ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Aerotech Unidex U14R-2-A-40* at our website: *Click HERE*

QI QUERY INTERRUPT STATUS

The QI command returns the same information for all axes when in the AA mode, as the QA command does in the single axis mode. The 4 character fields for each axis are separated by commas and the string is surrounded by carriage returns and line feeds.

QUEUE REQUIREMENTS

MODE	
AX-AT	Not valid
AA,AM	Immediate
AA/CD	Not valid

Syntax: AA QI
will return LF CRCR PNNN, PNNN, PNNN,PNNN LFCRCR

SECTION 8-12: COMMANDS FOR USER UNITS

The following commands allow specification of move parameters in user defined units. The controls will automatically convert all move parameters to these units once they have been initialized.

UU# USER UNITS

The UU command converts all move velocities, distances, etc. to user specified units by multiplying by the specified parameter. The multiplier parameter can be either an integer or floating point number. This command must be given in the single axis mode but will remain effective in the AA mode. The Unidex 14 defaults to user units off at power up or reset.

NOTE: The command UU-1; will reverse the programmed direction.

QUEUE REQUIREMENTS

MODE	
AX-AT	Immediate
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX UU1000

UF USER OFF

The UF command turns off the user units. This command is equivalent to and preferred over UU1 since it turns off the mode thus minimizing unnecessary overhead.

QUEUE REQUIREMENTS

MODE	
AX-AT	Immediate
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX UF

SECTION 8-13: POSITION MAINTENANCE COMMANDS

ER#,# ENCODER RATIO

The ER command allows specification of encoder ratio by entering encoder counts followed by motor counts for position maintenance mode. This ratio must be equal to one i.e. encoder counts must match motor counts when slip detection is enabled. All distance, velocity and acceleration parameters are input in encoder counts when this mode is enabled. The correct number of motor counts are generated, while this mode is enabled the user need only be concerned with encoder counts. This mode can be combined with user units allowing units such as inches or revolutions to be specified in encoder counts. All parameters are then input in the user units which have been defined. The ratio defaults to 1 on power up or reset.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	1
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX ER2000, 4000;

HV# HOLD VELOCITY

The HV command specifies maximum position hold correction velocity. This is the peak velocity which will be used while making position corrections.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX HV2000;

HG# HOLD GAIN

The HG allows the user to specify the position hold gain parameter. The gain parameter is multiplied by the position error in determining the velocity during correction. The parameter must be between 1 and 32,000. The parameter should be set experimentally by increasing it until the system is unstable then reducing it slightly below the threshold of stability.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX HG30;

NOTE: **Do not use this command if a DC Brush or Aerodrive system is being used for a given axis.**

HD# HOLD DEADBAND

The HD command specifies deadband counts for position hold. If the motor is within this limit, it is considered in position and no further correction will be made. This parameter interacts with the HG command, i.e. a larger deadband will allow a larger gain parameter in many applications. A parameter of zero is allowed.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

(See HN for an example)

NOTE: Do not use this command if a DC Brush or Aerodrive system is being used for a given axis.

HF HOLD OFF

The HF command disables position hold, stall detection and tracking modes. This is the default mode at power up or reset.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX HF

HN HOLD ON

The HN command enables position correction after a move and activates the HV, HG and HD commands. Hold and slip detection are disabled if an LP, SA, ST or KL command is entered.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Example: The following commands could be used to set up the position correction mode. This sequence sets up a move velocity of 100,000 steps per second and an acceleration of 500,000 steps per second. The position correction velocity is set for 50,000 steps per second, a dead band of 10 steps and correction gain of 2,000. The correction is then enabled. A 200,000 step move is performed, then that position is maintained within the 10 step deadband until commanded to a new position.

Enter:
```
AX VL100000 AC500000
HV50000 HD10 HG2000 HN
MR200000 GO
```

NOTE: Do not use this command if a DC Brush or Aerodrive system is being used for a given axis.

SECTION 8-14: SLIP AND STALL DETECTION COMMANDS

ES# ENCODER SLIP TOLERANCE

The ES command parameter specifies tolerance before slip or stall is flagged in the status register. The mode must be turned on with an IS command and off with an HF command.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX ES3;IS

NOTE: The Encoder Ratio must be equal to zero.

IS INTERRUPT ON SLIP

The IS command enables the Unidex 14 to interrupt the host upon a slip or stall detection if the appropriate bit has been set in the interrupt control register and the Encoder ratio is set to one.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	1
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX IS

SECTION 8-15: ENCODER TRACKING COMMANDS

ET ENCODER TRACKING

The ET command energizes the tracking mode. The axis will track it's encoder input thus allowing one axis to follow the activity of another. No acceleration or deceleration ramps are generated. The axis will duplicate the encoder input. The ER command allows the user to scale the motor's movements relative to the encoder.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX ET

HF HOLD OFF

The HF command disables position hold, stall detection and tracking modes.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX HF

SECTION 8-16: ENCODER STATUS REQUEST COMMANDS

EA ENCODER STATUS

The EA command returns encoder status of currently addressed axis in the following format:

EA COMMAND RESPONSE DESCRIPTION

CHAR	SENT		DESCRIPTION
1	LF		Line feed
2	CR		Carriage return
3	CR		Carriage return
4	E	Either Character	Slip detection enabled
	D		Slip detection disabled
5	E	Either Character	Position maintenance enabled
	D		Position maintenance disabled
6	S	Either Character	Slip or stall detected (reset by execution of EA command)
	N		No slip or stall detected
7	P	Either Character	Position Maintenance within deadband
	N		Position not within deadband
8	H	Either Character	Axis is home
	N		Axis is not home
9	N		Unused/reserved
10	LF		Line feed
11	CR		Carriage return
12	CR		Carriage return

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	Immediate
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX EA

The following may be returned:

LFCRCR DDNNNN LFCRCR

RE REQUEST ENCODER POSITION

The RE command returns current encoder position of the addressed axis, in encoder counts. The ASCII string is surrounded by carriage returns. The position is not scaled by User Units.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX RE

The following may be returned:

LFCR -45678 LFCR

SECTION 8-17: VELOCITY STAIRCASE COMMANDS

The following commands describe the velocity staircase mode. This mode is useful in applications requiring a change in velocity at a prescribed position without stopping.

MP MOVE POSITIVE

The MP command sets the direction logic to move in the positive direction.

QUEUE REQUIREMENTS	
MODE	
AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX MP MO

MM MOVE MINUS

The MM command sets the direction logic to move in the negative direction.

QUEUE REQUIREMENTS	
MODE	
AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX MM MO

MV#,# MOVE VELOCITY

The MV command causes the motor to run to the new ABSOLUTE position (parameter 1) at the new velocity (parameter 2). When the destination is reached control, will be passed to the next command which should be another MV command or an SP command. If the command is not received in time the controller will continue to move at the specified velocity. Note that this is a slave mode and it is the responsibility of the user to provide the commands in time. They may be queued ahead of time.

If a new MV command is sent after the controller has already passed the destination specified in the command, the controller will continue to move at the old velocity. Any number of steps can be specified in this manner with both acceleration and deceleration. The controller will not reverse direction if the position has already passed, but will behave as explained above. Thus the direction of the move must be specified before starting the move with the MP or MM commands above. All destinations must be in absolute position, no position relative moves are allowed due to the nature of these commands.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	4	5
AA,AM	Not valid	
AA/CD	Not valid	

Example Generate a velocity staircase with the breakpoints given in absolute position.

Enter:
```
MP
MV10000,30000
MV20000,50000
MV30000,10000
SP35000
```

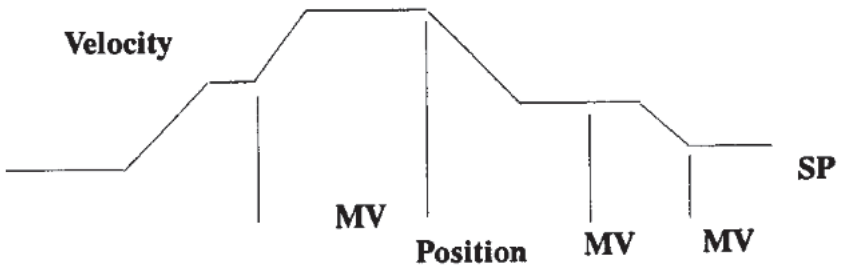


Figure 8-1: Velocity Staircase Profile

SP# STOP AT POSITION

The SP command will cause the axis to stop at the specified position. The controller will attempt to stop at the specified destination. If there is insufficient distance to stop at the previously specified deceleration when the command is received, the controller will stop as soon as possible at that deceleration. This command is not compatible with the JG command and will probably overshoot the desired position if used with it.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	3	4
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX MR5000; GO SP3000;

FP# FORCE POSITION

The FP command will flush the command queue and attempt to stop at the specified position. The axis will overshoot if there is insufficient distance left to stop at the programmed acceleration. This command is not compatible with the JG command and will probably overshoot the desired position if used with it.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Flush +4	Flush +4
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX MR5000; GO FP3000;

SECTION 8-18: CONSTANT VELOCITY CONTOURING

The following commands are available in the Unidex 14 for the constant velocity contouring option:

CD#,#; CONTOUR DEFINE

The CD command enters contour definition mode. It allows entry of commands for contouring mode. Commands are queued for execution by the CX command. The parameters define the axes for which the contour is defined and the starting position of the contour in absolute units. The contour may be defined on all 4 axes if circular interpolation is not used or 2 axes with circular mixed with linear interpolation. Attempting to do circular interpolation in a contour which is being defined for more than 2 axes will be flagged as a command error. This command is executed in the AA mode. The contouring axes must be at positions which allow them to reach the specified contouring velocity by the specified position when the contour is executed. If the actual position of the stage is equal to the starting position as defined by the CD command, the stage will jump to the contouring velocity with no ramp up. This could cause the stage to stall if it is not able to accelerate at this high rate. It is recommended that some ramp up distance be allowed. The distance required may be calculated from the equations in Section 1.

QUEUE REQUIREMENTS

MODE	
AX-AT	Not valid
AA,AM	0
AA/CD	Not valid

Example: The following demonstrates cutting a hole with a 10,000 count radius using constant velocity contouring and circular interpolation. The contouring velocity is set to 1000 pulses per second. A contour is then defined beginning at coordinates 0,0 on the Z and T axes. The auxiliary output of the Y axis is turned on, which could turn on the cutting torch or laser starting the cut at the center of the circle. A half circle is cut from the center to the outside of the hold positioning the cutting tool at the start of the desired hole. The hole is then cut, the torch turned off, the stage stopped and the definition is complete. The stage is then positioned and the hole cut with the CX command. **Note that no commas are provided in the MT and CR commands for the inactive X and Y axes within the contour definition.** The AN and AF commands must have commas for all axes since they can all be addressed from within the contour definition.

Enter:

```
CV1000 CD,,0,0;
AN,0; CR0,5000,3.1415926
CR0,0,6.2831853
AF,0; MT-10,10000
CE
MT,,-1000,0; GO CX
```

CE CONTOUR END

End of the contour sequence, terminate the CD mode or ramp to a stop and exit to AA command mode when executed. The end of the contour should contain at least a short linear segment just prior to the CE command to initialize the parameters for the deceleration of the stage.

QUEUE REQUIREMENTS

MODE	
AX-AT	Not valid
AA,AM	Not valid
AA/CD	1

(See CD command for an example)

CK CONTOUR END AND KILL

The CK command will end the contour with no ramp down, i.e. the pulses will stop abruptly. This command should be used with caution to prevent the stage from missing steps or loosing it's correct position.

QUEUE REQUIREMENTS

MODE	
AX-AS	Not valid
AA,AM	Not valid
AA/CD	1

Enter:
```
CV1000 CD,,0,0;
AN,0; CR0,5000,3.1415926
CR0,0,6.2831853
AF,0; MT-10,10000
CK
MT,,-1000,0; GO CX
```

CR#,#,# CIRCULAR INTERPOLATION

Move in a circular pattern from the entry position. The first two parameters are the center of the circle in absolute units and the third parameter is the distance to move in radians. The distance parameter should be supplied to seven significant digits if a full circle is to be generated.

QUEUE REQUIREMENTS

MODE	
AX-AT	Not valid
AA,AM	Not valid
AA/CD	8

(See CD command for an example)

CV# CONTOUR VELOCITY

Allows specification of contouring velocity. This command is executed from the AA mode before contour definition. A contour defined by a CD command can not be executed if followed by a CV command. The contour velocity defaults to 1000 on power up or reset.

QUEUE REQUIREMENTS

MODE	
AX-AT	Not valid
AA,AM	Immediate
AA/CD	Not valid

(See CD command for an example)

CX CONTOUR EXECUTE

Execute the previously entered contour sequence. The stage must be positioned such that it can accelerate to speed by the absolute position specified by the CD command it is executing and must be traveling in the proper direction. Once a contour is defined it may be executed at any time by executing a CX command until it is replaced by another contour definition. The CX command can not be placed within a loop or while construct.

QUEUE REQUIREMENTS

MODE	
AX-AT	Not valid
AA,AM	6
AA/CD	Not valid

The Unidex 14 will attempt to generate any profile which it is asked to do. It is the responsibility of the host to be sure the acceleration required when generating a circle or any other change in direction is possible within the mechanical constraints of the system. All corners must be defined by arcs and tangents to those arcs else the change in direction will be instantaneous and generate very large accelerations. The arc radius must be chosen so that the acceleration constraints of the system are met.

Note: **A new contour may be defined, but cannot be executed until the previous contour has finished execution.**

(See CD command for an example)

SECTION 8-19: COMMAND SUMMARY

The following commands are included in the Unidex 14 family of motor controllers. The '#' indicates a signed integer input parameter or a signed fixed point number of the format ##.# when user units are enabled. With User Units enabled, distances, velocity and acceleration parameters may be input in inches, revolutions, etc.

SUMMARY OF COMMANDS

PAGE NO.	COMMAND	COMMAND DESCRIPTION
8-3	AA	Any following commands are for the AA (All Axes) mode
8-16	AC#	Acceleration, set acceleration//deceleration register
8-13	AF	Auxiliary off
8-4	AM	Axis multitasking mode
8-12	AN	Auxiliary on
8-5	AT	Any following commands are for the T axis
8-5	AX	Any following commands are for the X axis (default on reset)
8-5	AY	Any following commands are for the Y axis
8-5	AZ	Any following commands are for the Z axis
8-14	BH#	Set selected I/O bit high (off)
8-14	BL#	Set selected I/O bit low (on)
8-15	BX	Return bit status in hex format
8-55	CD#,#;	Define a contour
8-57	CE	Contour End, exits to AA command mode
8-57	CK	Contour End, immediate pulse stop
8-9	CN	Cosine on, enables cosine velocity profiles
8-58	CR#,#,#	Circular interpolation, move in a circle
8-58	CV#	Contouring velocity, definition
8-30	CW	Clear while flag, i.e. terminate WH/WG loop
8-59	CX	Contour execute
8-50	EA	Encoder status, return encoder status of currently addressed axis
8-7	EF	Echo off, turn off echo to host (default on power on)
8-6	EN	Echo on, turn on echo to host

SUMMARY OF COMMANDS

PAGE NO.	COMMAND	COMMAND DESCRIPTION
8-44	ER#,#	Encoder ratio, sets encoder count to motor count ratio
8-48	ES#	Encoder slip tolerance, sets tolerance before slip or stall is flagged
8-49	ET	Encoder tracking, sets encoder tracking mode
8-54	FP#	Force position, will flush queue and attempt to stop at specified position
8-24	GD	Go and reset done flags
8-23	GO	Go command, starts execution of motion
8-46	HD#	Hold deadband, specifies dead band tolerance for position hold
8-46/8-49	HF	Hold off, disable position hold, slip detection and tracking modes
8-45	HG#	Hold gain, specifies position hold gain parameter
8-7	HH	Home high, home switches are active high
8-8	HL	Home low, home switches are active low
8-31	HM#	Home, find home and initialize the position counter
8-47	HN	Hold on, enables position correction after move
8-32	HR#	Home reverse, find home in reverse direction and initialize position counter
8-45	HV#	Hold velocity, specifies maximum position hold correction velocity
8-37	IC	Interrupt clear, clear done interrupt status and error flags
8-34	ID	Interrupt host when done and set done flag
8-35	II	Interrupt independent
8-36	IN#	Interrupt when nearly done
8-48	IS	Interrupt slip, interrupts host on slip or stall detection
8-24	JG#	Jog command, motor will run at specified velocity until a new velocity command is sent or it is stopped by a stop or kill command

SUMMARY OF COMMANDS

PAGE NO.	COMMAND	COMMAND DESCRIPTION
8-26	KL	Kill; flush queue and terminate pulse generation immediately on all axes without decelerating
8-32	KM	Home and kill pulse generation
8-33	KR	Home in reverse and kill pulse generation
8-28	LE	Loop end, terminates most recent LS command
8-8	LF	Disable limit switches for selected axis
8-9	LN	Enable limit switches for selected axis
8-10	LP#	Load position, loads position counter with parameter
8-27	LS#	Loop start, set loop counter, from 1 to 32000 loops; (may be nested to 4 levels)
8-20	MA#	Move absolute, move to absolute position
8-21	ML#,#;	Move linear, move specified distance relative from current position
8-52	MM	Move minus, sets minus direction for MV type move
8-22	MO	Move one pulse in current direction
8-52	MP	Move plus, sets positive direction for MF type move
8-20	MR#	Move relative, move specified distance from current position
8-22	MT#,#;	Move to, move to specified position
8-53	MV#,#	Move velocity, move to first parameter (absolute position) at second parameter velocity without stopping at end of move
8-13	PA	Power automatic, turn power on before each move and off after the move
8-10	PF	Parabolic off, disable parabolic ramps, i.e. linear ramps will be generated
8-10	PN	Parabolic on, enable parabolic ramps
8-42	QA	Query status of switches and flags for addressed axis without affecting flags
8-43	QI	Query status of switches and flags on all axes without affecting flags
8-41	RA	Return status of switches and flags and reset flags

SUMMARY OF COMMANDS

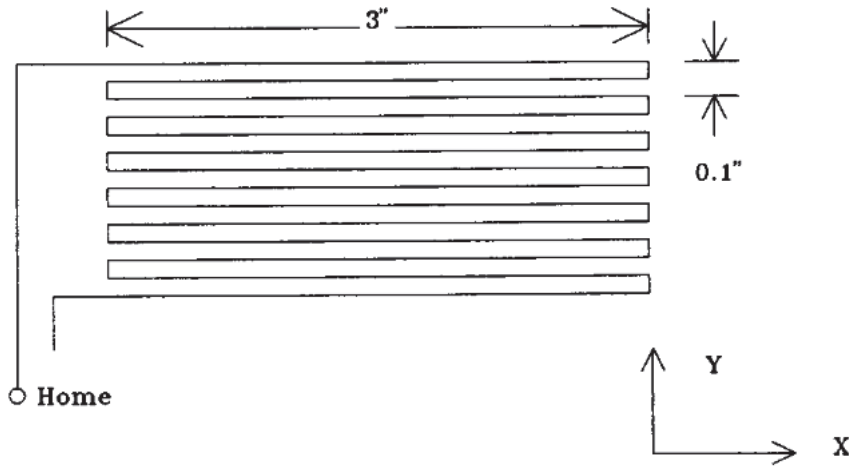
PAGE NO.	COMMAND	COMMAND DESCRIPTION
8-51	RE	Request encoder position, returns current encoder position
8-42	RI	Return status of switches and flags for all axes and reset flags
8-23	RM#	Return remainder of position divided by parameter in position counter
8-39	RP	Request position, returns current position
8-40	RQ	Request queue status, return number of queue entries available
8-11	RS	Software reset of Unidex 14
8-26	SA	Stop all, flushes queue and stops all axes with deceleration
8-54	SP#	Stop at position, will stop at specified position if possible after all commands have been executed
8-25	ST	Stop, flush queue and decelerate to stop
8-38	SW#	Sync wait, wait for the specified inputo be HI.
8-44	UF	User units off, turn off user unit translation
8-43	UU#	User units, multiply acceleration, velocity and distance parameters by specified parameter
8-18	VB#	Base velocity, sets base velocity
8-17	VL#	Sets maximum velocity to be used in profile
8-25	VS#,#,#	Velocity stream, slave velocity mode for profiling
8-29	WD	While end, WH loop terminator
8-30	WG	Terminates WH loop
8-29	WH	While, execute all commands until WG loop terminator until flag cleared by CW command
8-28	WS#	While sync, execute while sync is true (low)
8-38	WT#	Wait, wait for specified number of milliseconds
8-39	WY	Who are you, returns software revision

SECTION 8-20: APPLICATION PROGRAM EXAMPLES

The following program examples illustrate the versatility of the Unidex 14 programming language. These programs generate common X-Y contours that are found in many applications. The programs also show how easily repeat loops and I/O control can be incorporated into programs.

8-20-1: RASTER SCAN

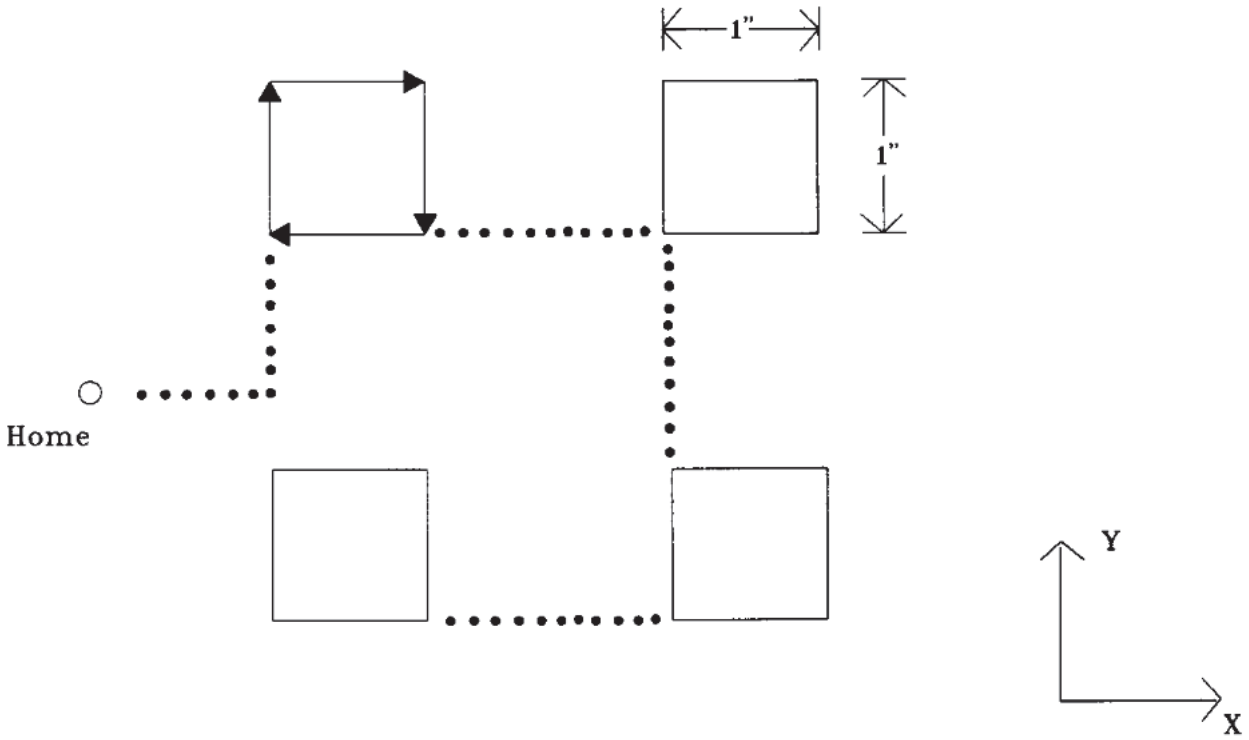
BL8	Bit 8 low (output OFF)
AX UU12700	Set X axis to inches
AC25 VL.1	Set accel to 25"/sec2 and velocity to 0.1"sec
LF HM0 MA0 GO	Turn off limits and go home
AY UU12700	Set Y axis to inches
AC25 VL.1	Set accel to 25"/sec2 and velocity to 0.1"sec
LF HM0 MA0 GO	Turn off limits and go home
AX LN	Turn ON X axis limits
AY LN	Turn ON Y axis limits
AA	All axes motion
VL2,6;	Set velocity to 2"/sec for X and 6"/sec for Y
BH8	Bit 8 high (output ON)
MR,1.5; GO	Move Y axis 1.5"
MR0.1; GO	Move X axis 0.1"
LS7	Loop start 7 times
MR3; GO	Move X axis 3"
MR,-0.1; GO	Move Y axis -0.1"
MR-3; GO	Move X axis -3"
MR,-0.1;GO	Move Y axis -0.1"
LE	Repeat loop end
BL8	Bit low (output OFF)



8-20-2: REPEATING PATTERN AND PROGRAMMED OUTPUTS

BL8	Bit 8 set low (output OFF)
AX UU12700	Set X axis to inches
AC25 VL.1	Set accel to 25"/sec2 and velocity to 0.1"sec
LF HM0 MA0 GO	Turn off limits and go home
AY UU12700	Set Y axis to inches
AC25 VL.1	Set accel to 25"/sec2 and velocity to 0.1"sec
LF HM0 MA0 GO	Turn off limits and go home
AX LN	Turn ON X axis limits
AY LN	Turn ON Y axis limits
AA	All axes motion
VL5,5;	Set velocity to 5"/sec for X and Y axis
MR0.5; GO	Move X axis 0.5"
MR,0.5; GO	Move Y axis 0.5"
BH8	Bit 8 high (output ON)
MR,1; GO	Move Y axis 1"
MR1; GO	Move X axis 1"
MR,-1; GO	Move Y
MR-1; GO	Move X axis -1"
BL8;	Bit 8 low (output OFF)

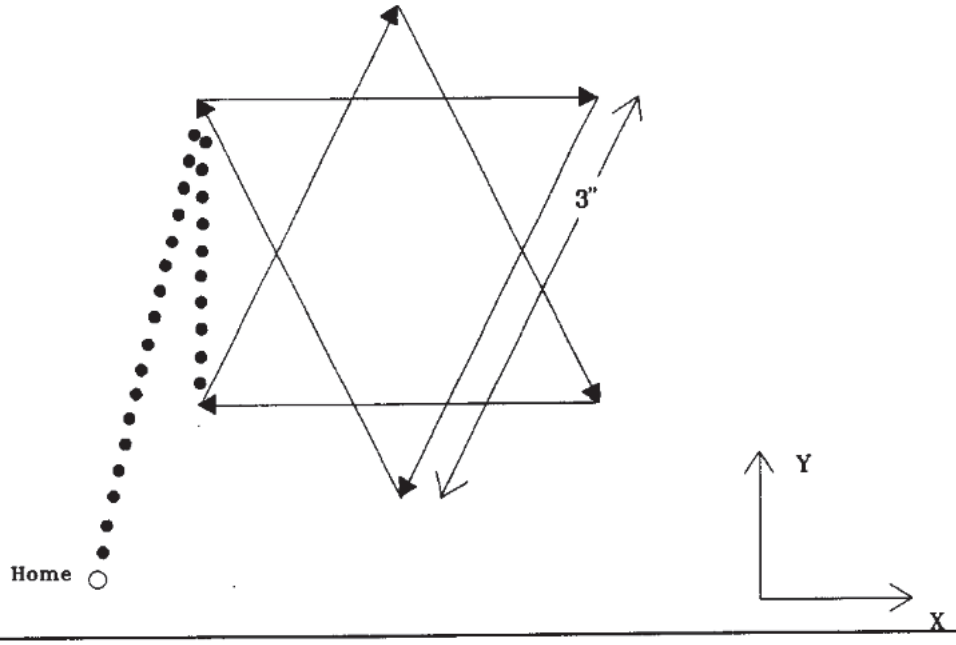
MR2; GO BH8 MR,1; GO MR1; GO MR,-1; GO MR-1; GO BL8	Move X axis 2" (Repeat square)
MR,-2; GO BH8 MR,1; GO MR1; GO MR,-1; GO MR-1; GO BL8	Move Y axis -2" (Repeat square)
MR-2; GO BH8 MR,1; GO MR1: GO MR,-1; GO MR-1; GO BL8	Move X axis -2" (Repeat square)



REPEATING PATTERN

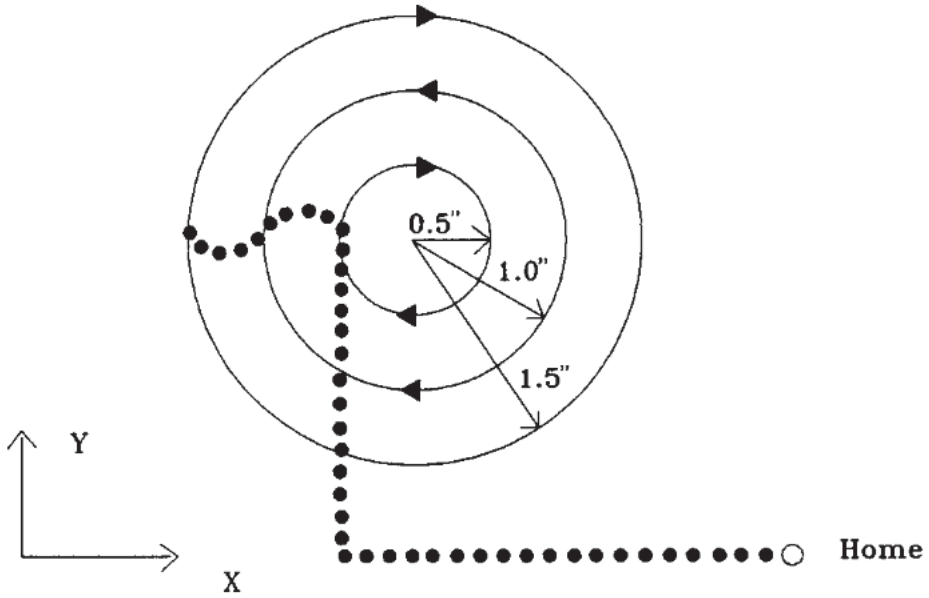
8-20-3: LINEAR INTERPOLATION - STAR

BL8	Bit 8 set low (output OFF)
AX UU12700	Set X axis to inches
AC25 VL.1	Set accel to 25"/sec2 and velocity to 0.1"sec
LF HM0 MA0 GO	Turn off limits and go home
AY UU12700	Set Y axis to inches
AC25 VL.1	Set accel to 25"/sec2 and velocity to 0.1"sec
LF HM0 MA0 GO	Turn off limits and go home
AX LN	Turn ON X axis limits
AY LN	Turn ON Y axis limits
AA	All axes motion
VL4.9,4.9	Set X and Y velocity to 4.9"/sec.
ML,5,3; GO	Interpolate X axis 0.5" and Y axis 3" at 5"
BH8	Bit high (output ON)
AC25,25; VL5;	Reset X,Y accel and X velocity
ML3; GO	Move X axis 3"
VL4.3,4.3	Set X and Y velocity to 4.3"/sec.
ML-1.5,-2.6; GO	Interpolate X axis -1.5" and Y axis -2.6" at 5"/sec.
BL8	Bit 8 low (output OFF)
AC25,25; VL,5;	Reset X,Y accel and velocity
ML,-1.75; GO	Move Y axis -1.75"
BH8	Bit 8 high (output ON)
VL4.3,4.3;	Set X and Y velocity to 4.3"/sec.
ML1.5,2.6; GO	Interpolate X axis 1.5" and Y axis 2.6" at 5"/sec.
AC25,25; VL4.3,4.3;	Reset X,Y accel and velocity
ML1.5,-2.6; GO	Interpolate X axis 1.5" and Y axis -2.6" at 5"/sec.
AC25; VL5;	Reset X axis accel and velocity
ML-3; GO	Move X axis -3"
BL8	Bit 8 low (output OFF)



8-20-4: THREE CONCENTRIC CIRCLES

AA AF0;	X axis aux output OFF
AX UU12700	Set X axis to inches
AC25 VL.1	Set accel to 25"/sec2 and velocity to 0.1"sec
LF HM0 MA0 GO	Turn off limits and go home
AY UU12700	Set Y axis to inches
AC25 VL.1	Set accel to 25"/sec2 and velocity to 0.1"sec
LF HM0 MA0 GO	Turn off limits and go home
AX LN	Turn ON X axis limits
AY LN	Turn ON Y axis limits
AA	All axes motion
VL5,5;	Set X,Y velocity to 5"/sec.
CV2	Set contouring velocity to 2"/sec.
CD-1.5,2;	Begin contour at position -1.5,2
AN0;	X axis aux output ON
CR-1,2,-6.2831853	Clockwise circle centered at position -1,2
AF0;	X axis aux output OFF
CR-1.75,2,3.1415926	Counterclockwise semicircle centered at -1.75,2
AN0;	X axis aux output ON
CR-1,2,6.2831853	Counterclockwise circle centered at -1,2
AF0;	X axis aux output OFF
CR-2.25,2,-3.14155826	Clockwise semicirclr centered at -2.25,2
AN0;	X axis aux output ON
CR-1,2,-6.2831853	Clockwise circle centered at -1,2
AF0;	X axis aux output OFF
CE	End contour
MA-1.5; GO	Move X axis -1.5"
MA,1.9; GO	Move Y axis 1.9"
CX	Execute contour
MT,1.2; GO	Move Y axis 2.1"



CHAPTER 9: HOST SOFTWARE

SECTION 9-1: DEMONSTRATION SOFTWARE

The following is a print-out of the programs available at the time that this manual was printed that are provided on the Support Disk. Read the provided disk for the most current programs.

A software disk is supplied with the initial purchase of a Unidex 14 controller to aid the user in developing their application program. It is supplied on a PC compatible floppy disk in both executable and source form. Install the U14C in your computer, install the disk in drive A: and type A:U14_POLL. The program will ask for a file name. Press enter and the program will announce it is in "Interactive mode". You are then communicating with the U14C board and can execute moves by typing them at the keyboard. Enter the following commands:

EN WY

This will turn on the echo from Unidex 14 so you can see what you are typing and should respond with the current firmware revision number so that you know that communication is established with the U14C board.

You are welcome to use the subroutines in this software for your application. No license is required. These programs are usable with all the PC/XT/AT compatible motion control products.

NOTE: **Software support Doc. —— fault routine provides fault indication from all servo axis (not available on steppers) thru a limit and output line, which also indicates a cw or ccw limit has been encountered, Unidex 14's status register may be polled to test for a limit condition, which would invalidate the fault input, unless it occurred simultaneously with a limit.**

The following routines were written specifically for programming the Unidex 14. These routines perform the same function in both BASIC and C languages.

Home -	Sends all axes to the Home Limit Switch and then moves it out to the first Marker.
Init Sys -	Resets Unidex 14 and sets defaults
In Pos -	Waits until all drives are in position
Send File -	Reads a file containing commands and then sends them to the Unidex 14
Write Chr -	Sends a character to the Unidex 14, if there is room in the buffer, if there is no room, will wait
Check Status -	Checks to see if the Unidex 14 is flagging an error or has executed a id
Fault -	Returns true "1" if a fault condition exists

Write String -	Checks to see if the Unidex 14 has room for a character, if not, waits until there is room and then sends the character to Unidex 14
In Bit -	Returns the status of the user inputs
Remote -	Controls the Remote Clock and Direction inputs to the Unidex 14 Power Chassis (L is PC control, R is Clock and Direction)
Read Chr -	Waits for a character from Unidex 14
Read Data -	Checks to see if Unidex 14 has a character to send, if the Unidex 14 does it is returned
Read LF -	Reads characters until a CR LF is received then begins to input data until a CR LF is received
Read String -	Wait and input a specified number of characters from the Unidex 14

SECTION 9-2: README.DOC

Below is a description of the programs found on this disk.

U14_BAS. BAS	46508	--- Microsoft QuickBasic Version 4.50
JOG. C	29939	--- Microsoft C 5.1 prog. for cursor control of motors
U14_POLL. C	11960	--- MS QuickC 2.5 program using polling method
U14_INTR. C	16443	--- Microsoft C 5.1 program using interrupts
U14_CDMA. EXE	33430	--- DOS executable version of U14_CDMA.C
U14_BAS. EXE	80988	--- DOS executable version of U14_BAS.BAS
U14_PAS. PAS	4934	--- Microsoft QuickPascal
U14_POLL. EXE	27763	--- DOS executable version of U14_POLL.C
U14_CDMA. C	17418	--- Microsoft C Version 5.1 for DMA example
README. DOC	3531	--- This file
JOG. EXE	59778	--- DOS executable version of JOG.C

SECTION 9-3: BASIC DEMONSTRATION PROGRAM

```
DECLARE FUNCTION Limit$ (cmd!)
DECLARE FUNCTION Valid$ (char$)
DECLARE SUB Filter (outletter%)
DECLARE SUB LoadYCal ()
DECLARE SUB LoadZCal ()
DECLARE SUB LoadTCal ()
DECLARE FUNCTION Trans.Y$ (mode$, NewPos!)
DECLARE FUNCTION Trans.Z$ (mode$, NewPos!)
DECLARE FUNCTION Trans.T$ (mode$, NewPos!)
DECLARE FUNCTION Trans.X$ (mode$, NewPos!)
DECLARE SUB LoadXCal ()
DECLARE SUB Home ()              ' Unidex 14 Ver. 1.12
DECLARE SUB InitSys ()           ' 9/9/91
DECLARE SUB InPos (move$)
DECLARE SUB GetPosition ()
DECLARE SUB SendFile (filename$)
DECLARE SUB WriteChr (outletter%)
DECLARE SUB CheckStatus ()
DECLARE FUNCTION Fault% ()
DECLARE SUB WriteString (cmd$)
DECLARE FUNCTION InBit! ()
DECLARE SUB Remote (mode$)
DECLARE FUNCTION Power16& (power%)
DECLARE FUNCTION ReadChr% ()
DECLARE FUNCTION ReadData% ()
DECLARE FUNCTION ReadLF$ ()
DECLARE FUNCTION ReadString$ (count%)

' define Unidex 14 register I/O addresses, & bits of status register.

CONST datareg = 768, donereg = 769, controlreg = 770, statusreg = 771
CONST donebit = 16, ibfbit = 32, tbebit = 64, irqbit = 128
CONST cmderrbit = 1, initbit = 2, encbit = 4, ovrtbit = 8
CONST cr = 13, LF = 10           ' Carraige Return, Line Feed

DEFINT A-Z
COMMON SHARED status%, sendmode
COMMON SHARED fileerror ' this will be set to a 1 in "handler -
    ' if the named file cannot be opened
COMMON SHARED AbsPosX AS LONG, AbsPosY AS LONG, AbsPosZ AS LONG, AbsPosT AS LONG
COMMON SHARED LastAbsX AS SINGLE, LastAbsY AS LONG, LastAbsZ AS LONG, LastAbsT AS LONG
COMMON SHARED LastErrX AS INTEGER, LastErrY AS INTEGER, LastErrZ AS INTEGER, LastErrT
AS INTEGER
```

```
COMMON SHARED CalSizX AS LONG, CalSizY AS LONG, CalSizZ AS LONG, CalSizT AS LONG
COMMON SHARED XUsrU!, YUsrU!, ZUsrU!, TUsrU!      ' user units
COMMON SHARED Xratio&, Yratio&, Zratio&, Tratio&   ' encoder ratio
DIM SHARED XErr(1000) AS INTEGER, YErr(1000) AS INTEGER, ZErr(1000) AS INTEGER, TErr(1000) AS INTE-
GER
COMMON SHARED name$

  XUsrU! = 1: YUsrU! = 1   ' reset all user units to 1
  ZUsrU! = 1: TUsrU! = 1
  Xratio& = 1: Yratio& = 1 ' reset encoder ratio to 1
  Zratio& = 1: Tratio& = 1
  fileerror = 0

ON ERROR GOTO handler

  COLOR 14, 1: CLS
  OUT controlreg, 0          ' disable Unidex 14 interrupts
  InitSys                    ' initialize system

  INPUT " Name of a Unidex 14 command file to download ( for no file): ", name$
  PRINT CHR$(13); " Terminal emulation mode active, press CTRL C to exit"
  SendFile name$
  keystroke = 0
  sendmode = 1               ' display U14 output mode
  WHILE (keystroke < > 3)          ' 3 is a Control-C
    key$ = INKEY$
    IF key$ < > "" THEN            ' if a key was pressed
      keystroke = ASC(key$)
      WriteChr (keystroke)   ' send key value to Unidex 14
    END IF

      result = ReadData
      IF result > 0 THEN     ' get Unidex 14's output if there is any
      IF (result < > 0) AND (result < > 3) AND (result < > 13) THEN
        ' don't print null's or ^C or carriage returns
        PRINT CHR$(result);
      END IF
      END IF
      CheckStatus
  WEND
  END

handler:   'this routine handles run time errors
```

```
  SELECT CASE ERR
    CASE 52 'no name given or illegal name'
      fileerror = 1
      RESUME NEXT
    CASE 53
      PRINT "This program could not find file: "; name$
      fileerror = 1
      RESUME NEXT
    CASE ELSE
      'else display error message and attempt to resume program
      PRINT "Error # "; ERR: RESUME NEXT
  END SELECT  ' end of handler routine

DEFSNG A-Z
SUB CheckStatus

' Check to see if Unidex 14 is flagging an error or has executed an id

IF (status% AND donebit)  < > 0 THEN
  IF (status% AND cmderrbit)  < > 0 THEN
    PRINT
    PRINT "Unidex 14 Status Register indicates a command error"

  ELSEIF (status% AND encbit)  < > 0 THEN
    PRINT
    PRINT "Unidex 14 Status Register indicates an Encoder Slip error"

  ELSEIF (status% AND ovrtbit) < >  0 THEN
    PRINT
    PRINT "Unidex 14 Status Register indicates a limit switch is closed"

  ELSE          ' an ID has been encountered, print the done axis
    doneflag = INP(donereg)
    IF doneflag  < > 0 THEN
      PRINT
      PRINT "Unidex 14 Status Register indicates a DONE flag has been set."
      PRINT "AXES DONE is/are: ";
      IF doneflag MOD 2 = 1 THEN PRINT "x ";
      doneflag = doneflag \ 2
      IF doneflag MOD 2 = 1 THEN PRINT "y ";
      doneflag = doneflag \ 2
      IF doneflag MOD 2 = 1 THEN PRINT "z ";
      doneflag = doneflag \ 2
      IF doneflag MOD 2 = 1 THEN PRINT "t ";
      PRINT
```

```
PRINT "Enter Cntrl-Y or IC command to clear Done Flags"
  END IF
 END IF
 status% = INP(statusreg)
 WHILE ((status% AND tbebit) = 0)    ' wait till Unidex 14 is ready
 status% = INP(statusreg)
 WEND
 OUT datareg, 25       ' send CTRL-Y to reset status register

END IF
END SUB ' checkstatus

DEFINT A-Z
FUNCTION Fault

' This function returns true "1" if a fault condition exists, on any servo
'  axis, stepper drives do not have a fault output to Unidex 14.

 inputs& = InBit                ' read the inputs
 IF ((inputs& AND 64) = 0) THEN
  Fault = 0                 ' no fault condition
  ELSE
  Fault = 1                 ' fault condition exists
  END IF

END FUNCTION

DEFSNG A-Z
SUB GetPosition

' This sub-program returns the absolute position from Unidex 14.

SHARED AbsPosX AS LONG, AbsPosY AS LONG, AbsPosZ AS LONG, AbsPosT AS LONG

 WriteString ("AzRP")          ' request z position from Unidex 14
 AbsPosZ = VAL(ReadLF$)          ' input z position

 WriteString ("AtRP")          ' request t position from Unidex 14
 AbsPosT = VAL(ReadLF$)          ' input t position

 WriteString ("AyRP")          ' request y position from Unidex 14
 AbsPosY = VAL(ReadLF$)          ' input y position

 WriteString ("AxRP")          ' request x position from Unidex 14
 AbsPosX = VAL(ReadLF$)          ' input x position
```

```
' Translate Unidex 14's absolute positions to real world positions.
  IF CalSizX = 0 THEN GOTO gskp1        ' is axis cal. present ?
  IF AbsPosX = 0 THEN GOTO gskp1        ' only neg. # calibrated
  APos = 0 - AbsPosX                    ' make it positive
  Ap! = APos / CalSizX                  ' find index into table
  FPos = Ap! - INT(Ap!)                 ' find fractional cal. step
  Ap! = INT(Ap!)
  AbsPosX = AbsPosX + FIX(FPos * (XErr(Ap! + 1) - XErr(Ap!)) + XErr(Ap!) + .5)

gskp1: IF CalSizY = 0 THEN GOTO gskp2   ' is axis cal. present ?
  IF AbsPosY = 0 THEN GOTO gskp2        ' only neg. # calibrated
  APos = 0 - AbsPosY                    ' make it positive
  Ap! = APos / CalSizY                  ' find index into table
  FPos = Ap! - INT(Ap!)                 ' find fractional cal. step
  Ap! = INT(Ap!)
  AbsPosY = AbsPosY + FIX(FPos * (YErr(Ap! + 1) - YErr(Ap!)) + YErr(Ap!) + .5)

gskp2: IF CalSizZ = 0 THEN GOTO gskp3   ' is axis cal. present ?
  IF AbsPosZ = 0 THEN GOTO gskp3        ' only neg. # calibrated
  APos = 0 - AbsPosZ                    ' make it positive
  Ap! = APos / CalSizZ                  ' find index into table
  FPos = Ap! - INT(Ap!)                 ' find fractional cal. step
  Ap! = INT(Ap!)
  AbsPosZ = AbsPosZ + FIX(FPos * (ZErr(Ap! + 1) - ZErr(Ap!)) + ZErr(Ap!) + .5)

gskp3: IF CalSizT = 0 THEN GOTO gskp4   ' is axis cal. present ?
  IF AbsPosT = 0 THEN GOTO gskp4        ' only neg. # calibrated
  APos = 0 - AbsPosT                    ' make it positive
  Ap! = APos / CalSizT                  ' find index into table
  FPos = Ap! - INT(Ap!)                 ' find fractional cal. step
  Ap! = INT(Ap!)
  AbsPosT = AbsPosT + FIX(FPos * (TErr(Ap! + 1) - TErr(Ap!)) + TErr(Ap!) + .5)

gskp4:

END SUB

SUB Home

' This sub-program homes all the axis by the hardware homing circuit.

' the next two lines home to limit switch at pre-programmed velocity, (VL cmd)
  cmd$ = "axlfhlhm;aylfhlhm;azlfhlhm;atlfhlhm;": GOSUB wrt
  cmd$ = "axhhhr;ln;ayhhhr;ln;azhhhr;ln;athhhr;ln;": GOSUB wrt
  FOR X = 1 TO 3000: NEXT X
```

```
' the next line homes the axis to the marker (absolute reference)
  cmd$ = "aasaBL13;WT9;BH13;": GOSUB wrt  ' initiate hardware home cycle
  PRINT CHR$(13); "Homing !"
  cmd$ = "AX": GOSUB wrt
  FOR X = 1 TO 3000: NEXT X          ' wait for home to end

  cmd$ = "ATLP0;": GOSUB wrt         ' set t axis position to zero
  LastErrT = 0: LastAbsT = 0         ' reset t cal. variables
  cmd$ = "AZLP0;": GOSUB wrt         ' set z axis position to zero
  LastErrZ = 0: LastAbsZ = 0         ' reset z cal. variables
  cmd$ = "AYLP0;": GOSUB wrt         ' set y axis position to zero
  LastErrY = 0: LastAbsY = 0         ' reset y cal. variables
  cmd$ = "AXLP0;": GOSUB wrt         ' set x axis position to zero
  LastErrX = 0: LastAbsX = 0         ' reset x cal. variables
  EXIT SUB

wrt:  FOR X% = 1 TO LEN(cmd$)            ' loop till all chr's are sent
    WHILE (INP(statusreg) AND tbebit = 0)  ' wait till Unidex 14 is ready
    WEND
    OUT datareg, ASC(MID$(cmd$, X%, 1))    ' send an ASCII chr. from string
  NEXT X%
  RETURN

END SUB

FUNCTION InBit

' This function returns the status of the user inputs.

  cmd$ = "AXBX": GOSUB wrti     ' ask for input bit states
  result$ = ReadLF$             ' input the hex values
  FOR X% = 0 TO 5       ' next line pre-scales the hex 'A-F' values
  IF (ASC(MID$(result$, X% + 1, 1)) > = 65) THEN MID$(result$, X% + 1, 1) = CHR$(65 - 7)
  accum = accum + (Power16(5 - X%) * (ASC(MID$(result$, X% + 1, 1)) - 48))' calc. base 10 value
  NEXT X%
  InBit = accum                 ' return dec. value of inputs
  EXIT FUNCTION

wrti:  FOR X% = 1 TO LEN(cmd$)           ' loop till all chr's are sent
    WHILE (INP(statusreg) AND tbebit = 0)  ' wait till Unidex 14 is ready
    WEND
    OUT datareg, ASC(MID$(cmd$, X%, 1))    ' send an ASCII chr. from string
  NEXT X%
  RETURN
```

```
END FUNCTION

SUB InitSys

SHARED cal AS INTEGER

' This sub-program resets Unidex 14 and sets all system defaults.

  WriteString ("RS")              ' reset to known state
  LoadXCal                        ' load X axis cal. table
  LoadYCal                        ' load Y axis cal. table
  LoadZCal                        ' load Z axis cal. table
  LoadTCal                        ' load T axis cal. table
  WriteString ("AXEF")            ' disable Unidex 14 echo
  WHILE (INP(statusreg) AND 32)   ' buffer not empty ?
    X = ReadData                  ' clr. it !
  WEND
  Remote ("L")                    ' disable rem. clk. & dir.
  WriteString ("WY")
  result$ = ReadLF$               ' get version #
  PRINT : PRINT " Unidex 14 Version # is = "; result$: PRINT
REM   WriteString ("AAPA1,1,1,1;")    ' high I. mode during motion
  WriteString ("AXBH13;")         ' reset home cycle pulse

END SUB

DEFINT A-Z
SUB InPos (move$)

' Wait for all servo axis & stepper drives with 1MR ramper bds to come to rest.
'  A "GO" command is expected within Move$ !

  WriteString (move$)             ' start the move
  PRINT "Waiting for drives to come to rest"
nloop: inputs& = InBit            ' read the inputs
    IF ((inputs& AND 128) = 0) THEN GOTO nloop  ' wait for move to start
innloop: inputs& = InBit          ' read the inputs
    IF ((inputs& AND 128) = 128) THEN GOTO innloop ' wait for move to end

END SUB

DEFSNG A-Z
FUNCTION Limit$ (cmd)

' Limit a cmd. string to the proper number of digits.
```

```
  temp$ = STR$(cmd)                ' make it a string
  dec.pt = INSTR(cmd$, ".")        ' find the decimal point
  IF dec.pt = 0 THEN
    Limit$ = RIGHT$(temp$, 10)     ' no dec. pt. return all
  ELSE
    Limit$ = LEFT$(temp$, 11)      ' return 11 char.s
  END IF

END FUNCTION

DEFINT A-Z
SUB LoadTCal

' Load T axis calibration table into memory.

  LastAbsT = 0                     ' initialize to start = 0
  name$ = "t_axis.cal"
  OPEN name$ FOR INPUT AS #1       ' open t cal. table
  WHILE (char$ < > CHR$(3))        ' ignore header before CTRL-C
    char$ = INPUT$(1, #1)          ' get a character from file
  WEND
  INPUT #1, ClrCrLf$               ' clear CR LF from file
  INPUT #1, CalSizT                ' get calibrated step size
  INPUT #1, ClrCrLf$               ' clear CR LF from file
  number = 0                       ' home pos = 0
  WHILE (NOT EOF(1))               ' load T error table till EOF
    INPUT #1, TErr(number)         ' input error
    INPUT #1, ClrCrLf$             ' clear CR LF from file
    number = number + 1            ' increment subscript
  WEND
  CLOSE #1                         ' close the file

END SUB

SUB LoadXCal

' Load X axis calibration table into memory.

  LastAbsX = 0                     ' initialize to start = 0
  name$ = "x_axis.cal"
  OPEN name$ FOR INPUT AS #1       ' open x cal. table
  WHILE (char$ < > CHR$(3))        ' ignore header before CTRL-C
    char$ = INPUT$(1, #1)          ' get a character from file
  WEND
  INPUT #1, ClrCrLf$               ' clear CR LF from file
```

```
  INPUT #1, CalSizX              ' get calibrated step size
  INPUT #1, ClrCrLf$             ' clear CR LF from file
  number = 0                   ' home pos = 0
  WHILE (NOT EOF(1))               ' load X error table till EOF
    INPUT #1, XErr(number)       ' input error
    INPUT #1, ClrCrLf$           ' clear CR LF from file
    number = number + 1          ' increment subscript
  WEND
  CLOSE #1                     ' close the file

END SUB

SUB LoadYCal

' Load Y axis calibration table into memory.

  LastAbsY = 0                 ' initialize to start = 0
  name$ = "y_axis.cal"
  OPEN name$ FOR INPUT AS #1         ' open y cal. table
  WHILE (char$ < > CHR$(3))          ' ignore header before CTRL-C
    char$ = INPUT$(1, #1)        ' get a character from file
  WEND
  INPUT #1, ClrCrLf$             ' clear CR LF from file
  INPUT #1, CalSizY              ' get calibrated step size
  INPUT #1, ClrCrLf$             ' clear CR LF from file
  number = 0                   ' home pos = 0
  WHILE (NOT EOF(1))               ' load Y error table till EOF
    INPUT #1, YErr(number)       ' input error
    INPUT #1, ClrCrLf$           ' clear CR LF from file
    number = number + 1          ' increment subscript
  WEND
  CLOSE #1                     ' close the file

END SUB

SUB LoadZCal

' Load Z axis calibration table into memory.

  LastAbsZ = 0                 ' initialize to start = 0
  name$ = "z_axis.cal"
  OPEN name$ FOR INPUT AS #1         ' open z cal. table
  WHILE (char$ < > CHR$(3))          ' ignore header before CTRL-C
    char$ = INPUT$(1, #1)        ' get a character from file
  WEND
  INPUT #1, ClrCrLf$             ' clear CR LF from file
```

```
  INPUT #1, CalSizZ              ' get calibrated step size
  INPUT #1, ClrCrLf$             ' clear CR LF from file
  number = 0                   ' home pos = 0
  WHILE (NOT EOF(1))               ' load Z error table till EOF
    INPUT #1, ZErr(number)       ' input error
    INPUT #1, ClrCrLf$         ' clear CR LF from file
    number = number + 1          ' increment subscript
  WEND
  CLOSE #1                     ' close the file

END SUB

FUNCTION Power16& (power)

' This is called by the InBit function only !
' This function returns 16 raised to the power of power.

  p& = 1
  FOR i% = 1 TO power
    p& = p& * 16
  NEXT i%
  Power16 = p&                 ' return result

END FUNCTION

DEFSNG A-Z
FUNCTION ReadChr%

' Wait for a character from Unidex 14.

rc1:   status% = INP(statusreg)
  IF ((status% AND ibfbit) = 0) THEN GOTO rc1    ' read status register
  X% = INP(datareg)                    ' get a chr.
  IF X% = 0 THEN GOTO rc1                  ' if null, get another
  ReadChr = X%                         ' return valid chr.

END FUNCTION ' ReadChr

FUNCTION ReadData%

' Check to see if Unidex 14 has a character to send, if it does, get it.

  status% = INP(statusreg)             ' read Unidex 14's status reg.
  IF (status% AND ibfbit) < > 0 THEN     ' if a chr. is available,
    ReadData = INP(datareg)      '  return it,
  ELSE                         ' else,
```

```
    ReadData = 0              ' return 0.
  END IF

END FUNCTION ' readdata

FUNCTION ReadLF$

' This function reads chr's till a CR is received then begins inputting
'     the data until a CR LF is received.

  WHILE (ReadChr  < > cr)                ' read till leading CR
  WEND
lp2:   temp = ReadChr                    ' input a chr.
  IF (temp = LF) THEN GOTO done          ' end of string ?
  result$ = result$ + CHR$(temp)         ' no; build string
  GOTO lp2                               ' wait for LF
done:  temp = ReadChr                    ' input trailing CR
  ReadLF$ = result$                      ' return string

END FUNCTION

FUNCTION ReadString$ (count%)

' Wait and input count% number of chr's from Unidex 14.

  FOR X% = 1 TO count%                   ' read till count% chr's
    result$ = result$ + CHR$(ReadChr)    ' build string
  NEXT X%
  ReadString = result$                   ' return result$

END FUNCTION

DEFINT A-Z
SUB Remote (mode$)

' This sub-program controls the remote clock & direction inputs to the
'     Unidex 14 Power chassis. (L is PC control, R is clock & dir.)

  IF UCASE$(mode$) = "R" THEN
    WriteString ("AXBL12;")        ' set remote mode
  ELSE
    WriteString ("AXBH12;")        ' set local mode
  END IF

END SUB
```

```
SUB SendFile (filename$)

' This sub-pgm. reads a file containing commands and sends them to Unidex 14.

  OPEN filename$ FOR INPUT AS #1
  IF fileerror = 0 THEN
    sendmode = 1              ' display U14 output mode
    DO UNTIL EOF(1)
      temp$ = INPUT$(1, #1)  ' get one character from file
      charval = ASC(temp$)   ' convert character to ASCII
      WriteChr (charval)     ' send the chr. to Unidex 14
      charval = ReadData     ' get chr.from Unidex 14's buffer
      IF (charval  < > 13 AND charval  < > 0) THEN PRINT CHR$(charval); ' display it !
    LOOP
  END IF
  CLOSE #1
  sendmode = 0                ' don't display U14 output

END SUB ' sendfile

DEFSNG A-Z
FUNCTION Trans.T$ (mode$, NewPos)

' Translate Absolute or Incremental T position to corrected position through
'  that axis lookup table within memory. Axes must be homed before initially
'  calling this function.

  IF CalSizT = 0 THEN             ' is calibration active ?
    temp$ = STR$(NewPos)          ' return same # as string
    GOTO xtt
  END IF
  mode$ = UCASE$(mode$)                                  ' convert to upper case
  Steps = NewPos * Tratio& * TUsrU!     ' convert to steps
  Orig = NewPos                 ' hold original move
  NewPos = 0 - Steps             ' make it positive

  IF mode$  < > "A" THEN NewPos = LastAbsT + NewPos - LastErrT ' convert to new absolute
ErrEt:  IF NewPos  < 0 THEN             ' if abs. pos. 0 then
    Trans.T = MID$(STR$(Orig), 2)  ' return same # as a string
    EXIT FUNCTION             ' exit
  END IF

  temp = NewPos / CalSizT          ' find index into error table
  pt = INT(temp)                ' find int. pointer
  IF pt  > 1000 THEN NewPos = -1: PRINT "T axis move out of range !": GOTO ErrEt
  FracStep = temp - pt           ' find fractional cal. step
```

```
  ' find corrected absolute position
  NewErr = FracStep * (TErr(pt + 1) - TErr(pt)) + TErr(pt)
  NewAbsT = NewPos + NewErr          ' find new abs. position
  IF mode$ < > "A" THEN              ' if incremental mode then
    IF TUsrU! = 1 THEN
      temp$ = STR$(0 - FIX(NewAbsT - LastAbsT / (Tratio& * TUsrU!) + .5))' convert to incremental
    ELSE
      temp$ = Limit(0 - ((NewAbsT - LastAbsT) / (Tratio& * TUsrU!))) ' convert to incr. int.
    END IF
  ELSE                               ' else if absolute mode then
    IF TUsrU! = 1 THEN
      temp$ = STR$(0 - FIX(NewAbsT / (Tratio& * TUsrU!) + .5))' NewAbsT is correction value
    ELSE
      temp$ = Limit(0 - (NewAbsT / (Tratio& * TUsrU!)))
    END IF
  END IF
  LastAbsT = NewAbsT                 ' remember LastAbsT position
  LastErrT = NewErr                  ' remember last error
xtt:  IF VAL(temp$) < > 0 THEN       ' do i need to remove leading space
    Trans.T = temp$                  ' no, return whole string
  ELSE                               ' else
    Trans.T = MID$(temp$, 2)         ' return without leading space
  END IF

END FUNCTION

FUNCTION Trans.X$ (mode$, NewPos)

' Translate Absolute or Incremental X position to corrected position through
' that axis lookup table within memory. Axes must be homed before initially
' calling this function.

  IF CalSizX = 0 THEN                ' is calibration active ?
    temp$ = STR$(NewPos)             ' no; return same # as string
    GOTO xtx
  END IF
  mode$ = UCASE$(mode$)              ' convert to upper case
  Steps = NewPos * Xratio& * XUsrU!  ' convert to steps
  Orig = NewPos                      ' hold original move
  NewPos = 0 - Steps                 ' make it positive

  IF mode$ < > "A" THEN NewPos = LastAbsX + NewPos - LastErrX ' convert to new absolute
ErrEx:  IF NewPos < 0 THEN           ' if abs. pos. 0 then
    Trans.X = MID$(STR$(Orig), 2)    ' return same # as a string
    EXIT FUNCTION                    ' exit
```

```
  END IF

  temp = NewPos / CalSizX          ' find index into error table
  px = INT(temp)                   ' find int. pointer
  IF px  > 1000 THEN NewPos = -1: PRINT " X axis move out of range !": GOTO ErrEx
  FracStep = temp - px             ' find fractional cal. step

  ' find corrected absolute position
  NewErr = FracStep * (XErr(px + 1) - XErr(px)) + XErr(px)
  NewAbsX = NewPos + NewErr        ' find new abs. position
  IF mode$ < > "A" THEN            ' if incremental mode then
    IF XUsrU! = 1 THEN
      temp$ = STR$(0 - FIX(NewAbsX - LastAbsX / (Xratio& * XUsrU!) + .5))' convert to incr.
    ELSE
      temp$ = Limit(0 - ((NewAbsX - LastAbsX) / (Xratio& * XUsrU!))) ' convert to incr. int.
    END IF
  ELSE                             ' else if absolute mode then
    IF XUsrU! = 1 THEN
      temp$ = STR$(0 - FIX(NewAbsX / (Xratio& * XUsrU!) + .5))' NewAbsX is correction value
    ELSE
      temp$ = Limit(0 - (NewAbsX / (Xratio& * XUsrU!)))' NewAbsX is correction value
    END IF
  END IF
  LastAbsX = NewAbsX               ' remember LastAbsX position
  LastErrX = NewErr                ' remember last error
xtx:  IF VAL(temp$) < 0 THEN       ' do i need to remove leading space
    Trans.X = temp$                ' no, return whole string
  ELSE                             ' else
    Trans.X = MID$(temp$, 2)       ' return without leading space
  END IF

END FUNCTION

FUNCTION Trans.Y$ (mode$, NewPos)

' Translate Absolute or Incremental Y position to corrected position through
' that axis lookup table within memory. Axes must be homed before initially
' calling this function.

  IF CalSizY = 0 THEN              ' is calibration active ?
    temp$ = STR$(NewPos)           ' return same # as string
    GOTO xty
  END IF
  mode$ = UCASE$(mode$)            ' convert to upper case
  Steps = NewPos * Yratio& * YUsrU!    ' convert to steps
  Orig = NewPos                    ' hold original move
```

```
  NewPos = 0 - Steps                ' make it positive

  IF mode$ < > "A" THEN NewPos = LastAbsY + NewPos - LastErrY ' convert to new absolute
ErrEy: IF NewPos < 0 THEN           ' if abs. pos. < 0 then
    Trans.Y = MID$(STR$(Orig), 2)   ' return same # as a string
    EXIT FUNCTION                   ' exit
  END IF

  temp = NewPos / CalSizY           ' find index into error table
  py = INT(temp)                    ' find int. pointer
  IF py > 1000 THEN NewPos = -1: PRINT "Y axis move out of range !": GOTO ErrEy
  FracStep = temp - py              ' find fractional cal. step

  ' find corrected absolute position
  NewErr = FracStep * (YErr(py + 1) - YErr(py)) + YErr(py)
  NewAbsY = NewPos + NewErr         ' find new abs. position
  IF mode$ < > "A" THEN             ' if incremental mode then
    IF YUsrU! = 1 THEN
      temp$ = STR$(0 - FIX(NewAbsY - LastAbsY / (Yratio& * YUsrU!) + .5))' convert to incremental
    ELSE
      temp$ = Limit(0 - ((NewAbsY - LastAbsY) / (Yratio& * YUsrU!))) ' convert to incr. int.
    END IF
  ELSE                              ' else if absolute mode then
    IF YUsrU! = 1 THEN
      temp$ = STR$(0 - FIX(NewAbsY / (Yratio& * YUsrU!) + .5))' NewAbsY is correction value
    ELSE
      temp$ = Limit(0 - (NewAbsY / (Yratio& * YUsrU!)))
    END IF
  END IF
  LastAbsY = NewAbsY                ' remember LastAbsY position
  LastErrY = NewErr                 ' remember last error
xty:  IF VAL(temp$) < 0 THEN        ' do i need to remove leading space
    Trans.Y = temp$                 ' no, return whole string
  ELSE                              ' else
    Trans.Y = MID$(temp$, 2)        ' return without leading space
  END IF

END FUNCTION

FUNCTION Trans.Z$ (mode$, NewPos)

' Translate Absolute or Incremental Z position to corrected position through
' that axis lookup table within memory. Axes must be homed before initially
' calling this function.

  IF CalSizZ = 0 THEN               ' is calibration active ?
```

```
    temp$ = STR$(NewPos)          ' return same # as string
    GOTO xtz
  END IF
  mode$ = UCASE$(mode$)              ' convert to upper case
  Steps = NewPos * Zratio& * ZUsrU!     ' convert to steps
  Orig = NewPos                   ' hold original move
  NewPos = 0 - Steps                ' make it positive

  IF mode$ < > "A" THEN NewPos = LastAbsZ + NewPos - LastErrZ ' convert to new absolute
ErrEz: IF NewPos < 0 THEN
    Trans.Z = MID$(STR$(NewPos), 2) ' return same # as string
    EXIT FUNCTION              ' exit
  END IF

  temp = NewPos / CalSizZ             ' find index into error table
  pz = INT(temp)                  ' find int. pointer
  IF pz > 1000 THEN NewPos = -1: PRINT "Z axis move out of range!": GOTO ErrEz
  FracStep = temp - pz              ' find fractional cal. step

  ' find corrected absolute position
  NewErr = FracStep * (ZErr(pz + 1) - ZErr(pz)) + ZErr(pz)
  NewAbsZ = NewPos + NewErr           ' find new abs. position
  IF mode$ < > "A" THEN              ' if incremental mode then
    IF ZUsrU! = 1 THEN
      temp$ = STR$(0 - FIX(NewAbsZ - LastAbsZ / (Zratio& * ZUsrU!) + .5))' convert to incremental
    ELSE
      temp$ = Limit(0 - ((NewAbsZ - LastAbsZ) / (Zratio& * ZUsrU!))) ' convert to incr. int.
    END IF
  ELSE                        ' else if absolute mode then
    IF ZUsrU! = 1 THEN
      temp$ = STR$(0 - FIX(NewAbsZ / (Zratio& * ZUsrU!) + .5))' NewAbsZ is correction value
    ELSE
      temp$ = Limit(0 - (NewAbsZ / (Zratio& * ZUsrU!)))
    END IF
  END IF
  LastAbsZ = NewAbsZ                ' remember LastAbsZ position
  LastErrZ = NewErr                 ' remember last error
xtz:  IF VAL(temp$) < 0 THEN             ' do i need to remove leading space
    Trans.Z = temp$             ' no, return whole string
  ELSE                        ' else
    Trans.Z = MID$(temp$, 2)      ' return without leading space
  END IF

END FUNCTION

FUNCTION Valid$ (char$)
```

```
' This sub - program returns the character it receives if it is valid,
' else it returns null.

  char$ = UCASE$(char$)              ' make upper case character
  SELECT CASE (char$)
    CASE (" ")
      char$ = ""            ' space is not valid
    CASE (CHR$(13))
      char$ = ""            ' CR is not valid
    CASE (CHR$(10))
      char$ = ""            ' LF is not valid
    CASE (";")
      char$ = ""            ' ; is not valid
  END SELECT
  Valid = char$                     ' return character

END FUNCTION

DEFINT A-Z
SUB WriteChr (outletter%) STATIC

' Send a chr. to Unidex 14 if there is room in its buffer; else wait !

' if axis calibration is not desired, un-comment the next line !
' goto sema

  IF mode$ = "" THEN mode$ = "X"        ' AX mode default
  IF echo THEN                      ' should i display it ?
    IF (outletter% 0) AND (outletter% 3) AND (outletter% 10) THEN
      ' don't print null's or ^C or Carriage return's
      PRINT CHR$(outletter%);
    END IF
  END IF

  IF term THEN
    IF ((outletter% = 13) OR (outletter% = 10) OR (outletter% = 32) OR (outletter = 59)) THEN term = 0
    GOTO sema
  END IF
  IF sema.colon THEN            ' am i waiting for a sema colon ?
    IF (outletter% = 59) THEN sema.colon = 0   ' got it, exit !
    GOTO sema
  END IF
  IF param THEN GOTO bld.prm            ' build parameters
  char$ = char$ + Valid(CHR$(outletter%)) ' build valid 2 char. cmd$
  IF LEN(char$)  2 THEN GOTO sema ELSE GOTO here ' wait for 2 char. cmd$
```

```
bld.prm: IF mode$ = "A" THEN          ' look for proper mode terminator
   IF (outletter% = 59) THEN GOTO bexit  ' if ";" goto bexit
  ELSE
   IF ((outletter% = 13) OR (outletter% = 10) OR (outletter% = 32) OR (outletter = 59)) THEN GOTO bexit ' termi-
nate on CR,LF,SPC,or sema
  END IF
  IF (outletter% = 44) THEN pnum = pnum + 1: GOTO wexit ' if "," goto wexit
  prm$(pnum) = prm$(pnum) + CHR$(outletter%) ' build parameter number pnum
  GOTO wexit

aa.at:  SELECT CASE (mode$)          ' translate & output positions
    CASE ("A")
      prm$(0) = Trans.X(md$, VAL(prm$(0)))
      prm$(1) = Trans.Y(md$, VAL(prm$(1)))
      prm$(2) = Trans.Z(md$, VAL(prm$(2)))
      prm$(3) = Trans.T(md$, VAL(prm$(3)))
    CASE ("M")
      prm$(0) = Trans.X(md$, VAL(prm$(0)))
      prm$(1) = Trans.Y(md$, VAL(prm$(1)))
      prm$(2) = Trans.Z(md$, VAL(prm$(2)))
      prm$(3) = Trans.T(md$, VAL(prm$(3)))
    CASE ("X")
      prm$(0) = Trans.X(md$, VAL(prm$(0)))
    CASE ("Y")
      prm$(1) = Trans.Y(md$, VAL(prm$(1)))
    CASE ("Z")
      prm$(2) = Trans.Z(md$, VAL(prm$(2)))
    CASE ("T")
      prm$(3) = Trans.T(md$, VAL(prm$(3)))
  END SELECT
writeit: FOR X = 0 TO pnum                ' write each parameter
  FOR y = 1 TO LEN(prm$(X))           ' write each chr. of param.
  WHILE (INP(statusreg) AND tbebit = 0)  ' wait till Unidex 14 is ready
  WEND
  OUT datareg, ASC(MID$(prm$(X), y, 1))  ' send character to Unidex 14
  NEXT y
  IF (X  pnum) THEN
    WHILE (INP(statusreg) AND tbebit = 0)  ' wait till Unidex 14 is ready
    WEND
    OUT datareg, ASC(",")         ' send "," to Unidex 14
  END IF
  NEXT X
    WHILE (INP(statusreg) AND tbebit = 0)  ' wait till Unidex 14 is ready
    WEND
    OUT datareg, ASC(";")         ' send terminator to Unidex 14
```

```
  CheckStatus
  RETURN

bexit: do.it = 1                    ' set execution flag

here:  SELECT CASE (char$)
    CASE ("AA")
      mode$ = "A"          ' all axis mode
    CASE ("AC")
      IF mode$ = "A" THEN
        sema.colon = 1 ' ignore all till ";" received
      ELSE
        term = 1      ' wait for terminator
      END IF
    CASE ("AF")
      IF mode$ = "A" THEN
        sema.colon = 1 ' ignore all till ";" received
      ELSE
        term = 1      ' wait for terminator
      END IF
    CASE ("AM")
      mode$ = "M"          ' multitasking axis mode
    CASE ("AN")
      IF mode$ = "A" THEN
        sema.colon = 1 ' ignore all till ";" received
      ELSE
        term = 1      ' wait for terminator
      END IF
    CASE ("AT")
      mode$ = "T"          ' T axis mode
    CASE ("AX")
      mode$ = "X"          ' X axis mode
    CASE ("AY")
      mode$ = "Y"          ' Y axis mode
    CASE ("AZ")
      mode$ = "Z"          ' Z axis mode
    CASE ("BH")
      term = 1             ' ignore all till terminator
    CASE ("BL")
      term = 1             ' ignore all till terminator
    CASE ("CD")
      IF (do.it = 0) THEN
        param = 4
      ELSE
        temp.mode$ = mode$    ' remember mode
        md$ = "A"             ' set abs. mode
```

```
    mode$ = "X"          ' fake AX mode
    GOSUB aa.at          ' translate & output positions
    mode$ = temp.mode$   ' restore mode
    ' remember contour plane definition
    p1 = 1: ax$ = ""     ' reset pointer & string
    IF LEN(prm$(0)) > 0 THEN MID$(ax$, p1, 1) = ax$ + "X": p1 = p1 + 1 ' set X axis
    IF LEN(prm$(1)) > 0 THEN MID$(ax$, p1, 1) = ax$ + "Y": p1 = p1 + 1 ' set Y axis
    IF LEN(prm$(2)) > 0 THEN MID$(ax$, p1, 1) = ax$ + "Y": p1 = p1 + 1 ' set Z axis
    IF LEN(prm$(3)) >0 THEN MID$(ax$, p1, 1) = ax$ + "Y": p1 = p1 + 1 ' set T axis
    ax$ = MID$(ax$, 1, 2)  ' limit to 2 axis
  END IF
CASE ("CR")
  IF (do.it = 0) THEN
    param = 3
  ELSE        ' translate param. 1&2 thru proper cal. tables
    SELECT CASE (ax$)
      CASE ("XY")
        prm$(0) = Trans.X(md$, VAL(prm$(0)))
        prm$(1) = Trans.Y(md$, VAL(prm$(1)))
      CASE ("XZ")
        prm$(0) = Trans.X(md$, VAL(prm$(0)))
        prm$(1) = Trans.Z(md$, VAL(prm$(1)))
      CASE ("XT")
        prm$(0) = Trans.X(md$, VAL(prm$(0)))
        prm$(1) = Trans.T(md$, VAL(prm$(1)))
      CASE ("YZ")
        prm$(0) = Trans.Y(md$, VAL(prm$(0)))
        prm$(0) = Trans.Z(md$, VAL(prm$(1)))
      CASE ("YT")
        prm$(0) = Trans.Y(md$, VAL(prm$(0)))
        prm$(1) = Trans.T(md$, VAL(prm$(1)))
      CASE ("ZT")
        prm$(0) = Trans.Z(md$, VAL(prm$(0)))
        prm$(1) = Trans.T(md$, VAL(prm$(1)))
    END SELECT
    GOSUB writeit  ' output corrected parameters
  END IF
CASE ("CV")
  term = 1          ' ignore all till terminator
CASE ("EF")
  echo = 0          ' kill echo
CASE ("EN")
  outletter% = 70     ' change it to E'F'
  echo = 1          ' but; enable my echo
CASE ("ER")             ' enc. counts, motor counts
  IF (do.it = 0) THEN
```

```
    param = 2
  ELSE
    SELECT CASE (mode$)
      CASE ("X")
        Xratio& = INT(VAL(prm$(1)) / VAL(prm$(0)))
      CASE ("Y")
        Yratio& = INT(VAL(prm$(1)) / VAL(prm$(0)))
      CASE ("Z")
        Zratio& = INT(VAL(prm$(1)) / VAL(prm$(0)))
      CASE ("T")
        Tratio& = INT(VAL(prm$(1)) / VAL(prm$(0)))
    END SELECT
    GOSUB writeit
  END IF
CASE ("ES")
  term = 1          ' ignore all till terminator
CASE ("FP")
  IF (do.it = 0) THEN
    param = 1
  ELSE
    md$ = "A"     ' set abs. mode
    GOSUB aa.at   ' translate & output positions
  END IF
CASE ("HD")
  term = 1          ' wait for terminator
CASE ("HG")
  term = 1          ' ignore all till terminator
CASE ("HM")
  IF (do.it = 0) THEN
    outletter% = 72      ' change it to H'H"
    param = 1
  ELSE
    Home
    GOSUB writeit
  END IF
CASE ("HR")
  IF (do.it = 0) THEN
    outletter% = 72      ' change it to H'H"
    param = 1
  ELSE
    Home
    GOSUB writeit
  END IF
CASE ("HV")
  term = 1          ' ignore all till terminator
CASE ("IN")
```

```
  IF mode$ = "A" THEN
    sema.colon = 1  ' ignore all till ";" received
  ELSE
    term = 1        ' wait for terminator
  END IF
CASE ("JG")
  term = 1          ' ignore all till terminator
CASE ("KM")
  Home
CASE ("KR")
  Home
CASE ("LP")               ' load position register
  IF (do.it = 0) THEN
    param = 1
  ELSE
    GOSUB writeit         ' send to Unidex 14
  END IF
CASE ("LS")
  term = 1          ' ignore all till terminator
CASE ("MA")
  IF (do.it = 0) THEN
    IF mode$ = "A" THEN param = 4 ELSE param = 1
  ELSE
    md$ = "A"       ' set abs. mode
    GOSUB aa.at     ' translate & output positions
  END IF
CASE ("ML")
  IF (do.it = 0) THEN
    IF mode$ = "A" THEN param = 4 ELSE param = 1
  ELSE
    md$ = "I"       ' set incremental mode
    GOSUB aa.at     ' translate & output positions
  END IF
CASE ("MR")
  IF (do.it = 0) THEN
    IF mode$ = "A" THEN param = 4 ELSE param = 1
  ELSE
    md$ = "I"       ' set mode
    GOSUB aa.at     ' go translate inc. positions
  END IF
CASE ("MT")
  IF (do.it = 0) THEN
    IF mode$ = "A" THEN param = 4 ELSE param = 1
  ELSE
    md$ = "A"       ' set absolute mode
    GOSUB aa.at     ' translate & output positions
```

```
    END IF
  CASE ("MV")          ' move vel. (abs. pos., velocity)
    IF (do.it = 0) THEN
      param = 2
    ELSE
      md$ = "A"      ' set abs. mode
      GOSUB aa.at    ' translate abs. positions
    END IF
  CASE ("PA")
    IF mode$ = "A" THEN
      sema.colon = 1 ' ignore all till ";" received
    ELSE
      term = 1       ' wait for terminator
    END IF
  CASE ("PN")
    IF mode$ = "A" THEN
      sema.colon = 1 ' ignore all till ";" received
    ELSE
      term = 1       ' wait for terminator
    END IF
  CASE ("RE")
    IF (mode$ < > "A") AND (mode$ < > "M") THEN
    IF sendmode THEN
      WHILE (INP(statusreg) AND tbebit = 0)
      WEND               ' wait till U14 is rdy
      OUT datareg, outletter% ' send "E" to U14
      CheckStatus
      result$ = ReadLF$      ' clr misc. encoder position
      EncPos& = VAL(result$) ' get encoder pos.

' translate Unidex 14's absolute positions to real world positions.

      IF EncPos& = > 0 THEN GOTO skp.cal      ' only neg. # calibrated
      SELECT CASE (mode$)
        CASE ("X")
          IF CalSizX = 0 THEN GOTO skp.cal
          IF EncPos& = > 0 THEN GOTO skp.cal ' only neg. # calibrated
          APos! = 0 - EncPos&          ' make it positive
          Ap! = APos! / CalSizX        ' find index into table
          FPos! = Ap! - INT(Ap!)       ' find fractional cal. step
          Ap! = INT(Ap!)
          EncPos& = EncPos& + FIX(FPos! * (XErr(Ap! + 1) - XErr(Ap!)) + XErr(Ap!) + .5)
        CASE ("Y")
          IF CalSizY = 0 THEN GOTO skp.cal
          IF EncPos& = > 0 THEN GOTO skp.cal ' only neg. # calibrated
          APos! = 0 - EncPos&          ' make it positive
```

```
        Ap! = APos! / CalSizY        ' find index into table
        FPos! = Ap! - INT(Ap!)       ' find fractional cal. step
        Ap! = INT(Ap!)
        EncPos& = EncPos& + FIX(FPos! * (YErr(Ap! + 1) - YErr(Ap!)) + YErr(Ap!) + .5)
      CASE ("Z")
        IF CalSizZ = 0 THEN GOTO skp.cal
        IF EncPos& > = 0 THEN GOTO skp.cal ' only neg. # calibrated
        APos! = 0 - EncPos&          ' make it positive
        Ap! = APos! / CalSizZ        ' find index into table
        FPos! = Ap! - INT(Ap!)       ' find fractional cal. step
        Ap! = INT(Ap!)
        EncPos& = EncPos& + FIX(FPos! * (ZErr(Ap! + 1) - ZErr(Ap!)) + ZErr(Ap!) + .5)
      CASE ("T")
        IF CalSizT = 0 THEN GOTO skp.cal
        IF EncPos& > = 0 THEN GOTO skp.cal ' only neg. # calibrated
        APos! = 0 - EncPos&          ' make it positive
        Ap! = APos! / CalSizT        ' find index into table
        FPos! = Ap! - INT(Ap!)       ' find fractional cal. step
        Ap! = INT(Ap!)
        EncPos& = EncPos& + FIX(FPos! * (TErr(Ap! + 1) - TErr(Ap!)) + TErr(Ap!) + .5)
    END SELECT
skp.cal:            PRINT CHR$(13); EncPos&
      char$ = ""          ' clr cmd$
      GOTO wexit
    END IF
    END IF
  CASE ("RM")
    term = 1          ' ignore all till terminator
  CASE ("RP")
    IF sendmode THEN
      WHILE (INP(statusreg) AND tbebit = 0)
      WEND            ' wait till U14 is rdy
      OUT datareg, outletter% ' send "P" to U14
      CheckStatus
      result$ = ReadLF$     ' clr misc. position
      prm$(0) = "AA;RP"     ' define pos. cmd. string
      pnum = 0: GOSUB writeit ' write it to U14
      prm$(0) = ""          ' clr before exit
      result$ = ReadLF$                  ' input all 4 position's
      AbsPosX = VAL(result$)             ' get x abs. pos.
      LastComma = INSTR(result$, ",") + 1        ' find start of y pos.
      AbsPosY = VAL(MID$(result$, LastComma, 25))    ' get y abs. pos.
      LastComma = INSTR(LastComma, result$, ",") + 1 ' find start of z pos.
      AbsPosZ = VAL(MID$(result$, LastComma, 25))    ' get z abs. pos.
      LastComma = INSTR(LastComma, result$, ",") + 1 ' find start of t pos.
      AbsPosT = VAL(MID$(result$, LastComma, 25))    ' get t abs. pos.
```

```
' translate Unidex 14's absolute positions to real world positions.
    IF CalSizX = 0 THEN GOTO skp1  ' is axis cal. present ?
    IF AbsPosX > = 0 THEN GOTO skp1  ' only neg. # calibrated
    APos! = 0 - AbsPosX          ' make it positive steps
    Ap! = APos! / CalSizX        ' find index into table
    FPos! = Ap! - INT(Ap!)       ' find fractional cal. step
    Ap! = INT(Ap!)
    AbsPosX = AbsPosX + FIX(FPos! * (XErr(Ap! + 1) - XErr(Ap!)) + XErr(Ap!) + .5)

skp1:           IF CalSizY = 0 THEN GOTO skp2  ' is axis cal. present ?
    IF AbsPosY > = 0 THEN GOTO skp2  ' only neg. # calibrated
    APos! = 0 - AbsPosY          ' make it positive
    Ap! = APos! / CalSizY        ' find index into table
    FPos! = Ap! - INT(Ap!)       ' find fractional cal. step
    Ap! = INT(Ap!)
    AbsPosY = AbsPosY + FIX(FPos! * (YErr(Ap! + 1) - YErr(Ap!)) + YErr(Ap!) + .5)

skp2:           IF CalSizZ = 0 THEN GOTO skp3  ' is axis cal. present ?
    IF AbsPosZ > = 0 THEN GOTO skp3  ' only neg. # calibrated
    APos! = 0 - AbsPosZ          ' make it positive
    Ap! = APos! / CalSizZ        ' find index into table
    FPos! = Ap! - INT(Ap!)       ' find fractional cal. step
    Ap! = INT(Ap!)
    AbsPosZ = AbsPosZ + FIX(FPos! * (ZErr(Ap! + 1) - ZErr(Ap!)) + ZErr(Ap!) + .5)

skp3:           IF CalSizT = 0 THEN GOTO skp4  ' is axis cal. present ?
    IF AbsPosT > = 0 THEN GOTO skp4  ' only neg. # calibrated
    APos! = 0 - AbsPosT          ' make it positive
    Ap! = APos! / CalSizT        ' find index into table
    FPos! = Ap! - INT(Ap!)       ' find fractional cal. step
    Ap! = INT(Ap!)
    AbsPosT = AbsPosT + FIX(FPos! * (TErr(Ap! + 1) - TErr(Ap!)) + TErr(Ap!) + .5)

skp4:           SELECT CASE (mode$)
      CASE ("A")
        PRINT CHR$(13); AbsPosX; ","; AbsPosY; ","; AbsPosZ; ","; AbsPosT
      CASE ("M")
        PRINT CHR$(13); AbsPosX; ","; AbsPosY; ","; AbsPosZ; ","; AbsPosT
      CASE ("X")
        PRINT CHR$(13); AbsPosX
      CASE ("Y")
        PRINT CHR$(13); AbsPosY
      CASE ("Z")
        PRINT CHR$(13); AbsPosZ
      CASE ("T")
```

```
      PRINT CHR$(13); AbsPosT
    END SELECT

    WHILE (INP(statusreg) AND tbebit = 0)
    WEND              ' wait till U14 is rdy
    OUT datareg, 65       ' send "A" to U14
    WHILE (INP(statusreg) AND tbebit = 0)
    WEND              ' wait till U14 is rdy
    OUT datareg, ASC(mode$) ' send char. to reset mode
    CheckStatus
    char$ = ""            ' clr cmd$
    GOTO wexit
  END IF
CASE ("RS")
  mode$ = "X"           ' reset to AX mode
  XUsrU! = 1: YUsrU! = 1 ' reset all user units to 1
  ZUsrU! = 1: TUsrU! = 1
  Xratio& = 1: Yratio& = 1 ' reset encoder ratio to 1
  Zratio& = 1: Tratio& = 1
  echo = 0              ' kill echo
  LastAbsX = 0: LastAbsY = 0 ' reset last axis cal. pos.
  LastAbsZ = 0: LastAbsT = 0
  LastErrX = 0: LastErrY = 0 ' reset last cal. error
  LastErrZ = 0: LastErrT = 0
CASE ("SP")
  IF (do.it = 0) THEN
    param = 1
  ELSE
    md$ = "A"      ' set abs. mode
    GOSUB aa.at    ' translate positions
  END IF
CASE ("SW")
  term = 1              ' ignore all till terminator
CASE ("UF")                 ' set user units off
  SELECT CASE (mode$)     ' kill user units
    CASE ("X")
      XUsrU! = 1    ' user units = 1
    CASE ("Y")
      YUsrU! = 1    ' user units = 1
    CASE ("Z")
      ZUsrU! = 1    ' user units = 1
    CASE ("T")
      TUsrU! = 1    ' user units = 1
  END SELECT
CASE ("UU")
  IF (do.it = 0) THEN
```

```
        param = 1
      ELSE
        SELECT CASE (mode$)    ' store user units
          CASE ("X")
            XUsrU! = VAL(prm$(0))
          CASE ("Y")
            YUsrU! = VAL(prm$(0))
          CASE ("Z")
            ZUsrU! = VAL(prm$(0))
          CASE ("T")
            TUsrU! = VAL(prm$(0))
        END SELECT
        GOSUB writeit
      END IF
    CASE ("VB")
      IF mode$ = "A" THEN
        sema.colon = 1 ' ignore all till ";" received
      ELSE
        term = 1      ' wait for terminator
      END IF
    CASE ("VL")
      IF mode$ = "A" THEN
        sema.colon = 1 ' ignore all till ";" received
      ELSE
        term = 1      ' wait for terminator
      END IF
    CASE ("VS")
      term = 1          ' wait for terminator
    CASE ("WS")
      term = 1          ' wait for terminator
    CASE ("WT")
      term = 1          ' wait for terminator
  END SELECT
  IF do.it THEN
    pnum = 0              ' clear param. pointer
    param = 0             ' clear # of param's
    do.it = 0           ' i did it, so clear it
    prm$(0) = "": prm$(1) = ""    ' clear param. storage
    prm$(2) = "": prm$(3) = ""
  END IF
  IF (param = 0) THEN char$ = ""       ' clear cmd$

sema:  status% = INP(statusreg)
  WHILE ((status% AND tbebit) = 0)  ' wait till Unidex 14 is ready
    status% = INP(statusreg)
  WEND
```

```
  OUT datareg, outletter%          ' send character to Unidex 14

wexit:

END SUB ' writechr

DEFSNG A-Z
SUB WriteString (cmd$)

' Check if Unidex 14 has room for a chr., if it doesn't, wait until there is
'     room, then send the chr. to Unidex 14.

  FOR X% = 1 TO LEN(cmd$)          ' loop till all chr's are sent
  WriteChr (ASC(MID$(cmd$, X%, 1)))    ' send an ASCII chr. from string
  CheckStatus
  NEXT X%

END SUB ' writestring
```

SECTION 9-4: PASCAL DEMONSTRATION PROGRAM

```
PROGRAM U14_PAS;

{This program allows the user to interact with a Unidex 14 motion
control board.  It uses a "polling" method to communicate with the
board as opposed to using interrupts as the "C" version does.

Portions of this program may be adapted by Unidex 14 users for application
programs.  No license is required.

This program was written using Microsoft QuickPascal Version 1.0 Copyright 1989
Version 1.00
RELEASED July 4, 1990 }

USES
  CRT;

CONST
  DATA_REG = $300;
  DONE_REG = $301;
  CONTROL_REG = $302;
  STATUS_REG = $303;
```

```
  CMDERR_BIT = $01;
  INIT_BIT = $02;
  ENC_BIT = $04;
  OVRT_BIT = $08;
  DONE_BIT = $10;
  IBF_BIT = $20;
  TBE_BIT = $40;
  IRQ_BIT = $80;

VAR
  keystroke, u14dataout, status : byte;
  name : string;


PROCEDURE  outp( outaddress : word; outdata : byte);
{output a byte to the given I/O address}

BEGIN
  INLINE
  (
  $8B/$96/outaddress/ { mov dx,outaddress ;Load I/O Address into DX reg.}
  $8A/$86/outdata/    { mov al,outdata    ;Load Data byte into AL reg.}
  $EE                 { out dx,al         ;Output byte in AL to DX addr}
  );
END; {procedure outp}


FUNCTION inp(inaddress: word) : byte;
{input a byte from given I/O address}

VAR
  return : byte;

BEGIN
  INLINE
  (
  $8B/$96/inaddress/ { mov  dx, inaddress ;Load I/O Address into DX reg}
  $EC/               { in   al, dx    ;Get byte at DX add, put it in AL}
  $88/$86/return     { mov  return, a    ;Put AL byte in Return variable}
  );

  inp := return;
END; {function inp}


PROCEDURE  write_u14_data(outletter : byte);
```

```
{check if Unidex 14 has room for a byte, if it doesn't wait until there is
room, then send the byte to it }

BEGIN
  WHILE inp(STATUS_REG) AND TBE_BIT = 0 DO
    BEGIN
    END;
  outp(DATA_REG,outletter);
END; {procedure write_u14_data}


PROCEDURE checkstatus (status : byte);
{check to see if Unidex 14 is flagging an error or has executed an id}

VAR
  doneflag : byte;
BEGIN
  IF (status AND DONE_BIT) < > 0 THEN
    BEGIN
      IF (status AND CMDERR_BIT) < > 0 THEN
        BEGIN
          writeln;
          writeln ('Unidex 14 Status Register indicates a command error');
        END

      ELSE IF (status AND ENC_BIT) < > 0 THEN
      BEGIN
          writeln;
          writeln ('Unidex 14 Status Register indicates an Encoder Slip error');
        END

      ELSE IF (status AND OVRT_BIT) < > 0 THEN
        BEGIN
          writeln;
          writeln ('Unidex 14 Status Register indicates limit switch closed');
        END

      ELSE   {an ID has been encountered, print the done axis}
       BEGIN
          doneflag : = INP(DONE_REG);
          IF doneflag < > 0 THEN
            BEGIN
              writeln;
              writeln ('Unidex 14 Status Register indicates a DONE flag set');
              write ('AXES DONE is/are: ');
              IF doneflag MOD 2 = 1 THEN
```

```
            write('x ');
          doneflag := doneflag DIV 2;
          IF doneflag MOD 2 = 1 THEN
            write('y ');
          doneflag := doneflag DIV 2;
          IF doneflag MOD 2 = 1 THEN
            write('z ');
          doneflag := doneflag DIV 2;
          IF doneflag MOD 2 = 1 THEN
            write('t ');
        writeln;
        writeln('Press Cntrl-Y or enter IC to clear done flags.');
        END;
    END;
    write_u14_data(24); {clear error status}
  END;
END; {procedure checkstatus}


FUNCTION read_u14_data : byte ;

BEGIN
{check to see if Unidex 14 has a character to send, if it does, print it}
 status := inp(STATUS_REG);
 IF (status AND IBF_BIT) 0 THEN
   read_u14_data := inp(DATA_REG)
 ELSE
   read_u14_data := 0;

END; {procedure read_u14_data}

PROCEDURE  sendfile(filename:string);
{this procedure reads a file containing Unidex 14 commands and send them to
Unidex 14}

VAR
 c : char;
 filepointer : FILE OF char;

{$I-}  {turn off run time I/O error checking}

BEGIN
 assign(filepointer , filename);
 reset(filepointer);
 IF (IOresult  < > 0) THEN
   BEGIN
```

```
      writeln('Unable to open file: ',filename);
      exit;
    END;
  WHILE NOT eof(filepointer) DO
    BEGIN
      read(filepointer,c);
      write_u14_data( ord (c) );
    END;

{$I + } {turn on run time I/O error checking}

  close(filepointer);
END; {procedure sendfile}


BEGIN {MAIN PROGRAM}

  outp(CONTROL_REG,$00); {disable Unidex 14's interrupts}
  clrscr;
  write('Name of a Unidex 14 command file to download (< CR > for no file):');
  readln(name);
  sendfile(name);
  writeln(output,'Interactive mode, Enter Unidex 14 commands, Press ^C to exit');
  keystroke : = 0;
  WHILE keystroke  < > 3 DO { 3 is a Control-C}
    BEGIN
      IF keypressed THEN
        BEGIN
          keystroke : = Ord(readkey);
          write_u14_data(keystroke);
          IF keystroke  = 13 THEN  { if it's a CR print an LF too}
            write( chr(10));
        END;

        u14dataout : = read_u14_data; {get Unidex 14 output }
        IF (u14dataout  0) AND (u14dataout  3) THEN
        {don't print null's or ^C}
        write( chr(u14dataout));

        checkstatus(status);
    END;

END.
```

SECTION 9-5: C DEMONSTRATION PROGRAM WITH INTERRUPTS

```
/* PROGRAM U14_INTR.C          Version 1.01

  This program allows the user to interact with the Unidex 14
  motion control board. This program uses the PC's interrupt line
  IRQ5 to communicate between the PC and Unidex 14. All C functions
  changing the mode of Unidex 14 will return it to the AX mode.

  This program will have to be restarted if an RS (reset) command is sent
  to Unidex 14.

  Portions of this program may be adapted by users for application
  programs using Unidex 14. No license is required.

  This program was written using Microsoft C 5.1 Compiler.

  RELEASED Aug. 27, 1991
*/

#include <stdio.h>
#include <dos.h>
#include<conio .h>

#define DATA_REG      0x300  /* Unidex 14 I/O port definitions */
#define DONE_REG      0x301
#define CONTROL_REG   0x302
#define STATUS_REG    0x303

#define CMDERR_BIT 0x1   /* CONTROL and STATUS register bit definitions */
#define INIT_BIT   0x2
#define ENC_BIT    0x4
#define OVRT_BIT   0x8
#define DONE_BIT   0x10
#define IBF_BIT    0x20
#define TBE_BIT    0x40
#define IRQ_BIT    0x80
        /* PC interrupt definitions */
#define ICNT      0x20        /* interrupt terminator byte for 8259 */
#define IMREG     0x21        /* interrupt mask register I/O address */
#define ILEVEL    0x20        /* interrupt mask level 5 byte */

#define CNTRL_C  3            /* character definitions */
#define CLEARD   24           /* Control-X */
```

```
#define LF      10
#define CR      13

#define BSIZE    1000          /* input and output buffer size */

int    inxti, inxto, onxti, onxto; /* buffer pointers */

char done_flag, error_flag;
char inbuf[ BSIZE ], outbuf[ BSIZE ];  /* define buffers arrays */

char fname[80];                 /* file name buffer */

long atol(char *);          /* function prototypes */
char toupper(char);
void main(void);
void sendfile(void);
void interact(void);
void checkstatus(void);
char getcm(void);
void init_sys(void);
void putcm(char c);
void writestring(char *cmd);
int  mhit(void);
interrupt far service(void);
void readstring(int cnt, char *result);
char readchr(void);
int fault(void);
void getposition(long *);
void home(void);
long inbit(void);
void inpos(char move[]);
long power_16(int power);
void readlf(char *result);
void remote(char mode);
void system(char *);
int strlen(char *);
char *strtok(char *,char *);

void main(void)
{
long lpoint[4];                 /* pointer to all 4 axis pos. */
  inxti = inxto = onxti = onxto = 0;    /* set buffer pointers to beginning */
  done_flag = 0;                /* clear done flag variable */
  error_flag = 0;               /* clear error flag variable */

  init(service);                /* initialize interrupts */
```

```
  system("cls");              /* clear PC's screen */
  init_sys();                 /* set some defaults */
  sendfile();                 /* send file to Unidex 14 */
  interact();                 /* interactive mode */
  printf("\nInputs = %li",inbit()); /* display input states */
  if(fault())                 /* test for drive faults */
    printf("\nERROR drive fault exists !");
  printf("\nDrives are now In Position");
  getposition(lpoint);        /* update the absolute position's */
  printf("\n X axis position = %li", lpoint[0]);
  printf("\n Y axis position = %li", lpoint[1]);
  printf("\n Z axis position = %li", lpoint[2]);
  printf("\n T axis position = %li", lpoint[3]);
/*    home();                   /* send all axis home */
}

void sendfile(void)    /* Open file and send it to Unidex 14 */
{
  FILE   *fp;

  printf("\nName of Unidex 14 command file to download (< CR > for no file):");
  gets(fname);
  if(strlen(fname)  > 0)
  {     if ((fp = fopen(fname, "r")) = = NULL)
      printf("\nCan't open %s\n", fname);
    else
    {     while(feof(fp) = = 0)      /* while not end of file */
      {     putcm(fgetc(fp)); /* get a char from file, send it to Unidex 14 */
    /* band aid - if DOS turns interrupt off, turn it back on */
        outp(IMREG, inp(IMREG) & ~ ILEVEL);
        while(mhit())         /* while a char available from Unidex 14 */
        {     putch(getcm());     /* print it */
        }
        checkstatus();
      }
      fclose(fp);         /* close file when done */
      printf("%s transmitted.\n", fname);
    }
  }
}

void interact(void)  /* Sends keyboard entry to Unidex 14, and displays Unidex 14's output */
{
char outchar = 0, inchar;

  printf("\nInteractive mode, Enter Unidex 14 commands, Press Control-C to exit.\n");
```

```
  while (outchar != CNTRL_C)
  {     if (kbhit())        /* char available from keyboard ? */
    {     outchar = getch();    /* get char from keyboard */
      putcm(outchar);       /* send it to Unidex 14 */
      if(outchar == CR)
        putch(LF);    /* add linefeeds when return is pressed to
              avoid overstriking letters on the screen */
    }
    if (mhit())         /* char available from Unidex 14 ? */
    {     inchar = getcm();  /* get char */
      putch(inchar);   /* display it */
    }
    checkstatus();
  }
}

void checkstatus(void)  /* this routine decodes Unidex 14's status and displays it's info */
{
extern char done_flag, error_flag;

  if (error_flag != 0)
  {     if ((error_flag & CMDERR_BIT) != 0)       /* command error */
      printf("\nUnidex 14's Status Register indicates a command error\n");
    else
    if ((error_flag & ENC_BIT) != 0)      /* encoder slip */
      printf("\nUnidex 14's Status Register indicates an Encoder Slip error\n");
    else
    if ((error_flag & OVRT_BIT) != 0)       /* limit switch is active */
      printf("\nUnidex 14's Status Register indicates a limit switch is closed\n");
    error_flag = 0; /* clear error_flag */
  }

  if(done_flag != 0) /* an ID has been encountered, print the done axis */
  {     printf("\nUnidex 14's Status Register indicates a DONE flag has been set\n");
    printf("AXES DONE: ");
    if(done_flag % 2 == 1)
      printf("x ");
    done_flag = done_flag / 2;
    if(done_flag % 2 == 1)
      printf("y ");
    done_flag = done_flag / 2;
    if(done_flag % 2 == 1)
      printf("z ");
    done_flag = done_flag / 2;
    if(done_flag % 2 == 1)
      printf("t ");
```

```
    printf("\nPress Control-Y, or type IC to clear Unidex 14's done status.\n");
    done_flag = 0; /* clear done flag */
  }
}

char getcm(void) /* Returns character from Unidex 14 */
{
char   c;
  if(inxti = = inxto)          /* no char available in buffer */
    return (0);
  else
  {     c = inbuf[inxto + +];    /* get char. and  incr. pointer */
    if(inxto = = BSIZE)     /*if end of buffer */
      inxto = 0;     /* reset pointer */
    outp(CONTROL_REG, inp(CONTROL_REG) | IBF_BIT); /* enable U14 intr. */
    return (c);
  }
}

void putcm(char c) /* Put character into Unidex 14's buffer */
{
  int    i, j;

  i = onxti;   /* set local pointer to global input pointer of output buffer*/
  j = i + 1;
  if(j = = BSIZE) /* if next pointer points to the end, wrap pointer to start */
    j = 0;
  while(j = = onxto)  /* while there is no room in the buffer for more */
    ;            /* wait until Unidex 14's interrupt routine makes room in buffer */
  outbuf[i + +] = c;   /* get character i points to and increment i */
  if(i = = BSIZE)   /* if i points to the end of buffer wrap it around */
    i = 0;
  onxti = i;       /* set global pointer to local pointer's value */
  outp(CONTROL_REG, inp(CONTROL_REG) | TBE_BIT); /* enable Unidex 14 to interrupt */
}

mhit()  /* returns non zero if input buffer has character */
{
  return(inxto - inxti);
}

interrupt far service()  /* interrupt driven routine to communicate with Unidex 14 */
{
char  c;
int  i;
```

```
c = inp(STATUS_REG) & (inp(CONTROL_REG) | 0x0f);
/* get the status of interrupts that are enabled */

if(c & DONE_BIT)        /* if the DONE bit is true */
{    done_flag | = inp(DONE_REG);  /* save done axis flags */
  error_flag | = (c & 0x0f);    /* save error bits also */
  outp(DATA_REG, CLEARD);      /* reset Unidex 14 status register DONE bit
          without clearing flags shown in RA or
          RI commands which ^Y does */
}
else
if((c & TBE_BIT) = = TBE_BIT) /* if Unidex 14 will accept a char */
{    if(onxto ! = onxti)  /* do we have a char to send to Unidex 14 ? */
  {    outp(DATA_REG, outbuf[onxto]); /* send char to Unidex 14 */
    if( + +onxto = = BSIZE)     /* take care of wrap around */
      onxto = 0;
    if(onxto = = onxti)     /* if the buffer is now empty */
    /* turn off Unidex 14 interrupt so it won't keep asking for chars */
      outp(CONTROL_REG, inp(CONTROL_REG) & ~TBE_BIT);
  }
  else    /* no char in buffer to send to Unidex 14 */
       /* turn off Unidex 14's interrupt so it won't keep asking for chars */
    outp(CONTROL_REG, inp(CONTROL_REG) & ~TBE_BIT);
}
if((c & IBF_BIT) = = IBF_BIT) /* if Unidex 14 has a char to send to the PC */
{
  inbuf[inxti] = inp(DATA_REG); /* get the char and put it in the buffer */
  if( + +inxti = = BSIZE)     /* take care of wrap around */
    inxti = 0;
  if((i = inxti + 1) = = BSIZE) /* room for another char in the buffer? */
    i = 0;
  if(i = = inxto)
    outp(CONTROL_REG, inp(CONTROL_REG) & ~IBF_BIT);
    /* no room in buffer, turn off interrupt so Unidex 14 cannot send more */
}
outp(ICNT, 0x20);        /* send 8259 chip a return from interrupt */

/* PC/XT/AT interrupt control is edge sensitive only.
The following generates a rising edge if a second interrupt has occured
prior to the first being reset.  This can happen if more than one
source is enabled on the Unidex 14 as in this program. */

c = inp(CONTROL_REG);
outp(CONTROL_REG, c & ~IRQ_BIT);      /* turn off interrupts */
outp(CONTROL_REG, c);         /* and back on to create edge */
}
```

```
init(intvec) /* connect hardware interrupt IRQ5 to interrupt handler */
int (*intvec)();
{
  extern int   intdosx();
  union  REGS  inregs, outregs;
  struct SREGS segregs;

  inregs.h.ah = 0x25;          /* set interrupt vector DOS call */
  inregs.h.al = 13;           /* interrupt vector level 5 */
  segregs.ds = FP_SEG(*intvec);  /* send the vector */
  inregs.x.dx = FP_OFF(*intvec);
  intdosx(&inregs, &outregs, &segregs);
  outp(IMREG, inp(IMREG) & ~ILEVEL);   /* enable interrupts */
  outp(CONTROL_REG, IBF_BIT | DONE_BIT | IRQ_BIT); /* turn on Unidex 14's interrupts */
}

void home(void)  /* this function will hardware home all axis */
{
long result;
  writestring("BL13;WT9;BH13;");       /* send all axis home */
  printf("\nHoming");
  writestring ("AX");                 /* set x axis active */
  for(result=0; result < 100; result++)  /* wait for home to start */
    {}
  result = inbit();                  /* read the inputs */
  while ((result & 32) != 0)          /* wait for home completion */
    result = inbit();           /* read the inputs */
  writestring ("ATLP0;");             /* set t axis pos. 0 */
  writestring ("AZLP0;");             /* set z axis pos. 0 */
  writestring ("AYLP0;");             /* set y axis pos. 0 */
  writestring ("AXLP0;");             /* set x axis pos. 0 */
  printf("\nAt Home");
}

long inbit(void)       /* this function returns the state of the inputs */
{
int x;
long accum=0;                    /* decimal equiv. accumulator */
char result[20];                /* init. string */
  writestring("AXBX");          /* ask for input bit states */
  readlf(result);             /* input the hex-ascii values */
  for(x=0; x < 5; x++)
    {  if (result[x] = 'A')          /* is it 'A' thru 'F' ? */
      result[x] = result[x]-7; /* pre-scale hex values */
    accum = accum + (power_16(5-x) * (result[x]-48)); /* calc. base ten value */
```

```
    }
    return(accum);                  /* return inputs in dec. */
}

long power_16(int power)  /* this function returns 16 raised to the power of
        power. This is called by the inbit function only ! */
{
int i;
long p=1;
  for(i=1; i < = power; i++)
    p = p * 16;
  return(p);
}

int fault(void)  /* this function returns true '1' if any drive fault exists,
       on any servo axis, stepper drives do not have a fault
       output to Unidex 14. */
{
  if((inbit() & 64) = = 0)       /* read the inputs */
    return(0);          /* drive fault not present */
  else
    return(1);          /* drive fault present */
}

void inpos(char move[255])  /* this function starts a move, then waits for all
         servos, & steppers with 1MR bds. to come to rest.
      the move cmd string is expected to include a "go" cmd */
{
  writestring(move);                /* start the move */
  printf("\nWaiting for drives to come to rest");
  while(!(inbit() & 128))            /* wait for move to start */
    ;
  while(inbit() & 128)               /* wait for move to end */
    ;
}

void remote(char mode)  /* this function controls the remote clk. & dir. inputs */
     /* to the Unidex 14 power chassis. (L is PC control, */
{                /*  R is remote clock & direction */
  if((toupper(mode) = = 'R'))       /* convert mode chr. to upper case */
    writestring ("AXBL12;");  /* set Remote mode */
  else
    writestring ("AXBH12;");  /* set Local mode  */
}

void getposition(long *position)  /* this function returns a pointer to 4 long int's */
```

```
{
char result[25];
  writestring("ATRP");          /* request position from Unidex 14 */
  readlf(result);
  position[3] = atol(result);   /* find the t axis position */
  writestring("AZRP");          /* request position from Unidex 14 */
  readlf(result);
  position[2] = atol(result);   /* find the z axis position */
  writestring("AYRP");          /* request position from Unidex 14 */
  readlf(result);
  position[1] = atol(result);   /* find the y axis position */
  writestring("AXRP");          /* request position from Unidex 14 */
  readlf(result);
  position[0] = atol(result);   /* find the x axis position */
}

void init_sys(void)       /* this function initializes Unidex 14 system */
{
int c;
char result[40];
  writestring("AXEF");          /* kill echo if active */
  for (c=0; c < 3000; c++);     /* delay till echo stops */
  {;}
  while ((inp(STATUS_REG) & IBF_BIT)) /* while there's a chr. in buffer */
    c = getcm();          /* input it */
  remote('L');              /* disable remote clk. & dir. */
  writestring ("WY");           /* get Unidex 14 version number */
  readlf(result);           /* input version number */
  printf ("Version # is %s\n",result);
/*    writestring ("AAPA1,1,1,1;");   /* set high i mode during moves */
  writestring ("AXBH13;");      /* reset home cycle pulse */
}

void writestring(char cmd[255]) /* this function writes a string to Unidex 14 */
{
int x = 0;                      /* set initial value  */
  putcm('\x0d');                /* write a leading CR */
  while (cmd[x] != '\0')        /* loop till all chr's sent  */
  {     putcm(cmd[x++]);        /* write a chr. to Unidex 14 */
    checkstatus();          /* check Unidex 14's status register */
  }
}

char readchr(void)  /* wait for a valid chr. and then return it */
{
```

```
char chr = 0;
  while(chr = = NULL)          /* read till valid chr. */
    chr = getcm();       /* get a chr. from Unidex 14 */
  return(chr);
}

void readstring(int cnt, char *result)  /* read cnt number of chr's and modify pointer to string */
{
int x;
  for(x=0; x= =(cnt-1); x++);          /* read cnt number of chr's */
  {     result[x] = readchr();       /* get a valid chr. */
  }
  result[x++] = '\0';                /* add NULL terminator */
}

void readlf(char *result)  /* read chr's till a LF is received then begin inputting string until CR LF */
{
int temp = 0;

  while(readchr() != CR)             /* read till leading CR */
    ;
  while((result[temp] = readchr()) != '\n') /* read till LF terminator */
   {    temp++;                     /* build chr. string */
   }
  result[temp] = '\0';               /* add terminator */
  while(readchr() != '\r')           /* read till LF terminator */
    ;
}
```

SECTION 9-6: U14_POLL.C

```
/* PROGRAM U14_POLL.C            Version 1.01

  This program allows the user to interact with the Unidex 14
  motion control board.  This program uses polling to
  communicate between the PC and Unidex 14. All C functions
  changing the mode of Unidex 14 will return it to the AX mode.

  Portions of this program may be adapted by users for application
  programs using Unidex 14.  No license is required.

  This program was written using Microsoft QuickC 2.5 Compiler.
```

```
RELEASED Aug. 27, 1991
*/

#include <stdio.h
#include <dos.h>
#include <conio.h>

#define DATA_REG      0x300  /* Unidex 14 I/O port definitions */
#define DONE_REG      0x301
#define CONTROL_REG   0x302
#define STATUS_REG    0x303

#define CMDERR_BIT 0x1   /* CONTROL and STATUS register bit definitions */
#define INIT_BIT   0x2
#define ENC_BIT    0x4
#define OVRT_BIT   0x8
#define DONE_BIT   0x10
#define IBF_BIT    0x20
#define TBE_BIT    0x40
#define IRQ_BIT    0x80

#define CNTRL_C  3              /* character definitions */
#define CLEARD   24             /* Control-X */
#define LF       10
#define CR       13

int status;
char fname[80];                 /* file name buffer */

long atol(char *);              /* function prototypes */
char toupper(char);
void main(void);
void sendfile(void);
void interact(void);
void checkstatus(void);
char getcm(void);
void init_sys(void);
void putcm(char c);
void writestring(char *cmd);
void getposition(long *position);
void readlf(char *result);
void writechr(int c);
char readchr(void);
int  fault(void);
void home(void);
```

```
long inbit(void);
void inpos(char move[]);
long power_16(int power);
void remote(char mode);
void system(char *);
int  strlen(char *);
char *strtok(char *, char *);

void main(void)
{
long lpoint[4];                    /* array of all four axis pos's */

  system("cls");                   /* clear PC's screen */
  init_sys();                      /* set some defaults */
/*    home();                        /* send all axis home */
  sendfile();                      /* send file to Unidex 14 */
  interact();                      /* interactive mode */
  printf("\nInputs = %li",inbit());     /* display input states */
  if(fault())                      /* test for drive faults */
    printf("\nERROR drive fault exists !");
  getposition(lpoint);             /* update the absolute position's */
  printf("\n X axis position = %li",lpoint[0]);
  printf("\n Y axis position = %li",lpoint[1]);
  printf("\n Z axis position = %li",lpoint[2]);
  printf("\n T axis position = %li",lpoint[3]);
}

void sendfile(void)    /* Open file and send it to Unidex 14 */
{
  FILE   *fp;
  char x;

  printf("\nName of Unidex 14 command file to download (< CR > for no file):");
  gets(fname);
  if(strlen(fname)  0)
  {
    if ((fp = fopen(fname, "r")) = = NULL)
      printf("\nCan't open %s", fname);
    else
    {
      while(feof(fp) = = 0)      /* while not end of file */
      {

        putcm(fgetc(fp)) ; /* get a char from file, send it to Unidex 14 */
        while((x = getcm()) ! = NULL)        /* while a char available from Unidex 14 */
        {
```

```
            putch(x);      /* print it */
          }
          checkstatus();
        }
        fclose(fp);        /* close file when done */
        printf("%s transmitted.\n", fname);
      }
    }
  }

void interact(void)  /* Sends keyboard entry to Unidex 14, and displays Unidex 14's output */
{
  char outchar = 0;
  int inchar;

  printf("\nInteractive mode, Enter Unidex 14 commands, Press Control-C to exit.\n");
  while (outchar != CNTRL_C)
  {
    if (kbhit())        /* char available from keyboard ? */
    {
      outchar = getch();     /* get char from keyboard */
      putcm(outchar);        /* send it to Unidex 14 */
      if(outchar == CR)
        putch(LF);     /* add linefeeds when return is pressed to
              avoid overstriking letters on the screen */
    }
    if ((inchar=getcm()) != NULL)         /* char available from Unidex 14 ? */
    {
      putch(inchar);   /* display it */
    }

    checkstatus();
  }
}

void checkstatus(void)  /* this routine decodes Unidex 14's status and displays it's info */
{
extern int status;
int done_flag;
  if ((status & DONE_BIT) != 0)
  {     if ((status & CMDERR_BIT) != 0)       /* command error */
      printf("\nUnidex 14's Status Register indicates a command error\n");
    else
    if ((status & ENC_BIT) != 0)          /* encoder slip */
      printf("\nUnidex 14's Status Register indicates an Encoder Slip error\n");
    else
```

```
    if ((status & OVRT_BIT) != 0)        /* limit switch is active */
      printf("\nUnidex 14's Status Register indicates a limit switch is closed\n");
    else
    {     done_flag = inp(DONE_REG);
      if(done_flag != 0)          /* if an axis finished, display it */
      {     printf("\nUnidex 14's Status Register indicates a DONE flag has been set\n");
        printf("AXES DONE: ");
        if(done_flag % 2 == 1)
          printf("x ");
        done_flag = done_flag / 2;
        if(done_flag % 2 == 1)
          printf("y ");
        done_flag = done_flag / 2;
        if(done_flag % 2 == 1)
          printf("z ");
        done_flag = done_flag / 2;
        if(done_flag % 2 == 1)
          printf("t ");
        printf("\n");
        done_flag = 0;
      }
    }
  putcm(24);  /* clr. error status */
  }
}

char  getcm(void)  /* Returns character from Unidex 14 */
{
extern int status;
  status = inp(STATUS_REG);     /* get U14's status */
  if((inp(STATUS_REG) & IBF_BIT))       /* if char available in buffer */
    return (inp(DATA_REG));       /* return it, */
  else                          /* else */
    return (0);                 /* return 0 */
}

void putcm(char c) /* Put character into Unidex 14's buffer */
{

  while((inp(STATUS_REG) & TBE_BIT) == 0)  /* wait till Unidex 14's ready */
    ;
  outp(DATA_REG,c);                   /* write the chr. */
}

void home(void)  /* this function will hardware home all axis */
{
```

```
long result;
  writestring("BL13;WT9;BH13;");        /* send all axis home */
  printf("\nHoming");
  writestring ("AX");                   /* set x axis active */
  for(result=0; result < 100; result++)  /* wait for home to start */
    {}
  result = inbit();                     /* read the inputs */
  while ((result & 32) != 0)            /* wait for home completion */
    result = inbit();                   /* read the inputs */
  writestring ("ATLP0;");               /* set t axis pos. 0 */
  writestring ("AZLP0;");               /* set z axis pos. 0 */
  writestring ("AYLP0;");               /* set y axis pos. 0 */
  writestring ("AXLP0;");               /* set x axis pos. 0 */
  printf("\nAt Home");
}

long inbit(void)       /* this function returns the state of the inputs */
{
int x;
long accum=0;                           /* decimal equiv. accumulator */
char result[20];                        /* init. string */
  writestring("AXBX");                  /* ask for input bit states */
  readlf(result);                       /* input the hex-ascii values */
  for(x=0; x < 5; x++)
    {  if (result[x] = 'A')             /* is it 'A' thru 'F' ? */
      result[x] = result[x]-7; /* pre-scale hex values */
    accum = accum + (power_16(5-x) * (result[x]-48)); /* calc. base ten value */
    }
    return(accum);                      /* return inputs in dec. */
}

long power_16(int power)  /* this function returns 16 raised to the power of
        power. This is called by the inbit function only ! */
{
int i;
long p=1;
  for(i=1; i < power; i++)
    p = p * 16;
  return(p);
}

int fault(void)  /* this function returns true '1' if any drive fault exists,
       on any servo axis, stepper drives do not have a fault
       output to Unidex 14. */
{
  if((inbit() & 64) == 0)       /* read the inputs */
```

```
    return(0);          /* drive fault not present */
  else
    return(1);          /* drive fault present */
}

void inpos(char move[255]) /* this function starts a move, then waits for all
        servos, & steppers with 1MR bds. to come to rest.
      the move cmd string is expected to include a "go" cmd */
{
  writestring(move);                /* start the move */
  printf("\nWaiting for drives to come to rest");
  while(!(inbit() & 128))            /* wait for move to start */
    ;
  while(inbit() & 128)              /* wait for move to end */
    ;
}

void remote(char mode) /* this function controls the remote clk. & dir. inputs */
      /* to the Unidex 14 power chassis. (L is PC control, */
{               /*  R is remote clock & direction */
  if((toupper(mode) = = 'R'))       /* convert mode chr. to upper case */
    writestring ("AXBL12;"); /* set Remote mode */
  else
    writestring ("AXBH12;"); /* set Local mode  */
}

void getposition(long *position)  /* this function returns a pointer to 4 long int's */
{
char result[25];
  writestring("ATRP");         /* request position from Unidex 14 */
  readlf(result);
  position[3] = atol(result);   /* find the t axis position */
  writestring("AZRP");         /* request position from Unidex 14 */
  readlf(result);
  position[2] = atol(result);   /* find the z axis position */
  writestring("AYRP");         /* request position from Unidex 14 */
  readlf(result);
  position[1] = atol(result);   /* find the y axis position */
  writestring("AXRP");         /* request position from Unidex 14 */
  readlf(result);
  position[0] = atol(result);   /* find the x axis position */
}

void init_sys(void)      /* this function initializes Unidex 14 system */
{
int c;
```

```
char result[40];
  writestring("AXEF");          /* kill echo if active */
  for (c=0; c < 767; c++);      /* delay till echo stops */
  {;}
  while ((inp(STATUS_REG) & IBF_BIT)) /* while there's a chr. in buffer */
    c = getcm();          /* input it */
  remote('L');              /* disable remote clk. & dir. */
  writestring ("WY");           /* get Unidex 14 version number */
  readlf(result);           /* input version number */
  printf ("Version # is %s\n",result);
/*    writestring ("AA PA1,1,1,1;"); /* set high i mode during moves */
  writestring ("AXBH13;");      /* reset home cycle pulse */
}

void writestring(char cmd[255]) /* this function writes a string to Unidex 14 */
{
int x = 0;                      /* set initial value  */
  putcm('\x0d');                /* write a leading CR */
  while (cmd[x] != '\0')        /* loop till all chr's sent  */
  {     putcm(cmd[x++]);        /* write a chr. to Unidex 14 */
    checkstatus();          /* check Unidex 14's status register */
  }
}

char readchr(void)  /* wait for a valid chr. and then return it */
{
extern int status;
char chr = 0;
  status = inp(STATUS_REG);     /* get U14's status */
  while(chr == NULL)            /* read till valid chr. */
    chr = getcm();          /* get a chr. from Unidex 14 */
  return(chr);
}

void readstring(int cnt, char *result)  /* read cnt number of chr's and modify pointer to string */
{
int x;

  for(x=0; x==(cnt-1); x++);            /* read cnt number of chr's */
    {   result[x] = readchr();      /* get a valid chr. */
    }
  result[x++] = '\0';               /* add NULL terminator */
}

void readlf(char *result)  /* read chr's till a LF is received then begin inputting string until CR LF */
{
```

```
int temp = 0;

  while(readchr() != CR)            /* read till leading CR */
    ;
  while((result[temp] = readchr()) != '\n') /* read till LF terminator */
   {   temp++;                      /* build chr. string */
   }
  result[temp] = '\0';              /* add terminator */
  while(readchr() != '\r')          /* read till LF terminator */
    ;
}
```

SECTION 9-7: C DEMONSTRATION PROGRAM WITH DMA

```
/* PROGRAM U14_CDMA.C              Version 1.01
  This program allows the user to interact with Unidex 14.

  It includes a routine to test the DMA link from the host computer to
  Unidex 14. This requires that a jumper be placed on the DRQ1 and
  DACK1 pins of JP26 on Unidex 14.

  Portions of this program may be adapted by Unidex 14 users for
  application programs. No license is required.

  This program was written using Microsoft C 5.1 Optimizing Compiler.

  This version released Aug. 27, 1991
*/

#include <stdio.h
#include <dos.h>
#include  <direct.h>

#define DATA_REG    0x300    /* Unidex 14 I/O port definitions */
#define DONE_REG    0x301
#define CONTROL_REG 0x302
#define STATUS_REG  0x303

#define CMDERR_BIT  0x1   /* CONTROL and STATUS register bit definitions */
#define ENC_BIT     0x4
#define OVRT_BIT    0x8
#define DONE_BIT  0x10
#define IBF_BIT   0x20
```

```
#define TBE_BIT  0x40
#define IRQ_BIT  0x80

#define DMASTAT 0x8     /* DMA definitions */
#define MASKREG 0x0a
#define DMAMODE 0x0b    /* dma mode register */
#define BPFF  0x0c
#define ADRREG 0X2      /* address register channel 1 */
#define COUNT  0x3      /* byte counter register channel 1 */
#define PGREG  0x83     /* page register channel 1 */
#define DMAEN  0x2
#define DMARD  0x1

#define ICNT   0x20             /* interrupt terminator for 8259 */
#define IMREG  0x21             /* interrupt mask register */
#define ILEVEL  0x20            /* interrupt mask level 5 */

#define CNTRLC  3       /* character definitions */
#define CLEARD 0x18     /* Control-X */
#define LF    10
#define CR    13

#define BSIZE  1000     /* buffer size */

void checkstatus(void);         /* function prototypes */
long atol(char *);
char toupper(char);
void system(char *);
int  strlen(char *);
char *strtok(char *, char *);
void home(void);
long inbit(void);
long power_16(int power);
int  fault(void);
void inpos(char move[]);
void remote(char mode);
void init_sys(void);
void writestring(char cmd[]);
void getposition(long *position);
char readchr(void);
void readlf(char *result);
void sendfile(void);
interrupt far service();
void dma(int count, char *buffer);
void initdma(int count, long address);
void interact(void);
```

```
char getcm(void);
void putcm(char c);
int  mhit(void);

int  inxti, inxto, onxti, onxto;    /* buffer pointers */
char inbuf[BSIZE], outbuf[BSIZE];   /* buffers */
char done_flag, error_flag;
char fname[80];
FILE *fp, *fopen();

/* sample command string that is sent to Unidex 14 via DMA*/

#define MSG "en wy rp mr100000 go wy rp"
char buffer[] = MSG;
int count = sizeof(buffer);

void main(void)
{
long lpoint[4];
  system ("cls");              /* clear the screen */
  printf("\nThis program includes a routine to test the DMA link from the host\
 computer\nto Unidex 14.  This requires that a jumper be placed on the DRQ1\
 and DACK1\npins of JP26 on Unidex 14.  It does not use DMA to interact with\
 the board.\nIt uses interrupts and I/O ports, the same as the program U14_INTR.C does.\n");
  inxti = inxto = onxti = onxto = 0;  /* buffers empty */
  init( service );      /* init interrupts */
  init_sys();          /* set some defaults */
  dma( count, buffer );  /*send sample string to Unidex 14 board via DMA */
/*      home();              /* send all axis home */
  interact();          /* go into interactive mode */
  printf("\nInput's = %li",inbit());     /* display input states */
  if(fault())
    printf("\nERROR drive fault exists !");
  printf("Drives are now In Position");
  getposition(lpoint);   /* update the absolute positions */
  printf("\n X axis position = %li", lpoint[0]);
  printf("\n Y axis position = %li", lpoint[1]);
  printf("\n Z axis position = %li", lpoint[2]);
  printf("\n T axis position = %li", lpoint[3]);
}

void sendfile(void)    /* Open file and send it to Unidex 14 */
{
  FILE   *fp;
  char x;
```

```
    printf("\nName of Unidex 14 command file to download (<CR> for no file):");
    gets(fname);
    if(strlen(fname) > 0)
    {   if ((fp = fopen(fname, "r")) == NULL)
          printf("\nCan't open %s", fname);
        else
        {   while(feof(fp) == 0)    /* while not end of file */
            {   putcm(fgetc(fp)) ; /* get a char from file, send it to Unidex 14 */
              /* band aid - if DOS turns interrupt off, turn it back on */
              outp(IMREG, inp(IMREG) & ~ILEVEL);
              while((x = getcm()) != NULL)      /* while a char available from Unidex 14 */
              {   putch(x);    /* print it */
              }
              checkstatus();
            }
            fclose(fp);       /* close file when done */
            printf("%s transmitted.\n", fname);
        }
    }
}

void interact(void) /* Sends keyboard entry to Unidex 14 and display its response */
{
    printf("\nInteractive mode, Enter Unidex 14 commands, Press Control-C to exit.\n");
    for (;;)
    {   if ( kbhit() )          /* char available from keyboard? */
          putcm( getch() );  /* send it to Unidex 14 */
        if ( mhit() )          /* char available from Unidex 14 ? */
          putch( getcm() );  /* get it and print it */
        checkstatus();          /* check U14's status register */
    }
}

void checkstatus(void)  /* this routine decodes Unidex 14's status and displays it's info */
{
extern char done_flag, error_flag;

    if (error_flag != 0)
    {   if ((error_flag & CMDERR_BIT) != 0)      /* command error */
          printf("\nUnidex 14's Status Register indicates a command error\n");
        else
        if ((error_flag & ENC_BIT) != 0)      /* encoder slip */
          printf("\nUnidex 14's Status Register indicates an Encoder Slip error\n");
        else
        if ((error_flag & OVRT_BIT) != 0)      /* limit switch is active */
          printf("\nUnidex 14's Status Register indicates a limit switch is closed\n");
```

```
    error_flag = 0; /* clear error_flag */
  }

  if(done_flag != 0) /* an ID has been encountered, print the done axis */
  {     printf("\nUnidex 14's Status Register indicates a DONE flag has been set\n");
    printf("AXES DONE: ");
    if(done_flag % 2 == 1)
      printf("x ");
    done_flag = done_flag / 2;
    if(done_flag % 2 == 1)
      printf("y ");
    done_flag = done_flag / 2;
    if(done_flag % 2 == 1)
      printf("z ");
    done_flag = done_flag / 2;
    if(done_flag % 2 == 1)
      printf("t ");
    printf("\nPress Control-Y, or type IC to clear Unidex 14's done status.\n");
    done_flag = 0; /* clear done flag */
  }
}

char getcm(void)      /* Returns character from input buffer */
{
char c;
  if(inxti == inxto)          /* no char available in buffer */
    return (0);
  else
  {     c = inbuf[inxto++];   /* get char. and  incr. pointer */
    if(inxto == BSIZE)    /* if end of buffer */
      inxto = 0;    /* reset pointer */
    outp(CONTROL_REG, inp(CONTROL_REG) | IBF_BIT); /* enable U14 intr. */
    return (c);
  }
}

void putcm(char c)    /* Put character into output buffer */
{
int  i, j;

  i = onxti;
  j = i + 1;
  if( j == BSIZE )
    j = 0;
  while( j == onxto )    /* wait till there's room for another chr.*/
    ;
```

```
  outbuf[i++] = c;
  if( i == BSIZE )
    i = 0;
  onxti = i;
  outp( CONTROL_REG, inp( CONTROL_REG ) | TBE_BIT );
}

int mhit(void)      /* returns non zero if input buffer has character */
{
  return(inxto - inxti);
}

interrupt far service()  /* interrupt driven routine to communicate with Unidex 14 */
{
char  c;
int  i;

  c = inp(STATUS_REG) & (inp(CONTROL_REG) | 0x0f);
  /* get the status of interrupts that are enabled */

  if(c & DONE_BIT)         /* if the DONE bit is true */
  {     done_flag |= inp(DONE_REG);  /* save done axis flags */
    error_flag |= (c & 0x0f);    /* save error bits also */
    outp(DATA_REG, CLEARD);      /* reset Unidex 14 status register DONE bit
            without clearing flags shown in RA or
            RI commands which ^Y does */
  }
  else
  if((c & TBE_BIT) == TBE_BIT) /* if Unidex 14 will accept a char */
  {     if(onxto != onxti)  /* do we have a char to send to Unidex 14 ? */
    {     outp(DATA_REG, outbuf[onxto]); /* send char to Unidex 14 */
      if(++onxto == BSIZE)     /* take care of wrap around */
        onxto = 0;
      if(onxto == onxti)     /* if the buffer is now empty */
      /* turn off Unidex 14 interrupt so it won't keep asking for chars */
        outp(CONTROL_REG, inp(CONTROL_REG) & ~TBE_BIT);
    }
    else    /* no char in buffer to send to Unidex 14 */
        /* turn off Unidex 14's interrupt so it won't keep asking for chars */
      outp(CONTROL_REG, inp(CONTROL_REG) & ~TBE_BIT);
  }
  if((c & IBF_BIT) == IBF_BIT) /* if Unidex 14 has a char to send to the PC */
  {
    inbuf[inxti] = inp(DATA_REG); /* get the char and put it in the buffer */
    if(++inxti == BSIZE)     /* take care of wrap around */
      inxti = 0;
```

```
    if((i = inxti + 1) = = BSIZE) /* room for another char in the buffer? */
      i = 0;
    if(i = = inxto)
      outp(CONTROL_REG, inp(CONTROL_REG) & ~IBF_BIT);
      /* no room in buffer, turn off interrupt so Unidex 14 cannot send more */
  }
  outp(ICNT, 0x20);        /* send 8259 chip a return from interrupt */

  /* PC/XT/AT interrupt control is edge sensitive only.
  The following generates a rising edge if a second interrupt has occured
  prior to the first being reset. This can happen if more than one
  source is enabled on the Unidex 14 as in this program. */

  c = inp(CONTROL_REG);
  outp(CONTROL_REG, c & ~IRQ_BIT);      /* turn off interrupts */
  outp(CONTROL_REG, c);          /* and back on to create edge */
}

init(intvec) /* connect hardware interrupt IRQ5 to interrupt handler */
int (*intvec)();
{
  extern int   intdosx();
  union REGS   inregs, outregs;
  struct SREGS  segregs;

  inregs.h.ah = 0x25;          /* set interrupt vector DOS call */
  inregs.h.al = 13;           /* interrupt vector level 5 */
  segregs.ds = FP_SEG(*intvec);  /* send the vector */
  inregs.x.dx = FP_OFF(*intvec);
  intdosx(&inregs, &outregs, &segregs);
  outp(IMREG, inp(IMREG) & ~ILEVEL);   /* enable interrupts */
  outp(CONTROL_REG, IBF_BIT | DONE_BIT | IRQ_BIT); /* turn on Unidex 14's interrupts */
}

void dma(int count, char *buffer)
{
  long  addr;
  char  far  *message;

  message = buffer;
  addr = FP_SEG( message );
  addr = ( addr < < 4 ) + FP_OFF( message );
  printf( "\n%x, %lx, %s\n", count, addr, buffer );
  initdma( count, addr );
}
```

```
void initdma(int count, long address)
{
  outp( MASKREG, 5 );       /* turn off channel 1 */
  outp( CONTROL_REG, inp( CONTROL_REG ) & ~ TBE_BIT );/*TBE_BIT interrupts off*/
  outp( DMAMODE, 0x49 );    /* single transfers, read memory */
  outp( BPFF, 0);     /* reset byte pointer F/F */
  outp( ADRREG, address & 0xff );  /* low order byte of address */
  outp( ADRREG, (address & 0xff00) > > 8 );
  outp( PGREG, (address & 0xf0000) > > 16 );
  count - = 1;        /* dma transfers one more than count */
  outp( COUNT, count & 0xff );       /* low order of byte count */
  outp( COUNT, (count & 0xff00) > > 8 );
  outp( CONTROL_REG, inp( CONTROL_REG ) & ~ DMARD); /* prepare for writing */
  outp( CONTROL_REG, inp( CONTROL_REG ) | DMAEN ); /* enable the controller*/
  outp( MASKREG, 1 );       /* turn on channel 1 */
  while( (inp( DMASTAT ) & 0x2) = = 0 )
    ;         /* wait for terminal count */
  outp( MASKREG, 5 );       /* turn off channel 1 */
  outp( CONTROL_REG, inp( CONTROL_REG ) & ~ DMAEN ); /* and turn off dma */
}

void home(void)  /* this function will hardware home all axis */
{
long result;
  writestring("BL13;WT9;BH13;");       /* send all axis home */
  printf("\nHoming");
  writestring ("AX");                  /* set x axis active */
  for(result = 0; result < 100; result + + )   /* wait for home to start */
    {}
  result = inbit();                  /* read the inputs */
  while ((result & 32) ! = 0)          /* wait for home completion */
    result = inbit();           /* read the inputs */
  writestring ("ATLP0;");              /* set t axis pos. 0 */
  writestring ("AZLP0;");              /* set z axis pos. 0 */
  writestring ("AYLP0;");              /* set y axis pos. 0 */
  writestring ("AXLP0;");              /* set x axis pos. 0 */
  printf("\nAt Home");
}

long inbit(void)       /* this function returns the state of the inputs */
{
int x;
long accum = 0;                    /* decimal equiv. accumulator */
char result[20];                  /* init. string */
  writestring("AXBX");         /* ask for input bit states */
  readlf(result);             /* input the hex-ascii values */
```

```
  for(x=0; x < 5; x++)
    {   if (result[x] = >'A')          /* is it 'A' thru 'F' ?  */
      result[x] = result[x]-7;  /* pre-scale hex values */
    accum = accum + (power_16(5-x) * (result[x]-48)); /* calc. base ten value */
    }
    return(accum);                  /* return inputs in dec. */
}

long power_16(int power)  /* this function returns 16 raised to the power of
        power. This is called by the inbit function only ! */
{
int i;
long p=1;
  for(i=1; i < power; i++)
    p = p * 16;
  return(p);
}

int fault(void)  /* this function returns true '1' if any drive fault exists,
       on any servo axis, stepper drives do not have a fault
       output to Unidex 14. */
{
  if((inbit() & 64) = = 0)       /* read the inputs */
    return(0);          /* drive fault not present */
  else
    return(1);          /* drive fault present */
}

void inpos(char move[255])  /* this function starts a move, then waits for all
         servos, & steppers with 1MR bds. to come to rest.
      the move cmd string is expected to include a "go" cmd */
{
  writestring(move);                /* start the move */
  printf("\nWaiting for drives to come to rest");
  while(!(inbit() & 128))             /* wait for move to start */
    ;
  while(inbit() & 128)                /* wait for move to end */
    ;
}

void remote(char mode)  /* this function controls the remote clk. & dir. inputs */
     /* to the Unidex 14 power chassis. (L is PC control, */
{                /*  R is remote clock & direction */
  if((toupper(mode) = = 'R'))       /* convert mode chr. to upper case */
    writestring ("AXBL12;");  /* set Remote mode */
  else
```

```
    writestring ("AXBH12;");  /* set Local mode  */
}

void getposition(long *position)  /* this function returns a pointer to 4 long int's */
{
char result[25];
  writestring("ATRP");          /* request position from Unidex 14 */
  readlf(result);
  position[3] = atol(result);    /* find the t axis position */
  writestring("AZRP");          /* request position from Unidex 14 */
  readlf(result);
  position[2] = atol(result);    /* find the z axis position */
  writestring("AYRP");          /* request position from Unidex 14 */
  readlf(result);
  position[1] = atol(result);    /* find the y axis position */
  writestring("AXRP");          /* request position from Unidex 14 */
  readlf(result);
  position[0] = atol(result);    /* find the x axis position */
}

void init_sys(void)      /* this function initializes Unidex 14 system */
{
int c;
char result[40];
  writestring("AXEF");          /* kill echo if active */
  for (c=0; c < 3000; c++);       /* delay till echo stops */
  {;}
  while ((inp(STATUS_REG) & IBF_BIT)) /* while there's a chr. in buffer */
    c = getcm();          /* input it */
  remote('L');               /* disable remote clk. & dir. */
  writestring ("WY");           /* get Unidex 14 version number */
  readlf(result);             /* input version number */
  printf ("\nVersion # is %s\n",result);
/*    writestring ("AAPA1,1,1,1;");  /* set high i mode during moves */
  writestring ("AXBH13;");      /* reset home cycle pulse */
}

void writestring(char cmd[255]) /* this function writes a string to Unidex 14 */
{
int x = 0;                    /* set initial value  */
  putcm('\x0d');              /* write a leading CR */
  while (cmd[x] != '\0')        /* loop till all chr's sent  */
  {    putcm(cmd[x++]);      /* write a chr. to Unidex 14 */
    checkstatus();        /* check Unidex 14's status register */
  }
}
```

```
char readchr(void)  /* wait for a valid chr. and then return it */
{
char chr = 0;
  while(chr = = NULL)          /* read till valid chr. */
    chr = getcm();       /* get a chr. from Unidex 14 */
  return(chr);
}

void readstring(int cnt, char *result)  /* read cnt number of chr's and modify pointer to string */
{
int x;

  for(x=0; x= =(cnt-1); x+ +);          /* read cnt number of chr's */
  {   result[x] = readchr();         /* get a valid chr. */
  }
  result[x+ +] = '\0';               /* add NULL terminator */
}

void readlf(char *result)  /* read chr's till a LF is received then begin inputting string until CR LF */
{
int temp = 0;

  while(readchr() != CR)             /* read till leading CR */
    ;
  while((result[temp] = readchr()) != '\n') /* read till LF terminator */
  {   temp+ +;                       /* build chr. string */
  }
  result[temp] = '\0';               /* add terminator */
  while(readchr() != '\r')           /* read till LF terminator */
    ;
}
```

SECTION 9-8: JOG.C

```
/* PROGRAM JOG.C            Version 1.01

   This program allows the user to interact with the Unidex 14
  motion control board.  It uses the PC's interrupt line IRQ5 to
  communicate between the PC and Unidex 14.  This program will have to
  be restarted if an RS (reset) command is sent to Unidex 14 because
  the interrupt control registers on the board are also reset causing
  the board to no longer create interrupts.

  This program uses the computer's cursor keys to control the movement
  of two axes. The default are X and Y but the user may change this.

  Portions of this program may be adapted by Unidex 14 users for
  application programs.  No license is required.

  This program was written using Microsoft C 5.1 Compiler.

  RELEASED Aug. 27, 1991
*/


#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <bios.h>
#include <time.h>
#include <signal.h>
#include <graph.h>

/* Unidex 14 I/O port definitions */
#define DATA_REG      0x300
#define DONE_REG      0x301
#define CONTROL_REG   0x302
#define STATUS_REG    0x303

/* CONTROL and STATUS register bit definitions */
#define CMDERR_BIT 0x1
#define INIT_BIT   0x2
#define ENC_BIT    0x4
#define OVRT_BIT   0x8
#define DONE_BIT   0x10
#define IBF_BIT    0x20
#define TBE_BIT    0x40
```

```
#define IRQ_BIT    0x80

/* PC interrupt definitions */
#define ICNT       0x20        /* interrupt terminator byte for 8259 */
#define IMREG      0x21        /* interrupt mask register I/O address*/
#define ILEVEL     0x20        /* interrupt mask level 5 byte*/

/* character definitions */
#define CNTRL_C  3
#define CNTRL_Z  26
#define CLEARD   24
#define LF       10
#define CR       13

#define BSIZE    1000          /* input and output buffer size */

int    inxti, inxto, onxti, onxto; /* input and output buffer pointers */

char  udaxis = 'y'; /*default axis controlled by up and down arrow keys*/
char  lraxis = 'x'; /*default axis controlled by left and right arrow keys*/
char  left = '-';   /*default step direction controlled by left arrow key*/
char  right = '+';  /*default step direction controlled by right arrow key*/
char  up = '+';     /*default step direction controlled by up arrow key*/
char  down = '-';   /*default step direction controlled by down arrow key*/
char  joyexitmode = 'x';  /*axis mode when leaving joykey mode*/
long int stepvalue = 200; /*amount speed changes is multiplied by when a cursor key is pressed*/
long int speed = 200;  /*current speed of joykey jog */
char *cmdstring = "joy keys not yet used"; /*joy key Unidex 14 command string*/

char   done_flag, error_flag;
char   inbuf[ BSIZE ], outbuf[ BSIZE ]; /* define buffers arrays */
char   fname[80];                  /* file name buffer */
int    exit_handler(); /*traps control-C*/
char   getc_u14();
int keyboard(); /*this routine decodes function and cursor keys and send commands to Unidex 14 */
interrupt  far service();
unsigned char   sameas();  /*returns complement cntrol or non-contol key for cursor keys*/

main()
{
  char inchar;
  inxti = inxto = onxti = onxto = 0;    /* set buffer pointers to beginning*/
  done_flag = 0;                  /* clear done flag variable*/
  error_flag = 0;                 /* clear error flag variable*/

  _clearscreen(_GCLEARSCREEN);
```

```
  printf("This program allows the user to control Unidex 14.\n");

  init(service);     /* initialize interrupts */
  if(signal (SIGINT,exit_handler) = = (int (*) () ) -1 ) /*enable Cntrl-C trapping*/
  {
    printf("Could not set up Control-C exit routine\n");
    abort();
  }

/*      sendfile(); /* get file name, open file, send it to Unidex 14 */

  /*display anything returned from Unidex 14 resulting from the file sent to it*/
  while(mhit() != 0)        /* char available from Unidex 14 ? */
  {
    inchar = getc_u14();  /* get char */
    putch(inchar);   /*display it */
  }
  interact(); /* interactive mode */
}

exit_handler()
{
  char ch;

  signal(SIGINT, SIG_IGN);  /*turn off Control-C trapping while in this routine*/
  printf("\nPress y, Y or Esc to kill all axes' motors, any other key to continue.\n");
  ch = getch();
  if((ch = = 'y') || (ch = = 'Y') || (ch = = 27) )
  {
        putc_u14(' '); /*stop all axis*/
        putc_u14('k');
        putc_u14('l');
  }
  printf("Press y, Y or Esc to stop this program, any other key to continue.\n");
  ch = getch();
  putch('\n');
  if((ch = = 'y') || (ch = = 'Y') || (ch = = 27))
  {
    exit(0);
  }
  signal(SIGINT,exit_handler); /*turn on Control-C trapping*/
}

sendfile()    /* Open file and send it to Unidex 14 */
{
  FILE   *fp;
```

```
  printf("Enter name of Unidex 14 command file to download. (
 for no file)\n");
  gets(fname);
  if(strlen(fname)  > 0)
  {
    if ((fp = fopen(fname, "r")) = = NULL)
      printf("Can't open %s\n", fname);
    else
    {
      while(feof(fp) = = 0)      /* while not end of file */
      {
        putc_u14(fgetc(fp)) ; /* get a char from file, send it to Unidex 14*/
        /* band aid - if DOS turns interrupt off, turn it back on */
        outp(IMREG, inp(IMREG) & ~ILEVEL);
        while(mhit())         /* while a char available from Unidex 14 */
        {
          putch(getc_u14());      /* print it*/
        }
        checkstatus();
      }
      fclose(fp);         /* close file when done */
      printf("The file \"%s\" has been sent to Unidex 14.\n", fname);
    }
  }
}


interact()  /* Sends keyboard entry to Unidex 14 and displays any response's */
{
  int x;
  int outchar = 0, inchar;

  printf("\nEnter Unidex 14 commands. Press F1 for help. Use cursor keys to control movement.\n");
  printf("If double characters are displayed enter \"ax ef\" to turn off Unidex 14's  echo.\n");
  printf("Esc, ^Z or ^C to exit.\n\n");
  while (outchar != CNTRL_Z)
  {
    if (kbhit())        /* char available from keyboard? */
    {
      outchar = getch(); /* get char from keyboard */
      if(outchar = = 27) /*if ESC is pressed ask if user wants to exit*/
        exit_handler();
      else
      if(outchar = = 0)  /*if a function or cursor key has been pressed*/
      {
```

```
        keyboard();   /*go to keyboard handler*/
      }
      else
      {
        putch(outchar); /*echo character to screen*/
        putc_u14(outchar);  /* send it to Unidex 14 */
      }
      if(outchar = = CR)
        putch(LF);    /* add linefeeds when return is pressed to
            avoid overstriking letters on the screen */
    }

    if (mhit())         /* char available from Unidex 14 ? */
    {
      inchar = getc_u14();  /* get char */
      putch(inchar);   /*display it */
    }

    checkstatus(); /*see if any errors occured or if done flag was set*/
  }
}

help()
{
  _clearscreen(_GCLEARSCREEN);
  printf("SPECIAL PURPOSE KEYS ARE PROGRAMMED AS FOLLOWS:\n");
  printf("FUNCTION KEYS \(keys not specified are not used\)\n");
  printf("F1-This message is displayed.\n");
  printf("F2-sends U14 command \"rp\" to the board and displays values returned.\n");
  printf("F9-displays information about current programming of cursor keys.\n");
  printf("F10-allows user to change programmed value of cursor keys.\n");
  printf("\nCURSOR KEYS\n");
  printf("HOME (7)-Jog both axis defined by the left (4) and up (8) keys at the same\
\n       time, at the same speed, resulting in a diagonal move.\n");
  printf("Up-arrow (8)-Jog its predefined axis in its predefined direction.\n");
  printf("          Power up defaults are Y axis and + direction.\n");
  printf("Pg Up (9)-Jog both axis defined by the right (6) and up (8) keys at the same\
\n       time, at the same speed, resulting in a diagonal move.\n");
  printf("Left-arrow (4)-Jog its predefined axis in its predefined direction.\n");
  printf("          Power up defaults are X axis and - direction.\n");
  printf("Right-arrow (6)-Jog its predefined axis in its predefined direction.\n");
  printf("          Power up defaults are X axis and + direction.\n");
  printf("End (1)-Jog both axis defined by the left(4) and down (2) keys at the same\
\n       time, at the same speed, resulting in a diagonal move.\n");
  printf("Down-arrow (2)-Jog its predefined axis in its predefined direction.\n");
  printf("          Power up defaults are Y axis and - direction.\n");
```

```
  printf("Pg Dn (3)-Jog both axis defined by the right(4) and down(2) keys at the same\
\n       time, at the same speed, resulting in a diagonal move.\n");
  printf("PRESS A KEY TO CONTINUE");
  getch();
  _clearscreen(_GCLEARSCREEN);
  printf("The first time a cursor key is pressed the jog speed is the programmed speed.\n");
  printf("Each time the same key is pressed or if the key is held down the speed will\n");
  printf("increase by the programmed step value.  Pressing any other key will stop\
\nmovement by sending an \"sa\" command to Unidex 14.\n\n");
  printf("If the CONTROL key is held down while tapping the same cursor key \n");
  printf("the speed is reduced by the step value. \n");
  printf("\nOTHER SPECIAL KEYS\n");
  printf("Ins (0)-Sends an \"sa\" command to Unidex 14, causing all axis to\
\n      decelerate to a stop.\n");
  printf("Del (.)-Sends a \"kl\" command to Unidex 14, causing all axis to\
\n      stop instantly.\n");
  printf("Esc - Asks user if they want to stop all motor movement and if they want\
\n    to terminate the program and performs these things if commanded to.\n");
  printf("Control-C - Same function as Esc key.\n");
  printf("Control-Z - Terminates program immediately, leaves Unidex 14 running.\n");
  printf("\nEND OF HELP MESSAGE\n");
}

changejoyinfo()
{
  extern long int stepvalue; /*the size of the jumps in velocity when a cursor key is pressed*/
  extern long int speed; /*jog speed for joy keys*/
  extern char  udaxis;  /*default axis controlled by up and down arrow keys*/
  extern char  lraxis;  /*default axis controlled by left and right arrow keys*/
  extern char  left;    /*default step direction controlled by left arrow key*/
  extern char  right;   /*default step direction controlled by right arrow key*/
  extern char  up;      /*default step direction controlled by up arrow key*/
  extern char  down;    /*default step direction controlled by down arrow key*/
  int ok,result;
  long int tempstep;
  char *string;
  char input,yn;

  printf("CHANGE CURSOR KEY PROGRAMMING\n");

  /*change step speed*/
  ok = 0;
  while(ok = = 0)
  {
    printf("\nCurrently each time a cursor key is pressed the jog speed changes by: %ld\n",stepvalue);
```

```
    gets(string);
    if(strlen(string) != 0)
    {
      tempstep = atol(string);
      if( (tempstep > (long) 0 ) && (tempstep < 524000))
      {
        ok = 1;
        stepvalue = tempstep;
      }
      else
        printf("Step speed must be an integer greater that 0 and less than 524000.\n");
    }
    else
    {
      printf("Step speed not changed\n");
      ok = 1;
    }
  }

  /*change left-right axis*/
  ok = 0;
  while(ok = = 0)
  {
    printf("\nCurrent axis controlled by left (4) and right (6) cursor keys: %c\n",lraxis);
    printf("Legal values are: x,y,z,t\n");
    printf("\n");
    printf("\n");
    printf("Axis to be controlled by left and right cursor keys (<CR> for no change):");
    input = getche();
    printf("\n");
    if(strchr("xyztXYZT",input) != 0)
    {
      ok = 1;
      lraxis = input;
    }
    else
    if(input < 0x20) /*a <CR> or <LF>or or other control char was entered so no change*/
      ok = 1;
    else
      printf("Illegal axis given\a\n");
  }

  /* change left direction */
  ok = 0;
  while(ok = = 0)
  {
```

```
printf("\nCurrent stepping direction controlled by left (4) cursor key: %c\n",left);
  printf("Legal directions are: + and - \n");
  printf("Enter direction to be controlled by left cursor key (<CR> for no change):");
  input = getche();
  printf("\n");
  if(strchr("+-",input) != 0)
  {
    ok = 1;
    left = input;
  }
  else
  if(input < 0x20) /*a <CR> or <LF> or  or other control char was entered so no change*/
    ok = 1;
  else
    printf("Illegal direction given\a\n");
}
/* change right direction */
ok = 0;
while(ok == 0)
{
  printf("\nCurrent stepping direction controlled by right (6) cursor key: %c\n",right);
  printf("Legal directions are: + and - \n");
  printf("Enter direction to be controlled by right cursor key (<CR> for no change):");
  input = getche();
  printf("\n");
  if(strchr("+-",input) != 0)
  {
    ok = 1;
    right = input;
  }
  else
  if(input < 0x20) /*a  <CR> or <LF> or  or other control char was entered so no change*/
    ok = 1;
  else
    printf("Illegal direction given\a\n");
}

/*change up-down axis*/
ok = 0;
while(ok == 0)
{
  printf("\nCurrent axis controlled by up (8) and down (2) cursor keys: %c\n",udaxis);
  printf("Legal values are: x,y,z,t\n");
  printf("\n");
  printf("\n");
```

```
  printf("Axis to be controlled by up and down cursor keys (<CR>for no change):");
  input = getche();
  printf("\n");
  if(strchr("xyztXYZT",input) != 0)
  {
    ok = 1;
    udaxis = input;
  }
  else
  if(input < 0x20) /*a <CR> or <LF>or or other control char was entered so no change*/
    ok = 1;
  else
    printf("Illegal axis given\a\n");
}

/* change up direction */
ok = 0;
while(ok == 0)
{
  printf("\nCurrent stepping direction controlled by up (8) cursor key: %c\n",up);
  printf("Legal directions are: + and - \n");
  printf("Enter direction to be controlled by up cursor key (<CR>for no change):");
  input = getche();
  printf("\n");
  if(strchr("+-",input) != 0)
  {
    ok = 1;
    up = input;
  }
  else
  if(input < 0x20) /*a <CR> or <LF>or or other control char was entered so no change*/
    ok = 1;
  else
    printf("Illegal direction given\a\n");
}

/* change down direction */
ok = 0;
while(ok == 0)
{
  printf("\nCurrent stepping direction controlled by down (2) cursor key: %c\n",up);
  printf("Legal directions are: + and - \n");
  printf("Enter direction to be controlled by down cursor key (<CR>for no change):");
  input = getche();
  printf("\n");
  if(strchr("+-",input) != 0)
```

```
    {
      ok = 1;
      down = input;
    }
    else
    if(input < 0x20) /*a <CR> or <LF>or or other control char was entered so no change*/
      ok = 1;
    else
      printf("Illegal direction given\a\n");
  }
  printf("\nReturned To Unidex 14 Control\n");

}

keyboard() /*this routine decodes function and cursor keys and sends
    them to Unidex 14 */
{
  int length, letter, joymode, skip = 0;
  extern long int speed;
  unsigned int key,key1,key2,oldkey,opposkey;
  extern char *cmdstring;

  joymode = 0;

  key = getch(); /*get second part of function key code*/

  do  /*while(joymode = = 1)*/
  {
    switch (key)
    {
      case ';':  /* F1 key, display help message*/
        help();
        if(joymode = = 0) /*if not in the joy key mode*/
        {
          skip = 1;
        }
        else
          key = oldkey;
        break;

      case '<':  /* F2 key, /*do an rp*/
        putc_u14(' ');
        putc_u14('r');
        putc_u14('p');
        while (mhit() != 0)        /* char available from Unidex 14 */
        {
```

```
    putch( getc_u14());  /* get char, display it */
  }
  if(joymode = = 0)  /*if not in the joy key mode*/
  {
    skip = 1;
  }
  else
    key = oldkey;
  break;

case 'C':   /* F9 key, display current joykey values*/
  showjoyinfo();
  if(joymode = = 0)  /*if not in the joy key mode*/
  {
    skip = 1;
  }
  else
    key = oldkey;
  break;

case 'D':   /* F10 key, change current joykey values*/
  changejoyinfo();
  if(joymode = = 0)  /*if not in the joy key mode*/
  {
    skip = 1;
  }
  else
    key = oldkey;
  break;

case 'G':  /* home (7) key, move N.W. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld a%c jg%c%ld aa\n",lraxis,left,speed,udaxis,up,speed);
  joyexitmode = 'a';
  break;

case 'H':  /*up arrow (8) key, move N */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld\n",udaxis,up,speed);
  joyexitmode = udaxis;
  break;

case 'I':  /*pg up (9) key, move N.E. */
  joymode = 1;
```

```
  sprintf(cmdstring , " a%c jg%c%ld a%c jg%c%ld aa\n",lraxis,right,speed,udaxis,up,speed);
  joyexitmode = 'a';
  break;

case 'K':  /*left arrow (4) key, move W. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld\n",lraxis,left,speed);
  joyexitmode = lraxis;
  break;

case 'M':  /*right arrow (6) key, move E. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld \n",lraxis,right,speed);
  joyexitmode = lraxis;
  break;

case 'O':  /*end (1) key, move S.W. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld a%c jg%c%ld aa\n",lraxis,left,speed,udaxis,down,speed);
  joyexitmode = 'a';
  break;

case 'P':  /*down arrow (2) key, move S. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld\n",udaxis,down,speed);
  joyexitmode = udaxis;
  break;

case 'Q':  /*pg dn (3) key, move S.E. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld a%c jg%c%ld aa\n",lraxis,right,speed,udaxis,down,speed);
  joyexitmode = 'a';
  break;

case 119:  /* CNTRL-home (7) key, move N.W. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld a%c jg%c%ld aa\n",lraxis,left,speed,udaxis,up,speed);
  joyexitmode = 'a';
  break;
```

```
case 141:  /* CNTRL-up arrow (8) key, move N */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld\n",udaxis,up,speed);
  break;

case 132:  /* CNTRL-pg up (9) key, move N.E. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld a%c jg%c%ld aa\n",lraxis,up,speed,udaxis,right,speed);
  break;

case 115:  /* CNTRL-left arrow (4) key, move W. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld\n",lraxis,left,speed);
  break;

case 116:  /* CNTRL-right arrow (6) key, move E. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld \n",lraxis,right,speed);
  break;

case 117:  /*CNTRL-end (1) key, move S.W. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld a%c jg%c%ld aa\n",lraxis,left,speed,udaxis,down,speed);
  break;

case 145:  /* CNTRL-down arrow (2) key, move S. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld\n",udaxis,down,speed);
  break;

case 118:  /* CNTRL-pg dn (3) key, move S.E. */
  joymode = 1;

  sprintf(cmdstring , " a%c jg%c%ld a%c jg%c%ld aa\n",lraxis,right,speed,udaxis,down,speed);
  break;

case 'R':  /*ins (0) key, stop everything */
  joymode = 1;
  sprintf(cmdstring," sa");
```

```
    break;
  case 'S':  /*del (.) key, stop everything */
    joymode = 1;
    sprintf(cmdstring," kl");
    break;
  default:   /*all other non used function keys */
    joymode = 0;
    putch(key);
    printf("=%d ",key);
    printf("KEY IS NOT USED\n");
    break;
}


if(joymode = = 1)  /*if a cursor control key was pressed*/
{
  /*send string to Unidex 14 */
  length = strlen(cmdstring);
  for(letter = 0 ; letter  < length; + +letter)
  putc_u14(cmdstring[letter]);

  while(!kbhit())
  {
    while (mhit() != 0)        /* char available from Unidex 14 ? */
    {
      putch( getc_u14());  /* get char, display it */
    }
    checkstatus();
  }
  key1 = getch(); /*get another keystroke*/
  if( key1 = = 0) /*if it's a function key*/
  {
    key2 = getch(); /*get the rest of the keycode*/
    opposkey = sameas(key2);
    if((key2 = = key) || (opposkey = = key)) /*if the same key is pressed increase the speed*/
    {
      if( (opposkey  > = 71) && (opposkey  < 81)) /* if we are slowing down*/
      {
        speed - = stepvalue;
      }
      else  /*we are speeding up*/
      {
        speed + = stepvalue;
      }
      if(speed  > 524000)  /*cant go faster than this*/
        speed = 524000;
```

```
        if(speed < 1)  /*cant go slower than this*/
          speed = 0;
        skip = 0;
        key = key2;
      }
      else
      if((key2 > = 59) && (key2 < =68))  /*if its F1 to F10*/
      {
        /*don't change anything and save the last cursor key*/
        oldkey = key;
        key = key2;
        continue;
      }
      else  /* a different cursor control key was pressed*/
      {
        putc_u14(' ');
        putc_u14('s');
        putc_u14('a');
        putc_u14(' '); /*stop all axis */
        putc_u14('a'); /*so it will be prepared to go in any direction*/
        putc_u14(joyexitmode);  /*change default mode when leaving joykeys*/
        key = key2;
        speed = stepvalue;
        skip = 0;
      }

    }
    else  /* a non cursor control, non-function key was pressed return it to main()*/
    {
      if(skip = = 0)
      {
        putc_u14(' ');
        putc_u14('s');
        putc_u14('a');
        putc_u14(' '); /*stop all axis */
        putc_u14('a'); /*so it will be prepared to go in any direction*/
        putc_u14(joyexitmode);  /*go to default mode after leaving joykeys*/
        printf("a%c mode\n",joyexitmode);
      }
      joymode = 0;
      ungetch(key1);
      return;
    }
  }
}
while(joymode = = 1);
```

```
  return;
}

unsigned char sameas(key) /*look-up to find the code for the control or non-control version of cursor key*/
unsigned char key;
{
  unsigned char sameas;


  switch (key)
  {
    case 'G':  /* home (7) key, move N.W. */
      sameas = 119;
      break;

    case 'H':  /*up arrow (8) key, move N */
      sameas = 141;
      break;

    case 'I':  /*pg up (9) key, move N.E. */
      sameas = 132;
      break;

    case 'K':  /*left arrow (4) key, move W. */
      sameas = 115;
      break;

    case 'M':  /*right arrow (6) key, move E. */
      sameas = 116;
      break;

    case 'O':  /*end (1) key, move S.W. */
      sameas = 117;
      break;

    case 'P':  /*down arrow (2) key, move S. */
      sameas = 145;
      break;

    case 'Q':  /*pg dn (3) key, move S.E. */
      sameas = 118;
      break;

    case 119:  /* CNTRL-home (7) key, move N.W. */
      sameas = 'G';
      break;
```

```
    case 141:  /* CNTRL-up arrow (8) key, move N */
      sameas = 'H';
      break;

    case 132:  /* CNTRL-pg up (9) key, move N.E. */
      sameas = 'I';
      break;

    case 115:  /* CNTRL-left arrow (4) key, move W. */
      sameas = 'K';
      break;

    case 116:  /* CNTRL-right arrow (6) key, move E. */
      sameas = 'M';
      break;

    case 117:  /*CNTRL-end (1) key, move S.W. */
      sameas = 'O';
      break;

    case 145:  /* CNTRL-down arrow (2) key, move S. */
      sameas = 'P';
      break;

    case 118:  /* CNTRL-pg dn (3) key, move S.E. */
      sameas = 'Q';
      break;

    case 'R':  /*ins (0) key, stop everything */
      sameas = 146;
      break;

    case 'S':  /*del (.) key, stop everything */
      sameas = 147;
      break;

    default:   /*all keys */
      sameas = 0;
  /*    printf("sameas() passed illegal value:%c\n",key);*/
      break;
   }
   return(sameas);
}

showjoyinfo()
```

```
{
  extern char *cmdstring; /*Unidex 14 commands created by keyboard();
  extern unsigned long int speed; /*jog speed for joy keys*/
  extern char  udaxis;  /*default axis controlled by up and down arrow keys*/
  extern char  lraxis;  /*default axis controlled by left and right arrow keys*/
  extern char  left;    /*default step direction controlled by left arrow key*/
  extern char  right;   /*default step direction controlled by right arrow key*/
  extern char  up;      /*default step direction controlled by up arrow key*/
  extern char  down;    /*default step direction controlled by down arrow key*/
  extern char  joyexitmode;  /*axis mode when leaving joykey mode*/

  printf("\n              JOYKEY INFORMATION\n\n");
  printf("Last \"joy key\" command sent to Unidex 14: %s\n", cmdstring);
  printf("Currently each time a cursor key is pressed the jog speed changes by: %ld\n\n",stepvalue);

  printf("%c :axis controlled by up (8) and down (2) cursor keys.\n",udaxis);
  printf("%c :stepping direction controlled by up (8) cursor key.\n",up);
  printf("%c :stepping direction controlled by down (2) cursor key.\n\n",down);

  printf("%c :axis controlled by left (4) and right (6) cursor keys.\n",lraxis);
  printf("%c :stepping direction controlled by left (4) cursor key.\n",left);
  printf("%c :stepping direction controlled by right (6) cursor key.\n\n",right);

  printf("a%c :axis control mode when leaving \"joykey\" mode.\n",joyexitmode);
  printf("\n END OF JOYKEY INFORMATION\n");

}

checkstatus()  /*this routine decodes Unidex 14's status and displays it */
{
  extern char done_flag, error_flag;

  if (error_flag != 0)
  {
    if ((error_flag & CMDERR_BIT) != 0)       /*command error*/
      printf("\nUnidex 14's Status Register indicates a COMMAND ERROR.\n");
    else
    if ((error_flag & ENC_BIT) != 0)      /*encoder slip*/
      printf("\nUnidex 14's Status Register indicates an ENCODER SLIP ERROR.\n");
    else
    if ((error_flag & OVRT_BIT) != 0)      /*limit switch is active*/
      printf("\nUnidex 14's Status Register indicates a limit switch has closed.\n");

    error_flag = 0; /*clear error_flag*/
  }
```

```
if(done_flag != 0) /*an ID has been encountered, print the done axis*/
{
  printf("\nUnidex 14's Status Register indicates a DONE flag has been set.\n");
  printf("AXES DONE: ");
  if(done_flag % 2 == 1)
    printf("x ");
  done_flag = done_flag / 2;
  if(done_flag % 2 == 1)
    printf("y ");
  done_flag = done_flag / 2;
  if(done_flag % 2 == 1)
    printf("z ");
  done_flag = done_flag / 2;
  if(done_flag % 2 == 1)
    printf("t ");
  printf("\nPress Cntrl-Y or type IC to clear Unidex 14's done status.\n");
  done_flag = 0; /*clear done flag */
 }
}

char getc_u14() /* Returns character from Unidex 14's buffer */
{
 char  c;

 if(inxti == inxto)      /* no char available in buffer */
  return (0);
 else
 {
  c = inbuf[inxto++]; /* get char that pointer points to, incr. pointer*/
  if(inxto == BSIZE) /* if we are pointing to the end of the buffer */
   inxto = 0;    /* wrap pointer around to beginning of buffer */
  outp(CONTROL_REG, inp(CONTROL_REG) | IBF_BIT); /* enable Unidex 14
            to interrupt PC to put more chars. in buffer */
  return (c);
 }
}


putc_u14(c) /* Puts character into output buffer to Unidex 14 */
char  c;
{
 int   i, j;

 i = onxti;  /* set local pointer to global input pointer of output buffer*/
 j = i + 1;
 if(j == BSIZE) /*if next pointer points to the end, wrap pointer to start*/
```

```
    j = 0;
  while(j = = onxto)  /* while there is no room in the buffer for more */
    ;          /*wait until Unidex 14's interrupt routine makes room in buffer*/
  outbuf[i + +] = c;  /* get character i points to and increment i */
  if(i = = BSIZE)  /* if i points to the end of buffer wrap it around */
    i = 0;
  onxti = i;      /*set global pointer to local pointer's value*/
  outp(CONTROL_REG, inp(CONTROL_REG) | TBE_BIT); /* enable Unidex 14 to interrupt
            PC so it can get chars from the buffer*/
}


mhit()  /* returns non zero if input buffer has character */
{
  return(inxto - inxti);
}

interrupt far  service()  /* interrupt driven routine to communicate with Unidex 14 */
{
  char  c;
  int  i;

  /* get the status of interrupts that are enabled */
  c = inp(STATUS_REG) & (inp(CONTROL_REG) | 0x0f);

  if(c & DONE_BIT)        /* if the DONE bit is true */
  {
    done_flag | = inp(DONE_REG);  /* save done axis flags */
    error_flag | = (c & 0x0f);    /* save error bits also */
    outp(DATA_REG, CLEARD);      /* reset Unidex 14 done status */
  }
  else
  if((c & TBE_BIT) = = TBE_BIT) /* if Unidex 14 board will accept a char */
  {
    if(onxto ! = onxti)  /* do we have a char to send to Unidex 14 ? */
    {
      outp(DATA_REG, outbuf[onxto]); /* send char to Unidex 14 */
      if( + + onxto = = BSIZE)     /* take care of wrap around */
        onxto = 0;
      if(onxto = = onxti)     /* if the buffer is now empty */
      /* turn off Unidex 14's interrupt so it won't keep asking for chars */
        outp(CONTROL_REG, inp(CONTROL_REG) & ~TBE_BIT);
    }
    else    /* no char in buffer to send to Unidex 14 */
        /* turn off Unidex 14's interrupt so it won't keep asking for chars */
      outp(CONTROL_REG, inp(CONTROL_REG) & ~TBE_BIT);
```

```
    }
    if((c & IBF_BIT) = = IBF_BIT) /* if Unidex 14 has a char to send to the PC*/
    {
      inbuf[inxti] = inp(DATA_REG); /* get the char and put it in the buffer*/
      if(+ +inxti = = BSIZE)    /* take care of wrap around */
        inxti = 0;
      if((i = inxti + 1) = = BSIZE) /* room for another char in the buffer? */
        i = 0;
      if(i = = inxto)
        outp(CONTROL_REG, inp(CONTROL_REG) & ~IBF_BIT);
        /*no room in buffer, turn off interrupt so Unidex 14 cannot send more*/
    }
    outp(ICNT, 0x20);        /* send 8259 chip a return from interrupt */

    /* PC/XT/AT interrupt control is edge sensitive only.
    The following generates a rising edge if a second interrupt has occured
    prior to the first being reset.  This can happen if more than one
    source is enabled on Unidex 14 as in this program. */

    c = inp(CONTROL_REG);
    outp(CONTROL_REG, c & ~IRQ_BIT);      /* turn off interrupts */
    outp(CONTROL_REG, c);         /* and back on to create edge */
}


init(intvec) /*connect hardware interrupt IRQ5 to interrupt handler */
int   (*intvec)();
{
  extern int   intdosx();
  union  REGS   inregs, outregs;
  struct SREGS  segregs;

  inregs.h.ah = 0x25;          /* set interrupt vector DOS call */
  inregs.h.al = 13;           /* interrupt vector level 5 */
  segregs.ds = FP_SEG(*intvec);   /* send the vector */
  inregs.x.dx = FP_OFF(*intvec);
  intdosx(&inregs, &outregs, &segregs);
  outp(IMREG, inp(IMREG) & ~ILEVEL);    /* unmask Unidex 14's interrupts */
  outp(CONTROL_REG, IBF_BIT | DONE_BIT | IRQ_BIT); /*turn on Unidex 14 interrupts*/
}
```

SERVICE AND REPAIR

If necessary, any on-site service should be performed by an experienced electronic technician, preferably one trained by Aerotech, Inc.

SHIPMENT

The procedure for shipping equipment back to Aerotech described below, pertains to warranty as well as non-warranty returns.

1. Before shipping any equipment back to Aerotech, the person making the return must call ahead for a "*Return Authorization Number*".

2 .The equipment being returned must be encased in a proper cushioning material and enclosed in a cardboard box.

Call for a "Return Authorization Number" if it is necessary to ship any part to the factory.

NOTE: Aerotech is not responsible for equipment damage that is due to improper packaging.

Aerotech Service offices are listed on the following page. For service and information, contact the office servicing your area.

AEROTECH SERVICE CENTERS

World Headquarters
AEROTECH, INC.
101 Zeta Drive
Pittsburgh, PA 15238
Phone (412) 963-7470
FAX (412) 963-7459
TLX (710) 795-3125

AEROTECH LTD.
3 Jupiter House, Calleva Park
Aldermaston
Berkshire RG7 4QW England
Phone (07356) 77274
TLX 847228
FAX (07356) 5022

AEROTECH GMBH
Neumeyerstrasse 90
8500 Nuernberg 10
West Germany
Phone (0911) 521031
TLX 622474
FAX (0911) 521235

Warranty and Field Service Policy

Aerotech, Inc. warrants its products to be free from defects caused by faulty materials or poor workmanship for a minimum period of one year from date of shipment from Aerotech. Aerotech's liability is limited to replacing, repairing or issuing credit, at its option, for any products which are returned by the original purchaser during the warranty period. Aerotech makes no warranty that its products are fit for the use or purpose to which they may be put by the buyer, whether or not such use or purpose has been disclosed to Aerotech in specifications or drawings previously or subsequently provided, or whether or not Aerotech's products are specifically designed and/or manufactured for buyer's use or purpose. Aerotech's liability on any claim for loss or damage arising out of the sale, resale or use of any of its products shall in no event exceed the selling price of the unit.

Laser Product Warranty

Aerotech, Inc. warrants its laser products to the original purchaser for a minimum period of one year from date of shipment. This warranty covers defects in workmanship and material and is voided for all laser power supplies, plasma tubes and laser systems subject to electrical or physical abuse, tampering (such as opening the housing or removal of the serial tag) or improper operation as determined by Aerotech. This warranty is also voided for failure to comply with Aerotech's return procedures.

Return Products Procedure

Claims for shipment damage (evident or concealed) must be filed with the carrier by the buyer. Aerotech must be notified within (30) days of shipment of incorrect materials. No product may be returned, whether in warranty or out of warranty, without first obtaining approval from Aerotech. No credit will be given nor repairs made for products returned without such approval. Any returned product(s) must be accompanied by a return authorization number. The return authorization number may be obtained by calling an Aerotech service center. Products must be returned, prepaid, to an Aerotech service center (no C.O.D. or Collect Freight accepted). The status of any product returned later than (30) days after the issuance of a return authorization number will be subject to review.

Returned Product Warranty Determination

After Aerotech's examination, warranty or out-of-warranty status will be determined. If upon Aerotech's examination a warrantied defect exists, then the product(s) will be repaired at no charge and shipped, prepaid, back to the buyer. If the buyer desires an air freight return, the product(s) will be shipped collect. Warranty repairs do not extend the original warranty period.

Returned Product Non-Warranty Determination

After Aerotech's examination, the buyer shall be notified of the repair cost. At such time the buyer must issue a valid purchase order to cover the cost of the repair and freight, or authorize the product(s) to be shipped back as is, at the buyer's expense. Failure to obtain a purchase order number or approval within (30) days of notification will result in the product(s) being returned as is, at the buyer's expense. Repair work is warranted for (90) days from date of shipment. Replacement components are warranted for one year from date of shipment.

Rush Service

At times, the buyer may desire to expedite a repair. Regardless of warranty or out-of-warranty status, the buyer must issue a valid purchase order to cover the added rush service cost. Rush service is subject to Aerotech's approval.

On-Site Warranty Repair

If an Aerotech product cannot be made functional by telephone assistance or by sending and having the customer install replacement parts, and cannot be returned to the Aerotech service center for repair, and if Aerotech determines the problem could be warranty- related, then the following policy applies.

Aerotech will provide an on-site field service representative in a reasonable amount of time, provided that the customer issues a valid purchase order to Aerotech covering all transportation and subsistence costs. For warranty field repairs, the customer will not be charged for the cost of labor and material. If service is rendered at times other than normal work periods, then special service rates apply.

If during the on-site repair it is determined the problem is not warranty related, then the terms and conditions stated in the following "On-Site Non-Warranty Repair" section apply.

On-Site Non-Warranty Repair

If an Aerotech product cannot be made functional by telephone assistance or purchased replacement parts, and cannot be returned to the Aerotech service center for repair, then the following field service policy applies.

Aerotech will provide an on-site field service representative in a reasonable amount of time, provided that the customer issues a valid purchase order to Aerotech covering all transportation and subsistence costs and the prevailing labor cost, including travel time, necessary to complete the repair.

AEROTECH, Inc., 101 Zeta Drive, Pittsburgh, Pennsylvania 15238

Phone (412) 963-7470 • TWX 710-795-3125 • FAX (412) 963-7459

INDEX

A

Auxiliary Control,4-18

C

Command Structure
- Axis Specification,8-3
- Command Summary,8-60
- Constant Velocity Contouring,8-55
- Encoder Status,8-50
- Encoder Tracking,8-49
- Home,8-31
- Initialization,8-31
- Loop Control,8-27
- Move Execution,8-23
- Move Specifications,8-16
- Move Synchronization,8-34
- Move Termination,8-25
- Position Maintenance,8-44
- Slip/Stall Detection,8-48
- System Control,8-6
- System Statu Request,8-39
- User I/O,8-12
- User Units,8-43
- Velocity Staircase,8-52

Connections,4-5
- Auxiliary Control,4-16
- Encoder,4-7
- I/O,4-19
- Limit Switch,4-7
- Marker,4-7,4-13
- Power,4-29
- Serial Output,4-22
- Cosine Ramp,2-20

D

Demonstration Software,9-1
- Basic,9-3
- C,9-35
- Jog.C,9-41
- Pascal,9-11
- Poll.C,9-27
- Readme,9-2

Description,1-1

Dimensions,7-12

DMA,2-6,3-2,3-5

Drive Chassis,5-1

Drive Modules,6-1
- Aerodrive
  - Adjustments,6-54
  - Analog Signal,6-59
  - Circuit Description,6-48
  - Encoder Parameters,6-53
  - Home Direction,6-51
  - Home Limit,6-51
  - Home Reference,6-48
  - Limit Switch,6-51
  - Marker Input,6-52
  - Operation,6-48
  - Specifications,6-62
  - Torque/Speed,4-63
- DC Servo Drives
  - Adjustments,6-34
  - Analog Signal,6-41
  - Circuit Description,6-26
  - Encoder Parameters,6-33
  - Fuse Ratings,6-29
  - Home Direction,6-31
  - Home Limit,6-31
  - Home Reference,6-30
  - Limit Switch,6-33
  - Marker Input,6-32
  - Operation,6-26
  - Specifications,6-45
  - Tachometer,6-29
  - Torque/Speed,4-46
- Stepping Drives,6-1
  - Circuit Description,6-1
  - Direction,6-5,6-18
  - Home Clock,6-8,6-21
  - Home Limit,6-3,6-15
  - Home Reference,6-2,6-14
  - Limit Switch,6-5,6-17
  - Lo/Hi Current,6-8,6-21
  - Personality Module,6-7,6-20
  - Marker Buffer,6-4,6-16
  - Resolution,6-6,6-18
  - Specifications,6-12
  - Torque/Speed,6-13

H

Home Marker,4-13

I

I

Interrupts,2-6,3-2,3-5
Interface Board
    Jumpers,2-12
Internal Structure,5-1
Introduction,2-1
I/O
    Address,2-5,3-3
    Channel,3-1,3-4
    Clock/Osc,3-3
    Data Bus,3-1
    Input,2-7
    Memory,3-1
    Power Supply,3-8
    Output,2-7
    Registers,3-6
    Reset,3-3
    Pull-Jumpers,2-6

L

Linear Ramps,2-18
Limit Inputs,2-3
Limit Switch,7-11

M

Motors,7-1

P

Packaging,1-1
Parabolic Ramp,2-19
Power Supply,6-64
Program Examples,8-64

R

Rear Panel,4-2

S

Software
    Installation,2-9
Specifications, 1-3,1-4,1-5

T

Theory of Operation,2-2

U

U14C
    Description,5-7
    Jumpers,2-3
    Installation,2-8
    Motor Control Connector,2-10
    Schematics,5-8

V

Velocity Profiles,2-17

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP