC         ; *
ENDSC     POP     AF         ; Return

*4 - 28a*
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,NOTBF ; Point to last drive table
  LD B,6 ; B = motor count
  LD A,(HL) ; Get motor value
  OR A ; Is it zero?
  JR NZ,TORQ2 ; No then skip
  LD A,1 ; Reset CTPOS(N) to position 1
  LD (DB),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 TQQCL ; Exit with register restoration

CLRMT
  PUSH AF ; * clear all motors torque
  PUSH BC ; *
  PUSH DE ; * Save Registers
  PUSH HL ; *
  LD HL,NOTOR ; Print "NO TORQUE" message
  CALL PSTR ;
  LD D,ØFØH ; Pattern for motors off
  LD B,6 ; B = Motor count
  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 CLRMT ; Do next motor
  CALL SETDT ; Clear CTPOS array to value of 1

TQQCL
  POP HL ; *
  POP DE ; *
  POP BC ; * Restore Registers
  POP AF ; *
  RET ; Done, exit

*4 - 29*
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 CTFCS array
LD (HL),1 ; Set CTPOS(N) to start position
INC HL ; Increment HL
DJNZ NSET1 ; Do set up next CTPCS element
POP HL ; *
POP DE ; * Restore used registers
POP BC ; *
RET ;

*4 - 30*
SUBROUTINE DRAMT

; DRAMT drives all six motors directly and uses
; FTABLE to output the correct pulse patterns.
; For half stepping the pattern must be changed in FTABLE
; 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 FTABLE offset array
NMTDT
LD A, (DE)       ; Get MOTBF(N)
OR A             ; Is it zero?
JR Z, IGMTN      ; If zero, then skip
BIT l,A          ; Test direction
CALL OUTAM       ; Step motor
JR Z, REVMT      ; If direction negative then jump
INC A            ; Increment table counter
CP 5             ; Upper bound?
JR C, NORST      ; No then continue
LD A, 1          ; Reset table offset
NORST
LD (HL), A       ; Store in CTPOS (N)
IGMTN
INC HL           ; Increment CTPOS pointer
DEC DE           ; Decrement MOTBF pointer
DJNZ NMTDT       ; Do for next motor
CALL DELT        ; Delay after all pulses out
CALL DELS        ; *
POP HL           ; *
POP DE           ; *
POP BC           ; * Restore Registers
POP AF           ; *
RET              ; Exit

REVMT
DEC A            ; Move table pointer on
CP 1             ; Compare with lower bound
JR NC, NORST     ; If no overflow then continue
LD A, 4          ; Reset table offset
JR NORST         ; Do next motor
OUTAN
LD A, (HL)       ; Get table offset l-4
PUSH AF          ; *
PUSH DE          ; * Save Registers
PUSE HL          ; *
LD HL, FTABLE-1  ; Get table start
LD D, @         ; DE now equals 1-4
LD E, A          ; Add to FTABLE -1 to get address
ADD HL, DE       ; Get motor pulse pattern
LD A, (HL)       ; Get address field in C and
LD C, B          ; shift it one to the left
SLA C            ; or in the pulse pattern
DE C             ; Output to interface circuitry
CUT (PGRT), A    ; *
POP HL           ; * Restore Registers
POP DE           ; *
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.

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

*4 - 32*
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 ; HL 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

*4 - 33*
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 slt
; 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  1X, DBUF+6 ; 1X = previous direction buffer
LD  1Y, MOTBF+6 ; 1Y = current buffer
LD  HL, TBUF +6 ; HL = step buffer

NTMOT
DEC 1Y ; move pointers
DEC 1X
DEC HL
LD  A, (1Y +Ø) ; Get current motor direction
OR  A ; No work to do
JR  Z, NODRV ; skip, if so
CP  1 ; Reverse
JR  Z, REVDR ; Yes then skip
FORDR
LD  A, (1X +Ø) ; Get previous direction
CP  1 ; Direction change?
JR  NZ, CFORD ; No then advance TBUF(N) step
CALL SETST ; Set the store flag
LD  (1Y +Ø), Ø ; Clear MOTBF element.
JR  NODRV ; Do next motor
CFORD
INC (HL) ; Increment motor step in TBUF
LD  A, (HL) ; Get new value
CP  127 ; Check against upper board
CALL Z, SETST ; Limit reached then store flag
LD  (1X+Ø), 3 ; Set previous direction
NODRV
DJNZ NTMOT ; Do next motor
CALL STEPM ; Step motors to be driven
LD  A, (STRFG) ; Examine store flag
OR  A ; Zero?
JR  NZ, STORE ; No then do store operation
RET ; Exit
REVDR
LD  A, (1X +Ø) ; Get previous direction
CP  3 ; Direction reversed?
JR  NZ, CREVL ; No then continue
CALL SETST ; Else set store TBUF in ARST fl
JR  NODRV ; clear MOTBF element
CREVL
DEC (HL) ; Advance step count in TBUF (N)
LD  A, (HL) ; Get element
CP  -128 ; Compare with upper negative bc
CALL Z, SETST ; Limit reached so set store fls
CREVD
LD  (1X +Ø), 1 ; Set Direction
JR  NODRV ; Do next motor
SETST
PUSH AF ; Save AF
LD  A, 1 ; Set store flag STRFG
SETSC
LD  (STRFG), A ; to one
PCH AF ; Restore AF
RET ; Continue
SUBROUTINE KEYIN

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

KEYIN     CALL CLRMF ; Clear MOTBF array
         LD  A,(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,Ø ; Clear MOTBF offset in BC
         BIT 1,A ; See if '1' key is pressed
         JP  Z,TRYN2 ; No then skip else
         CALL FORMT ; Set up motor 1 position in MOTBF

TRYN2     INC  BC ; Increment MOTBF offset
         BIT 2,A ; See if '2' key pressed
         JP  Z,TRYN3 ; No skip
         CALL FORMT ; Set second motor forward

TRYN3     INC  BC ; Advance offset
         BIT 3,A ;
         JP  Z,TRYN4 ; See if '3' key pressed, No skip
         CALL FORMT ; Set forward direction on Motor 3

TRYN4     INC  BC ; Increment offset in BC
         BIT 4,A ; See if key '4' is pressed
         JP  Z,TRYN5 ; No then test key '5'
         CALL FORMT ; Do forward direction for Motor 4

TRYN5     INC  BC ; Advance offset
         BIT 5,A ; Key '5' pressed
         JP  Z,TRYN6 ; No skip
         CALL FORMT ; Do set up for motor 5

TRYN6     INC  BC ; Advance offset
         BIT 6,A ; Key '6' pressed
         JP  Z,TRYQT ; No then try 'Q'
         CALL FORMT ; Do for motor 6

*4 - 35*
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRYQT</td>
<td>LD BC,$0</td>
</tr>
<tr>
<td></td>
<td>LD A,(3804H)</td>
</tr>
<tr>
<td>TRYQ</td>
<td>EIT 1,A</td>
</tr>
<tr>
<td></td>
<td>JP Z,TRYW</td>
</tr>
<tr>
<td></td>
<td>CALL BACMT</td>
</tr>
<tr>
<td></td>
<td>INC BC</td>
</tr>
<tr>
<td></td>
<td>BTT 7,A</td>
</tr>
<tr>
<td></td>
<td>JP Z,TRYR</td>
</tr>
<tr>
<td></td>
<td>CALL BACMT</td>
</tr>
<tr>
<td></td>
<td>INC BC</td>
</tr>
<tr>
<td>TRYE</td>
<td>LD A,(3801H)</td>
</tr>
<tr>
<td></td>
<td>BIT 5,A</td>
</tr>
<tr>
<td></td>
<td>JR Z,TRYR</td>
</tr>
<tr>
<td></td>
<td>CALL BACMT</td>
</tr>
<tr>
<td>TRYR</td>
<td>INC BC</td>
</tr>
<tr>
<td></td>
<td>LD A,(3804H)</td>
</tr>
<tr>
<td></td>
<td>BIT 2,A</td>
</tr>
<tr>
<td></td>
<td>JR TRYT</td>
</tr>
<tr>
<td></td>
<td>CALL BACMT</td>
</tr>
<tr>
<td></td>
<td>INC BC</td>
</tr>
<tr>
<td>TRYT</td>
<td>BIT 4,A</td>
</tr>
<tr>
<td></td>
<td>JP Z,TRYY</td>
</tr>
<tr>
<td></td>
<td>CALL BACMT</td>
</tr>
<tr>
<td></td>
<td>LD A,(3808H)</td>
</tr>
<tr>
<td>TRYY</td>
<td>INC BC</td>
</tr>
<tr>
<td></td>
<td>BIT 1,A</td>
</tr>
<tr>
<td>SOMEN</td>
<td>JR Z,SOMEN</td>
</tr>
<tr>
<td></td>
<td>CALL BACMT</td>
</tr>
<tr>
<td>NOTNG</td>
<td>OR l</td>
</tr>
<tr>
<td></td>
<td>RET</td>
</tr>
<tr>
<td></td>
<td>LD A,(MAN)</td>
</tr>
<tr>
<td></td>
<td>OR A</td>
</tr>
<tr>
<td></td>
<td>CALL Z,STORE</td>
</tr>
<tr>
<td></td>
<td>XOR A</td>
</tr>
<tr>
<td>FORMT</td>
<td>RET</td>
</tr>
<tr>
<td></td>
<td>LD E,3</td>
</tr>
<tr>
<td>BACMT</td>
<td>JR SETMT</td>
</tr>
<tr>
<td>SETMT</td>
<td>LD E,1</td>
</tr>
<tr>
<td></td>
<td>LD HL,MOTBF</td>
</tr>
<tr>
<td></td>
<td>ADD HL,BC</td>
</tr>
<tr>
<td></td>
<td>PUSH AF</td>
</tr>
<tr>
<td></td>
<td>LD A,(HL)</td>
</tr>
<tr>
<td></td>
<td>OR A</td>
</tr>
<tr>
<td></td>
<td>JR Z,DOMOT</td>
</tr>
<tr>
<td></td>
<td>XOR A</td>
</tr>
<tr>
<td></td>
<td>LD (HL),A</td>
</tr>
<tr>
<td></td>
<td>POP AF</td>
</tr>
<tr>
<td></td>
<td>RET</td>
</tr>
<tr>
<td>DOMOT</td>
<td>LD (HL),E</td>
</tr>
<tr>
<td></td>
<td>LD A,1</td>
</tr>
<tr>
<td></td>
<td>LD (KEYP),A</td>
</tr>
<tr>
<td></td>
<td>POP AF</td>
</tr>
</tbody>
</table>

*4 - 36*
SUBROUTINE CBTAS

; This subroutine makes a signed binary value in HL into an ASCII string and stores the string
; in the locations pointed to by IX

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

*4 - 37*
DEFW 10000
DEFW 1000
DEFW 100
DEFW 10
DEFW 1

; Table of subtraction constants
; for conversion routine
CLEARING AND RESETING ROUTINES

; CLRMF clears the MOTBF array

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

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

CTBUF
PUSH BC          ; * Save Registers
PUSH DE          ; * Save Registers
PUSH HL
LD HL,TBUF       ; HL points to TBUF(Ø)
LD DE,TBUF +1    ; DE points to TBUF(1)
LD BC,17         ; BC = Count of 17
LD (HL),Ø        ; Clear first element
LDIR             ; Now clear next 17 elements
POP HL
POP DE          ; * Restore Registers
POP BC
RET             ; Exit
S U E R C O U T I N E G I N T

; This subroutine gets a signed 16 bit integer
; from the TRS80 Keyboard.
; If a bad number is typed it returns with the
; Status flag - non zero.
; The 2's complement number is returned in HL

G I N T
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
G I N T 1
CALL GCHRA ; Get a character and display
CP SP,C ; Is it a space?
JR Z,G I N T 1 ; Yes then skip
CP N I ; Is it a newline?
JP Z,P R E T 1 ; Done if new line, return ze:
CP MINUS ; A minus number?
JR NZ,POS N ; No then see if positive
LD A,1 ; Set minus flag
LD (MIN),A
JR G I N T 2 ; Get rest of number
CP '4' ; Is number a positive number
JR NZ,NUM 1 ; See if numeric
G I N T 2
CALL GCHRA ; Get next character
CP N L ; Newline?
JR Z,N U M E T ; 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 $ ;
JR C,E R R N 2 ; If number less than ASCII Ø
CP '9' + 1 ; If number greater than ASCII 1
JR NC,E R R N 2 ; 9 then error
SUB NUMBA ; Number input OK, so make int
LD E,A ; Binary and
LD D,Ø ; load into DE
ADD HL,DE ; Now add to total
DJNZ G I N T 2 ; Do for next digit
CALL PN E W L ; Print a new'line
N U M E T
LD A,(MIN) ; Is number negative?
OR A ;
JR Z,P R E T 1 ; No then finish off
LD A,L ; else complement
C P L ; The value in HL
LD L,A ;
LD A,H ; (2's Complement)

*4 - 40*
CPL
LD  H, A
INC  HL
XOR  A
FPOP  DE
POP  BC
RET

Clear A and flags
* Restore Registers
* and return

ERRN2
CALL  PNEWL
LD  A, 1
OR  A
SBC  HL, HL
OR  A
JR  PRET2

Print a newline
Set A to 1
Clear carry flag
Clear HL
Clear carry flag
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  EC    ; *
        PUSH  DE    ; * Save all registers
        PUSH  HL    ; *
        LD     HL,POSS1 ; Print "FELPCS=
        CALL   PSTR    ; String
        LD     B,6    ; Motor count into B
        LD     DE,POSAR ; Point to array containing offs
                       :
                       :
        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   NPGSA    ; Do for next motor
        CALL   PNEWL    ; Print a new line, all done
        POP    HL    ; *
        POP    DE    ; *
        POP    BC    ; * Restore all Registers
        POP    AF    ; *
        RET    ; Now return
SUBROUTINE PCSIC

; PCSIC 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 3/2 times to cause such an event.

FCSIC
    PUSH AF
    PUSH BC
    PUSH DE
    PUSH HL
    LD E,6
    LD DE,MOTBF+5
    LD HL,POSAR+10

NPOS1
    PUSH BC
    LD C,(HL)
    INC HL
    LD B,(HL)
    LD A,(DE)
    AND 3
    OR A
    JR NZ,NONZM
    JR NPOS2

NONZM
    BIT 1,A
    JR NZ,RPOS
    JR STPCS

RPOS
    DEC (HL),B
    INC BC
    JR STPOS

STPOS
    DEC HL
    DEC HL,(HL),C

NPOS2
    DEC HL
    DEC HL
    DEC DE
    POP BC
    DJNZ NPOS1
    POP HL
    POP DE
    POP BC
    POP AF
    RET

; Done, Exit
SUBROUTINE STORE

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

STORE

PUSH BC ; * Save registers
PUSH HL ;
LD HL TBUF ; Point to TBUF
LD B 6 ; B = motor count
STEST
LD A (HL) ; Get TBUF (N)
OR A ; Is TBUF element zero
JR NZ STOR1 ; No then do store
INC HL ; Point to next element
DJNZ STEST ; Go do next element check
JR EXIT ; All TBUF zero so exit
STOR1
LD (1x+0), O ; 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 (CURROW) ; Get current row pointer in DE
LD HL TBUF ; Get TBUF pointer in HL
LD BC 0600 ; Count for six motors
LD LP ; Copy TBUF to ARST(1)
CALL (CURROW), LE ; Replace updated row pointer (CALL CTBUF) ; Clear buffers
EXIT
POP HL ; * Restore Registers
POP BC
RET ; Now return to caller
OVRFW
LD HL, CNT ; Print overflow situation
CALL PSTR ; Message
CALL GCHRA ; Get response
CALL PNEWL ; Print a new line
CP 'D' ; User typed a 'D'
JP Z REDO ; Yes then clear all
CP 'S' ; User typed an 'S'
JR Z, EXIT2 ; Yes exit with sequence saved
JR OVRFW ; Bad input, try again
REDO
EXIT2
CALL INIT ; Clear all arrays etc
POP HL ; * Restore Registers
POP BC
POP BC ; Throw away return address
JP CUES1 ; Back to main loop

*4 - 44*
SUBROUTINE RESET

; This subroutine clears the POSAR array

RESET   PUSH BC          ; *
         PUSH DE          ; * Save Registers
         PUSH HL          ; *
         LD HL,POSAR      ; Point to POSAR start
         LD DE,POSAR+1    ; Point to next element
         LD (HL),""       ; Clear first POSAR element
         LL 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 PUTCHR ; 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
PCF 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,9DH ; 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 KBH ; See if character is there
POP DE ; Restore
RET ; Done, Exit

; GCHRRA gets a character from keyboard and displays it

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

*4 - 46*
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

<table>
<thead>
<tr>
<th>DELSW</th>
<th>PUSH</th>
<th>BC</th>
<th>Delay for $1\cdot E + 1\cdot M$ cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELS1</td>
<td>PUSH</td>
<td>BC</td>
<td>Save BC</td>
</tr>
<tr>
<td></td>
<td>NOP</td>
<td></td>
<td>Delay for $11$ T states</td>
</tr>
<tr>
<td></td>
<td>NOP</td>
<td></td>
<td>$4$ T state delay</td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>BC</td>
<td>$4$ T state delay</td>
</tr>
<tr>
<td></td>
<td>DJNZ</td>
<td>DELS1</td>
<td>Do delay times value in B</td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>BC</td>
<td>Restore BC</td>
</tr>
<tr>
<td></td>
<td>RET</td>
<td></td>
<td>Exit</td>
</tr>
<tr>
<td>DELS</td>
<td>PUSH</td>
<td>BC</td>
<td>Save BC</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>B,20</td>
<td>Set B for $0.001$ sec delay</td>
</tr>
<tr>
<td></td>
<td>CALL</td>
<td>DELSW</td>
<td>Do delay</td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>BC</td>
<td>Restore EC</td>
</tr>
<tr>
<td></td>
<td>RET</td>
<td></td>
<td>Exit</td>
</tr>
<tr>
<td>DELT</td>
<td>PUSH</td>
<td>BC</td>
<td>Save BC</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>E,0</td>
<td>Set B for $0.01$ sec delay (?)</td>
</tr>
<tr>
<td></td>
<td>CALL</td>
<td>DELSW</td>
<td>Do delay</td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>BC</td>
<td>Restore BC</td>
</tr>
<tr>
<td></td>
<td>RET</td>
<td></td>
<td>Exit</td>
</tr>
<tr>
<td>DELRN</td>
<td>PUSH</td>
<td>EC</td>
<td>Save BC</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>B,20B</td>
<td>Set B for $1.0$ sec delay (a)</td>
</tr>
<tr>
<td></td>
<td>CALL</td>
<td>DELSW</td>
<td>Do delay</td>
</tr>
<tr>
<td></td>
<td>DJNZ</td>
<td>DELSW</td>
<td>Do next delay section</td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>BC</td>
<td>Restore BC</td>
</tr>
<tr>
<td></td>
<td>RET</td>
<td></td>
<td>Exit</td>
</tr>
</tbody>
</table>

*4 - 48*
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

<table>
<thead>
<tr>
<th>QA</th>
<th>QB</th>
<th>QC</th>
<th>OD</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>φ</td>
<td>1</td>
<td>φ</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>φ</td>
<td>1</td>
<td>φ</td>
<td>2</td>
</tr>
<tr>
<td>φ</td>
<td>1</td>
<td>φ</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>φ</td>
<td>1</td>
<td>φ</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

HALF STEPPING PULSE SEQUENCE

<table>
<thead>
<tr>
<th>QA</th>
<th>QB</th>
<th>QC</th>
<th>OD</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>φ</td>
<td>1</td>
<td>φ</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>φ</td>
<td>1</td>
<td>φ</td>
<td>2.5</td>
</tr>
<tr>
<td>φ</td>
<td>1</td>
<td>φ</td>
<td>1</td>
<td>3.5</td>
</tr>
<tr>
<td>φ</td>
<td>1</td>
<td>φ</td>
<td>1</td>
<td>4.5</td>
</tr>
</tbody>
</table>

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

FULL STEP TABLE

<table>
<thead>
<tr>
<th>FTABL</th>
<th>DEFB</th>
<th>Step number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>192</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>144</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>48</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>4</td>
</tr>
</tbody>
</table>

HALF STEP TABLE

<table>
<thead>
<tr>
<th>FTABLH</th>
<th>DEFB</th>
<th>Step number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>192</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>144</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>48</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>3.5</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4.5</td>
</tr>
</tbody>
</table>

*4 - 49*
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 input 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-DIRECTIONAL)

READ OR WRITE FROM PORT (4)

TRS80 BUS (SEE BUS DESCRIPTION)

A7
A6
A5
A4
A3
A1
A2
A0

IC 1a
IC 1b
IC 1c
IC 2a

ENABLE

IC 3a
IC 3b

IN

OUT

DIRECTION

IC 4

GND

5 VOLTS SUPPLY FROM ROBOT CONNECTOR
+ GND

Coine Robotics

A.J. LENNARD 20/6/1981

D8
D7
D6
D5
D4
D3
D2
D1

Pin 14: 5 Volts, Pin 7: GND

IC 1: 74LS27
IC 2: 74LS20
IC 3: 74LS00
IC 4: 74LS245

Pin 14: 5 Volts, Pin 7: GND

2.55 OCTAL BUS TRANSCEIVER (Tri-state)

3a3 INPUT NOR
2#4 INPUT NAND
4#2 INPUT NAND

0 VOLTS
5 VOLTS

TO ROBOT 8 BIT INTERFACE

*4.52*
CONNECTION OF ARMDROID TO PET/VIC COMPUTERS

PET/VIC USER PORT CONNECTOR

<table>
<thead>
<tr>
<th>PIN NO</th>
<th>PET/VIC NOTATION</th>
<th>ARMDROID NOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>PA0</td>
<td>D1</td>
</tr>
<tr>
<td>D</td>
<td>PA1</td>
<td>D2</td>
</tr>
<tr>
<td>E</td>
<td>PA2</td>
<td>D3</td>
</tr>
<tr>
<td>F</td>
<td>PA3</td>
<td>D4</td>
</tr>
<tr>
<td>H</td>
<td>PA4</td>
<td>D5</td>
</tr>
<tr>
<td>J</td>
<td>PA5</td>
<td>D6</td>
</tr>
<tr>
<td>K</td>
<td>PA6</td>
<td>D7</td>
</tr>
<tr>
<td>L</td>
<td>PA7</td>
<td>D8</td>
</tr>
<tr>
<td>N</td>
<td>GROUND</td>
<td>GROUND</td>
</tr>
</tbody>
</table>

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>6 THEN 70
90 INPUT "FORWARD BACKWARD"; A$
100 IF A$ = "F" THEN D = 0: 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
200 IF D = 0 THEN 230
210 I = I + 1: IF I>T THEN I = 1
220 GOTO 240
230 I = I - 1: IF I<1 THEN I = 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 will 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

\[
7.5^\circ \times \frac{20 \text{ teeth}}{72 \text{ teeth}} \times \frac{12 \text{ teeth}}{108 \text{ teeth}} = 0.2314 \text{ degree step or 4.32152 steps per degree.}
\]

**Shoulder**

\[
7.5^\circ \times \frac{11 \text{ teeth}}{72 \text{ teeth}} \times \frac{12 \text{ teeth}}{108 \text{ teeth}} = 0.162 \text{ degree per step or 6.17284 steps per degree}
\]

**Elbow**

Same as shoulder joint

**Wrist**

Same as base joint calculations

**Hand**

\[
7.5^\circ \times \frac{20 \text{ teeth}}{72 \text{ teeth}} \times \frac{12 \text{ teeth}}{108 \text{ teeth}} = 0.231 \text{ degree per step}
\]

\[
\frac{x \times \frac{231}{360}}{2} = (0.0524/2) \text{mm}
\]

\[
= 0.0262 \text{mm = hand pulley motion per step}
\]

Total hand open to close pulley movement = 20.0 mm

Angle traversed by single finger = 50°

\[
\frac{50^\circ}{20.0} \times 0.0262 \text{mm} = 0.655^\circ \text{ per step or 15.2672 step per degree}
\]

d = 26mm = pulley diameter
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 grommot
   will first have to be fitted to the LED and its holder
   can then be super glued if necessary into the grommot.

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

[Signature]

J. R.
THE ARMDROID 1
ROBOTIC ARM

COLNE ROBOTICS CO. LTD.
BEAUFORT ROAD, off RICHMOND ROAD, TWICKENHAM TW1 2PQ, ENGLAND
Telephone: 01-892 8197/8241  Telex: 8814066
TO: Bob Gilling
FROM: John Hercus
SUBJECT: POWER SUPPLY : ARMDROID
DATE: 18/7/83

Please proceed to construct a power supply as you suggest.

If you need an order number, this could be obtained from Gay and charge it to 310 183.
October 21 1983

The Manager
Colne Robotics Ltd
Beaufort Road
East Twickenham
Middlesex TW1 4LL
ENGLAND.

Dear Sir,

Thank you for the cassette which arrived safely last term. We had by then typed in the text from the manual so that cassette was useful to check this for accuracy.

There were a number of errors in the text as given in the manual. These were mainly missing labels, misspelt labels and most of these I picked up by inspection and trial assembly. Others were picked up by operating the program under monitor control. May I suggest that for future editions of the manual that you simply lump the source code from your editor-assembler, preferably an assembled dump, rather than the manually typed version.

However, there are two major areas where there are problems with the program.

(1) The WRITE/READ modules do not store the value of COUNT. As this is set to zero by INIT, when the stored sequence is read in, COUNT is at zero. If the sequence is then added to, the CUROW pointer (which gets its value from COUNT) points to the first row, and the fresh data is written over the existing data. I modified WRITE and READ accordingly.

(2) This is the most serious of the two. The EDIT module, when in the ROW COUNT mode, does not update CUROW (next row pointer) after the array has been truncated. The consequence is that it still points to the same place. The following code was placed before JP QUESI just prior to EDMOT to cure this.

```
ADD HL, HL ; double the count
PUSH HL ; save it
ADD HL, HL ; Count * 4
POP BC ; Restore Count * 2
ADD HL, BC ; Count * 6
LD B, ARST ; Get buffer pointer
ADD HL, BC ; Calc. new CUROW
LD (CUROW), HL ; and save it
JP QUESI
```

I hope this will be of some value to you.

Yours faithfully,

R. N. GILLING,
Tutor
Machine Tool Engineering Department.
June 10 1983

The Manager
Colne Robotics Ltd
1 Station Road
Twickenham
Middlesex TW1 4LL
ENGLAND.

Dear Sir,

Reference my letter 13 April 1983.

If you refer to paragraph 2 of the above letter you will see that I regarded the non supply of the cassette of software of prime importance.

Please will you send this URGENTLY, by the fastest available means.

It is also important to indicate on the package that this was part of a consignment not sent and that the cost has already been met.

We have typed in the listing in the manual, but this has many errors and we have had problems due to this and need to cross check our code.

Yours faithfully,

R N GILLING,
Tutor
Machine Tool Engineering Department,
1 August 1983

The Manager
Colne Robotics Ltd
Beaufort Road
Twickenham
Middlesex TW1 2PH
England

Dear Sir

Ref: Your letter dated 27 July 1983

Our computer is a Model 1 TRS-80 with 48 K memory on board. Unfortunately this information was not given, so it appears in the original order.

Yours faithfully

R N Gilling
Tutor
Department of Machine Tool Engineering

RNG:CMD
Mr. R.N. Gilling,
Tutor, Machines Tool Eng Dept.
Christchurch Polytechnic,
Madasas Street,
Christchurch 1,
New Zealand.

27th July 1983

Dear Mr. Gilling,

Ref: Your Letter dated 10th June 1983

Please accept our apologies for not despatching the cassette, unfortunately we cannot until we know which computer you have. As soon as I have this information I can forward the cassette, providing it works on the computer you have, which I will confirm with our technicians. If there are any problems I will contact you.

Yours sincerely,
for Colne Robotics Co. Ltd.

Mrs. E. Viner
Sales Administration
25 April 1983

R N Gilling
Tutor, Machines Tool Eng Dept
Christchurch Polytechnic
Madras Street
Christchurch 1
New Zealand

Dear Mr Gilling

We hope that our enclosures will ensure that you are soon able to achieve full operation from your Armédroid and that you will gain the same satisfaction that many other owners now have.

We are also enclosing some literature about other products which we are developing and hope that these may be of interest.

You are the first Armédroid owner in New Zealand and we hope that there will be many more in due course. We wonder whether you could suggest to us any Companies in New Zealand who might be interested in acting as agents and distributors for our products?

Your assistance in this matter would be greatly appreciated.

Yours sincerely

A F I Macmillan
Director and General Manager
25 April 1983

R N Gilling
Tutor, Machines Tool Eng Dept
Christchurch Polytechnic
Madras Street
Christchurch 1
New Zealand

Dear Mr Gilling,

Thank you for your letter of 13 April, I'm afraid that our instruction manual is not as up to date in some respect as we would hope, so I will reply to the questions you ask.

The omission

a) 6mm long x 8mm dia bore spacer
b) 3mm long x 8mm dia bore spacer

You will have received nine 1 mm steel washers which we now use in place of the spacers (six for the 6mm spacer and three for the 3mm spacer).

c) The magnets for the reed switch switcher are now only supplied with the reed switch kit.

The items observed by your technician

a) The belts. If the belts appear to be tight, check you have the pulleys the right way round, the pulley with the alloy extension should operate the wrist gears. The motors can be moved a little on their mountings to enable a small amount of belt adjustment. They should not be too tight as this puts extra load on the motors.

b) This is an omission in the manual

c) A useful point which will add to the new manual.

d) This could have been avoided by stringing the wrist drive with the spring on the inside.
25 April 1983

R N Gilling

e) The metal bar on the hand gear (part 25) acts as a stop against the composite gear spindle (part 21) to prevent the hand from opening too far, when adjusting the hand string tension make sure the stop is hard against the spindle with the hand open.

I hope the above answers help you to get full use out of your Armdroid, and if I can be of any other assistance do not hesitate to get in touch with me.

Yours sincerely

[Signature]

D Boothroyd
for Colne Robotics Co Ltd
25 April 1983

R N Gilling
Tutor, Machines Tool Eng Dept
Christchurch Polytechnic
Madras Street
Christchurch 1
New Zealand

Dear Mr Gilling

We hope that our enclosures will ensure that you are soon able to achieve full operation from your Armdroid and that you will gain the same satisfaction that many other owners now have.

We are also enclosing some literature about other products which we are developing and hope that these may be of interest.

You are the first Armdroid owner in New Zealand and we hope that there will be many more in due course. We wonder whether you could suggest to us any Companies in New Zealand who might be interested in acting as agents and distributors for our products?

Your assistance in this matter would be greatly appreciated.

Yours sincerely

A F I Macmillan
Director and General Manager
April 13 1983

The Manager
Colne Robotics Ltd
1 Station Road
Twickenham
Middlesex TW1 4LL
ENGLAND.

Dear Sir,

The ARMROID robot arm ordered by us on 27 September 1982 arrived on 31 March 1983. One of our technicians has assembled the kit, while I have the responsibility to get the arm working under software control.

The most notable omission was of the cassette of software (containing, I presume, the LEARN program). Would you please send this out by airmail as we need this to check out that the finished arm and associated electronics are working correctly.

Other less obvious omissions were:

(a) 6mm long x 8mm dia. bore spacer.
(b) 3mm long x 8mm dia. bore spacer.
   These go on shaft Pt No 29.
(c) The magnets to work the reed switches. Although not specifically ordered, these appear in the parts list.

The following items were observed by the technician while assembling the arm and the electronics:

(a) Four out of the six belts seemed to be of incorrect length.
(b) The circuit diagram for the interface board did not match the printed circuit board in several areas.
(c) In the instructions it would be useful to indicate that the wires are to be soldered to the motors before fixing the motors in place.
(d) He found it necessary to make spacers to hold the pulleys on the elbow pivot from moving against the sides of the arm.
(e) One of the gears (either Pt 24, 25 or 26) has a small metal bar attached to one face. This protrudes beyond the periphery of the gear, but was not shown as such in any of the pages of the manual. He is not sure that where he has placed this gear is correct. Could you inform us as to the function of this bar and which position the gear should be in?

Apart from these problems, the arm appears to be satisfactory and we look forward to making good use of it.

Yours faithfully,

R N GILLING,
Author, Machine Tool Engineering Department.
CHAIRMAN'S LETTER

Since the launch of Armdroid I in September 1981, Colne Robotics has been the focus of considerable customer interest. We are now ready to introduce to our customers, new products which will further establish Colne’s place as a leader in the field of micro-robotics.

In 1983 the company intends to increase the competitive attraction of Armdroid I, by making available a low-cost computer vision system. This is designed to meet the growing world interest in computer vision, but at very low cost. Coupled to the Armdroid this will familiarize students, managers and development engineers with the software requirements for visual recognition, orientation and robotic interfacing.

Other developments, such as Armdroid II, a small Turtle-type mobile robot, and X-Y plotters — all at low cost — will follow throughout '83 to ensure that the company remains in the forefront of micro-robotic technology. Please read on for further details of these exciting new developments.

Many thanks to all our customers for their support and patience.

John Reekie
Chairman
ARMDROID I achieves worldwide sales in first twelve months

Colne Robotics' low-cost robotic arm, the Armroid I, has achieved outstanding sales success since its introduction in 1981. Among our customers have been a variety of schools, colleges and universities, as well as many leading world companies. The primary intention of buyers has been to use the arm for education and training in robotics as well as for the development of software. However, Armroid I has also been put to such varied uses as radio-active loading, clean-room packing, and the dipping of components into dangerous liquids. In quite a different setting, the arm has been used to help the disabled. Armroid I's success against competitors worldwide is due to its mechanical reliability, the wide range of software now available, and of course to its markedly lower cost. Overwhelmed by orders, Colne Robotics was initially unable to meet the demand for Armroid I. Our move to a new factory, coupled with recent backing by Prutech (a subsidiary of Prudential Corporation Ltd.) has enabled us largely to overcome delivery lags.

A subsidiary company, Colne Robotics Inc. in Florida, is starting production of Armroid I early in 1983, to supply the large U.S. market. This has included major companies such as Bell Telephones and I.B.M., as well as educational establishments - Princeton, M.I.T. and many leading U.S. colleges. We fully anticipate that U.S. sales will reflect as strong an interest as that shown by our customers on this side of the Atlantic.

THE LOW-COST ARMROID II — a 7-axis, applications micro-robot with 4lb lift

Buyers of our small Armroid I micro-robotic arm have developed many different applications for the robot. Its general use in laboratories is outlined above. However, Colne Robotics has frequently received enquiries from customers for a faster and more accurate robot, capable of lifting heavier loads.

To meet this demand we are developing Armroid II, which we believe will surpass the performance of any other small robotic arm in the world. In line with the low cost of Armroid I, the new robot will be available remarkably cheaply, at less than £1,500.

The outline specifications of this new and improved Armroid are as follows:

**MECHANICAL SPECIFICATION**

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load capacity</td>
<td>2 Kg</td>
</tr>
<tr>
<td>Arm length to wrist pivot</td>
<td>600 mm</td>
</tr>
<tr>
<td>Spherical envelope with STD gripper</td>
<td>1340 mm</td>
</tr>
</tbody>
</table>

**AXIS**

<table>
<thead>
<tr>
<th>Axis</th>
<th>Motor Load Capacity (Ncm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>70</td>
</tr>
<tr>
<td>2</td>
<td>70</td>
</tr>
<tr>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>4</td>
<td>70</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
</tr>
</tbody>
</table>

**ANGULAR MOVEMENT**

<table>
<thead>
<tr>
<th>Movement</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shoulder</td>
<td>1270</td>
</tr>
<tr>
<td>Elbow</td>
<td>130</td>
</tr>
<tr>
<td>Wrist yaw</td>
<td>140</td>
</tr>
<tr>
<td>Wrist pitch</td>
<td>180</td>
</tr>
<tr>
<td>Wrist roll</td>
<td>135</td>
</tr>
<tr>
<td>Gripper</td>
<td>200</td>
</tr>
</tbody>
</table>

**ANGULAR SPEED**

<table>
<thead>
<tr>
<th>Speed (°/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>180</td>
</tr>
<tr>
<td>135</td>
</tr>
<tr>
<td>180</td>
</tr>
<tr>
<td>220</td>
</tr>
<tr>
<td>250</td>
</tr>
</tbody>
</table>

**Accuracy of repetition ± 0.5 mm (theoretical)**

*1 Ncm = Torque exerted by 1 Newton Force at 1 cm radius

**ELECTRONIC SPECIFICATION**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>On board microprocessor</td>
<td>(Z80)</td>
</tr>
<tr>
<td>Key pad, Led display</td>
<td></td>
</tr>
<tr>
<td>On board EPROM learning program</td>
<td></td>
</tr>
<tr>
<td>Ability to communicate with other computers</td>
<td>Closed loop</td>
</tr>
</tbody>
</table>

Launch is planned for Summer 1983. Please let us have your name and address, and we will be happy to keep you informed of developments.
COLVIS — Colne develops world’s first low-cost computer vision system

"Intelligence" depends on the ability to acquire information about oneself and one's surroundings. So think of the benefits to be gained from enabling a computer or a robot to perceive such information for itself. Clearly, sensors have an important role to play in robotics engineering and, with this in mind, Colne Robotics has developed a revolutionary new computer vision system, which permits a computer to see objects and remember their shapes.

Previous vision systems have been in the £20,000 — £40,000 price range, but the Colne Robotics system, COLVIS, will be priced at only £395.

It consists of a solid-state camera connected to a powerful micro-computer capable of extracting and learning information from the image produced. This information, such as area, perimeter and centre of gravity of the image, is used to recognise the object in view as well as to deduce its position and orientation. The system can be used in conjunction with any micro-computer which has, or can be fitted with, an 8-bit, parallel bi-directional port.

As with our existing Armroid I micro-robotic arm, the vision system is aimed at the educational market. A versatile teaching-aid, equally at home in the University department or the classroom, it is also appropriate to the teaching carried out in Technical Colleges and by Industrial Training and Development Organisations.

This new product constitutes an invaluable low-cost peripheral to existing robotic arms which we expect to interest all our present customers, and attract many new ones.

Here is the V.D.U. display after COLVIS has learnt 5 objects. It is seeking an object described by the selected parameters in the top R.H. corner and represented by the picture within the square. The first object examined (coded BLNK) was identified as false, as were the 2nd, 4th and 5th objects. The third object, 2RNG, was recognised as true by the similarity of its parameters to those selected.

GOLDMANN PERIMETER AUTOMATED CONTROL — Colne Robotics expands into the medical field

For many years the standard equipment for clinically testing the peripheral vision of the eye, has been the Goldmann perimeter device. In conjunction with the Institute of Ophthalmology, London, Colne Robotics has developed an additional unit which largely automates the testing procedure.

The unit consists of a microprocessor, an E.P.R.O.M. and a stepper motor to drive the mechanism. It substantially speeds up the process of testing a patient, gives pre-determined testing programs and automatically re-tests areas of failed recognition.

The Colne Robotics unit has itself undergone exhaustive tests at Moorfields Eye Hospital, London. Priced at £495, a worldwide launch is scheduled for the unit in March 1983.