COLNE ROBOTICS

The

Colne Robotics

ARMDROID

Construction and Operation Manual

Published by
COLNE ROBOTICS LIMITED
Beaufort Road, Twickenham
Middlesex TW1 2PQ
England
Telephone: 01-892 8197
© Copyright 1981
Introduction

The word Robot comes from the Czechoslovak language and means simply worker. It entered the English vocabulary when a play by Karel Capek was translated in 1923 for the London stage entitled R.U.R. (Rossum's Universal Robots). In the play Rossum created humanoid devices to work for him. Almost all that is left of the play is the word Robot and all that it now means to us.

The image of the robot has been created in part by science fiction writer and part by the manufacturers of industrial robots. As a result most laymen do not have a clear idea of the capability or limitations of a robot or even in some cases what exactly a robot is. The Robot Institute of America puts it like this:

A robot is a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialised devices through variable programmed motions for the performance of a variety of tasks*. In other words it is an intelligent handling device. There are of course specialised pieces of machinery which are able to handle industrial work which are not robotic. What then is the difference? The significance is in the word specialised. A robot is not a specialised piece of machinery and is in fact a part of a flexible automated production line which comes “off the shelf”. The robot can be programmed and is therefore infinitely variable. That is the difference and it is a big one.

The industrial robot has been in existence for a number of decades but the development of microelectronics and microprocessors over the last couple of years is the technology that has made them cost effective. Five years ago a robot had a cost ratio of about 60% electronics and 40% mechanical content. The effect of the silicon chip revolution is that it is now 20% electronics and 80% mechanics. The drop in the total price of robots as a result of the cost ratio has been dramatic. This and the rise in wage rates has caused the increase in the usage of industrial robots.

The development of the small robotic arm was largely accidental. The decision to design this small arm came from what was intended as a survey of industrial robots. It was noticed that the microcomputer hobbyist in the USA was experimenting with medium and small robotic devices of all kinds. There was however, no educational or experimental robotic device on sale anywhere in the world that gave the capability of control and handling at a price which made the robot a good microcomputer peripheral.

*1 - 1*


The design of the robot as a kit is deliberate policy of the Colne Group. It enables the person assembling the device to understand the principles of the robot, although of course the machine is also available in made up form. The main opportunity that the kit form gives to hobbyists and experimental roboticists is the possibility to modify or add to their robot. There are, even at this time of writing a number of experimental projects planned. These will obviously increase in a number of developments and to keep owners of robot arms informed of events we intend to run a newsletter.

Typical of the projects will be the fitting of sensors on the grippers. This will enable the arm to have feedback from the manipulator to give a closed loop control of the device. Another form of feedback could come from sensors such as fibre optic or electro/optic devices on the joints of the machine to sense the exact position of those joints. Variations can be made around the manipulators, the substitution of the gripper for the other kinds of handling device such as a pneumatic sucker for handling sheet or paper or an electro magnetic handler for picking up ferrous metal workpieces, are planned.

Other variations will be seen in the computer programs which will drive the machine. The basis of these programs and the intelligence that they can have are shown in the chapter on software. The variations in mechanical and electronic drives in the robot give the operator the ability to simulate many of the functions of a real industrial robot. The various projects about which we are informed will be the subject of articles in the monthly newsletter which robot arm owners can subscribe to (see details in the appendix of this manual).

We are hoping that this robot arm will serve as a helpful pilot scheme for the evaluation, economical analysis and hopefully installation of its big industrial brothers in manufacturing facilities. The factors that you will have to take into consideration are:-

- Ability of the gripper and machine to handle jobs in the factory.
- Economic factors and evaluation
- Social impact of the robot

Some consideration on the three aspects are given below.
Handling jobs in the Factory

Non Gripping Applications

Cleaning  
Drilling  
Spraying  
Paint  
Power  
Welding  
Arc  
Flame Cutting  
Spotwelding  
Stud

Low Precision Gripping

Investment Moulding  
Diecasting  
Forging  
Foundary Practice  
Press Work Applications  
Plastic Moulding

High Precision Gripping

Interstage Tooling  
Transfer between Stations in Forging  
"  "  " in Metal Stamping and Drawing  
Loading and Unloading Tools in General Work  
Inspection Probing  
Filament Winding  
Wire Wrapping  
Sprue Cutting  
Sorting and Packing  
Precision Drilling  
Precision Routing  
Deburring

Advanced Assembly and Parts Control

Palletizing  
Brick Manufacturing  
Glass Manufacturing

These applications are listed in order of difficulty for robotic handling. It would be unwise for the inexperienced to consider attempting more than those in the first two categories.

*1 - 3*
In surveying the existing use of industrial robots nearly 90% of all applications fall within the first two areas where no gripping or standard gripping surfaces are available. Such applications require low levels of precision, say plus or minus a couple of millimeters. It is in these two areas that the most cost effective applications will at this time be found until robots can be made more intelligent.

The other important parameters are weight of workpieces and speed of handling required. Light handling is generally associated with speed of operation and it is normally difficult for a robot to compete with human response in this kind of handling. The speed of robots is much more suited to handling forgings etc., but this often leads to a variable scenario which require intelligence, and the variation of the machine by operator or computer.

It is possible for the production technician who is inexperienced in robots to identify a robot application in his own manufacturing facility. He must bear in mind that the above mentioned limitations and possibilities but a robot installation starts with the requirements of the production line. The first thing that he must look for is what skills are needed at a particular workstation, by the operative. His work must be assessed for the use of:-

a. Sight
b. Touch
c. Hearing

and how often does the pattern for the use of these skills repeat or be made to repeat without variation. For the purpose of this exercise we will assume that the production engineer is looking for robotic installation opportunities in the non gripping or low precision gripping application areas.

Having looked at a,b and c he will probably discard 'c' as a factor in most operations. Therefore any workstation that needs hearing to do the job will not be a candidate for robotics. The 'a' aspect will now have to be considered from the point of view of how sight is used. It is necessary to decide if sight is used in establishing position or has it got other functions such as quality control etc? If anything other than position is involved then this application must also be discarded.
You will see that we are coming down one factor and that is position. Such a positioned workpiece must have little variability. The workpiece must be offered up to the robot at the same place (+ a small discrepancy), in the same orientation and with little or no variation in the receiving station although now it can be reorientated by other machinery if required and possibly may not need the accuracy of placement. Timing may or may not be important in this last operation but speed to integrate the working of the robot into production line is obviously important.

It will be obvious to you that we are looking at a single robot installation as a hybrid approach using both robots and workpeople on the production line. We are also looking at the possible installation of the simplest robotic device. This of course means safety considerations such as not mixing machines and people in the workspace of the robot. It does however, give a bonus which will allow the robot to work within the safety guards of a machine and could therefore speed up cycle times of the machines working.

You will see that an installation will require modifications both to the robot and to the production machinery. It may be that you need to change parameters of the production line such as workpiece flow and timing, size of batches of items, positioning of the workpiece but remember that the robot, through the computer that controls it can also be very variable to meet a production situation. The main variability of the robot being given by the programming (see the chapter on software).
Economic Factors

The evaluation of the installation of a robot in your factory include cost versus savings and the throughput of your factory. The details of costs and savings are as follows:

Financial Evaluation - Robot Costs

1. Purchase price of robot.

2. Purpose made tooling. This cost may be considered as a part of the cost of the robot for your specific task and may be regarded as part of the cost of the robot.

3. Installation costs may be a constituent of the robot project or may be partially carried as a part of plant changes that were to be made in the normal programme for production machinery depreciation and replacement.

4. Maintenance can be a variable figure depending upon usage. Such costs in foundry work are usually more than in plastic moulding. There is a general rule of thumb with production equipment that maintenance costs are about 10% of the purchase price per annum on a two shift basis.

5. Operating Power is a minor cost which can be assessed in conjunction with the robotic device manufacturer.

6. Finance allows for the current cost of money or alternatively estimates the expected return on investment.

7. Depreciation of ordinary general purpose multi shift equipment is usually about 10 years on a straight line basis.

Robot Savings

a. Labour displaced is the crux of robot usage although protection of workers in hazardous working conditions is a close second. It is obvious that if a robot can be made to work more than one shift the savings will be greater. This cost should also include fringe benefits and be shown as an hourly rate.

b. Quality improvement is a considerable factor particularly if the job is repetitive to the extent of becoming intensely boring so that the task is subject to moods and attitudes of the workers. In addition there can be aspects of environment to be taken into consideration such as physically hazardous or demanding which enables robotic control to produce a better or more thorough job.
c. Increase in throughout either from higher quality or increased output or a mixture of both gives a capability which might be mixed. If the throughput increases the workpieces available in a certain part of the production scheme could mean a modification of the installation (Robot Costs 3). This will have to be costed and set against the savings seen in this item of increased output. The improvement of utilisation of capital assets should not be ignored and may assist the benefit of one for one displacement of a worker.

These lead to a simple payback formula which would give you a timespan, say eighteen months which will be the time that the robot will take to return its investment.

**Payback Formula**

\[
\text{Payback years} = \frac{\text{Robot Cost (1, 2 & 3)}}{\text{Labour (a) - Robot Expense (4, 5, 6, 7)}}
\]

This simple formula does not include 'b' and 'c' in the robot savings.

**Example**

*Single Shift Operation*

\[
P = \frac{\text{£25,000}}{\text{£12 (250 x 8 hrs)} - \text{£1.20 (250 x 8 hrs)}}
\]

The capacity and throughput of your factory is an important factor in the installation of a robotic installation and must be examined.

Here are some of the questions that the robot manufacturer will ask you. You will probably find that it is not worth installing a device unless you have a throughput of two or three million workpieces a year. Production should be fairly steady and the need for the level of the production to last for some years is important to the payback calculation. Other aspects of the handling of the workpiece are more complex but the potential installer must know these answers so do your homework.
Capacity and Throughput

Volume of Throughput

1. Number of workpieces per annum
2. Is the volume of production steady?
3. If there is a variation can you give an indication of the throughput over 12 months.
4. Is the annual production going to last at the present level or above over the next three years.

Workpiece Configuration

1. What is the size of the workpiece?
2. What is the weight of the workpiece?
3. Will it deform under its own weight?
4. Will it break or crack if dropped from a height of 3 inches onto a hard surface?
Social Impact of the Robot

This checklist enables you to evaluate the positive and negative aspects of robot acceptance in your workforce. This is quite different to factors in management acceptance and should be treated as two separate parts of the total equation in robot installation.

Each item has a positive and negative share of the total percentage of the checklist findings. The total can be treated as an indicator and the points can be modified by item and total using management action.

<table>
<thead>
<tr>
<th>Total points</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Will workers be given assurance of keeping their jobs</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2. Can displaced workers be retained in equal rated jobs</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3. Will workers benefit in terms of</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relief from Boredom</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Health</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Safety</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Difficult or dirty tasks</td>
<td>7 total 15</td>
<td></td>
</tr>
<tr>
<td>4. Is present worker management climate in favour of discussing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Economic Conditions</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Labour Unrest</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Is it usually distrustful</td>
<td>7 total 15</td>
<td></td>
</tr>
<tr>
<td>5. Has the company enough economic strength to guarantee promises are kept</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>6. Have management particularly in the engineering department shown ability to establish dialogue with the workers</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>7. Is management concern for the quality of the job or is it only for the economic aspect</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>8. Will there be a plan to upgrade workers who will supervise robotic systems</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>9. Will workers pay rates suffer by robot breakdown etc.</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>
10. Has management respected the skills and intelligence of the workforce in the past 3

11. Will this checklist be shared with the workforce and will the result be discussed 3

12. Will robot training be on company time and will workers be sent to a 'robot training' school. 2

13. Can workers air their views without concern or fear of ridicule 2

Total: 100

Positive points

Negative points

Nett Score

Score Indicator Probability of Acceptance

80 - 100 There is a high acceptance of the concept of robotic and automation systems being installed in your factory.

60 - 80 There is a tolerable possibility of acceptance of robotic and automation systems.

40 - 60 There is little possibility of acceptance, of robotic and automation systems unless action is taken to increase the score and therefore acceptance level.

0 - 40 Forget it.

The examination of the above aspects of robot installation and the understanding of the principles involved using the robot will give you an understanding of the factors involved in an installation of a robot. More information can be obtained from the Department of Industry, the Production Engineering Research Association and the British Robot Association all of whose addresses are shown elsewhere in this manual.
MECHANICS

2.1 Description

The ARMDROID consists of five main parts.

The base

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

The Shoulder

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

The Upper Arm

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

The Forearm

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

The Wrist and Hand

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

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

1. FITTING BELTS ONTO PULLEYS

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

2. FITTING SWITCHES

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

3. FITTING PULLEYS TO MOTORS

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

4. LUBRICATION

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

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

Micro-switches are included in the assembled and unassembled Armdroid packages as optional extras. It must be stressed, however, that the machine will function perfectly well without the micro-switches, but a check must be kept on the number of complete revolutions of the base. Any more than 1½ turns will put a strain on the stepping motor leads where they connect to the printed circuit boards.

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

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

PART NUMBERS INVOLVED: *09*10*15*16*18/16*18/12*
2.3 TOOLS LIST INC. Lubricants etc

General and small circlip pliers
7mm spanner - supplied
5.5mm spanner - supplied

Metric steel rule, (part identification)

Hypodermic syringe or small oilcan and 3 in 1 oil
"Superglue" and if possible "Loctite"

Cold vaseline or cycle bearing grease

Tweezers

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

Light weight hammer (fitting rollpins)
## 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>06m</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**

<table>
<thead>
<tr>
<th>Item</th>
<th>Part No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shoulder Side Plates</td>
<td>017</td>
</tr>
<tr>
<td>Switch support bar 107mm x M3 at ends</td>
<td>019</td>
</tr>
<tr>
<td>Support bar spacers M3 clearance X</td>
<td>018/16</td>
</tr>
<tr>
<td>Motor support bracket stiffener 107mm x M3 at ends</td>
<td>018/12</td>
</tr>
<tr>
<td>Support bar spacers</td>
<td>019</td>
</tr>
<tr>
<td>Reduction gears</td>
<td>020</td>
</tr>
<tr>
<td>Reduction gear spindle 96mm x 6mm</td>
<td>021</td>
</tr>
<tr>
<td>Drive belts</td>
<td>08/1</td>
</tr>
<tr>
<td>medium = 114 teeth</td>
<td>08/m</td>
</tr>
<tr>
<td>short = 87 teeth</td>
<td>08/s</td>
</tr>
<tr>
<td>Uprer Arm Drive Gear</td>
<td>021</td>
</tr>
<tr>
<td>small internal dim no drum</td>
<td>022</td>
</tr>
<tr>
<td>Upper arm side plates</td>
<td>023</td>
</tr>
<tr>
<td>Upper arm brace</td>
<td>024</td>
</tr>
<tr>
<td>Gears wrist action</td>
<td>025</td>
</tr>
<tr>
<td>hand action</td>
<td>026</td>
</tr>
<tr>
<td>fore arm</td>
<td>027</td>
</tr>
<tr>
<td>Idler pulley</td>
<td>029</td>
</tr>
<tr>
<td>Shoulder pivot 96mm x 8mm spindle</td>
<td>030</td>
</tr>
<tr>
<td>Fore arm side plates</td>
<td>031</td>
</tr>
<tr>
<td>Fore arm brace</td>
<td>032</td>
</tr>
<tr>
<td>Fore arm pulley</td>
<td></td>
</tr>
</tbody>
</table>

*2 - 5*
<table>
<thead>
<tr>
<th>DESCRIPTION OF ITEM</th>
<th>Part No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elbow Idler pulleys</td>
<td>033</td>
</tr>
<tr>
<td>wrist</td>
<td></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
<table>
<thead>
<tr>
<th>DESCRIPTION</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 - 8*
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 (C46). 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 20'clock as seen from inside the pan with the flange at the top. (M4 countersunk x 16mm bolts, nuts)

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

*2 - 9*
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) than the hand motor 03h(M4) and the hand motor pulley. Then fit the hand reed switch bracket (M3) and the switch (M3 x 10 cheesehead bolts).

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

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

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

Assemble reduction gear support bar (21), assemble with the correct length drive belts (08s/m/1) 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 acrute 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.

*2 – 10*
Fit pulley (32) to the outside of the forearm side plate (30) (M3x5mm) (countersunk screws). Fit a brace to one forearm side plate, then fit the other side plate to the brace. Check for squareness before finally tightening bolts.

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

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

Fit the wrist gears on the bushes (37) from the outside. Fit bevel gear carrier (35) between the wrist bevel gears (37), line up holes in end of wrist pivot (38) bore with tapped hole in carrier by peering down pivots. If you do not have a screw gripping or magnetic driver use a little piece of masking tape or sellotape to fix M3 cheesehead screw to the end of your screwdriver in such a way that it will pull off after tightening - check gears pivot freely on pivots and that the whole 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 bevel gear carrier (35). Tighten on a drop of loctite if available, gently by turning a pair of pliers inside it. The bevel gears should be positioned with their grub screws pointing towards the hand when the hand and the forearm are in line (see drawing).

Assemble the fingertip (42) and cable clamp (43) with the small spring (44) on the pivot (45), and clip together with large circlips on the cable clamp. The spring should be positioned so that the "back" of the spring is on the 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/L) (16mm x 3mm) and two small circlips (see drawing). Fix one circlip to the pin before one begins to assemble.

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

Cut off one end of the tip spring about 8mm-10mm beyond its hole. Level with its hole bend the spring through a right-angle to secure it. Repeat at the other end. Trim the inner end of the middle finger spring flush with the end of the finger end and treat the outer end as above.
Fit the small pulley (51) to the finger middle section using a short pin (13mm x 3mm) and two small circlips. Fit the larger pulley (52) to the finger base with a long pin (16mm x 3mm) and two small circlips.

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

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

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

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

Glue strips of rubber to finger tips using superglue.

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

Hints: Useful tools are:

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

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

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

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

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

e) When tensioning the cable, if it is passed through the clamp and back, then connected to the spring adequate tension can be applied by pulling the 'free tail' and then nipping it with the grub screw. A 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.
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.

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

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  
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 cheesehead and two nuts as a cable pillar
Third  - Single finger cable (fix to the hand sheave pulley using M3 x 6mm cheesehead and crimped cable clamp
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)
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.

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.

*2 - 14*
## INTERFACE DRIVER BOARD

<table>
<thead>
<tr>
<th>ITEM</th>
<th>VALUE</th>
<th>QUANTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Resistors</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>1K8</td>
<td>1</td>
</tr>
<tr>
<td>R2</td>
<td>10K</td>
<td></td>
</tr>
<tr>
<td>R3-8</td>
<td>2K2 resitor network</td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td>1K8</td>
<td>1</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><strong>Capacitors</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>100p polystyrene</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>1.0µf Tant</td>
<td>1</td>
</tr>
<tr>
<td>C3-C15</td>
<td>10nf ceramic</td>
<td>13</td>
</tr>
<tr>
<td><strong>Semiconductors</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IC1</td>
<td>74LS 125</td>
<td></td>
</tr>
<tr>
<td>IC2</td>
<td>74LS 125</td>
<td>2</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>2D1</td>
<td>BZX 13v ZENER</td>
<td>1</td>
</tr>
</tbody>
</table>

**Miscellaneous**
- MXJ 10 way edge connector
- 5 way PCB plug and socket connector
- Through PIns
- 16 pin IC sockets
- 14 pin IC sockets
- 4 way modified PCB plug and socket
GENERAL ASSEMBLY SEQUENCE FOR THE PC BOARD

A  Fit all of the through pins to the board.
B  Fit and screw the 5v regulator to the board.
C  Identify and fit the resistors and the 13v zener to the board. The black band v points to the motor connectors (on the zener D10DE).
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.

Blue = Pin 1 on I/P connector=5v 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 1-5 and q,w,e,r,t,y to the motors in the following way:

1. q. = GRIPPER. 2. w. = left wrist. 3. e. = right wrist.
4. r. = forearm. 5. t. = shoulder. 6. y. = base.

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

Ribbon Cable To Stepper Motor Connections
Qa Black or Green
Qb Red or Purple
Qc Brown or Blue
Qd Orange or Grey
+15V Yellow or white

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

15V
R14  18  5W
ZD1  BZX 13V

IC17 7805
5V
To rest of board

X pin 9 ICs 13,14,15,16.

* 3 - 5 *
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 resister (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.

*3 - 2*
CONNECTION OF THE ARMDROID TO THE TRS80 PRINTER PORT

The TRS80 printer port can be used to drive the robot arm, but when using the printer port it will not be possible to read the reed-switches connected to the arm as this port is not a bi-directional port. The TRS80 to ARMDROID connections are shown below.

<table>
<thead>
<tr>
<th>TRS80 PRINTER PORT PIN CONNECTIONS</th>
<th>ARMDROID CONNECTION ON INTERFACE BOARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>V5</td>
</tr>
<tr>
<td>17</td>
<td>D8 Macaro</td>
</tr>
<tr>
<td>15</td>
<td>D7 Pino</td>
</tr>
<tr>
<td>13</td>
<td>D6 Verde</td>
</tr>
<tr>
<td>11</td>
<td>D5 Menta</td>
</tr>
<tr>
<td>9</td>
<td>D4 Verde</td>
</tr>
<tr>
<td>7</td>
<td>D3 Verde</td>
</tr>
<tr>
<td>5</td>
<td>D2 Verde</td>
</tr>
<tr>
<td>3</td>
<td>D1 Verde</td>
</tr>
</tbody>
</table>

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

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

```
OR  $01H  ; Set bit D1
LD  PORTAD,A; Send data to port
AND  $0FH  ; Clear bit D1
LD  PORTAD,A; Now latch data pulse to
            ; selected motor
```

In the case of the TRS80 level 11 the printer port address is:

PORTAD equals 37E8H
SECTION 1

A SYSTEM EQUATES

B SYSTEM VARIABLES

C SYSTEM CONSTANTS
ARM TRAINER MK2AL

DIRECT FULL STEP MOTOR CONTROL

FOR TRS80 MODEL I, LEVEL II

BY ANDREW LENNARD

*** July 1981 ***
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 1 Level 11, and the loading instructions given here apply to that computer. But the program can be easily adapted to any 280 microprocessor with the necessary port, and versions made available for the leading makes with variations of these instructions where appropriate. But of course users can write their own software in whatever language they choose.

4.2 Loading

When in Basic type SYSTEM, press ENTER, answer the '*' with LEARN and then press ENTER again. The cassette tape will take about 1½ 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(LEARN)

Stores a sequence of manual movements in memory. The arm is moved using the commands explained under N(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(LEARN) 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 0 (zero).

DISPLAY

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 L-6/Q-Y keys (see MOVE). The first row (RELPOS) shows the current position. Each row represents a stage of the movement, and the actual figures are the number of steps each motor is to make, positive for forward, negative for reverse. The maximum number of steps stored in a row for one motor is +127 or −128, so if a movement consists of more than this number it is 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 0 (zero). To continue scrolling, press any other key. To display the figures one after the other, keep pressing 0.

E(DIT)

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

Truncate a sequence by pressing R(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.

S(ET ARM)

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

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

S(ET ARM) has another function. During a L(EARN), pressing S(ET ARM) at any moment when the arm is at rest will ensure that the movements before and after are separated from each other instead of being merged. This is the way to make quite sure that the arm passes through a particular point during a sequence. Try the same two movements without pressing S(ET ARM), and note the difference in the display.
It is important to realise that, if a sequence has been memorised and S(ET ARM) is pressed from the Menu when the arm is not in its original starting position, pressing G(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 the right or left), but it can lead to the arm running into objects if the new starting point is not selected with care.

W(RITE)

Writes a memorised sequence to cassette tape.

R(READ)

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

M(ANUAL)

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

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

The geometry of the arm is designed to give the maximum flexibility combined with maximum practicality. A movement of one joint affects only that joint: with some designs one movement involuntarily produces movement in other joints.
It is a feature of the ARMROID 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: Grippper
2/W: Wrist left
3/E: Wrist right
4/R: Forearm
5/T: Shoulder
6/Y: Base

B(00T)
Returns the computer to the program start and clears the memories.

Q(UI)T
Returns the computer to TRS80 System level.
4.3 INTRODUCTORY DEMONSTRATION SEQUENCE

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

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

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

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


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 1-6/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.

*4 - 5*
SYSTEM EQUATES

PORT EQU ø 4 ; ARM PORT NUMBER
FINAD EQU ø2B2 ; SYSTEM RESTART
FCHR EQU ø33H ; SYSTEM PRINT CHARACTER
GCHR EQU ø49H ; SYSTEM GET CHARACTER
KBD EQU ø2BH ; SCAN KEYBOARD
PUTSTR EQU 28A7H ; SYSTEM PRINT STRING
CASN EQU ø212H ; CASSETTE ON
CASOF EQU ø1F8H ; CASSETTE OFF
RDHDR EQU ø296H ; READ HEADER ON CASSETTE
READC EQU ø233H ; READ CHARACTER FROM CASSETTE
WRHDR EQU ø287H ; WRITE HEADER TO CASSETTE
WRCYC EQU ø264H ; WRITE CHARACTER TO CASSETTE
MINUS EQU '‐' ; ASCII MINUS
SPAC EQU '‐' ; ASCII SPACE
NL EQU øDH ; ASCII NEW LINE
NUMBA EQU 32H ; ASCII NUMBER BASE
MAXLE EQU 10 ; UPPER BOARD FOR ARST ROW COUNTER

; ORG 17408 ; = 4400 TRS80 HEX ADDRESS
; FOR START OF PROGRAM
VARIABLES USED

IN  DEFB 0$; Has value of one if number input negative
MAN  DEFB 0$; If MAN = zero then steps are stored
STRFG  DEFB 0$; If STRFG non zero then store TBUF array
KEYP  DEFB 0$; Set if key pressed in KEYIN Routine
PORFG  DEFB 0$; Set if sequence to be done forever

COUNT  DEFB 0000$; Number of motor slices stored
CURON  DEFB 0000$; Pointer to next free motor slice

ARRAYS

NUMAR  DEFS 1$; Store used for Binary to ASCII Conversion
        ; Routine CTRAS

POSAR  DEFS 12$; Each two bytes of this six element array
            ; contain one 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 arithmetic.

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.

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

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

ARST  DEFS N*6$; This array holds the sequence that
                ; the user teaches the system. The array
                ; consists of N*6 bytes where N is
                ; the number of rows needed to store the
                ; sequence.
FTABL is a small table which defines the order of the steps as they are sent out to the arm. To drive each motor the DRAM routine adds the motors offset which is obtained from CTPOS and adds this to the FTABL start address -1. This will now enable the DRAM routine to fetch the desired element from the FTABL array, and this value is then sent to the motor via the output port.
CONSTRANTS AND ARRAYS

SIGON
DEFM ***
DEFW
DEFB
DEFM 'REALLY QUIT? (Y/N)'
DEFW 09
DEFM 'YOU ARE NOW AT TRS@ SYSTEM LEVEL'
DEFW 09
DEFW 090DH
DEFM 'EDIT (M)OTOR STEP, OR (R) ON COUNT?'
DEFW 090DH
DEFW 09
DEFM 'NEW UPPER ROW BOUND IS?'
DEFW 09
DEFB 09
DEFM 'ROW NUMBER?'
DEFB 09
DEFM '**** BAD INPUT VALUE ****'
DEFW 090DH
DEFM 'CHANGE STEPS ON WHICH MOTOR?'
DEFM 09
DEFM 'REPLACEMENT STEP VALUE?'
DEFM 09
DEFM 'LEN, READ, CHECK, WRITE, GO, DISP, BOOT, MAN,
DEFM QUIT, SETA, TOAT, LFT, FREE
DEFW 090DH
DEFM 'DO (F)OREVER OR (O)nce?'
DEFM 09
DEFM 'TYPE SPACE BAR WHEN READY, OF FULL STOP TO EXIT'
DEFM 09
DEFM 'PARDON'
DEFW 090DH
DEFM 'WANT TO RE-START (Y/N)?'
DEFW 09
DEFM 'START AGAIN OR (C)ONTINUE FROM CURRENT POSITION
DEFM (.) TO EXIT
DEFW 090DH
DEFB 09
DEFM ' *** MOVEMENT ARRAY DISPLAY *** '
DEFB 09
DEFW 090DH
DEFM '**** NO SEQUENCE IN STORE ****'
DEFM 09
DEFW 090DH
DEFW 090DH
DEFM 'NO MORE ARM STORE LEFT, DELETE OR SAVE?'
DEFM 090DH
DEFW 090DH
DEFW 090DH
DEFM 'SEQUENCE COMPLETE'
DEFW 090DH
DEFM '**** READ ERROR ****'
DEFM 090DH
DEFB 090DH
DEFB 090DH
DEFB 090DH
DEFM 'ARM RESET'
DEFM 090DH
DEFM 'ARM NOW FREE TO MOVE.'
TORMS
DEFB  $00DH
DEFB  $0DH
DEFM  '*** TORQUE APPLIED ***'
DEFW  $00DH
POSST
DEFM  'FELPOS='
DEFB  $0
SECTION 2

COMMAND ROUTINES
COMMAND INDEX

STARM............. Program entry point
LEARN............. Learn a sequence command
EDIT................ Edit a sequence command
READ................ Read in sequence from tape command
WRITE................ Write sequence to tape command
CHECK............... Check stored sequence command
BOOT............. Re-start system command
FINISH.................. Exit from system command
SETARM............. Set start position command
TOSTM................ Move arm to start position command
FREEARM............. Free all arm joints
MANU................ Go into manual mode
GO ................ Execute stored sequence command
DISPLAY........... Display stored Sequence command
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 menu string
CALL PSTR ; Print it
CALL GCHRA ; Get response and print it
CALL PNEWL ; Print new line
CP NL ; Is response a newline
JR Z,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,SETAM ; Yes then do arm set
CP 'T' ; a 'T'
JP Z,TOSTM ; Yes then move arm to start
CP 'G' ; a 'G'
JP Z,GO ; Do execute movements stored
CP 'D' ; a 'D'
JP Z,DISP ; Yes then display ARST array
CP 'B' ; a 'B'
JP Z,BOOT ; Yes then restart system
CP 'M' ; an 'M'
JP Z,MANU ; Yes the Manual control of arm
CP 'F' ; a 'F'
JP Z,FREARM ; Yes then clear all motors
CP 'Q' ; a 'Q'
JP Z,FINSH ; Yes then quit program
LD HL,QMESS ; Point to 'FARDON' message
CALL PSTR ; Print it
JP QUESS1 ; 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 GCHRA ; Get response and print it
CALL PNEWL ; Print a new line
CP '.' ; Response a '.'
JP Z,QUEST1 ; 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 GCHRA ; Get response and print it
CALL PNEWL ; Print new line character
CP '.' ; Response a '.'
JP QUEST1 ; Exit to main loop if so
CP SPAC ; Is it a space?
JR NZ,WAIT2 ; If not then bad input, try again
CALL TORQUE ; Switch motors on
JR STLRN ; Do rest of learn
NOINT
LD HL,(COUNT) ; Get current count
LD A,L ; Is it zero?
OR H ;
JR Z,NOSTR ; Yes then can't add to nothing
STLRN
XOR A ; Clear manual flag
LD (MAN)A ; Because we are in learn mode
CONLN
CALL KEYIN ; Drive motors and store sequence
OR A ; Zero key pressed
JR NZ,CONLN ; No then continue
CALL MOVTO ; Move arm to start position
JP QUEST1 ; Back to main loop

*4 - 13*
<table>
<thead>
<tr>
<th>EDIT</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>HL, (COUNT)</td>
<td>Get row count</td>
</tr>
<tr>
<td>LD</td>
<td>A, L</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>H</td>
<td>Test for zero</td>
</tr>
<tr>
<td>JP</td>
<td>Z, NOSTR</td>
<td>Yes then nothing in store</td>
</tr>
<tr>
<td>ЕD3RT</td>
<td>CALL PSTR</td>
<td>Print edit message</td>
</tr>
<tr>
<td>CALL</td>
<td>GCHRRA</td>
<td>Get response</td>
</tr>
<tr>
<td>CALL</td>
<td>PNEWL</td>
<td>Print a new line</td>
</tr>
<tr>
<td>CP</td>
<td>'M'</td>
<td>Is response an 'M'</td>
</tr>
<tr>
<td>JR</td>
<td>Z, EDMOT</td>
<td>Yes then edit motor</td>
</tr>
<tr>
<td>CP</td>
<td>'R'</td>
<td>Is response an 'R'</td>
</tr>
<tr>
<td>JR</td>
<td>NZ, EDSRT</td>
<td>No then try again</td>
</tr>
<tr>
<td>LD</td>
<td>HL, COUTS</td>
<td>HL = New row count message</td>
</tr>
<tr>
<td>CALL</td>
<td>PSTR</td>
<td>Print it</td>
</tr>
<tr>
<td>CALL</td>
<td>GINT</td>
<td>Get 16 bit signed integer</td>
</tr>
<tr>
<td>JP</td>
<td>NZ, BADC</td>
<td>Non zero return means bad input</td>
</tr>
<tr>
<td>LD</td>
<td>A, H</td>
<td>Test top bit of HC</td>
</tr>
<tr>
<td>BIT</td>
<td>7, A</td>
<td></td>
</tr>
<tr>
<td>JP</td>
<td>NZ, BADC</td>
<td>If negative then bad input</td>
</tr>
<tr>
<td>LD</td>
<td>BC, (COUNT)</td>
<td>Get count value</td>
</tr>
<tr>
<td>PUSH</td>
<td>HL</td>
<td>Save response</td>
</tr>
<tr>
<td>OR</td>
<td>A</td>
<td>Clear carry flag</td>
</tr>
<tr>
<td>SBC</td>
<td>HL, BC</td>
<td>See if response &lt; current count</td>
</tr>
<tr>
<td>POP</td>
<td>HL</td>
<td>Restore response</td>
</tr>
<tr>
<td>JR</td>
<td>NC, BADC</td>
<td>Replace count with response</td>
</tr>
<tr>
<td>LD</td>
<td>(COUNT), HL</td>
<td>Back to main loop</td>
</tr>
<tr>
<td>JP</td>
<td>QUES1</td>
<td></td>
</tr>
<tr>
<td>EDMOT</td>
<td>CALL PSTR</td>
<td>Print 'row number'</td>
</tr>
<tr>
<td>CALL</td>
<td>GINT</td>
<td>Get integer response</td>
</tr>
<tr>
<td>JR</td>
<td>NZ, BADC</td>
<td>Bad answer</td>
</tr>
<tr>
<td>LD</td>
<td>A, H</td>
<td>No negative row count</td>
</tr>
<tr>
<td>BIT</td>
<td>7, A</td>
<td>allowed</td>
</tr>
<tr>
<td>JR</td>
<td>NZ, BADC</td>
<td>or zero row count</td>
</tr>
<tr>
<td>LD</td>
<td>BC, (COUNT)</td>
<td>Get row count into BC</td>
</tr>
<tr>
<td>INC</td>
<td>BC</td>
<td>Move count up one</td>
</tr>
<tr>
<td>PUSH</td>
<td>HL</td>
<td>Clear carry flag</td>
</tr>
<tr>
<td>SBC</td>
<td>HL, BC</td>
<td>Subtract count from response</td>
</tr>
<tr>
<td>POP</td>
<td>HL</td>
<td>Restore response</td>
</tr>
<tr>
<td>JR</td>
<td>NC, BADC</td>
<td>If greater than allowed error</td>
</tr>
<tr>
<td>DEC</td>
<td>HL</td>
<td>Move response down one</td>
</tr>
<tr>
<td>ADD</td>
<td>HL, HL</td>
<td>Double HL</td>
</tr>
<tr>
<td>PUSH</td>
<td>HL</td>
<td>Save it</td>
</tr>
<tr>
<td>ADD</td>
<td>HL, HL</td>
<td>Row count x 4</td>
</tr>
<tr>
<td>POP</td>
<td>BC</td>
<td>BC = row count x 2</td>
</tr>
</tbody>
</table>

*4 = 14*
ADD HL, BC
ADD HL, ARST
ADD HL, BC
PUSH HL
CALL HL, MOTNS
CALL PSTR
CALL GINT
JR NZ, BADNM
LD A, H
OR A
JR NZ, BADNM
LD A, L
CP 1
JR C, BADUM
CP 7
JR NC, BADNM
POP HL
DEC A
LD C, A
LD B, 0
ADD HL, BC
PUSH HL
LD HL, NVALS
CALL PSTR
CALL GINT
JR NZ, BADNM
LD A, H
CP $FFH
JR NZ, PEDIT
BIT 7, L
JR NZ, BADNM
JR NOTAS
PEDIT OR A
JR NZ, BADNM
BIT 7, L
JR NZ, BADNM
MOTAS LD A, L
POP HL
LD (HL), A
JP QUES1
BADNM POP HL
BADC LD HL, BADMS
CALL PSTR
JP QUES1

HL = Row count x 6
Get store start address
Add row offset
Save resulting pointer
Print
Motor number string
Get Answer
Bad answer
Response too large
No motor number < 1
No motor number > 6
Restore = Memory pointer
Motor offset 0 → 5
Add to memory pointer
Now we point to motor in store
Save pointer
Print new step value
Get response
Bad answer
We have a positive response
New negative step value too large
Step value OK
New positive step value too large
so exit
else ok
Get step value
Restore memory pointer
Place step value in store
Go do next operation
Print error message and
return to main loop
READ ROUTINE

; Reads stored sequence from cassette
; into memory

READ    LD     HL,CASRD ; Point to wait message
CALL    PSTR ; Print it
CALL    GCHRA ; Get response
CALL    PNWL ; Print new line
CP      ' ' ; Is response a dot?
JP      Z,QUEST ; 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
RROWNR  PUSH BC ; Same count
LD      E,A ; 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,RDBRR ; No then error
DEC     BC ; Decrement row count
LD      A,B ; See if row count
OR      C ; is zero
JR      NZ,RROWNR ; No then read next row
CALL    CASCF ; Switch cassette off
JP      TAPEF ; exit
RDBRR   LD      HL,RDMSG ; Error message for tape
CALL    PSTR ; Print it
JP      QUEST ; Go to main loop
WRITE ROUTINE

; Writes a stored sequence to tape

WRITE
LD BC, (COUNT) ; Get row count
LD A, B
OR C
JP Z, NOSTR ; If zero exit
LD HL, CASRD
CALL PSTR ; print message
CALL GCHRA ; Get answer
CALL PNEWL ; Print new line
CP '.' ; Is answer a dot
JP Z, QUESI1 ; 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
CALL DELT ; delay
PUSH BC ; Save row count
LD E, @$ ; Clear check sum
LD B, 6 ; Six motor slots per row
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
JR NZ, ROWNW ; No then try again
CALL CASOF ; Switch cassette off
JP QUESI1 ; 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
BADCI
LD HL, CASRD ; Print wait message
CALL PSTR
CALL GCHRA ; Get answer
CALL FNEWL ; Print new line
CP ',' ; Is response a ','
JP Z, QUESL1 ; 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 D ; Same?
JR NZ, RDERR ; No then error
CALL READC ; Read lower byte of count
CP C ; Same?
JR NZ, RDERR ; No then error
OR B ; Zero count from tape
JP Z, NOSTR ; So exit
LD HL, ARST ; Point to start of memory
ROWNC
PUSH BC ; Save count
LD E, 0 ; Check sum is zero
LD B, 6 ; Count is 6
CKBYT
CALL READC ; Read a motor step element
CP (HL) ; Same as in store?
JP NZ, RDERR ; Not the same so error
ADD A, E ; Add to check sum
LD E, A
INC HL ; Advance memory pointer
DJNZ CKBYT ; Do next row element
POP BC ; Restore row count
CALL READC ; Read check sum
CP E ; Same as check sum calculated
JP NZ, RDERR ; No then error
DEC BC ; Decrement count
LD A, B ; Is count zero?
OR C
JP NZ, ROWNC ; No then do next row
CALL CASO ; Switch cassette off
LD HL, TAP0X ; Print tape off message
CALL PSTR
JP QUESL ; and back to main loop
BOOT AND FINISH COMMANDS

; This routine restarts the program

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

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

FINISH
LD HL,RELYQ ; Print "REALY 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,FINISH ; No then try again 
CALL PNEWL ; Print a new line 
JP QUESL ; Back to main loop

*4 -19*
OTHER SHORT COMMANDS

; SETAM clears arm position array

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

; TOSTM moves the arm back to its start position

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

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

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

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

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

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

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

GO CALL PNEWL ; Print a new line
CALL MOVTO ; Move arm to start
XOR A ; Clear
LD (FORFG),A ; Forever Flag FORFG
LD HL,AORMN ; Print "DO ONCE OR FOREVER
CALL PSTR ; Message
CALL CCHRA ; 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 (FORFC),A ; to 1
ONECY LD A,',' ; Print a ','
CALL PUTCCH ; Using PUTCCH
CALL DCALL ; Execute the sequence
LD A,(FORFC) ; Test FORFC, if zero
OR A ; then we do not want
JF 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,DONNS ; Print sequence done
CALL PSTR ;
JF 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
CALL HL,DISPS ; Point to header string
CALL PSTR ; and display it
CALL POSDS ; Print out the relative position
LD HL,ARST ; Point to sequence start
LD BC,(COUNT) ; BC = how many rows to print
LD A,$ ;
OR c ;
JP NZ,SETBC ; Test if count is zero
LD HL,NODIS ; No then jump to rest of
CALL PSTR ; display else print message
JP QUES1 ; telling user nc display and
SETBC ; return to the main loop
LD EC,$0000 ; Clear BC for row count
DGROW ; Save it
PUSH HL ; Save memory position
PUSH H,B
LD L,C
TNC HL
LD IX,NUMAR ; HL = row count
CALL CBTRAS ; New row count =N+1
LE HL,NUMAR ; IX points to buffer for ASCII String
CALL PSTR ; Convert HL to ASCII
CALL PSTR ; Point to ASCII string
LD A,'.' ; row print it
CALL PUTC HR ; Print a '.
POP HL ; Restore memory pointer
LD B,5 ; Motor count to B (6 motors)
LD A,(HL) ; Get step value
PUSH HL ; Save memory pointer
PUSH BC ; Save motor count
BIT 7,A ; Test bit 7 of A for sign
JR Z,NUMPO ; If bit = 0 then positive step
LD H,$FFH ; Make H = negative number
JR EVAL ; Do rest
NUMPO ; Clear H for positive number
EVAL
LD E,$0 ; Get low order byte into L
LD L,A ; Point to result string
CALL CBTRAS ; Call conversion routine
LD HL,NUMAR ; HL points to result
CALL PSTR ; Print resulting conversion
LD A,(3819H) ; Get keyboard memory location
BIT 0,A ; Test for zero key pressed
JR 17,STIFS ; Not pressed, then skip
DSTIFS ; Wait till next character entered
CALL GCFIR ; Is it a dot?
CP '.' ;
JR NZ,NOSTP ; No then, carry on
CALL PNEWL ; else print a new line
POP BC ; and restore all the registers
POP HL
POP BC ; Jump back to main loop
JP QUEST ; Jump to main loop
POP BC ; Restore column count
POP HL ; Restore memory pointer
INC NL ; Increment memory pointer
CALL PSPAC ; Print a space between numbers
DJNZ OTEST ; 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 QUEST ; Back to main loop
SUBROUTINES INDEX

DOALL....................Execute a stored sequence once
DRVL......................Drives all motors directed by TBUF
INIT......................Set up system
MOVT0.....................Use FGSR to reset system arm
TORQUE....................Turn on off motors
CLRMT.....................Turn off all motors
SEITR.....................Reset CTPOS elements to one
DRAMT.....................Drive directed motors
STEPM.....................Step motors via DRAMT
DNEKD.....................Delay on direction change
SPANT.....................Update TBUF array during learn
KEYIN.....................Scan keyboard and build up motors to move
CBTAS.....................Convert 16 bit 2's complement number to ASCII
CLRMPF....................Clear MOTBF array
CTBUF.....................Clear TBUF, DRBUF & MOTBF arrays
GINT.....................Get 16 bit signed value from keyboard
POSDS.....................Display relative position array elements
POSIC.....................Increment relative position array elements
STORE.....................Copy TBUF to current ARST slice
RESET.....................Clear POSAH array
PUTCHR....................Print a character
PSTR.....................Print a string
PSPAC.....................Print a space
PNEWL.....................Print a carriage return
SECTION 3

SUBROUTINES.
SUBROUTINES INDEX (continued)

SCKBD.........................Scan the keyboard
GCHKa..........................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.........................Delay approx 1.0 sec
SUBROUTINE DOALL

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

<table>
<thead>
<tr>
<th>label</th>
<th>mnemonic1</th>
<th>op</th>
<th>mnemonic2</th>
<th>op</th>
<th>label</th>
<th>mnemonic1</th>
<th>op</th>
<th>mnemonic2</th>
<th>op</th>
<th>label</th>
<th>mnemonic1</th>
<th>op</th>
<th>mnemonic2</th>
<th>op</th>
<th>label</th>
<th>mnemonic1</th>
<th>op</th>
<th>mnemonic2</th>
<th>op</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOALL</td>
<td>LD</td>
<td>BC</td>
<td>(COUNT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>A,B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OR</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JR</td>
<td>Z,RET2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>HL,ARST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMOTS</td>
<td>LD</td>
<td>DE, TBUF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PUSH</td>
<td>BC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>EC, F0F0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDPIR</td>
<td>HL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PUSH</td>
<td>HL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL</td>
<td>DRIVL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL</td>
<td>SCRBD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>HL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>BC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL</td>
<td>DNEWD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CP</td>
<td>'.'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JR</td>
<td>NZ, CARON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RET2</td>
<td>XOR</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>(FORFG), A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CARON</td>
<td>DEC</td>
<td>BC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>A,B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OR</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JR</td>
<td>NZ, NMOTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

; Get sequence row count
; If count zero then
; exit
; HL points to memory start
; DE points to temporary buffer
; Save count
; Motor count of six
; Copy memory slice into TBUF
; Save new memory pointer
; Drive all motors for this slice
; See if keyboard input
; Restore memory pointer
; Restore row count
; User typed a '('.
; No then continue
; Clear A
; Clear flag to halt routine above
; exit
; Decrement count
; Test for zero
; No then carry on else
; return
SURTOUTINE DRIVL

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

DRIVL    LD    C,Ø            ; Set BC = motor count
SCANW    LD    B,6            ; Point to TBUF
LD      HL,TBUP               ; Get step value from TBUF
CR      A                        ; Is it zero?
JR     NZ,TBNZR               ; No then continue
INC     HL                    ; Point to next TBUF location
DJNZ   TBZER                  ; Do next motor check
RET                           ; If no motor to step, then return

TBNZR    LD    DE,MOTBF + 5    ; DB points to last direction array
LD      HL,TBUP + 5           ; HL points to TBUF
LD      B,6                   ; B = motor count
DGAGN    LD    A,(HL)         ; Get motor step value
CP      Ø                      ; Is it zero?
JR     Z,NOEL                 ; Yes then skip
JP     M,SNEG                 ; Is it negative ie, reverse
SFCG     LD    A,3             ; No positive, so load MOTBF (N)
LD      (DE),A                  ; With 3
DEC     (HL)                   ; Decrement motor count in TBUF
JR      NOFIL                 ; Complete the MOTBF array
SNEG     LD    A,1             ; Set MOTBF = 1 for
LD      (DE),A                ; a positive drive
INC     (HL)                   ; Decrement negative count
JR      NOFIL                 ; Do rest of MOTBF
NOEL     XOR     A              ; Clear MOTBF (N)
LD      (DE),A                  ; Move to next MOTBF element
NOFIL    DEC     DE              ; Move to next TBUF element
DEC     HL                    ; Do for all six motors
DJNZ   DOAGN                  ; Set key pressed flag
LD      A,1                    ; Step all motors once if
LD     (REWP),A               ; any to step
CALL    STEPM                 ; Do for maximum of 123 cycles
DEC     C                      ; then return
JF      NZ,SCANW
RET
SUBROUTINE INIT

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

INIT
    LD    HL, $0            ; Set HL = $0
    LD    (COUNT), HL      ; and clear the row count
    XOR   A                 ; Clear A
    LD    (MAN), A         ; Now clear MAN
    LD    HL, ARST         ; HL = start of arm store
    LD    (CUROW), HL      ; CUROW = start of arm store
    CALL   CLRBUF          ; Clear TBUF, DRBUF & MOTBF
    CALL   RESET           ; Clear the POSAR array
    CALL   CLMWT           ; Free all motors
    RET                 ; EXIT
; This routine takes the POSAR array and uses it to drive
; all the motors until the ARM is in its defined start position

AMOVTO
PUSH AF                   ; *
PUSH BC                   ; *
PUSH DE                   ; *
PUSH HL                   ; *
RES1
LD HL, POSAR             ; HL points to POSAR
LD B, 12                 ; B = count of 12
NRES1
LD A, (HL)               ; Get POSAR element
OR A                     ; Is it zero?
JR NZ, MTSA              ; No then continue
INC HL                   ; Point to next POSAR element
DJNZ NRES1               ; See if all zero
JR ENDSC                 ; All zero so end!
MTSA
LD HL, POSAR + 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
JP NZ, DOMPL             ; no skip
LD (DE), A               ; Zero MOTBF (N)
DEC HL                   ; advance POSAR pointer
JR NMDR                  ; Do next motor
DOMPL
LD A, B                  ; See direction to move in
BIT 7, A                 ;
JR Z, RMOVTL            ; Go in reverse
INC BC                   ; Go forward
LD A, 1                  ; A = forward
JR DOITL                ; Do rest
RMOVTL
DEC EC                   ; Dec count for reverse
LD A, 3                  ; Set reverse in A
DOITL
LD (DE), A               ; Store reverse in MOTBF (N)
LD (HL), B               ; Store updated POSAR count
DEC HL                   ; in POSAR (N)
LD (HL), C               ; Store lower byte
NMDR
DEC HL                   ; point to next POSAR element
DEC DE                   ; Move to next MOTBF element
POP BC                   ; Restore motor count
DJNZ RSCAN              ; Do for next motor
CALL ORAMT              ; Drive all motors to be driven
JR RES1                  ; Do till all POSAR slots zero
ENDSC
POP HL                   ; *
POP DE                   ; *
POP BC                   ; *
POP AF                   ; *
RET                      ; 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 CTPCS(1-6)
; SETDT sets all CTPOS elements to start offset
; position which equals l.

TORQUE
  PUSH AF    ; * Set clear motor-
  PUSH BC    ; * Save Registers
  PUSH DE
  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 (DE),A  ; in FTABL
  LD A,B     ; Get motor address in A
  SLA         ; 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 TCQCL   ; Exit with register restoration

CLRMT
  PUSH AF    ; * clear all motors torque
  PUSH BC    ; * Save Registers
  PUSH DE
  PUSH HL    ; *
  LD HL,NOTOR ; Print "NO TORQUE" message
  CALL PSTR
  LD D,OFH    ; 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
  CR 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

TCQCL
  POP HL     ; *
  POP DE
  POP BC     ; * Restore Registers
  POP AF     ; *
  RET        ; Done, exit
SETDT
PUSH BC  ; * Set CTPOS elements to start
PUSH DE  ; * Save used registers
PUSH HL  ; *
LD B,6 ; Motor count to B
LD HL,CTPOS ; HL points to CTPCS array
LL (HL),1 ; Set CTPOS(N) to start position = 1
INC HL ; Increment HL
DJNZ NST1 ; Do set up next CTPCS element
POP HL ; *
POP DE ; * Restore used registers
POP BC ; *
RET
SUBROUTINE DRAMT

; DRAMT drives all six motors directly and uses
; PTABL to output the correct pulse patterns.
; For half stepping the pattern must be changed in PTABL.
; 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 PTABL offset array
NMTDT       LD  A,(DE)    ; Get MOTEF(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
            LD  (HL),A   ; Store in CTPOS (N)
NORST       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  EC      ; * 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
NORST       LD  A,(HL)   ; Get table offset 1-4
            PUSH  AF    ; *
            PUSH  DE    ; * Save Registers
            PUSH  HL    ; *
            LD  HL,PTABL-1 ; Get table start
            LD  D,$0     ; DE now equals 1-4
            LD  E,A      ; Add to PTABL -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
            OR  C        ; Output to interface circuitry
            CALL  (PGRT),A
            POP  HL      ; *
            POP  DE      ; * Restore Registers
            POP  AF      ; *
            RET          ; Return

31 - 4*
SUBROUTINE STEPM

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

STEPM  PUSH  AF  ; * Save Register
        PUSH  HL
        PUSH  BC  ; *
        LD   HL,MOTBP   ; HL points to motor buffer
        LD   E,6
        B   =  Count
TRYØ   LD   A, (HL)  ; Get motor value 3 or 1
        OR   A  ; Zero?
        JR   NZ, CONT  ; No then continue
CONT   INC  HL  ; Point to next motor
        DJNZ TRYØ  ; Do next motor
        POP  BC  ; *
        POP  HL  ; * Restore Registers
        POP  AF  ; *
        RET  ; Exit
CONTA  POP  BC  ; *
        POP  HL  ; * Restore registers
        CALL  DRANT  ; Drive motors
        CALL  POSIC  ; Increment relative position
        POP  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              ; HC points to previous motor slice
       LD    D,H               ;
       LD    E,L               ; Move HL to DE
       POP   HL                 ; Restore current row pointer
       PUSH  HL                 ; Save again
       LD    E,C               ;
       NCOMP  LD    A,(HL)      ; Get contents of this row
       CP    Ø                  ; See if positive or negative
       LD    A,(DE)             ; Get identical previous motor slot
       JP    P,PDIR             ; if positive do for positive motor
       NDIR  CP    Ø              ; Compare if both in same
       JP    M,NXTCK            ; direction then skip else
       CDDEL  CALL  DELLN       ; delay and
       NCDSG  POP  HL            ; *
       POP  DE                ; *
       POP  BC                ; * Restore registers
       POP  AF                ; *
       RET                   ; Now return
       PDIR  CP    Ø              ; If previous motor is negative
       JP    P,NXTCK            ; then delay, else do for next
       JR    CDDEL              ; motor slot
       NXTCK  INC  HL            ; increment current row pointer
       INC  DE                ; increment lost row pointer
       DJNZ  NCOMP              ; do for next motor
       JR    NCDSG              ; Return with no large (1 sec) delay

*4 - 33*
SUBROUTINE SRAMT

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

SRAMT
LD A,(MAN); Get manual flag
OR A; Is it zero?
JP NZ,STEPM; Yes then just step motors
LD (STRFG),A; Clear the store flag
LD B,6; B = motor count
LD 1X,DRBUF+6; 1X = previous direction buffer
LD 1Y,MOTBF+5; 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 TBURF(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 TBURF
LD A,(HL); Get new value
CP 127; Check against upper board
CALL 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,CREVI; No then continue
CALL SETST; Else set store TBURF in ARST flag
JR NODRV; Do next motor

CREVI
DEC (HL); Advance step count in TBURF (N)
LD A,(HL); Get element
CP -128; Compare with upper negative bound
CALL Z,SETST; Limit reached so set store flag
JR NODRV; Do next motor

CREVD
LD (1X+Ø),1; Set Direction

SETST
PUSH AF; Save AF
LD A,1; Set store flag STRFG

SETSC
LD (STRFG),A; to one
POP AF; Restore AF
RET; Continue

*4 - 34*
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    CLRKM  ; Clear MCTBF array
LD       A,(3840H) ; Get TR5BØ 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 PRESSES flag
LD       (KEYP),A
LD       A,(3810H)
BIT      Ø,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
LC       A,(3810H) ; Restore memory value
JR       Z,TRYN1  ; No then skip
LD       A,(MAN)  ; See if in manual mode
CR       A
CALL     2,STORE ; No then store TBUF
OR       1        ; Not not finished flag
RET      ; and exit to caller
TRYN1    LD       BC,Ø  ; Clear MCTBF 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 MCTBF
TRYN2    INC      BC  ; Increment MCTBF 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      ; See if '3' key pressed, No skip
JP       Z,TRYN4  ; Set forward direction on Motor 3
CALL     FORMT    ; 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

*3 - 35*
LD BC, 0 ; Clear BC offset for motor 1
LD A, (3804H) ; See if 'Q' key pressed
JP Z, TRYQ ; No then skip
CALL BACMT ; Set motor 1 for backward
INC BC ; Advance pointer
BIT 7, A ; See if 'W' key pressed
JP Z, TYRE ; No skip
CALL BACMT ; Do backward for motor 2
INC BC ; Advance pointer offset
LD A, (3801H) ; See if
BIT 5, A ; 'E' key pressed
JR Z, TRYR ; No skip
CALL BACMT ; Set motor 3 for backward
INC BC ; Advance pointer offset
LD A, (3804H) ; See if
BIT 2, A ; Key 'R' is pressed
JP TRYT ; No skip
CALL BACMT ; Set motor 4 backward
INC BC ; Advance offset
BIT 4, A ; Is key 'T' pressed?
JP Z, TRYY ; No skip
CALL BACMT ; Set motor 5 backward
LD A, (3808H) ; Is the 'Y' key pressed?
INC BC ; Advance offset
BIT 1, A ; No key
JP Z, SOMEN ; 'Y' then skip
CALL BACMT ; Set motor 6 for backward
CALL SRANT ; Step motors, maybe store.
OR 1 ; Set zero key not pressed flag
RET ; Return to caller
LD A, (MAN) ; Zero was pressed so see
OR A ; if in learn mode
CALL 2, STORE ; Yes then store
XOR A ; Set zero flag and
RET ; Return to caller
LD E, 3 ; Set for forward direction
JR SETMT ; Do set motor slot in MOTBF
LD E, 1 ; Set for reverse direction
LD HL, MOTBF ; Point to MOTBF
ADD HL, BC ; Add in motor offset
PUSH AF ; Save AF
LD A, (HL) ; Get byte
OR A ; See if zero
JR Z, DOMOT ; Yes then set byte
XOR A ; Clear
LD (HL), A ; byte in MOTBF user wants both
PUSH AF ; directions clear byte
RET ; Restore AF and return
POP AF ;
LD (HL), E ; Set byte in MOTBF
LD A, 1 ; and set
LD (KEYP), A ; key pressed flag
POP AF ; Restore AF
RET
SUBROUTINE CBTA5

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

CBTA5

PUSH AF
PUSH HL
PUSH DE
PUSH 1X
BIT 7,H
JR Z,POSNO
LD A,H
CPL
LD H,A
LD A,L
CPL
LD L,A
INC HL
LD A,MINUS
INC 1X
JR CONUM
LD A,SPAC
JR PUTSN
LD 1Y,BTOAT
LD A,NUMBA
LD E,(1Y+0)
LD D,(1Y+1)
SUBBA
OR A
SBC HL,DE
JP C,CONEN
INC A
JR SUBBA
ADD HL,DE
LD (1X+0),A
INC 1X
INC L
INC L
DEC E
JR NZ,NULP
XOR A
LD (1X+0),A
POP 1Y
POP 1X
POP DE
POP HL
POP AF
RET

; Now 2's complement negative
; Pointed to by LX
; Do rest of conversion
; Place a space if number positive
; Jump to copy space to memory
; Save 1Y register
; Point to subtraction table
; Get ASCII $0$ in A
; Get table value
; Subtract table value from value input
; If carry then do for next digit
; Inc count (ASCII in A)
; Do next subtraction
; Restore value before last subtraction
; Store ASCII Number in memory
; Inc memory pointer
; Point to next table value
; Test if $E = \phi$
; No then try for next digit
; Clear A and place in store
; as EOS = End of string
; * Restore all saved registers
; * and
; Exit

*4 - 37*
BTOAT

DEFW 10000 ; Table of subtraction constants
DEFW 1000 ; for conversion routine
DEFW 100
DEFW 10
DEFW 1
CLEARING AND Resetting ROUTINES

; CLR MF clears the MOTBF array

CLR MF
PUSH BC
PUSH DE
POP HL
LD HL, MOTBF
LD DE, MOTBF +1
LD BC, 5
LD (HL), Ø
LDIR
POP HL
POP DE
POP BC
RET

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

CTBUF
PUSH BC
PUSH DE
PUSH HL
LD HL, TBUF
LD DE, TBUF +1
LD BC, 17
LD (HL), Ø
LDIR
POP HL
POP DE
POP BC
RET

*4 - 39*
SUEROUTINE GINT

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

GINT
PUSH BC     ; * 
PUSH DE     ; * Save Registers
XOR A       ; Clear A and carry
SBC HL, HL  ; Zero HL
LD B, 5     ; Maximum of 5 characters
LD (MIN), A  ; Clear MIN=Minus Flag
GINT1       
CALL GHRA   ; Get a character and display it
CP SPAC      ; Is it a space?
JR Z, GINT1  ; Yes then skip
CP NL        ; Is it a newline?
JR Z, PRET1  ; Done if new line, return zero
CP MINUS     ; A minus number?
JR NZ, POSON ; No then see if positive
LD A, 1      ; Set minus flag
LD (MIN), A  ; Get rest of number
JR GINT2     ; Is number a positive number
CP '+'       ; See if numeric
JR NZ, NUM1  ; Newline?
GINT2       
CALL GHRA   ; Get next character
CP NL        ; Yes then exit
ADD HL, HL   ; Double number
PUSH HL      ; Save X 2
ADD PL, HL   ; X 4
ADD HL, HL   ; X 8
POP DE       ; Restore X 2
ADD HL, DE   ; Now add to get X 10
CP 0         ;
JR C, EFRN2  ; If number less than ASCII 0 ERR
CP '9' + 1   ; If number greater than ASCII
JR NC, EFRN2 ; 9 then error
SUE NUMBA    ; Number input OK, so make into
LD E, A      ; Binary and
LD D, 0      ; load into DE
ADD HL, DE   ; Now add to total
DJNZ GINT2   ; Do for next digit
CALL PNEWL   ; Print a new line
NUMET        
LD A, (MIN)  ; Is number negative?
OR A         ;
JR Z, PRET1  ; No then finish off
LD A, L      ; else complement
CPL          ; The value in HL
LD L, A      ;
LD A, H      ; (2’s Complement)

*4 - 40*
PRET1
CPL
LD H,A
INC HL
XOR A
PCF DE
POP BC
RET

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

; Clear A and flags
; * Restore Registers
; * and return
; 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   BC       ; *
  PUSH   DE       ; * Save all registers
  PUSH   HL       ; *
  LD      HL,POSSR ; Print "POSAR="
  CALL   PSTR     ; String
  LD      B,6      ; Motor count into B
  LD      DE,POSSR ; Point to array containing offsets
  LD      A,(DE)   ; Get lower order byte into L
  INC     DE       ; Increment memory pointer
  LD      L,A      ; Get higher order byte into H
  INC     DE       ; Increment to next number
  LD      LX,NUMAR ; LX 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   NPOSRA    ; Do for next motor
  CALL   PNEAL     ; Print a new line, all done
  POP     HL       ; *
  POP     DE       ; *
  POP     BC       ; * Restore all Registers
  POP     AF       ; *
  RET       ; Now return

*4 - 42*
SUBROUTINE PGSIC

; PGSIC 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 360 times to cause such an event.

PGSIC    PUSH AF    ; *
PUSH BC    ; *
PUSH DE    ; * Save registers
PUSH HL    ; *
LD B,6     ; B = motor count
LD DE,MOTBF+5 ; Point to MCTLF
LD HL,POSAR+10 ; Point to POSAR (relative position)

NPOS1    PUSH BC    ; Save motor count
LD C,(HL) ; Get lower FOSAE byte in C
INC HL    ; Point to Higher byte
LD B,(HL) ; Get higher byte in B
LD A,(DE) ; Get direction byte from MCTLF
AND 3     ; Clear all higher bits from D7-D3
OR A      ; Is it zero?
JR NZ,NNZM ; No skip
DEC HL    ; Yes then move POSAR pointer back
JR NPOS2  ; and continue with next motor

NNZM     BIT l,A    ; Test direction bit
JR NZ,RDPOS ; Do for reverse direction
INC BC    ; Advance element
JR STFCS  ; Restore 16 bit POSAR element

RDPOS    DEC BC    ; Advance negative POSAR element
STPOS    LD (HL),B ; Store higher byte
DEC HL    ; Move pointer to lower byte
LD (HL),C ; Store lower byte

NPOS2    DEC HL    ; Back up POSAR pointer to
DEC HL    ; next motor position slot
DEC DE    ; Backup MOTBF pointer to next slot
POP BC    ; Restore Motor count
DJNZ NPOS1 ; Do next motor
POP HL    ; *
POP DE    ; * Restore used Registers
POP BC    ; *
POP AF    ; *
RET       ; Done, Exit
; STORE copies the TBUF array into the locations pointed to
; by CURCW. 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
PUSH HL
LD HL,TBUF
LD B,6
OR A
JR NZ,STOR1
INC HL
DJNZ STEST
JR EXIT
LD (1X+O),O
LD HL,(COUNT)
INC HL
LD A,H
CP L
JP NC,OVRFW
LD (COUNT),HL
LD DE,(CURROW)
LD HL,TBUF
LD BC,8F86
LDI (CURROW),DE
CALL CTBUF
POP HL
POP BC
RET
OVRFW
LD HL,(AVMS)
CALL PSTR
CALL GCHR
CALL PNSVL
CP 'D'
JP Z,REDO
CP 'S'
JR Z,EXIT2
JR OVRFW
REDO
CALL JN1T
POP HL
POP BC
POP BC
JP QUEST
*4 - 44*
SUBROUTINE RESET

; This subroutine clears the POSAR array

RESET    PUSH    BC                     ; *  Save Registers
          PUSH    DE                     ; *        
          PUSH    HL                     ; *
          LD    HL,POSAR                  ; Point to POSAR start
          LD    DE,POSAR+1                ; Point to next element
          LD    (HL),#0                   ; 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 PCCHR ; 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 Ts880
CALL PUTSTR ; Print the string
FCP DE ; * Restore Registers
FCP BC ;
RET ; Done, Exit

; PSPAC prints a space character

PSPAC
PUSH AF ; Save AF
LD A, 0 ; 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, $D8H ; 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 XBM ; See if character is there
POP DE ; Restore
RET ; Done, Exit

; GCHRA gets a character from keyboard and displays it

GCHRA
CALL GCCHR ; Get a character
CALL PUTCHR ; Print it
RET ; Done, Exit
CLEAR SCREEN ROUTINE

; Simple scrolling type screen clear

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

<table>
<thead>
<tr>
<th>DELSW</th>
<th>PUSH</th>
<th>BC</th>
<th>Delay for 10 * 5 + 10 M cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELSL</td>
<td>PUSH</td>
<td>BC</td>
<td>Save BC</td>
</tr>
<tr>
<td>NOP</td>
<td></td>
<td></td>
<td>Delay for 11 T state</td>
</tr>
<tr>
<td>NOP</td>
<td></td>
<td></td>
<td>4 T state delay</td>
</tr>
<tr>
<td>POP</td>
<td>BC</td>
<td></td>
<td>4 T state delay</td>
</tr>
<tr>
<td>DJNZ</td>
<td>DELSL</td>
<td></td>
<td>Delay for 11 T states</td>
</tr>
<tr>
<td>POP</td>
<td>BC</td>
<td></td>
<td>Do delay times value in B</td>
</tr>
<tr>
<td>RET</td>
<td></td>
<td></td>
<td>Restore BC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Exit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DELS</th>
<th>PUSH</th>
<th>BC</th>
<th>Save BC</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>B,20</td>
<td></td>
<td>Set B for 0.01 sec delay (apx)</td>
</tr>
<tr>
<td>CALL</td>
<td>DELSW</td>
<td></td>
<td>Do delay</td>
</tr>
<tr>
<td>POP</td>
<td>BC</td>
<td></td>
<td>Restore BC</td>
</tr>
<tr>
<td>RET</td>
<td></td>
<td></td>
<td>Exit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DELT</th>
<th>PUSH</th>
<th>BC</th>
<th>Save BC</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>E,0</td>
<td></td>
<td>Set B for 0.01 sec delay (apx)</td>
</tr>
<tr>
<td>CALL</td>
<td>DELSW</td>
<td></td>
<td>Do delay</td>
</tr>
<tr>
<td>POP</td>
<td>BC</td>
<td></td>
<td>Restore BC</td>
</tr>
<tr>
<td>RET</td>
<td></td>
<td></td>
<td>Exit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DELLN</th>
<th>PUSH</th>
<th>EC</th>
<th>Save BC</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>B,20</td>
<td></td>
<td>Set B for 1.0 sec delay (apx)</td>
</tr>
<tr>
<td>CALL</td>
<td>DELSW</td>
<td></td>
<td>Do delay</td>
</tr>
<tr>
<td>POP</td>
<td>DELLN</td>
<td></td>
<td>Do next delay section</td>
</tr>
<tr>
<td>RET</td>
<td>BC</td>
<td></td>
<td>Restore BC</td>
</tr>
<tr>
<td></td>
<td></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>QD</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ø</td>
<td>1</td>
<td>ø</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>ø</td>
<td>ø</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ø</td>
<td>1</td>
<td>ø</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ø</td>
<td>1</td>
<td>1</td>
<td>ø</td>
<td>4</td>
</tr>
</tbody>
</table>

### HALF STEPPING PULSE SEQUENCE

<table>
<thead>
<tr>
<th>QA</th>
<th>QB</th>
<th>QC</th>
<th>QD</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ø</td>
<td>1</td>
<td>ø</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>ø</td>
<td>ø</td>
<td>ø</td>
<td>1.5</td>
</tr>
<tr>
<td>1</td>
<td>ø</td>
<td>ø</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ø</td>
<td>ø</td>
<td>ø</td>
<td>1</td>
<td>2.5</td>
</tr>
<tr>
<td>ø</td>
<td>1</td>
<td>ø</td>
<td>1</td>
<td>3.Ø</td>
</tr>
<tr>
<td>ø</td>
<td>1</td>
<td>ø</td>
<td>ø</td>
<td>3.5</td>
</tr>
<tr>
<td>ø</td>
<td>1</td>
<td>1</td>
<td>ø</td>
<td>4</td>
</tr>
<tr>
<td>ø</td>
<td>ø</td>
<td>1</td>
<td>ø</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 CPT 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
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARM</td>
<td>Program entry point</td>
</tr>
<tr>
<td>LEARN</td>
<td>Learn a sequence command</td>
</tr>
<tr>
<td>EDIT</td>
<td>Edit a sequence command</td>
</tr>
<tr>
<td>READ</td>
<td>Read in sequence from tape command</td>
</tr>
<tr>
<td>WRITE</td>
<td>Write sequence to tape command</td>
</tr>
<tr>
<td>CHECK</td>
<td>Check stored sequence command</td>
</tr>
<tr>
<td>BOOT</td>
<td>Re-start system command</td>
</tr>
<tr>
<td>FINISH</td>
<td>Exit from system command</td>
</tr>
<tr>
<td>SETARM</td>
<td>Set start position command</td>
</tr>
<tr>
<td>TOSTM</td>
<td>Move arm to start position command</td>
</tr>
<tr>
<td>F'REARM</td>
<td>Free all arm joints</td>
</tr>
<tr>
<td>MANU</td>
<td>Go into manual mode</td>
</tr>
<tr>
<td>GO</td>
<td>Execute stored sequence command</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>Display stored Sequence command</td>
</tr>
</tbody>
</table>
MAIN LOOP

; Program start

STARM

CALL CLRSC ; Clear the TRS80 screen
LD HL,SICON ; 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 menu string
CALL PSTR ; Print it
CALL GCHR ; Get response and print it
CALL PNEWL ; Print new line
CP NL ; Is response a newline
JR Z,QUESS ; Yes then ignore
CP 'L' ; Is response an 'L'
JP Z,LEARN ; Yes do learn section
CP 'E' ; Is it an 'E'
JP Z,EDIT ; Yes do edit
CP 'R' ; Is it an 'R'
JP Z,READ ; Yes then do read command
CP 'W' ; Is it a 'W'
JP Z,WRITE ; Yes do write command
CP 'C' ; Is it a 'C'
JP Z,CHECK ; Yes do check routine
CP 'S' ; Is it an 'S'
JP Z,SETAM ; Yes then do arm set
CP 'T' ; a 'T'
JP Z,TOSTM ; Yes then move arm to start
CP 'G' ; a 'G'
JP Z,GO ; Do execute movements stored
CP 'D' ; a 'D'
JP Z,DISP ; Yes then display ARST array
CP 'B' ; a 'B'
JP Z,BOOT ; Yes then restart system
CP 'M' ; an 'M'
JP Z,MANU ; Yes the Manual control of arm
CP 'F' ; a 'F'
JP Z,PREARM ; Yes then clear all motors
CP 'Q' ; a 'Q'
JP Z,FINISH ; Yes then quit program
LD HL,QMESS ; Point to 'FARDON' message
CALL PSTR ; Print it
JP QUESS ; Try for next command
THE LEARN ROUTINE

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

LEARN

LD HL,RELNS ; Point to learn message
CALL PSTR ; Print the message
CALL CCHR ; Get response and print it
CALL PNEWL ; Print a new line
CP '.' ; Response a '.
JP Z,QUEST ; 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 CCHR ; Get response and print it
CALL PNEWL ; Print new line character
CP '.' ; Response a '.
JP QUEST ; Exit to main loop if so
CP SPAC ; Is it a space?
JR NZ,WAIT2 ; If not then bad input, try again
CALL TORQUE ; Switch motors on
JR STLRN ; Do rest of learn

NOINT

LD HL,(COUNT) ; Get current count
LD A,L ;
OR H ; Is it zero?
JR Z,NOSTR ; Yes then can't add to nothing
XOR A ; Clear manual flag
LD (MAN)A ; Because we are in learn mode
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 QUEST ; Back to main loop

*4 - 13*
<table>
<thead>
<tr>
<th>EDIT</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>HL, (COUNT)</td>
<td>Get row count</td>
</tr>
<tr>
<td>LD</td>
<td>A, L</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>H</td>
<td>Test for zero</td>
</tr>
<tr>
<td>JP</td>
<td>Z, NOSTR</td>
<td>Yes then nothing in store</td>
</tr>
<tr>
<td>CALL</td>
<td>PSTR</td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>GCHRA</td>
<td>Get response</td>
</tr>
<tr>
<td>CALL</td>
<td>PNEWL</td>
<td>Print a new line</td>
</tr>
<tr>
<td>CP</td>
<td>'M'</td>
<td>Is response an 'M'</td>
</tr>
<tr>
<td>JR</td>
<td>Z, EDMOT</td>
<td>Yes then edit motor</td>
</tr>
<tr>
<td>CP</td>
<td>'R'</td>
<td>Is response an 'R'</td>
</tr>
<tr>
<td>JR</td>
<td>NZ, EDSRT</td>
<td>No then try again</td>
</tr>
<tr>
<td>LD</td>
<td>HL, COUTS</td>
<td>HL = New row count message</td>
</tr>
<tr>
<td>CALL</td>
<td>PSTR</td>
<td>Print it</td>
</tr>
<tr>
<td>CALL</td>
<td>GINT</td>
<td>Get 16 bit signed integer</td>
</tr>
<tr>
<td>JP</td>
<td>NZ, BADC</td>
<td>Non zero return means bad input</td>
</tr>
<tr>
<td>LD</td>
<td>A, H</td>
<td>Test top bit of HC</td>
</tr>
<tr>
<td>BIT</td>
<td>7, A</td>
<td></td>
</tr>
<tr>
<td>JP</td>
<td>NZ, BADC</td>
<td>If negative then bad input</td>
</tr>
<tr>
<td>LD</td>
<td>BC, (COUNT)</td>
<td>Get count value</td>
</tr>
<tr>
<td>PUSH</td>
<td>HL</td>
<td>Save response</td>
</tr>
<tr>
<td>OR</td>
<td>A</td>
<td>Clear carry flag</td>
</tr>
<tr>
<td>SBC</td>
<td>HL, BC</td>
<td>See if response &lt; current count</td>
</tr>
<tr>
<td>POP</td>
<td>HL</td>
<td>Restore response</td>
</tr>
<tr>
<td>JR</td>
<td>NC, BADC</td>
<td>Replace count with response</td>
</tr>
<tr>
<td>LD</td>
<td>(COUNT), HL</td>
<td>Back to main loop</td>
</tr>
<tr>
<td>JP</td>
<td>QUEST</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EDMOT</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>PSTR</td>
<td>Print 'row number'</td>
</tr>
<tr>
<td>CALL</td>
<td>GINT</td>
<td>Get integer response</td>
</tr>
<tr>
<td>JR</td>
<td>NZ, BADC</td>
<td>Bad answer</td>
</tr>
<tr>
<td>LD</td>
<td>A, H</td>
<td>No negative row count allowed</td>
</tr>
<tr>
<td>BIT</td>
<td>7, A</td>
<td>or zero row count</td>
</tr>
<tr>
<td>JR</td>
<td>NZ, BADC</td>
<td>Get row count into BC</td>
</tr>
<tr>
<td>LD</td>
<td>BC, (COUNT)</td>
<td>Move count up one</td>
</tr>
<tr>
<td>INC</td>
<td>BC</td>
<td>Clear carry flag</td>
</tr>
<tr>
<td>PUSH</td>
<td>HL</td>
<td>Subtract count from response</td>
</tr>
<tr>
<td>SBC</td>
<td>HL, BC</td>
<td>Restore response</td>
</tr>
<tr>
<td>POP</td>
<td>HL</td>
<td>If greater than allowed error</td>
</tr>
<tr>
<td>JR</td>
<td>NC, BADC</td>
<td>Move response down one</td>
</tr>
<tr>
<td>DEC</td>
<td>HL</td>
<td>Double HL</td>
</tr>
<tr>
<td>ADD</td>
<td>HL, HL</td>
<td>Save it</td>
</tr>
<tr>
<td>PUSH</td>
<td>HL</td>
<td>Row count x 4</td>
</tr>
<tr>
<td>ADD</td>
<td>HL, HL</td>
<td>BC = row count x 2</td>
</tr>
<tr>
<td>POP</td>
<td>BC</td>
<td></td>
</tr>
</tbody>
</table>
ADD HL,BC
LD BC,ARST
ADD HL,BC
PUSH HL
CALL HL,MOTNS
CALL PSTR
JR NZ,BADNM
LD A,H
OR A
JR NZ,BADNM
LD A,L
CF 1
JR C,BADUM
CF 7
JR NC,BADNM
POP HL
DEC A
LD C,A
LD B,Ø
ADD HL,BC
PUSH HL
LD HL,NVALS
CALL PSTR
CALL GINT
JR NZ,BADNM
LD A,H
CP ZFFH
JR NZ,PEDIT
BIT 7,L
JR Z,BADNM
JR MOTAS
PEDIT CR A
JR NZ,BADNM
BIT 7,L
JR NZ,BADNM
MOTAS LD A,L
POP HL
LD (HL),A
JP QUESL
BADNM POP HL
LD HL,BADNM
CALL PSTR
JP QUESL

HL = Row count x 6
Get store start address
Add row offset
Save resulting pointer
Print
Motor number string
Get Answer
Bad answer
Response too large
No motor number < 1
No motor number > 6
Restore = Memory pointer
Motor offset Ø → 5
Add to memory pointer
Now we point to motor in store
Save pointer
Print new step value
Get response
Bad answer
We have a positive response
New negative step value too large
Step value OK
New positive step value too large
so exit
close ok
Get step value
Restore memory pointer
Place step value in store
Go do next operation
Print error message and
return to main loop

*4 = 15*
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    PNEXNL   ; Print new line
     CP      ','      ; Is response a dot?
     JP      Z,QUEST   ; Yes then exit
     CP      SPAC     ; Is it a space?
     JR      NZ,READ   ; No then try again
     XCR     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
     PUSH    BC       ; Same count
     LD      E,A      ; 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    CASOP    ; Switch cassette off
     JP      TAPEF    ; exit
     LD      HL,REMSG ; Error message for tape
     CALL    PSTR     ; Print it
     JP      QUEST    ; Go to main loop

*4 - 16*
WRITE ROUTINE

; Writes a stored sequence to tape

WRITE       LD     BC, (COUNT)   ; Get row count
            LD     A, B
            OR     C
            JP     Z, NOSTR
            LD     HL, CASRD
            CALL    PSTR
            CALL    GCHEA
            CALL    PNEWL
            CP     ' '
            JP     Z, QUES1
            CP     SPAC
            JR     N2, BADWI
            XOR    A
            CALL    CASON
            CALL    DELT
            CALL    WRLDR
            CALL    DELT
            LD     BC, (COUNT)
            LD     A, B
            CALL    WRBYA
            LD     A, C
            CALL    DELT
            CALL    WRBYA
            LD     HL, ARST
            PUSH    BC
            LD     E, 0
            LD     B, 6
            LD     A, (HL)
            CALL    DELS
            CALL    WRBYA
            CALL    DELS
            ADD    A, E
            ADD    E, A
            INC    HL
            DJNZ    WRBYT
            CALL    WRBYA
            POP     BC
            DEC     BC
            LD     A, B
            OR     C
            JR     NZ, ROWNW
            CALL    CASOH
            JP     QUES1

*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
BADCI  LD  HL, CASRD ; Print wait message
         CALL PSTR
         CALL GCHRA ; Get answer
         CALL FNEWL ; Print new line
         CP  ',' ; Is response a ','
         JP  Z, QUES1 ; Yes then go to main loop
         CP  SPAC ; Is it a space
         JR  N2, BACCI ; 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  D ; 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
ROWNC  CALL READC ; Read a motor step element
         CP  (HL) ; Same as in store?
         JP  NZ, RDERR ; Not the same so error
         ADD  A, E ; Add to check sum
         LD  HL ; Advance memory pointer
         DJNZ CKBYT ; Do next row element
         POP  BC ; Restore row count
         CALL READC ; Read check sum
         CP  E ; Same as check sum calculated
         JP  NZ, RDERR ; No then error
         DEC  BC ; Decrement count
         LD  A, B ; Is count zero?
         OR  C ; No then do next row
         JP  NZ, ROWNC ; Switch cassette off
         CALL CASOF ; Print tape off message
         CALL PSTR ; and back to main loop
         JP  QUES1 ;

*4 - 18*
BOOT AND FINISH COMMANDS

; This routine restarts the program

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

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

FINISH
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,FINISH ; No then try again
CALL PNEWL ; Print a new line
JP QUESl ; Back to main loop

*4 -19*
OTHER SHORT COMMANDS

; SETAM: clears arm position array
SETAM CALL RESET ; Clear Arm array (POSAR)
   JP QUESL ; Back to main loop

; TOSTM: moves the arm back to its start position
TOSTM CALL MOVTO ; Steps motors till POSAR elements
   JP QUESL ; are zero then back to main loop

; FPREARM: frees all motors for user to move arm
   by hand
PFREARM CALL CLRMT ; Output all ones to motors
   JP QUESL ; 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 QUESL ; Back to main loop
THE GO COMMAND

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

GO CALL PNEWL ; Print a new line
CALL MOVTO ; Move arm to start
XOR A ; Clear
LD (FORFG),A ; Forever Flag FCPEG
LD HL,AORM ; Print "DO ONCE OR FOREVER"
CALL PSTR ; Message
CALL CCHRA ; 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 (FORFC),A ; to 1
ONECY LD A,',' ; Print a ','
CALL PUTCHR ; Using PUTCHR
CALL DCALL ; Execute the sequence
LD A,(FORFC) ; Test FORFG, if zero
OR A ; then we do not want
JF 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,DONNS ; Print sequence done
CALL PSTR ;
JF 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       ; Point to header string
       CALL  PSTR            ; and display it
       CALL  POSDS          ; Print out the relative position
       LD   HL,ARST         ; Point to sequence start
       LD   BC,(COUNT)     ; BC = how many rows to print
       OR   C              ; Test if count is zero
       JP    NZ,SETBC      ; No then jump to rest of
       LD   HL,NODIS       ; display else print message
       CALL  PSTR          ; telling user nc display and
       JP    QUES1         ; return to the main loop
       LD   EC,OFF         ; Clear BC for row count
       PUSH  BC            ; Save it
       PUSH  HL            ; Save memory position
       LD    H,B           ; HL = row count
       LD    L,C           ; Now row count = N+1
       INC  HL             ;
       LD   1X,NUMAR       ; 1X points to buffer for ASCII string
       CALL  CBTAS         ; Convert HL to ASCII
       LD   HL,NUMAR       ; Point to ASCII string
       CALL  PSTR          ; row print it
       LD    A,'.'         ; Print a '.'
       POP   HL            ; Restore memory pointer
       LD    E,5           ; Motor count to B (6 motors)
       LD    A,(HL)        ; Get step value
       PUSH  HL            ; Save memory pointer
       PUSH  BC            ; Save motor count
       BIT   7,A           ; Test bit 7 of A for sign
       JR    Z,NUMFO       ; If bit = 0 then positive step
       LD    H,OFFH        ; Make H = negative number
       JR    EVAL          ; Do rest
       LD    E,0           ; Clear H for positive number
       LD    L,A           ; Get low order byte into L
       LD   1X,NUMAR       ; Point to result string
       CALL  CBTAS:        ; Call conversion routine
       LD   HL,NUMAR       ; HL points to result
       CALL  PSTR          ; Print resulting conversion
       LD   A,(3810H)      ; Get keyboard memory location
       BIT   0,A           ; Test for zero key pressed
       JR    Z,NOSTP       ; Not pressed, then skip
       CALL  GCFR          ; Wait till next character entered
       CP    '.'           ; Is it a dot?
       JR    NZ,NOSTP      ; No then, carry on
       CALL  PNEWL         ; else print a new line
       POP   BC            ; and restore all the registers
       POP   HL            ; and the stack level

*4 - 22*
POP
JP
POP
POP
INC
CALL
DJNZ
CALL
POP
INC
LD
CP
JR
LD
CP
JR
CALL
JP
BC
QUESL
BC
QUESL
HL
QUESL
NL
QUESL
PSPAC
PNEWL
BC
QUESL
BC
QUESL
(B,COUNT)
C
QUESL
NZ,DOROW
A, (COUNT+1)
B
QUESL
NZ,DOROW
QUESL
Print a space between numbers
; Jump back to main loop
; Restore column count
; Restore memory pointer
; Increment memory pointer
; Do for six motors
; Print a new line
; Restore row count
; Increment row count
; Get lower count byte
; Is it the same
; No then do next row
; Get higher order count byte
; Same?
; No then do next row else
; print a new line and then
; back to main loop

*4 - 23*
SUBROUTINES INDEX

DOALL......................Execute a stored sequence once
DRIVL......................Drives all motors directed by TBUF
INIT......................Set up system
MOVTO......................Use FACAR to reset system arm
TORQUE....................Turn on off motors
CLRMT......................Turn off all motors
SETLT......................Reset CTPOS elements to one
DRAMT......................Drive directed motors
STEPM......................Step motors via DRAMT
DNED.............Delay on direction change
SPAMT......................Update TBUF array during learn
KEYIN......................Scan keyboard and build up motors to move
CBTAS......................Convert 16 bit 2's complement number to ASCII
CLRMP......................Clear MOTBF array
CTBUF......................Clear TBUF, DRBUF & MOTBF arrays
CINT......................Get 16 bit signed value from keyboard
POSDS......................Display relative position array elements
POSCI......................Increment relative position array elements
STORE......................Copy TBUF to current ARST slice
RESET......................Clear POSAR array
PUTCHR.....................Print a character
PSTR......................Print a string
PSPAC......................Print a space
PNEWL......................Print a carriage return
SECTION 3

SUBROUTINES.
SUBROUNDES INDEX (continued)

SCKBD......................Scan the keyboard
GCHMC......................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......................Delay approx 1.0 sec
SUBROUTINE DOALL

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

DOALL

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

*4 - 26*
SUBROUTINE DRIVL

; This routine is given TBUP, it then drives all
; the motors that need to be driven, till TBUP = 0

DRIVL   LD   C, 0
SCANW   LD   B, 6
         LD   HL, TBUP
TBZER   LD   A, (HL)
         JR   NZ, TBNZR
         INC   HL
         DJNZ   TBZER
         RET

TBNZR   LD   DE, MOTBF + 5
         LD   HL, TBUP + 5
         LD   B, 6
DOAGN   LD   A, (HL)
         CP   0
         JR   NZ, NOEL
         JP   M, SNEG
         LD   (DE), A
         DEC   (HL)
         JR   NOFIL
SPOS    LD   A, 3
         LD   (DE), A
         DEC   (HL)
         JR   NOFIL
SNEG    LD   A, 1
         LD   (DE), A
         INC   (HL)
         JR   NOFIL
NOEL    XOR   A
         LD   (DE), A
NOFIL   DEC   DE
         DEC   HL
         DJNZ   DOAGN
         LD   A, 1
         LD   (KEYP), A
         CALL   STEPM
         DEC   C
         JP   NZ, SCANW
         RET

; Set BC = motor count
; Point to TBUP
; Get step value from TBUP
; Is it zero?
; No then continue
; Point to next TBUP location
; Do next motor check
; If no motor to step, then return
; DB points to last direction array
; HL points to TBUP
; B = motor count
; Get motor step value
; Is it zero?
; Yes then skip
; Is it negative i.e., reverse
; No positive, so load MOTBF (N)
; With 3
; Decrement motor count in TBUP
; Complete the MOTBF array
; Set MOTBF = 1 for
; a positive drive
; Decrement negative count
; Do rest of MOTBF
; Clear MOTBF (N)
; Move to next MOTBF element
; Move to next TBUP element
; Do for all six motors
; Set key pressed flag
; Step all motors once if
; any to step
; Do for maximum of 128 cycles
; then return

*4 - 27*
SUBROUTINE INIT

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

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

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

<table>
<thead>
<tr>
<th>MOVTO</th>
<th>PUSH AF</th>
<th>; *</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PUSH BC</td>
<td>; *</td>
</tr>
<tr>
<td></td>
<td>PUSH LE</td>
<td>; *</td>
</tr>
<tr>
<td></td>
<td>PUSH HL</td>
<td>; *</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RES1</th>
<th>LD HL,POSAR</th>
<th>; HL points to POSAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LD B,12</td>
<td>; B = count of 12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NRES1</th>
<th>LD A,(HL)</th>
<th>; Get POSAR element</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JR A</td>
<td>; Is it zero?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JNZ</th>
<th>INC HL</th>
<th>; No then continue</th>
</tr>
</thead>
<tbody>
<tr>
<td>DJNZ</td>
<td>LD NRES1</td>
<td>; Point to next POSAR element</td>
</tr>
<tr>
<td></td>
<td>JR ENDSC</td>
<td>; See if all zero</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MTSAM</th>
<th>LD HL,POSAR+10</th>
<th>; HL points to POSAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LD DE,MOTBF+5</td>
<td>; DE points to MOTBF</td>
</tr>
<tr>
<td></td>
<td>LD B,6</td>
<td>; B = count</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RSCAN</th>
<th>PUSH BC</th>
<th>; Save count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LD C,(HL)</td>
<td>; Get lower byte</td>
</tr>
<tr>
<td>INC</td>
<td>HL</td>
<td>; Advance HL pointer</td>
</tr>
<tr>
<td>LD</td>
<td>B,(HL)</td>
<td>; Get high byte of POSAR element</td>
</tr>
<tr>
<td>OR</td>
<td>B</td>
<td>; Get low byte into A</td>
</tr>
<tr>
<td>JP</td>
<td>NZ,DOMPL</td>
<td>; skip if POSAR(N) is zero</td>
</tr>
<tr>
<td>DEC</td>
<td>HL</td>
<td>; advance POSAR pointer</td>
</tr>
<tr>
<td>JR</td>
<td>NMDR</td>
<td>; Do next motor</td>
</tr>
</tbody>
</table>

| DOMTL | 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 |

| RMT1  | DEC EC    | ; Dec count for reverse |
| LD    | A,3       | ; Set reverse in A |

| DOIT1 | LD (DE),A | ; Store reverse in MOTBF (N) |
| LD    | (HL),B    | ; Store updated POSAR count |
| DEC   | HL        | ; in POSAR (N) |
| LD    | (HL),C    | ; Store lower byte |

| NMDR  | DEC HL    | ; point to next POSAR element |
| DEC   | DE        | ; Move to next MOTBF element |
| POP   | BC        | ; Restore motor count |
| DJNZ  | RSCAN     | ; Do for next motor |
| CALL  | ORMAT     | ; Drive all motors to be driven |
| JR    | RES1      | ; Do till all POSAR slots zero |

| ENDS  | POP HL    | ; * Restore all registers |
|       | POP DE    | ; * Return |
| POP   | DC        | ; * |
| POP   | AF        | ; * |
| RET   |           | ; 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 CTPCS(1-6)
; SETDT sets all CTPOS elements to start offset
; position which equals 1.

TORQ1
LD A, (HL) ; Get motor value
OR A ; Is it zero?
JR NZ, TORQ2 ; No then skip
LD A, 1 ; Reset CTPOS(N) to position 1
LD (DE), A ; in FTABL
LD A, B ; Get motor address in A
SLA A ; Shift it left for interface defin
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 TCQCL ; 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

OTMT
LD D, $0FH ; 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
OUT (PORT), A ; Combine with coils off pattern
DJNZ CLMT ; Output to selected motor
CALL SETDT ; Do next motor

TCQCL
PCP HL ; *
PCP DE
PCP BC ; * Restore Registers
PCP AF ; *
RET ; Done, exit
SETDT
PUSH BC
PUSH DE
PUSH HL
LD B,6
LD HL, CTPOS
LD (HL), 1
INC HL
DJNZ NSET1
POP HL
POP DE
POP BC
RET

; * Set CTPOS elements to start
; * Save used registers
; Motor count to B
; HL points to CTPOS array
; Set CTPOS(N) to start position = 1
; Increment HL
; Do set up next CTPOS element
; * Restore used registers

*4 - 30*
SUBROUTINE DRAMT

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

DRAMT
PUSH AF
PUSH BC
PUSH DE
PUSH HL
/*
LD B,6 ; B = motor count
LD DE,MOTBF +5 ; Point to MOTBF array
LD HL,CTPOS ; HL points to PTABL offset array
NMTDT
LD A,(DE) ; Get MOTBF(N)
OR A ; Is it zero?
JR Z,IGMTN ; If zero, then skip
BIT 1,A ; Test direction
CALL OUTAM ; Step motor
JR Z,REVMT ; If direction negative then jump
INC A ; Increment table counter
CP 5 ; Upper bound?
JR C,NORST ; No then continue
LD A,1 ; Reset table offset
NORST
LD (HL),A ; Store in CTPOS (N)
IGMTN
INC HL ; Increment CTPOS pointer
DEC DE ; Decrement MOTBF pointer
DJNZ NMTDT ; Do for next motor
CALL DELT ; Delay after all pulses out
CALL DELS ; *
POP HL ; *
POP DE ; *
POP EC ; * Restore Registers
POP AF ; *
RET ; Exit

REVMT
DEC A ; Move table pointer on
CP 1 ; Compare with lower bound
JR NC,NORST ; If no overflow then continue
LD A,4 ; Reset table offset
JR NORST ; Do next motor
OUTAM
LD A,(HL) ; Get table offset 1-4
PUSH AF
/*
PUSH DE ; * Save Registers
PUSH HL ; *
LD HL,PTABL-1 ; Get table start
LD D,0
LD E,A ; DE now equals 1-4
ADD HL,DE ; Add to PTABL -1 to get address
LD A,(HL) ; Get motor pulse pattern
LD C,B ; Get address field in C and
SLA C ; shift it one to the left
OR C ; or in the pulse pattern
OUT (PGRT),A ; Output to interface circuitry
POP HL ; *
POP DE ; * Restore Registers
POP AF ; *
RET ; Return
SUBROUTINE STEPM

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

STEPM       PUSH  AF           ; * Save Register
            PUSH  HL
            PUSH  BC
            LD    HL,MOTBP         ; HL points to motor buffer
            LD    E,6             ; B = Count
TRYØ        LD    A,(HL)       ; Get motor value 3 or 1
            OR    A              ; Zero?
            JR    NZ,CONTA      ; No then continue
CONT         INC   HL           ; Point to next motor
            DJNZ  TRYØ           ; Do next motor
            POP   BC
            POP   HL
            POP   AF
            RET               ; Exit
CONTA        POP   BC           ; * Restore registers
            POP   HL
            CALL  DRANT         ; Drive motors
            CALL  POSIC        ; Increment relative position
            POP   AF            ; * Restore AF
            RET               ; Exit
SUBROUTINE DNEWD
: This subroutine checks to see if any motors are changing direction, if so a delay is inserted into the sequence.

DNEWD
PUSH AF ; *
PUSH BC ; *
PUSH DE ; * save used registers
PUSH HL ; *
LD BC, 6 ; Load BC with count
OR A ; Clear carry
SBC HL, BC ; NC 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 E, C ;
NCOMP LD A, (HL) ; Get contents of this row
CP Œ ; See if positive or negative
LD A, (DE) ; Get identical previous motor slot
JP P, PDIR ; if positive do for positive motor
NDIR CP Œ ; Compare if both in same
JP M, NXTCK ; direction then skip else
CDDEL CALL DELLN ; delay end
NCDSG POP HL ; *
POP DE ; *
POP BC ; * Restore registers
POP AF ; *
RET ; Now return
PDIR CP Œ ; 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 slot
; boundary of -128 to 127.

SRAMT
LD A,(MAN) ; Get manual flag
OR A ; Is it zero?
JP NZ,STEPM ; Yes then just step motors
LD (STRFG),A ; Clear the store flag
LD B,6 ; B = motor count
LD 1X,DRBUP+6 ; 1X = previous direction buffer
LD 1Y,MOTBP+5 ; 1Y = current buffer
LD HL,TBUF +5 ; HL = step buffer

NMTOT
DEC 1Y ;
DEC 1X ;
DEC HL ; move pointers
LD A,(1Y+Ø) ; Get current motor direction
OR A ; No work to do
JR Z,NORDV ; 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 NORDV ; Do next motor

CFORD
INC (HL) ; Increment motor step in TBUF
LD A,(HL) ; Get new value
CP 0 ; Check against upper board
CALL SETST ; Limit reached then store flag
LD (1X+Ø),3 ; Set previous direction
JR NZ,CREQ ; No then continue

NORDV
DJNZ NMTOT ; Do next motor
CALL STEPM ; Step motors to be driven
LD A,(STRFG) ; Examine store flag
OR A ; Zero?
JP NZ,STORE ; No then do store operation
RET ; Exit

REVDR
LD A,(1X+Ø) ; Get previous direction
CP 3 ; Direction reversed?
JR NZ,CREVI ; No then continue
CALL SETST ; Else set store TBUF in ARST flag
JR NORDV ; Do next motor

CREVI
DEC (HL) ; Advance step count in TBUF (N)
LD A,(HL) ; Get element
CP -128 ; Compare with upper negative bound
CALL Z,SETST ; Limit reached so set store flag
JR NORDV ; Do next motor

CREVD
LD (1X+Ø),1 ; Set Direction
JR NORDV ; Do next motor

SETST
PUSH AF ; Save AF
LD A,1 ; Set store flag STRFG
POP AF ; Restore AF
RET ; Continue
SUBROUTINE KEYIN

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

KEYIN  CALL  CLRKF    ; Clear MOTBF array
LD      A,(3840H) ; Get TR580 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,(3510H);
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
LC      A,(3810H) ; Restore memory value
JR      Z,TRYN1   ; No then skip
LD      A,(MAN)   ; See if in manual mode
CR       A         ;
CALL    2,STORE   ; No then store TBUF
OR      1          ; Set not finished flag
RET       ; and exit to caller

TRYN1   LD      BC,0 ; Clear MOTBF offset in BC
BIT     1,A       ; See if '1' key is pressed
JP      Z,TRYN2   ; No then skip else
CALL    FORMT     ; Set up motor 1 position in MOTBF
TRYN2   INC       BC ; Increment MOTBF offset
BIT     2,A       ; See if '2' key pressed
JP      Z,TRYN3   ; No skip
CALL    FORMT     ; Set second motor forward
TRYN3   INC       BC ; Advance offset
BIT     3,A       ;
JP      Z,TRYN4   ; See if '3' key pressed, No skip
CALL    FORMT     ; Set forward direction on Motor 3
TRYN4   INC       BC ; Increment offset in BC
BIT     4,A       ; See if key '4' is pressed
JP      Z,TRYN5   ; No then test key '5'
CALL    FORMT     ; Do forward direction for Motor 4
TRYN5   INC       BC ; Advance offset
BIT     5,A       ; Key '5' pressed
JP      Z,TRYN6   ; No skip
CALL    FORMT     ; Do set up for motor 5
TRYN6   INC       BC ; Advance offset
BIT     6,A       ; Key '6' pressed
JP      Z,TRYQT   ; No then try 'Q'
CALL    FORMT     ; Do for motor 6
| TRYQT | LD    | BC, 0  | ; Clear BC offset for motor 1 |
| TRYQ  | LD    | A, (3804H) | ; See if 'Q' key pressed |
|       | BIT   | 1, A   | ; No then skip |
|       | JP    | 2, TRYQ | ; Set motor 1 for backward |
|       | CALL  | BACMT   | ; Advance pointer |
|       | INC   | BC    | ; See if 'W' key pressed |
| TRYW  | BIT   | 7, A   | ; No skip |
|       | JP    | 7, TRYR | ; Do backward for motor 2 |
|       | CALL  | BACMT   | ; Advance pointer offset |
| TRYE  | INC   | BC    | ; See if |
|       | LD    | A, (3801H) | ; 'E' key pressed |
|       | BIT   | 5, A   | ; No skip |
|       | JR    | 7, TRYR | ; Set motor 3 for backward |
|       | CALL  | BACMT   | ; Advance pointer offset |
| TRYR  | INC   | BC    | ; See if |
|       | LD    | A, (3804H) | ; Key 'R' is pressed |
|       | BIT   | 2, A   | ; No skip |
|       | JP    | TRYT   | ; Set motor 4 backward |
|       | CALL  | BACMT   | ; Advance offset |
| TRYT  | INC   | BC    | ; Is key 'T' pressed? |
|       | BIT   | 4, A   | ; No skip |
|       | JP    | 2, TRYY | ; Set motor 5 backward |
|       | CALL  | BACMT   | ; Is the 'Y' key pressed? |
| TRYY  | INC   | BC    | ; Advance offset |
|       | BIT   | 1, A   | ; No key |
|       | JP    | 2, SOMEN | ; 'Y' then skip |
| SOMEN | CALL  | BACMT   | ; Set motor 6 for backward |
|       | CALL  | SRANT   | ; Step motors, maybe store. |
|       | OR    | 1      | ; Set zero key not pressed flag |
| NOTNG | RET   |        | ; Return to caller |
|       | LD    | A, (MAN) | ; Zero was pressed so see |
|       | OR    | A      | ; if in learn mode |
|       | CALL  | 2, STORE | ; Yes then store |
|       | XOR   | A      | ; Set zero flag and |
|       | RET   |        | ; Return to caller |
| FORMT | LD    | E, 3   | ; Set for forward direction |
|       | JP    | SETMT   | ; Do set motor slot in MOTBF |
| BACMT | LD    | E, 1   | ; Set for reverse direction |
| SETMT | LD    | HL, MOTBF | ; Point to MOTBF |
|       | ADD   | HL, BC  | ; Add in motor offset |
|       | PUSH  | AF     | ; Save AF |
|       | LD    | A, (HL) | ; Get byte |
|       | OR    | A      | ; See if zero |
|       | JR    | 2, DOMOT | ; Yes then set byte |
| DOMOT | XOR   | A      | ; Clear |
|       | LD    | (HL), A | ; byte in MOTBF user wants both |
|       | POP   | AF     | ; directions clear byte |
|       | RET   |        | ; Restore AF and return |
|       | LD    | (HL), E | ; Set byte in MOTBF |
|       | LD    | A, 1   | ; and set |
|       | LD    | (KEYP), A | ; key pressed flag |
|       | POP   | AF     | ; Restore AF |
|       | RET   |        | ; exit from routine |

\*4 - 36\*
SUBROUTINE CB TAS

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

CB TAS

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

* - 37*
BTOAT

DEFW 10000 ; Table of subtraction constants
DEFW 1000 ; for conversion routine
DEFW 100
DEFW 10
DEFW 1
CLEARING AND resetting routines

; CLRMP clears the MOTBF array

CLRMP  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 MOTBF
;   Note all must be in order

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

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

GINT
  PUSH BC
  PUSH DE
  XOR A
  SBC HL, HL
  LD B, 5
  LD (MIN), A

GINT1
  CALL GCHRA
  CP SPAC
  JR Z, GINT1
  CP ML.
  JP Z, PRET1
  CP MINUS
  JR NZ, POSON
  LD A, 1
  LD (MIN), A
  JR GINT2

POSON
  CP '+'
  JR NZ, NUM1

GINT2
  CALL GCHRA
  CP NL.
  JR Z, NUMET
  ADD HL, HL
  ADD PL, HL
  ADD HL, HL
  CP $0
  CP '9' + 1
  JR NC, ERRN2
  CP '9' + 1
  JR NC, ERRN2
  SUB NUMBA
  LD E, A
  LD D, $0
  ADD HL, DE
  DJNZ GINT2
  CALL PNEWL

NUMET
  LD A, (MIN)
  OR A
  JR Z, PRET1
  LD A, L
  CPL
  LD L, A
  LD A, H

* 4 - 40 *
PRET1
CPL
LD H,A
INC HL
XOR A
FCP DE
POP BC
RET

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

; Clear A and flags
; * Restore Registers
; *
; and return
; 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  BC        ; *
       PUSH  DE        ; * Save all registers
       PUSH  HL        ; *
LD      HL,POSSST  ; Print "FELPOS=
CALL    PSTR      ; String
LD      B,6        ; Motor count into B
LD      DE,POSSAR  ; Point to array containing offsets
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      LX,NUMAR   ; LX points to result string
CALL    CBTAS      ; Convert HL and leave in (1X)
LD      HL,NUMAR   ; Point to result string
CALL    PSTR      ; Print it
CALL    PSPACE     ; Print a space
DJNZ    NPSCA      ; Do for next motor
CALL    PSEL      ; Print a new line, all done
POP     HL         ; *
POP     DE         ; *
POP     BC         ; * Restore all Registers
POP     AF         ; *
RET         ; Now return
SUBROUTINE PGSIC

; PGSIC 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 360 times to cause such an event.

PUSH AF ; *
PUSH BC ; *
PUSH DE ; * Save registers
PUSH HL ; *
LD B,6 B = motor count
LD DE,MOTBF+5 Point to MOTBF
LD HL,POFAR+1Q Point to POSAR (relative position)

NPOS1
PUSH BC ; Save motor count
LD C,(HL) Get lower POSAR byte in C
INC HL Point to Higher byte
LD B,(HL) Get higher byte in B
LD A,(DE) Get direction byte from MOTBF
AND 3 Clear all higher bits from D7-D3
OR A Is it zero?
JR NZ,NNZM No skip
DEC HL Yes then move POSAR pointer back
JR NPOS2 and continue with next motor

NNZM
BIT 1,A Test direction bit
JR NZ,RDPOS Do for reverse direction
INC BC Advance element
JR STPOS Restore 16 bit POSAR element

RDPOS
DEC BC Advance negative POSAR element
INC HL Store higher byte
DEC HL Move pointer to lower byte
LD (HL),C Store lower byte

STPOS
DEC HL Back up POSAR pointer to
DEC HL next motor position slot
DEC DE Backup MOTBF pointer to next slot
POP BC Restore Motor count
DJNZ NPOS1 Do next motor
POP HL ; *
POP DE ; * Restore used Registers
POP BC ; *
POP AF ; *
RET ; Done, Exit

*4 = 43*
STORE
PUSH BC
PUSH HL
LD HL,TBUF ;* Save registers
LD B,6 ; Point to TBUF
OR A ; B = motor count
INC HL ; Get TBUF (N)
JR NZ,STOR1 ; Is TBUF element zero
INC HL ; No then do store
DJNZ STEST ; Point to next element
JR EXIT ; Go do next element check
STOR1
LD (1X+0),0 ; All TBUF zero so exit
LD HL,(COUNT) ; Clear DRBUF element
INC HL ; Get current count value
LD A,0 ; Advance it
CP L ; See if over or at 512 bytes
JP NC,OVRFW ; Yes then overflow
LD (COUNT),HL ; Put back advanced count
LD DE,(CUROW) ; Get current row pointer in DE
LD HL,TBUF ; Get TBUF pointer in HL
LD BC,0666 ; Count for six motors
DIR (CUROW),LE ; Copy TBUF to ARST(1)
LD CALL CTBUF ; Replace updated row pointer CUROW
CALL CTBUF ; Clear buffers
EXIT
POP HL ; * Restore Registers
POP BC ; Now return to caller
RET
OVRFW
LD HL,(AFMS) ; Print overflow situation
CALL PSTR ; Message
CALL GCRA ; Get response
CALL PNSNL ; 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 JNLT ; Clear all arrays etc
POP HL ; * Restore Registers
POP BC ; Back to main loop
POP BC ; Throw away return address
JP QUES1 ; Back to main loop
SUBROUTINE RESET

: This subroutine clears the POSAR array

RESET
PUSH BC ; * Save Registers.
PUSH DE ; *
PUSH HL ; *
LD HL,PO SAR ; Point to POSAR start
LD DE,PO SAR+1 ; Point to next element
LD (HL),ØØ ; 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 FCCH ; 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 TF880
    CALL PUTSTR ; Print the string
    FCPO DE ; * Restore Registers
    FCPO BC ;
    RET ; Done, Exit

; PSPAC prints a space character

PSPAC
    PUSH AF ; Save AF
    LD A, 20 ; A = Space character
    CALL PUTCHR ; Print it
    POP AF ; Restore AF
    RET ; Done, Exit

; PNEWL prints a new line to the screen

PNEWL
    PUSH AF ; Save AF
    LD A, 0DH ; A = Newline character
    CALL PUTCHR ; Print it
    POP AF ; Restore AF
    RET ; Done, Exit

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

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

; GCHR gets a character from keyboard and displays it

GCHR
    CALL GCHR ; Get a character
    CALL PUTCHR ; Print it
    RET ; Done, Exit
CLEAR SCREEN ROUTINE

; Simple scrolling type screen clear

CLRSC  PUSH  BC      ; Save used register
        LD    B,16    ; Get screen row count
UPLRW  CALL  PNEWL   ; Print a new line
        DJNZ  UPLRW   ; Do 16 times
        POP   BC      ; Restore Register
        RET      ; Exit
<table>
<thead>
<tr>
<th>Delay Routines</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELSW</td>
<td>Delay for 10 * 5 + 10 M cycles</td>
</tr>
<tr>
<td>DELSL</td>
<td>Save BC</td>
</tr>
<tr>
<td>NOP</td>
<td>Delay for 11 T state</td>
</tr>
<tr>
<td>NOP</td>
<td>4 T state delay</td>
</tr>
<tr>
<td>POP</td>
<td>4 T state delay</td>
</tr>
<tr>
<td>DJNZ</td>
<td>Delay for 11 T states</td>
</tr>
<tr>
<td>DELSL</td>
<td>Do delay times value in B</td>
</tr>
<tr>
<td>POP</td>
<td>Restore BC</td>
</tr>
<tr>
<td>RET</td>
<td>Exit</td>
</tr>
<tr>
<td>DELS</td>
<td>Save BC</td>
</tr>
<tr>
<td>LD</td>
<td>Set B for 0.001 sec delay (apx)</td>
</tr>
<tr>
<td>CALL</td>
<td>Do delay</td>
</tr>
<tr>
<td>POP</td>
<td>Restore BC</td>
</tr>
<tr>
<td>RET</td>
<td>Exit</td>
</tr>
<tr>
<td>DELT</td>
<td>Save BC</td>
</tr>
<tr>
<td>LD</td>
<td>Set B for 0.01 sec delay (apx)</td>
</tr>
<tr>
<td>CALL</td>
<td>Do delay</td>
</tr>
<tr>
<td>POP</td>
<td>Restore BC</td>
</tr>
<tr>
<td>RET</td>
<td>Exit</td>
</tr>
<tr>
<td>DELLN</td>
<td>Save BC</td>
</tr>
<tr>
<td>LD</td>
<td>Set B for 1.0 sec delay (apx)</td>
</tr>
<tr>
<td>CALL</td>
<td>Do delay</td>
</tr>
<tr>
<td>POP</td>
<td>Restore BC</td>
</tr>
<tr>
<td>RET</td>
<td>Exit</td>
</tr>
</tbody>
</table>

*4 - 43*
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>OA</th>
<th>OB</th>
<th>OC</th>
<th>OD</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ø</td>
<td>1</td>
<td>Ø</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Ø</td>
<td>Ø</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Ø</td>
<td>1</td>
<td>Ø</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Ø</td>
<td>1</td>
<td>1</td>
<td>Ø</td>
<td>4</td>
</tr>
</tbody>
</table>

**HALF STEPPING PULSE SEQUENCE**

<table>
<thead>
<tr>
<th>OA</th>
<th>OB</th>
<th>OC</th>
<th>OD</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ø</td>
<td>1</td>
<td>Ø</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>1.5</td>
</tr>
<tr>
<td>1</td>
<td>Ø</td>
<td>Ø</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>1</td>
<td>2.5</td>
</tr>
<tr>
<td>Ø</td>
<td>1</td>
<td>Ø</td>
<td>1</td>
<td>3.5</td>
</tr>
<tr>
<td>Ø</td>
<td>1</td>
<td>1</td>
<td>Ø</td>
<td>4</td>
</tr>
<tr>
<td>Ø</td>
<td>Ø</td>
<td>1</td>
<td>Ø</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 CPT 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>DEFB</td>
<td>192</td>
<td>1</td>
</tr>
<tr>
<td>DEFB</td>
<td>144</td>
<td>2</td>
</tr>
<tr>
<td>DEFB</td>
<td>48</td>
<td>3</td>
</tr>
<tr>
<td>DEFB</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>DEFB</td>
<td>192</td>
<td>1</td>
</tr>
<tr>
<td>DEFB</td>
<td>128</td>
<td>1.5</td>
</tr>
<tr>
<td>DEFB</td>
<td>144</td>
<td>2</td>
</tr>
<tr>
<td>DEFB</td>
<td>16</td>
<td>2.5</td>
</tr>
<tr>
<td>DEFB</td>
<td>48</td>
<td>3</td>
</tr>
<tr>
<td>DEFB</td>
<td>32</td>
<td>3.5</td>
</tr>
<tr>
<td>DEFB</td>
<td>96</td>
<td>4</td>
</tr>
<tr>
<td>DEFB</td>
<td>64</td>
<td>4.5</td>
</tr>
</tbody>
</table>
If you compare the table values with the tables on the previous page you will note a difference, this is because QB and QC are exchanged in the above table due to the hardware switching these two lines.

NOTE

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

D8 = QA
D7 = QC
D6 = QB
D5 = QD
CONSTRUCTION OF A SUITABLE PORT FOR THE ARMDROID

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

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

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

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

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

The components for the described port should be easily available from most sources.
TRS80 8 BIT INTERFACE (NON LATCHED BI-DIRECTIONAL)

READ OR WRITE FROM PORT (4)

TRS80 BUS
(SEE BUS DESCRIPTION)

IC 1: 74LS27  Pin 14: 5 Volts, Pin 7: GND
IC 2: 74LS20  Pin 14: 5 Volts, Pin 7: GND
IC 3: 74LS30  Pin 14: 5 Volts, Pin 7: GND
IC 4: 74LS245  Pin 20: 5 Volts, Pin 10: GND

5 Volts Supply from Robot Connector + GND

Colne Robotics

A.J. Lennard 20/6/1981

0 Volts
5 Volts

To
Robot 8 Bit
Interface

* 4 - 52x *
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>ARMROID 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 ARM DROID 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**

\[
7.5 \times \frac{14 \text{ teeth}}{72 \text{ teeth}} \times \frac{12 \text{ teeth}}{108 \text{ teeth}} = 0.162 \text{ degree per step or 6.17284 steps per degree}
\]

**Elbow**

Same as shoulder joint

**Wrists**

Same as base joint calculations

**Hand**

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

\[
\frac{d \times \theta \times 0.231}{360} = (0.0524/2) \text{ mm}
\]

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

Total hand open to close pulley movement = 20.0 mm

Angle traversed by single finger = 50°

\[
\frac{50^\circ}{20.0 \text{ mm}} \times 0.0262 \text{ mm} = 0.0265 \text{ per step or 15.2672 step per degree}
\]

\[
f = 3.1415926
\]

\[
d = 26 \text{ mm = pulley diameter}
\]
SOME OVERALL DIMENSIONS

Shoulder pivot to pivot = 190mm
Forearm pivot to pivot = 190mm
Finger wrist pivot to fingers closed = 90mm
   wrist pivot to finger open (90) = 99mm
Bottom of base to shoulder pivot = 238mm

ANGULAR JOINT SPANS

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

(All above measurements are in degrees)

NOTE

The above measurements were taken with the arm joints held in a horizontal plane.
SOME EXTRA POINTS TO BEAR IN MIND

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

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

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

FINAL NOTE
BEST WISHES AND GOOD LUCK

[Signature]
METAL WASHERS (FOR SPACING) INDIVIDUAL THICKNESS = 1