2-1 LINKACI.BAT

This batch file will link a main module with the ACI library and other language specific libraries.

```
tlink /c c:\tc\lib\c0l %1 %2 %3 %4 %5 %6 %7,%1,%1,aci emu mathl cl
```

where the emu, mathl and cl libraries are the Turbo C libraries either in the current directory or with a library path set to the directory containing them. This directory can of course be explicitly defined on the command line above.

2-2 COMPILE.BAT

This batch file will compile the source file

```
tcc -C -c -d -K -a -ml -N- %1 %2 %3 %4 %5 %6 %7 %8 %9
```

The compiler directives are:

-C	Enable nested comments
-c	Compile to .OBJ but do not link
-d	Merge duplicate strings
-K	Default character type is unsigned
-a	Align on word boundaries
-ml	Compile with large memory model - 1Mbyte of code, 1Mbyte of data space
-N-	Disable stack overflow checking

2-3 UTIL00.C

Description:

This module will contain utility procedures that will be commonly used for robot interfacing.

NOTE:

It is important that the procedure rdparams() be executed before any other procedure in this module, as they depend upon the information returned by the rdparams( ) procedure. These procedures will utilize the structure 'mem' as the source for parameter values. In this way, the calling procedures do not have to worry about them.

Global Procedures:

```
int rdparams( n )
int n;
```
This procedure will read back the parameters and will update the 'mem' array. Only the first 'n' parameters will be read back. This is so that time is not wasted reading back those parameters that are not needed.

```
rdprogtable( t )
p_table *t;
```
Read back the program table. The mem.pt_ptr must be valid, as well as mem.pt_siz for this procedure work properly.

```
rdloctable( t )
l_table *t;
```
Read back the location table. The mem.lt_ptr must be valid, as well as mem.lt_size for this procedure work properly.

```
rdvartable( t )
v_table *t;
```
Read back the program table. The mem.vt_ptr must be valid, as well as mem.vt_size for this procedure work properly.

```
rdprogtable( b )
char *b;
```
Read back the valid program buffer. For this procedure to work, the values of mem.pb_cnt, mem.pb_ptr must be valid.

Global Variables:

```
params mem;
```
This structure of robot parameters is loaded by the rdparams() procedure, and must be valid when any of the other procedures in this module are used. The 'params' structure definition is provided in the 'ACIVAR.H' header file.

External Procedures:

readrobot( )
  This procedure will extract the data from the robot controller. It is defined in the ACI00.C module.
readptrs()
  This procedure is defined in the ACI00.C module. It is used to extract the RAPL pointer list.

2-4 ACIO1.C

Description:

This module will provide all of the ACI protocol support routines. The requirements for the ACI implementation can be divided into three procedure categories which are included in this module:

i) Utility routines,
ii) Protocol
iii) Application Level

Additional Information

Appendix B of the RAPL Technical reference manual discusses the ACI protocol in detail. This information is not reproduced here.

Global Procedures:

All of the global procedures provided by this module are presented here, in the appropriate hierarchy:

i) Utilities

These procedures are utilized by the higher level procedures for string transfer and comparison.

```
void aci_nstrout( str , n )
char str[];
int n;
```

Send a string of characters out. The string is pointed to by 'str', and the string is 'n' bytes long.

```
void aci_strin( tempstr,n )
char tempstr[];
int  n;
```

Read a string of characters in from the serial interface. The number of characters to be expected is 'n', while the destination character array is pointed to by 'tempstr'.

```
int lobyte(i)
int i;
```

Will return the low byte of the integer.

```
hibyte(i)
int i;
```

Will return the hi byte of the integer.

```
int string_cmp( s1 , s2 , n )
byte *s1,*s2;
int  n;
```

Compare two binary strings, using the specified pointers to the two, and given the length of the comparison.

ii) Protocol

These procedures actually drive the ACI protocol. Only two procedures are needed to interface to the programmer. These are 'readrobot' and 'writerobot' which will accept the communication parameters from the global data and the argument list and will transfer the information.

The following two procedures will provide better access to the robot memory by permitting all communication parameters to be set up as necessary.

```
aci_xfr( slave_dev, read_write, memofs, memseg, accesstype, bytes, memptr )
unsigned int  slave_dev, read_write, memofs, memseg, bytes;
byte *memptr;
char accesstype;
```

Main entry point for ACI protocol. All header parameters are entered here. Communication channel is initialized before the communication starts, and the channel is restored after use. The 'memptr' parameter indicates the starting location in the IBM memory for the data transfer. 'Bytes' is a word parameter which indicates how many bytes will be transferred. All other parameters are explained in the header block description, in the Technical manual, Appendix B.

```
aci_xfr1( slave_dev,read_write,memofs,memseg,accesstype,bytes,memptr )
int  slave_dev, read_write, memofs, memseg, bytes;
byte *memptr;
char accesstype;
```

Main entry point for ACI protocol. All header parameters are entered here. The 'memptr' parameter indicates the starting location in the IBM memory for the data transfer. 'Bytes' is a word parameter which indicates how many bytes will be transferred. All other parameters are explained in the header block description, in the Technical manual, Appendix B. Unlike aci_xfr(), the communication channel is not initialized or restored after use.

The remainder of the protocol procedures are used by the four main procedures declared above. They typically cannot be used by themselves.

```
void sendenq( slave_no )
int  slave_no;
```

Initialize the communication with the target robot slave number. Send the enquiry sequence. In a multi-drop environment, this sequence will alert the target robot and will shut off the other units from the remainder of the ACI communication cycle.

```
void data_out( n_bytes,data )
int  n_bytes;
byte data[];
```

Send data to the slave device. The data is in the data[] array. Send the specified number of bytes.

```
void data_read( n_bytes,data )
int  n_bytes;
byte data[];
```
  Read the specified number of bytes from the slave. Write it to the data[] array.

```
void send_header( slave,memtype,memofs,memseg,rw,data_length )
int  slave, memtype, memofs, memseg, data_length;
char rw;
```
  Issue the header to the slave device. Performed after the communication is successfully initialized.

```
void header( slave_no,memtype,memofs,memseg,rorw,message_length )
int  slave_no, memtype, memofs, memseg, message_length;
char rorw;
```
  Formulate the header block to the slave. See technical description for the correct format of the header block.

```
void enqout( slave_no )
int  slave_no;
```
  Issue the enquiry message to the slave. Expect to read back the correct response.

```
void error_state( err )
int err;
```
  Will set the global error flag 'aci_err'. If the input error number is zero, then the error condition will be removed.

iii) Application

The following procedures assume that the global slave_device defines the robot controller that we are talking to. It also assumes that a standard memory access, specified by the '0' memory type qualifier is being attempted. This is the most convenient robot memory access procedure, as it contains the fewest arguments, but assumes correct setup of the global parameter '**aci_slave**'.

```
writerobot( offset , segment , srcptr, cnt )
byte *srcptr;
unsigned int offset, cnt, segment;
```
  Transfer the 'cnt' bytes of data from system memory at 'srcptr' to the robot memory at the specified segment and offset.

```
readrobot( offset , segment , destptr, cnt )
byte *destptr;
unsigned int offset, cnt, segment;
```
  Transfer the 'cnt' bytes of data to system memory at 'destptr' from the robot memory at the specified segment and offset.

```
readptrs( p, num )
char *p;
int num;
```
  Will return the pointer list, using special hex code 0x40. 'num' refers to the number of pointers that will be returned. This procedure always returns

pointers starting at the top of the list. Each pointer is a standard Intel 4 byte pointer value. 'p' points to the area of computer memory which will contain the parameter list after the procedure has executed.

Global Variables:

The global parameters required by the ACI library are available to be adjusted by the system programmer. It is recommended that these parameters assume a default value that is present when the system is loaded for execution. Failing this, they must be given initial values by the programmer before any ACI communication can be attempted.

word aci_txto;
  Timeout value for transmit character - in msec
word aci_rxto;
  Timeout value for receive character - in msec
word aci_baud;
  Baud rate for ACI communication - the default is 2400
word aci_slave;
  Default is slave #1. Can assume a value of 1 to 127
int aci_err;
  Global aci error number. Gets set to non-zero if any ACI error occurs. When this happens, subsequent ACI programming is aborted.
int aci_channel
  Global parameter which will determine the communication channel of the host. This parameter is set mainly for the benefit of the hardware level interface that decides how to issue and receive characters.

Error Codes:

All ACI shell programs must recognize the possible error codes, and should determine a safe recovery strategy, depending upon the application.

04 An enquiry sequence was ignored 32 times.
12 A header was tried 4 times with no success.
16 A header was responded to with neither an ACK nor NAK.
20 A data block read was attempted from the slave 4 times, with a LRC failure each time.
22 A data block write was attempted to the slave 4 times, with a NAK returning each time.
24 Invalid response to a data block write to the slave; neither an ACK nor a NAK returned.
27 Invalid EOT response from the slave.
28 Invalid STX read in a data block read from the slave.
32 Invalid ETB read in a data block read from the slave.
34 Invalid ETX read in a data block read from the slave.
40 Timeout on receiving a character.
50 Timeout on Transmitting a character.
60 Operator pressed key at some time.
61 Bad baud rate selection.

External Procedures:

The ACI protocol module assumes that a hardware level interface exists. The default interface supplied by this library is intended for use with an IBM-PC/XT/AT series of computer, using the COM1 channel. 2400 baud is a recommended communication rate.

```
void clean_aci();
```
  Reset the communication channel. For interrupt driven communications, this procedure would typically clean out the character buffers.
```
void send_aci();
```
  Send a byte to the communication channel
```
int inz_aci();
```
  Initialize the communication channel. Set up interrupt vectors if necessary, and save old communication status. Set the baud rate according to 'aci_baud'.
```
void res_aci();
```
  Restore the communication channel to the pre-ACI state (if necessary).
```
char get_aci();
```
  Get the next byte from the interface.