Department of Electrical and Computer Engineering

The University of Alabama in Huntsville

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Robot Welding Process Control		5. Report Date Aug 1991
		6. Performing Organization Code
7. Author(s) Peter L, Romine		8. Performing Organization Report No.
		10. Work Unit No.
9. Performing Organization Name and Address University of Alabama in Huntsville Electrical and Computer Engineering Department Huntsville, Alabama 35899		11. Contract or Grant No. NAS8-36955 DO #75
		13. Type of Report and Period Covered Final Technical 25Feb90 - 25Feb91
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546-0001 George C. Marshal Space Flight Center		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract ✓

This final report documents the development and installation of software and hardware for Robotic Welding Process Control. Primary emphasis is on serial communications between the CYRO 750 robotic welder, Heurikon minicomputer running Hunter & Ready VRTX, and an IBM PC/AT, for offline programming and control and closed-loop welding control. The requirements for completion of the implementation of the Rocketdyne weld tracking control are discussed. The procedure for downloading programs from the Intergraph, over the network, is discussed. Conclusions are made on the results of this task, and recommendations are made for efficient implementation of communications, weld process control development, and advanced process control procedures using the Heurikon.

17. Key Words (Suggested by Author(s)) Robotic Welding, serial communications, CYRO, HEURIKON, welding control		18. Distribution Statement	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 72	22. Price

NASA FORM 1626 OCT 86

FINAL TECHNICAL REPORT

ROBOT WELDING PROCESS CONTROL

25 February 1990 to 25 February 1991

Contract Number NAS8-36955

Delivery Order 75

Prepared for:

George C. Marshall Space Flight Center

Marshall Space Flightcenter, Alabama 35812

22 July 1991

By

Peter L. Romine

Electrical and Computer Engineering Department

The University of Alabama in Huntsville

Huntsville, Alabama 35899

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Introduction

This report documents the new Weld Process Control interface hardware and software for the CYRO 750. To appreciate the significance of these changes, it is helpful to understand the work previously done to improve CYRO functionality, the type of interface desired for the CYRO, and the computer inside the CYRO.

The basic CYRO is designed to do repetitive welding tasks without outside intervention. The console and control pendant are normally the only ways for an operator to interact with the robot. These are used to teach the robot how to perform specific tasks. The SSME CYRO also includes an optional Sensor interface. In theory, the sensor is an external computer or intelligent device used for closed-loop control of the robot. The sensor can interface to the robot via a serial or parallel interface.

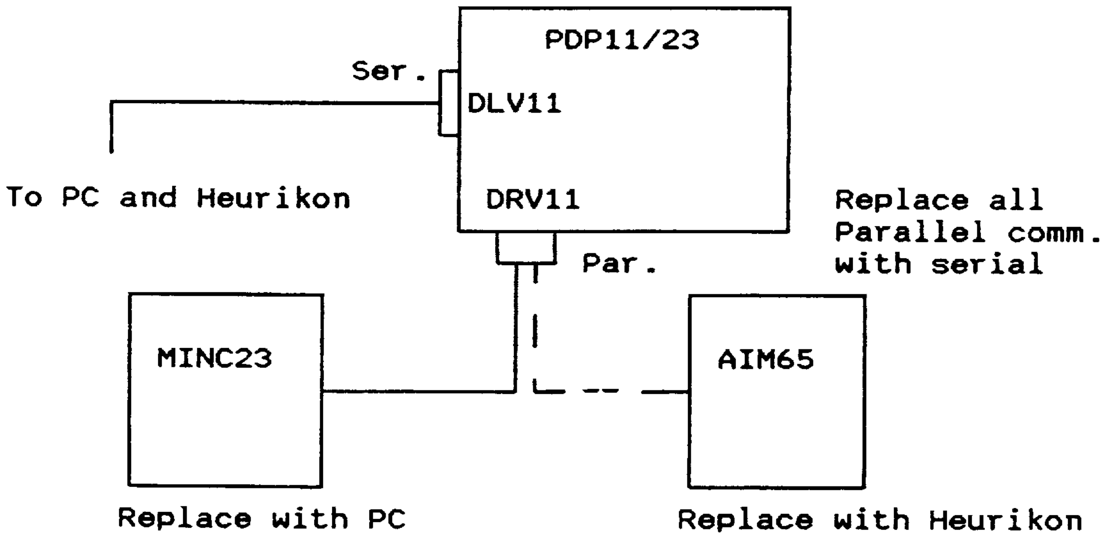

Figure 1. Old CYRO communications.

The parallel interface was previously implemented between the CYRO and a MINC-23 microcomputer. The MINC was used as the host computer for storage and maintenance of CYRO NC programs. Software was developed by Fred R. Sias to support transferring programs between the CYRO and MINC. The MINC editor could then be used to create and modify NC programs. It was also possible to download programs developed on any other system, such as the Intergraph, to the CYRO. The limitations in this configuration were due to the lack of computing power and development tools in the MINC, and the parallel interface did not allow other devices to readily communicate with the CYRO. Moreover, the MINC system did not utilize interrupts and was not multitasking, therefore the MINC could not be used to execute another task while the CYRO was running a program.

The PDP-11/23 inside the CYRO 750 is utilized as a dedicated controller. This computer contains only 128K words of memory, one TU58 tape drive, no floppy drives, and no hard-disk. The program loaded via the tape drive must contain all of the functions for initialization and basic control of the robot. In addition, it must leave space for user programming and the optional sensor interface. This does not leave room for an operating system, as we are accustomed to in modern computers, or any significant diagnostic tools. As a result, software development must be performed on a separate system, compatible with the PDP-11/23 and capable of generating TU58 tapes with a RT-11 format.

To extend the functionality of the CYRO it is highly desirable to implement the serial sensor interface. This will allow the CYRO to communicate with any modern computer via a standard RS-232 interface. In particular, this will allow the Heurikon to send positional updates to the CYRO, based on image processing performed on the captured weld image, to implement closed-loop weld process control. In addition, this will allow a PC or other computer to serve as the host for software storage and maintenance.

1.2 Objectives

a. Investigate OSU software communications to send offsets to CYRO robot from sensed vision data. Software written in Hunter & Ready VRTX. Implement via serial approach, Heurikon serial I/O card to CYRO robot's DEC PDP-11 computer.

b. Make recommendations on hardware and software required for efficient implementation of communications and weld process control software developments.

c. Implement software written by Rocketdyne in Canoga Park that analyzes the weld image, provides offsets to weld path, requires software to communicate with CYRO. Complete definition of software requirements, write code to communicate with robot.

d. Investigate the computer communications requirements for the welding robot system presently under contract to Hobart Brothers Co. to be installed in the welding laboratory late in FY90. Determine proper interface between Heurikon computer for vision process control as well as welding process control.

e. Develop software to allow downloading of programs from Intergraph to CYRO robot. These routines will run on Heurikon to communicate with Intergraph for downloading files - interactive with CYRO when the robot is not in motion.

f. Investigate procedures and recommend approaches to allow the Heurikon computer to more effectively support welding process control. These include, but are not limited to:

    Communications to new robot
    Peaking/Mismatch
    Automatic Robotic Torch Tooling (ARTT) gauging
    GDI wire feed contract for SSME applications
    Backpurge closed-loop control

1.3 Approach

The element central to all of the objectives is serial communications with the CYRO. The approach chosen is to perform this task first. Due to the wealth of development tools available for the PC and the expertise available in program development on the PC, it is expeditious to establish and validate serial communications by developing software on the PC first. The new software will replace the functions provided by the MINC system and also provide the services that are discussed in the PRINZ windows program user manual. The software will be written in C, for its suitability for hardware control and to facilitate the portability of the finished software to the Heurikon.

Dave Gutow of Rocketdyne Canoga Park discussed the overall operation of his seam tracking software and its communications requirements with the CYRO, during his visit to MSFC. He identified the information his software expected from the robot and the commands it sent to the robot. Since it is possible now to modify what the robot sends, it was agreed that it will be ideal to modify the CYRO to emulate the communications of the Canoga Park robot. This would greatly simplify exchange of new software between MSFC and Canoga Park.

2.0 COMMUNICATIONS HARDWARE

2.1 CYRO hardware

The CYRO initially contained a DLV-11E serial interface card. It was determined to be defective. This was done using the ODT monitor in the CYRO and an intelligent serial diagnostic box.

A replacement card was not readily available, instead a DLV-11J was removed from the MINC-23. This card contains 4 serial ports, but could not be configured to duplicate the DLV11-E settings. New I/O register locations and interrupt vector locations were selected for serial port #1.

RCSR	167710
RBUF	167712
XCSR	167714
XBUF	167716
DLVEC	350

Figure 2. Octal register addresses and interrupt vector.

A new CYRO executive tape was required to accommodate the new register locations and interrupt vectors for the DLV-11J. A new tape was created at MSFC as described in appendix B. The ODT monitor was used again to verify the operation of the new card.

Maintenance and troubleshooting was performed on the CYRO during November 1990. The troubleshooting was looking for noise problems that were disrupting robot performance, especially while welding. The technician working on the CYRO removed the serial and parallel cards (the option cards) from the card-cage and changed the location of some of the other cards. Since the two option cards are not required during default operation, he did not reinstall them.

Although the robot worked perfectly with the sensor disabled, it would work only intermittently and then lock-up, with the sensor enabled. Since the new software was working before the hardware changes, it was determined to be a hardware problem. After extensive hardware and software debugging, the original programmer for the sensor software, Russell Vires, emphasized the card location was critical to the full operation of the CYRO. A document that outlined the proper location of the I/O cards was located, and after replacing the cards in the proper positions, the software worked as before. The proper card placement is shown in the figure below.

KD11-HD	MSV11
DRV11 (Data Bus)	DRV11 (To CMC)
DRV11 (Addr Bus)	MXV11-AA
2nd Option (DRV11)	1st Option (DLV11)

Figure 3. Proper CYRO card placement.

2.2 PC and Heurikon hardware

The PC and Heurikon required no additional hardware to communicate with the CYRO. The PC communicates via serial port COM1: and the Heurikon can use any of its available RS-232 terminal lines.

2.3 Fiber-optic serial interface

Due to the harsh electrical environment to be expected in the vicinity of the CYRO, especially during welding, it was decided to replace the existing shielded wire link, between the CYRO and PC, with a fiber optic interface. The link chosen connects between the two devices exactly as the original wire. The only additional connection is for the standard wall-type external power supplies required to power the interfaces.

Two 100 foot links were purchased. One was installed between the PC and CYRO; this link will eventually be used to connect the Heurikon to the CYRO. The other link will be used to connect either the PC or the new robot to the Heurikon. The new interface is illustrated in the figure below.

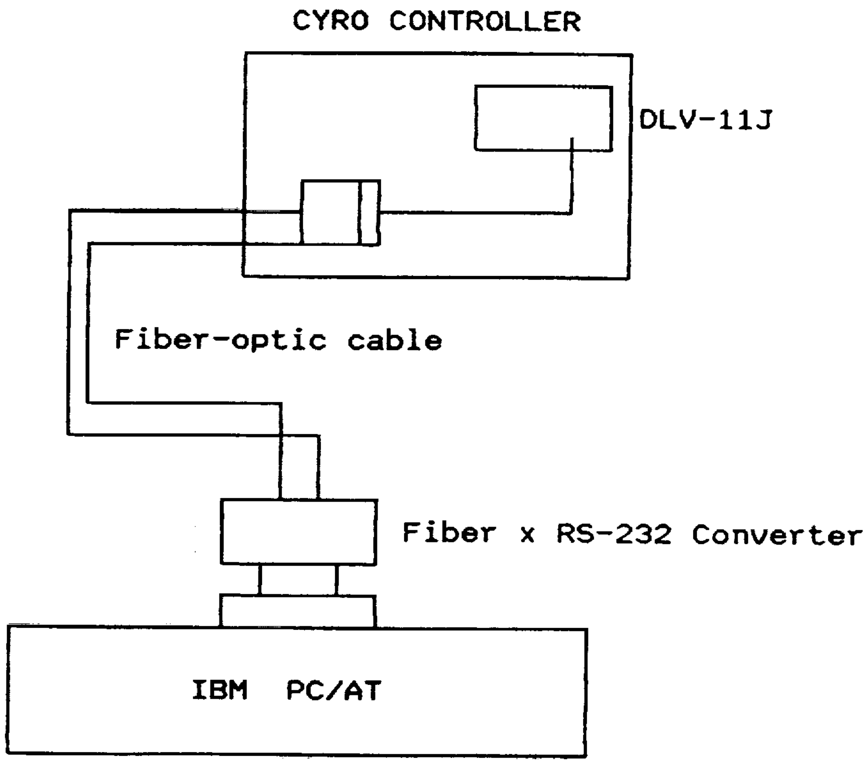

Figure 4. Fiber-Optic Serial Interface

3.0 THE PC INTERFACE

3.1 Introduction

The initial version of the PC-CYRO program was designed around a graphical user interface. The functions it provides place it somewhere between the MINC and Windows programs. In addition to being more complicated to write and maintain, the software would not be directly portable to the Heurikon. For these reasons, the one large program was separated and simplified into several stand-alone, single-function programs. These programs could then be directly transferred to the Heurikon.

3.2 The CYRO TSR

The low-level hardware/software interface evolved through three phases. The first phase of interface software did not utilize interrupts. This required the program to constantly wait for messages from the CYRO. The second phase used interrupts to receive messages from the CYRO. This freed the program to do other things while the CYRO was running. However, the program could not be exited to start another program because the CYRO periodically transmitts status information while it is running.

The final phase uses a Terminate and Stay Resident (TSR) program to communicate with the CYRO. The TSR runs continuously, in the background, until it is removed or the computer is reset. This allows any other program to execute simultaneously. Of course the program must not use the same serial port as the CYRO.

A novel feature of the implementation used is the way in which the TSR "shares" the serial line with the other interface programs. When the TSR is first executed, it grabs and initializes the serial interrupt. It then goes to sleep waiting to intercept messages from the CYRO. In this mode, the default character input function, inside the TSR, acknowledges each character recieved from the CYRO. It then provides the appropriate response to the CYRO once a complete message has been received.

When one of the interface functions desires to talk to the CYRO, they grab the interrupt from the TSR when they start and return it to the TSR before they terminate. The interface function also attaches a new character input function. This function accumulates characters in a circular buffer and feeds them to the main program as they are requested.

3.3 Overview of PC/CYRO Interface Functions

The PC/CYRO interface functions support four basic operations:

Robot Initialization

Report/Change Robot Status

Program Exchange

Robot Position Control

Separate functions have been written for each of these, and are listed below. The programs are envoked by typing the program name and any required parameters, at the DOS prompt. Installation instructions and a more detailed description of each function is included in appendix A.

INIT [*no arguments*] - Initializes the PC and CYRO communications. This program can be executed before the CYRO is switched to sensor mode or any time after. If it is executed after the CYRO is in sensor mode it will wait until the reset button is pressed on the control pendant.

LOADP *<Program #>* *<PC File>* - Loads the NC program in the PC to the CYRO in the specified program slot #.

RUNP *<Program #>* - Begins execution of the program previously loaded into the specified program slot #.

HALTP *<Program #>* - Halts the program currently running from the specified slot #.

SAVEP *<Program #>* *<PC File>* - Saves the NC program from the specified program slot # in the CYRO to the specified file name in the PC.

LISTP *<PC File>* - Displays the NC program contained in the specified file on the PC console.

EDTONC *<PC Edit File>* *<PC CYRO File>* - Converts the NC program file from the format necessary for editing to the format necessary for the CYRO.

4.0 The CYRO Executive

4.1 Introduction

The CYRO EXECUTIVE (EXEC) refers to the program that must be loaded into the CYRO PDP-11/23 before the robot can be operated. The EXEC initializes the robot software and hardware each time the robot is reset. It also controls all of the robots built-in functions, including the EXTERNAL SENSOR INTERFACE (SENSOR) which is the method used to interface external computers to the CYRO.

The SENSOR was not fully implemented by the original manufacturer, as indicated by their sensor interface specification. The original SENSOR does implement the functions necessary to interface to the CYRO. As discussed in chapter 1, the parallel SENSOR connection has been used successfully. To this point, the serial connection had not been used. The following sections discuss the software modifications that were necessary for proper operation of the SERIAL SENSOR INTERFACE.

4.2 Software Changes

The software changes were limited to two MACRO-11 source files; EXTIVE.MAC and SEN.MAC. Two new command files ASM.COM and LINK.COM were written for batch creation of new tapes. The command files were modified for the RSX-11 operating system. The ASM.COM file would typically only be run once. The LINK.COM file must be run each time a new tape is made.

```
STRTUP,STRTUP=SHWMAC/ML,STRTUP
ODTS,ODTS=SHWMAC/ML,ODTS
COMMON,COMMON=SHWMAC/ML,COMMON
TABLE,TABLE=SHWMAC/ML,TABLE
STACK,STACK=SHWMAC/ML,STACK
EXTIVE,EXTIVE=SHWMAC/ML,EXTIVE
SENSOR,SENSOR=SHWMAC/ML,SENSOR
GETCMD,GETCMD=SHWMAC/ML,GETCMD
MATHPK,MATHPK=SHWMAC/ML,MATHPK
PRGDCD,PRGDCD=SHWMAC/ML,PRGDCD
WMGPF,WMGPF=SHWMAC/ML,WMGPF
EXCUTE,EXCUTE=SHWMAC/ML,EXCUTE
DISP,DISP=SHWMAC/ML,DISP
CMC,CMC=SHWMAC/ML,CMC
DIAG,DIAG=SHWMAC/ML,DIAG
```

Figure 5. ASM.COM Assembly Batch File.

```
PETE3/-HD/-MM/SQ,PETE3/-SP/CR=STRTUP,ODTS,COMMON
TABLE,STACK,EXTIVE,SEN,GETCMD,MATHPK
PRGDCD,WMGPF,EXCUTE,DISP,CMC,DIAG
/
STACK=0
PAR=PETE3:1000:157000
//
```

Figure 6. LINK.COM link batch file.

The initial software change was due to the new register and vector values required for the DLV11-J, as discussed in chapter 2 and shown below.

```
RCSR  =  167710
RBUF  =  167712
XCSR  =  167714
XBUF  =  167716
XCSR0 =  167704
XBUF0 =  167706
XCSR1 =  167714
XBUF1 =  167716
XCSR2 =  167724
XBUF2 =  167726
DLVEC =  350
```

Figure 7. New SENSOR equates (SENSOR.MAC)

The CYRO still did not communicate serially after these changes. The problem was isolated to the software after the hardware was proven functional using the ODT.

The similarity between MOTOROLLA 68000 assembly language and MACRO-11 simplified analysis of the software. The software was downloaded to the PC and a programming editor was used to trace the flow of communications in the program. It was discovered that all of the software relating to the SENSOR was contained in the file SENSOR.MAC (copied to SEN.MAC). A logical error was discovered in the low-level serial interface functions. These functions were rewritten and a new tape was generated. The first tape did not work due to a memory overflow. The parallel I/O functions were removed from SEN.MAC to free sufficient program memory. A new tape was generated and was validated using a communications program on the PC.

The EXTIVE.MAC file was only changed to update the CYRO initialization message, show below. This message will appear on the CYRO console each time the CYRO is rebooted from the CYRO-SERIAL executive tape.

```
SSME Robotic Welding Project
Marshall Space Flight Center
UAH Tape Version 2.3
Generated by Peter L. Romine on Feb. 6, 1991
DLV 11-J 4-Channel Serial Card
Communications Via Chan. 1, BAUD set on card.
```

Figure 8. CYRO Initialization message.

```
MSGFLG: .BYTE    0
MSG1:   .BYTE    15,12
        .ASCII   /SSME ROBOTIC WELDING PROJECT/
        .BYTE    15,12
        .ASCII   /MARSHALL SPACE FLIGHT CENTER - /
        .BYTE    15,12
        .ASCII   /UAH Tape Version 2.3/
        .BYTE    15,12
        .ASCII   /Generated by Peter L. Romine on Feb. 6, 1991/
        .BYTE    15,12
        .ASCII   /DLV11-J 4-CHANNEL SERIAL CARD/
        .BYTE    15,12
        .ASCII   /Communications Via Chan.1, BAUD set on card./
        .BYTE    15,12
```

Figure 9. Modification to EXTIVE.MAC

More substantial changes were made to SEN.MAC, as discussed earlier. The low-level I/O functions are DRIN and DROUT, shown below. The parallel I/O support was removed from each to simplify the functions and free program memory.

```
;***************************************************************************
;   ROUTINE: DRIN
;   FUNCTION: READ A BYTE FROM A SERIAL PORT
;   OUTPUT: R0 = BYTE READ FROM PORT
;***************************************************************************

DRIN:   BIT     #RCVDNE,@#RCSR          ;CHAR READY?
                BNE     DRIN                    ;->NO
                MOV     @#RBUF,R0               ;READ BYTE
                BIC     #LBYTMS,R0
                RTS     PC


;***********************************************************************
;   ROUTINE: DROUT
;   FUNCTION: OUTPUT A BYTE TO A SERIAL PORT
;   INPUT: R0 = BYTE TO OUTPUT TO PORT
;***********************************************************************

DROUT:  BIT     #XMTRDY,@#XCSR          ;CAN I XMIT CHAR?
                BEQ     DROUT           ;->NO
                MOV     R0,@#XBUF               ;OUTPUT CHAR
                RTS     PC
```

5.0 MSFC FACILITIES USED

5.1 SSME CYRO Workcell

The primary facility used is the SSME CYRO workcell located in building 4705. This area contains the CYRO 750 robot, PDP-11/23 based controller, welding apparatus, an IBM PC/AT running MSDOS 3.30, and a Heurikon minicomputer running UNIX and VRTX. The PC serves as an alternative software development system and is used to demonstrate the operation of the new software. After initial development is complete, these functions will be shifted to the Heurikon.

5.2 Facilities for CYRO executive tape

As discussed in the first chapter, a separate computer is required to create new CYRO executive tapes. The complication is that this computer must contain a MACRO-11 assembler or cross-assembler and be able to either generate RT-11 program images or VAX EXE files. The computer must also have a TU58 tape drive.

Ten years ago this would have been a typical computer system. Today it is becoming more difficult to find a system that supports any of these. The latest revisions, documented in the following section, requires the use of two separate computers in two different buildings. To make things worse, the output from the first system must be physically transferred to the second on magnetic tape.

The two systems used are the PDP-11 running RSX-11M in building 4708 and the VAX running VMS in the back of 4705. The RSX system contains the source files, the MACRO-11 assembler, and the linker. Software changes can be made on a terminal at 4708 or remotely via the NASA network. The new files are then assembled and linked into an executable or image file.

The new executable file is then loaded on magnetic tape for transfer to the VAX in 4705. Once on the VAX, the RT11UTL is used to convert the RSX file to RT-11 format. The EXCHANGE utility is used to copy the file to the TU58 tape. Finally, the ZAPCSA1 or ZAPTU58 utility is used to mark the new tape as a CYRO Executive tape.

6.0 CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE DEVELOPMENT

6.1 Conclusions

Serial communications between the CYRO and PC is fully implemented and tested. The foundation for serial communications from the CYRO and PC to the Heurikon is established. The software interface for this communication is designed to facilitate operation of the OSU software, weld process control, Rocketdyne software, or other software developed on the Heurikon or PC. The Heurikon includes sufficient I/O capabilities to support several simultaneous communications.

Communication from Intergraph to CYRO is now possible. Files are transferred from Intergraph to the PC or Heurikon over the network, and then to the CYRO.

Experimentation with different BAUD rates on the PC and Heurikon has resulted in a rough indication of processing margin. Additional analysis will be required to better quantify processor utilization.

Familiarization was obtained for the requirements for peaking/mismatch, ARTT gauging, wire feed, backpurge closed-loop control, and other advanced weld process control methods. From this cursory investigation, it is feasible to run these processes on the existing equipment.

6.2 Heurikon development

The NC programming support software now running on the PC is written in ANSI standard C to facilitate its portability to the Heurikon. I propose we port this software over to the Heurikon, running on one of its spare serial lines. This will be a good first step in CYRO-Heurikon integration.

The NC programming software developed for the PC supports transfer of programs between the PC and CYRO. With this setup, files are transferred over the network from the Intergraph to the PC and then transferred to the CYRO. Porting the PC software to the Heurikon allows the Heurikon to duplicate this function. It may prove desirable to obtain an additional modem to connect the Heurikon directly to the network rather than using the one now connected to the PC.

After my discussion with Dave Gutow of Rocketdyne, I propose we make small modifications to the CYRO executive tape to make the CYRO emulate the communications protocol of the Rocketdyne robot. This will provide greater compatibility between the two versions of the weld tracking software. I now have sufficient experience with the process of making CYRO executive tapes to do this.

6.3 TU58 development

In an attempt to simplify the currently tedious software development cycle for the CYRO, I propose we make a brief attempt at emulating the CYRO's TU58 tape drive with the PC. I now have a portion of the software necessary for this and I believe I could quickly determine the feasibility of this.

6.4 The HOBART robot

At a minimum, the Hobart system will require at least one free serial line to be able to communicate within the present system . We must first define the role of the Hobart system as to the type and amount of information that it will need to share between the other devices, robots, computers, etc. The actual communications requirements will then be determined from this evaluation. In addition, due to the harsh electrical environment presented by the robots and the welding process, I propose that all important communications between the CYRO, Heurikon, PC's , Hobart, and any other computer equipment be made with fiber optics.

6.5 Advanced Welding Process Control

Once reliable communications is established between the CYRO, PC, Heurikon, Intergraph, etc. I anticipate the advanced welding process control work will begin. From my previous experimentation, the PC/AT can support full speed communications with the CYRO while running other, CPU intensive, processes. I expect the same to apply for the Heurikon. Still to be determined though is the ability of the Heurikon to simultaneously support advanced weld process control in addition to weld tracking. From a controls standpoint, it is essential that the control loop calculation time be sufficiently small to maintain stability. I propose to determine the maximum loop time allowable by adding processes to the CPU until stability is degraded. This will quantify the amount of additional processing power available for Peaking/Mismatch, ARTT, Wire Feed, Backpurge Closed-loop Control, and other control methods.

APPENDIX A   CYRO-PC INSTALLATION AND OPERATION MANUAL

A.1 Installation

The disk space requirements of CYRO-PC are small enough that it can be used directly from a floppy disk, or installed on a hard disk for faster operation. For the program to operate your machine must have the standard DOS 3.3 (or greater) files, 1 serial communications port designated COM1:, and 512K or more RAM. There are no special display requirements.

For hard disk installation, create a subdirectory to store the files in. For example, if your hard disk is C: and the program files are in drive A:, then at the C: prompt you would enter,

```
CD\
MKDIR CYRO
CD CYRO
COPY A:\*.*
```

Enter the DIR command and you should see the following files.

```
CYROINIT.BAT
CYROTSR.COM
EDTONC.EXE
HALTP.EXE
INIT.EXE
LISTP.EXE
LOADP.EXE
RUNP.EXE
SAVEP.EXE
```

The CYRO-PC programs need the DOS program MODE.EXE to initialize the serial port. Check to see if your system knows where this file is by typing MODE then pressing Enter. If you get a "BAD COMMAND" message, you must locate the MODE program and copy it to the CYRO subdirectory.

A.2 Operation

The initialization CYROINIT command file must be run before using any of the other programs, each time the PC is turned on or rebooted. This file contains the following commands.

```
MODE COM1:9600,N,8,1,P
CYROTSR
INIT
```

You should see a message saying the PC is waiting for the CYRO init message. If the CYRO is booted, press the RESET button on the pendant. Otherwise, power-up and boot the CYRO with the CYRO-PC executive tape. Be sure the key switch on the CYRO is turned to SERIAL mode.

If the PC does not return with a prompt, reset the CYRO several times. If this fails, press the Q key on the PC, this should exit from the INIT program; if it does not, reboot the PC and start over. If you get back to the prompt, enter INIT, then reset the CYRO; if this does not work, reboot the PC and start over.

Once communications have been initialized you can use any of the other programs to transfer files between the CYRO and PC; RUN or HALT a program loaded in the CYRO; or LIST and EDIT CYRO programs on the PC. Detailed instructions for each of these follows.

A.2.1 Transfer To/From CYRO

The SAVEP and LOADP programs are used to transfer files between the CYRO and PC. The SAVEP program copies a program from the CYRO and saves it in a file on the PC.

> **SAVEP** *<Program #>* *<PC File>* - Saves the NC program from the specified program slot # in the CYRO to the specified file name in the PC.

To save the CYRO program in slot 1 to the PC file MARY.NC you would enter

```
SAVEP 1 MARY.NC
```

The LOADP program copies a program from the pC and loads it into the CYRO.

> **LOADP** *<Program #>* *<PC File>* - Loads the NC program in the PC to the CYRO in the specified program slot #.

To load the program in file MARY.NC into slot 1 on the CYRO you would enter

```
LOADP 1 MARY.NC
```

The programs RUNP and HALTP are used to start and stop programs loaded in the CYRO.

> **RUNP** *<Program #>* - Begins execution of the program previously loaded into the specified program slot #.

> **HALTP** *<Program #>* - Halts the program currently running from the specified slot #.

The program LISTP is used to list CYRO programs on the PC screen, printer, or file.

> **LISTP** *<PC File>* - Displays the NC program contained in the specified file on the PC console.

The program EDTONC is used to convert a CYRO file that has been edited on the PC into the format required by the CYRO.

> **EDTONC** *<PC Edit File>* *<PC CYRO File>* - Converts the NC program file from the format necessary for editing to the format necessary for the CYRO.

The following file is an example of a CYRO NC program that could be entered into a PC file using any ASCII editor. You must use an editor that does not insert additional format information into the file.

```
N02G19F5/
N04G91XA10/
M2/
```

Figure 10. Example CYRO NC program.

APPENDIX B PROCEDURES FOR GENERATING EXECUTIVE TAPES

The following section details two methods for creating CYRO Executive tapes. The first method assumes you are initially operating over the network. The second assumes you are physically located in the 4708 computer room.

Two systems are used to generate the executive tapes. The systems are the PDP-11 running RSX-11M in building 4708 and the VAX running VMS in the back of 4705. The RSX system contains the source files, the MACRO-11 assembler, and the linker. Software changes can be made on a terminal at 4708 or remotely via the NASA network. The new files are then assembled and linked into an executable or image file.

The new executable file is then loaded on magnetic tape for transfer to the VAX in 4705. Once on the VAX, the RT11UTL is used to convert the RSX file to RT-11 format. The EXCHANGE utility is used to copy the file to the TU58 tape. Finally, the ZAPCSA1 or ZAPTU58 utility is used to mark the new tape as a CYRO Executive tape.

METHOD I WORKING OVER THE NETWORK

D	
TRW	
@LOCAL> C ISVX01	
USERNAME:	
PASSWORD	
SET HOST SCAT	(Bldg 4708 Room 1107A, Gene Dennis and Jon Scheidt)
>HELLO	(Required to start a login)
Account or name:	(Enter your account name)
PASSWORD:	(Enter your password)
(Use ED to edit files or upload files over the network)	
>RUN $MAC	(To run the RSX-11 assembler)
RTO>FILE,FILE=SHWMAC/ML,FILE	(Wait for prompt to return)
RTO>^Z	(To exit the assembler)
>RUN $TKB	
TKB>@LINK.COM	
>LO	(To logoff SCAT)
LO	(To logoff the VAX)

You must now go to Bldg 4708 to copy the new file to mag tape.

```
>HELLO
Account or name:
PASSWORD:
>MOUNT  MS0:SSME
>COPY  FILE.TSK  MS0:FILE.EXE
>DISMOUNT  MS0:
>LO
```

You must now go to Bldg 4705 room C-200 to generate the CYRO's TU58 tape

```
USERNAME:
PASSWORD
SET  DEFAULT  DUA0:[DEASON.SSME]
MOUNT/OVER=OWNER_ID  MSA0:
SSME
<CR>
COPY  MSA0:FILE.EXE  STA.EXE
RUN  RT11UTL
STA.EXE
DEL  FILE.EXE
DISMOUNT  MSA0:
RUN  SYS$SYSTEM:SYSGEN
SYSGEN>  CON  CON
EXIT
EXCHANGE
EXCHANGE>  INIT  CSA1:
COPY  STA.SAV  CSA1:
EXIT
RUN  ZAPCSA1
DEL  STA.SAV
LO
```

YOU NOW HAVE A NEW CYRO EXECUTIVE TAPE

METHOD II WORKING FROM BUILDING 4708

You should be sitting at a terminal in Bldg 4708 Room 1107A.

>HELLO <CR>	(Required to start a login)
Account or name:	(Enter your account name)
PASSWORD	(Enter your password)
(Use ED to edit files or upload files over the network)	
>RUN $MAC	(To run the RSX-11 assembler)
MAC>FILE,FILE=SHWMAC/ML,FILE	(Wait for prompt to return)
MAC>^Z	(To exit the assembler)
>RUN $TKB	
TKB>@LINK.COM	
MOUNT MS0:SSME	
COPY FILE.TSK MS0:FILE.EXE	
DISMOUNT MS0:	
LO	

(You must now go to Bldg 4705 room C-200 to generate the CYRO's TU58 tape)

USERNAME:
PASSWORD
SET DEF DUA0:[DEASON.SSME]
MOUNT/OVER=OWNER_ID MSA0:
SSME
<CR>
COPY MSA0:FILE.EXE STA.EXE
RUN RT11UTL
STA.EXE

```
DEL  FILE.EXE
DISMOUNT  MSA0:
RUN  SYS$SYSTEM:SYSGEN
SYSGEN>  CON  CON
EXIT
EXCHANGE
EXCHANGE>  INIT  CSA1:
COPY  STA.SAV  CSA1:
EXIT
RUN ZAPCSA1
DEL  STA.SAV
LO
```

APPENDIX C CYRO SERIAL INTERFACE SPECIFICATION

C.1 INTRODUCTION

This section details the actual functions supported by the new software. The functions added have been clearly identified. This chapter replaces the Advanced Robotics Corporation, External Device Interface manual (dated April 27, 1984).

The external device interface option provides the capability for other intelligent devices to communicate with Advanced Robotics Corporation's CYRO 750 and CYRO 2000 arc welding robot controllers. The interface consists of a software package for the 11/23 based robot controller that is designed to support one communication channel using a DEC DLV-11 (Serial) board. **Note:** The interface originally supported both serial and parallel interfaces. The parallel interface option was removed to free memory resources for other applications.

The software for the robot controller supports two classes of external device: SENSOR and/or COMPUTER. Each device class has a predefined set of allowable commands to perform the following communication functions:

ROBOT/ALL DEVICE INTERACTIONS

- Device Identification/Status
- Program Status
- Welding Status
- Robot Positions
- Message to/from Device
- Error Messages

Robot System Parameters

Device Mode Command

<u>ROBOT/SENSOR INTERACTION</u>

Sensor Setup Parameters

Sensor Table

Sensor Position Definition

Sensor Diagnostic

Sensor Calibration

Search for Seam

Sensor Override Data

<u>ROBOT/COMPUTER INTERACTIONS</u>

Save Program to Computer

Load Program from Computer

SERIAL INTERFACE HARDWARE

Digital Equipment Corporation DLV11-J 4 Channel Serial Line Unit

Full Duplex without echo from receiver

Device Address Selection

First Serial Device = 175620

Second Serial Device = 175630

Device Interrupt Vector

First serial device = 370

Second serial device = 380

Table 1 SERIAL INTERFACE

PIN#	SIGNAL
1	Protective ground
2	Transmitted Data
3	Received Data
4	Request to Send
5	Clear to Send
7	Signal Ground
8	Carrier Detect
20	Data Terminal Ready
22	Ring Indicator

SERIAL INTERFACE HANDSHAKING PROTOCOL

The handshaking on the CYRO side is performed in hardware. The receiver should be able to read each character via an interrupt or polling.

MESSAGE PROTOCOL

The message protocol describes the format that the actual data is transmitted in. The message format is as follows:

Length = a byte of information is transmitted by the sender indicating the length of the type code and data portion of the message. The length of a message can range from 1 to 254 bytes.

Sequence Number = a byte identifying each message. This number will be used to reference a particular message, for example, an error message may reference this number to indicate which message caused an error.

Type Code = a byte indicating the type of message that is being transmitted. This information is used to define the format of the data following, and is application dependent.

Data = 0 to 253 bytes of information that are application dependent. The number of data bytes plus the type code defines the length of the message.

Longitudinal Redundancy Check (LRC) = a byte transmitted by the sender to be used by the receiver for error detection. The LRC is computed by exclusive or-ing the length with xff then using the result to exclusive or with the sequence number, then using the result to exclusive or with the type code, then using the result to exclusive or with each byte of data.

The message will be complete when a byte is transmitted by the receiver to acknowledge the correct or incorrect receipt of a message from the sender. If the LRC computed by the receiver matches the LRC sent by the sender, then the message was received correctly.

Response is:

LRC correct =	1
LRC not correct	2

MESSAGE TYPES

There are six message types supported by the external device interface. Distances and angle measurements are referred to in many of the messages in these different message types. For consistency, the following scale factors will be used when referring to distance and angles:

Distances:	1/64 inch per bit
Angles:	1/10 degree per bit

<u>Messages from the Robot to all Devices</u>

1	Request Device Identification/Status
2	Program Status Mode
3	Welding Status Mode
4	Robot Positions
5	Special Message to Device
6	Error

7	Robot System Parameters
8	Device Modes

<u>Messages from the Robot to Sensor Devices</u>

33	Sensor Setup Parameters
34	Sensor Table
35	Sensor Position Definition
36	Sensor Diagnostic
37	Sensor Calibration

<u>Messages from the Robot to Computer Devices</u>

65	Load Program from Computer Acknowledge
66	Save Program to Computer Acknowledge
67	Save Program to Computer

<u>Messages from all Devices to the Robot</u>

129	Device Identification/Status
130	Set Program Mode
131	Set Welding Mode
132	Request Robot Positions
133	Special Message from Device
134	Error
138	Request Robot System Parameters

<u>Messages from Sensor Devices to the Robot</u>

161	Override Data
162	In Position Command

Messages from Computer Devices to the Robot

193	Request Save Program to Computer
194	Request Load Program from Computer
195	Load Program from Computer

MESSAGE CONTENTS - ROBOT TO ALL DEVICES

1 **Request Device Identification/Status** is a message sent at reset time requesting the device identification and hardware status of the device. The result of the request will be a Device Identification/Status message from the device, indicating existence, software and hardware version numbers, and the status of the hardware that can be determined by the device.

2 **Program Status Mode** - indicates to the device that the specified N/C program has been started or stopped.

```
Type Code = 2
Status (one byte):
      Program Started = 1
      Program Stopped = 2
Program Number (one byte - 1 to 9)
```

3 **Welding Status Mode** - indicates to the device that welding has been started or stopped by the N/C program.

```
Type Code = 3
Status (one byte):
      Welding Started = 1
      Welding Stopped = 2
```

4 **Robot Positions** - indicates to the device what the current robot positions are.

Type Code = 4
X axis position - inches (two bytes, low byte, then high byte)
Y axis position - inches (two bytes, low byte, then high byte)
Z axis position - inches (two bytes, low byte, then high byte)
A axis position - degrees (two bytes, low byte, then high byte)
C axis position - degrees (two bytes, low byte, then high byte)
X axis position - C positioner - degrees (two bytes)
Y axis position - C positioner - degrees (two bytes)
X axis position - D positioner - degrees (two bytes)
Y axis position - D positioner - degrees (two bytes)

5 **Special Message to Device** - is a message that will pass ASCII data that is placed in a corresponding N/C command to the device. This message is envisioned to allow special features of some devices to be enabled without the need to change the robot software. It may also be used to send information messages from the N/C program to the device. There is a corresponding message from the device to the robot that will display on the operator's terminal.

Type Code = 5
Variable number of ASCII bytes to be interpreted by the device for special function operation.

6 **Error** - is a message indicating that an error has occurred in the robot control, and what that error is. The device will be required to make a decision based on the error as to the proper course of action to take.

Type Code = 6
Error number - to be defined as needed

7 **Robot System Parameters** - is a message indicating that a robot system parameter has changed. Some of the system parameters will be torch feedrate, welding level, wirefeed speed, and left and right oscillations.

Type Code = 7
Torch Feedrate - inches per minute (two byte, low byte, then high)
Wirefeed Feedrate - percent of power supply output (two bytes)
* 1 bit = 0.1 percent
AVC/ACC setpoint Level - weld level setpoint as defined in the N/C program for Automatic Voltage Control and Automatic Current Control (two bytes)
* 1 bit = 0.1 percent

Oscillation - indicates that a left or right oscillation has occurred (one byte):

None	=	0
Left Osc.	=	1
Right Osc.	=	2

8 **Device Modes** - is a message telling the device whether the message being received by the robot will be executed or not. For example, this will tell a sensor when it should start sending override data, or a host computer that a safety switch has been released, and that it has control of the robot.

Type Code = 8

Device Type (one byte):

Sensor Device	=	1
Computer Device	=	2

Device Identification - three characters as defined in the Device Identification/Status Message.

Device Status (one byte):

Device On	=	1
Device Off	=	2

MESSAGE CONTENTS ROBOT TO SENSOR DEVICES

33	Sensor Setup Parameters
34	Sensor Table
35	Sensor Position Definition
36	Sensor Diagnostic
37	Sensor Calibration

Messages from the Robot to Computer Devices

65	Load Program from Computer Acknowledge
66	Save Program to Computer Acknowledge
67	Save Program to Computer

Messages from all Devices to the Robot

129	Device Identification/Status
130	Set Program Mode
131	Set Welding Mode
132	Request Robot Positions
133	Special Message from Device
134	Error
138	Request Robot System Parameters

Messages from Sensor Devices to the Robot

161	Override Data
162	In Position Command

Messages from Computer Devices to the Robot

193	Request Save Program to Computer
194	Request Load Program from Computer
195	Load Program from Computer

APPENDIX D SOURCE LISTING OF PC CODE

```
/* CYRO.H  --- Communications Program For The CYRO750 ROBOT */

/* Written by          Peter L. Romine   1990
**                     University of Alabama, Huntsville
**                     Electrical and Computer Engineering Department
**                     Copyright 1991 Peter L. Romine. All rights reserved.
*/

#define  UART_PTR    0x400
#define  COMM1       0x00
#define  COMM2       0x01

#define  XMIT        0x00
#define  RCV         0x00
#define  SIO_STATUS  0x05
#define  RCV_MASK    0x01
#define  XMIT_MASK   0x20

#define  DTR_MASK    1
#define  RTS_MASK    2
#define  CONTROL_232 4


#define  MAX_INTR    10
#define  BAD_ITYPE   0x100

typedef  long    IVEC_PTR;

#define  INTR_ENABLE 1
#define  ENABLE_RCV  1
#define  ENABLE_XMIT 2
#define  ENABLE_ERR  4
#define  ENABLE_232  8

#define  SIO_INIT_OK 'U'
#define  INTON_MASK  8

struct serial
{
   unsigned uart_base;
   unsigned commport;
   unsigned int_on;
};

typedef    unsigned char UCHAR;

struct cyro_msg
{
   UCHAR   len;
   UCHAR   seq;
   UCHAR   tc;
   UCHAR   data[255];
   UCHAR   lrc;
};
```

```
typedef    struct cyro_msg CYRO;

struct robot
{
   struct
   {
          char type[16];      /* Device type 1=sensor, 2=computer */
          char id[3];    /* Device ID */
          char status[16];    /* Device Status 1=on, 2=off */
   }device;
       struct
       {
              float  x;
              float  y;
              float  z;
              float  a;
              float  c;
       }arm;
       struct
       {
              float  x;
              float  y;
       }cpos;
       struct
       {
              float  x;
              float  y;
       }dpos;
       float  torch; /* Torch feed rate */
       float  wire;         /* Wire feed rate */
       float  weld;         /* Weld level */
       float avc_acc; /* avc/acc setpoint */
       UCHAR  osc;          /* 0=no osc, 1=left osc, 2=right osc */
       struct
       {
              char   program[16];
              char   welding[16];
       }stat;
       struct
       {
              UCHAR  number;        /* NC program number */
              UCHAR size_l,size_h;                /* Program size in bytes (low,high) */
              char   name[16];      /* Program file name */
              char   comment[80];/* Comment to store in file */
       }prog;
};

#include "cyroproto.h"
```

```
/* CYROPROTO.H  --- Communications Program For The CYRO750 ROBOT */

/* Written by          Peter L. Romine   1990
**                     University of Alabama, Huntsville
**                     Electrical and Computer Engineering Department
**                     Copyright 1991 Peter L. Romine. All rights reserved.
*/

/* Declared in CYRO.C */
extern int    main(int argc,char **argv);
extern void robot_status(void);
extern void nc_programming(void);
extern void pc_utilities(void);
extern void control_robot(void);

/* Declared in COMM.ASM */
extern void init_comm(void);     // ;initialize the comm port,
extern void uninit_comm(void);   // ;remove initialization,
extern void set_xoff(int flag);      // ;enable/disable XON/XOFF,
extern int get_xoff(void);       // ;read XON/XOFF state,
extern int rcvd_xoff(void);      // ;returns true if XOFF rcvd,
extern int sent_xoff(void);      // ;true if XOFF sent,
extern int inp_cnt(void);      // ;returns count of rcv chars,
extern UCHAR inp_char(void);     // ;get one char from buffer,
extern void inp_flush(void);     // ;flush input buffer,
extern void outp_char(UCHAR c);     // ;output a character,

/* Declared in SERIAL.C */
extern struct serial   sio;
extern   void outport(unsigned port,UCHAR c);
extern   UCHAR inport(unsigned port);
extern   unsigned peek(unsigned segmnt,unsigned offst);
extern   void s_xmit(UCHAR c);
extern   UCHAR s_rcv(void);
extern   UCHAR s_rcvstat(void);
extern   UCHAR s_xmstat(void);
extern   UCHAR s_getch(void);
extern   void s_putch(UCHAR c);
extern   UCHAR s_inchar( );
extern   UCHAR get_rs232(void);
extern   void rs232_on(UCHAR rs232_mask);
extern   void rs232_off(UCHAR rs232_mask);

void term(void);

/* Declared in MENU.C */

/* Declared in CYRO1.C */
extern void compute_lrc(CYRO *msg);
extern UCHAR send_msg(CYRO *msg);
extern UCHAR get_msg(CYRO *msg);
extern float to_inches(UCHAR low, UCHAR high);
extern float to_degrees(UCHAR low, UCHAR high);
```

```
/* Declared in CYRO2.C */
extern void cyro_init(CYRO *msg);
extern void send_device_id(CYRO *msg);

/* Declared in CYRO3.C */
extern void cyro_position(CYRO *msg);

/* Declared in CYRO4.C */
extern void cyro_parameters(CYRO *msg);

/* Declared in CYRO5.C */
extern int     Update_OK;
extern struct robot    robo;
extern void talk_to_cyro(void);
extern void update_robo(CYRO *msg);
extern void update_status(void);

/* Declared in CYRO6.C */
extern void cyro_run_prog(CYRO *msg);

/* Declared in CYRO7.C */
extern void cyro_save_prog(CYRO *msg);

/* Declared in CYRO8.C */
extern void cyro_load_prog(CYRO *msg);

/* Declared in CYRO9.C */
extern void cyro_stop_prog(CYRO *msg);

/* Declared in CYRO10.C */
extern void cyro_jog(CYRO *msg);

/* Declared in CYRO11.C */
extern  void    sorry(void);
extern void cyro_message_to_robot(CYRO *msg);

/* Declared in CYRO12.C */
extern void cyro_list_prog(CYRO *msg);

/* Declared in CYRO13.C */
extern void sorry(void);
extern void status( char *msg );
extern void cyro_time(void);
```

```
/* SERIAL.C  --- Communications Program For The CYRO750 ROBOT */

/* Written by          Peter L. Romine   1990
**                     University of Alabama, Huntsville
**                     Electrical and Computer Engineering Department
**                     Copyright 1991 Peter L. Romine. All rights reserved.
*/

#include <stdio.h>
#include "cyro.h"
#include "graph.h"

struct serial sio={0,0,0};

#pragma    check_stack (off)

void outport(unsigned port,UCHAR c)
{
   _asm
   {
      mov dx,[port]
      mov al,[c]
      out dx,al
   }
}

UCHAR inport(unsigned port)
{
   _asm
   {
      mov dx,[port]
      in  al,dx
      xor ah,ah
   }
}

unsigned peek(unsigned segmnt,unsigned offst)
{
   _asm
   {
      mov ax,[segmnt]
      mov es,ax
      mov si,[offst]
      mov ax,ES:[si]
   }
}

void s_xmit(UCHAR c)
{
   outport(sio.uart_base+XMIT,c);
}
```

```
UCHAR s_rcv(void)
{
   return ( sio.int_on ? inp_char( ) : inport(sio.uart_base+RCV) );
}

UCHAR s_rcvstat(void)
{
   return (sio.int_on ? inp_cnt( ) : ((UCHAR)(inport(sio.uart_base+SIO_STATUS) & RCV_MASK)) );
}

UCHAR s_xmstat(void)
{
   return ((UCHAR)(inport(sio.uart_base+SIO_STATUS) & XMIT_MASK));
}

/* waits forever for char */
UCHAR s_getch(void)
{
   while(s_rcvstat( ) == (UCHAR)NULL )
       if( kbhit( ) != (int)NULL )
           if( getch( ) == 'q' )
                       {
               _setvideomode( _DEFAULTMODE );
                              restore_int( );
//                             uninit_comm( );
               exit( );
                       }

   return ( s_rcv( ) );
}

void s_putch(UCHAR c)
{
   while(s_xmstat( ) == (UCHAR)NULL)
       ;
   s_xmit(c);
}

/* if char not available, return NULL */
UCHAR s_inchar( )
{
   return ((UCHAR)((s_rcvstat( )==(UCHAR)NULL) ? (UCHAR)NULL : s_rcv( )));
}

UCHAR get_rs232(void)
{
   return( inport(sio.uart_base+CONTROL_232) );
}

void rs232_on(UCHAR rs232_mask)
{
   outport(sio.uart_base+CONTROL_232,get_rs232( )|rs232_mask);
}
```

```
void rs232_off(UCHAR rs232_mask)
{
   outport(sio.uart_base+CONTROL_232,get_rs232( ) & ~rs232_mask);
}


void term(void)
{
   int c;

   while(1)
   {
       if( s_rcvstat( ) != (UCHAR)NULL )
           putch(s_rcv( ));

       if( kbhit( ) != (int)NULL )
       {
           c = getch( );
           switch( c )
           {
           case 'q':
               return;

           default:
               if( c >= 0 )
                   s_putch((UCHAR)c);
           }
       }
   }
}
```

```
/* CYRO1.C  --- Communications Program For The CYRO750 ROBOT */

/* Written by          Peter L. Romine   1990
**                     University of Alabama, Huntsville
**                     Electrical and Computer Engineering Department
**                     Copyright 1991 Peter L. Romine. All rights reserved.
*/

#include <stdio.h>
#include "cyro.h"

#pragma    check_stack (off)

struct robot   robo;

void compute_lrc(CYRO *msg)
{
   UCHAR          n;
   register UCHAR  i,lrc;

   lrc = msg->len ^ 0xff;
   lrc ^= msg->seq;
   lrc ^= msg->tc;
   n = msg->len - 1;
   for(i=0; i<n; i++)
       lrc ^= msg->data[i];

   msg->lrc = lrc;
}

UCHAR send_msg(CYRO *msg)
{
   UCHAR          n;
   register UCHAR  i;

       s_putch(msg->len);
       delay( );
       s_putch(msg->seq);
       delay( );
       s_putch(msg->tc);
   n = msg->len - 1;
   for(i=0; i<n; i++)
       {
              delay( );
           s_putch(msg->data[i]);
       }
       delay( );
       s_putch(msg->lrc);

   return s_getch( );
}
```

```
delay( )
{
        int     i,n=3

        for(i=0; i<0xfff; i++)
                n = n*(-1);
}

UCHAR get_msg(CYRO *msg)
{
   UCHAR          n;
   register UCHAR i,lrc;

        msg->len = s_getch( );
   lrc = msg->len ^ 0xff;
        msg->seq = s_getch( );
   lrc ^= msg->seq;
        msg->tc = s_getch( );
   lrc ^= msg->tc;
   n = msg->len - 1;
   for(i=0; i<n; i++)
   {
            msg->data[i] = s_getch( );
       lrc ^= msg->data[i];
   }
        msg->lrc = s_getch( );

   return lrc;
}


float to_inches(UCHAR low, UCHAR high)
{
        float   flow,fhigh;

        flow = (float) low;
        fhigh = (float) high;

        return (low + 256.0*fhigh)/64.0;
}


float to_degrees(UCHAR low, UCHAR high)
{
        float   flow,fhigh;

        flow = (float) low;
        fhigh = (float) high;

        return (low + 256.0*fhigh)/10.0;
}
```

```
/* CYRO11.C  --- Communications Program For The CYRO750 ROBOT */

/* Written by         Peter L. Romine   1990
**                    University of Alabama, Huntsville
**                    Electrical and Computer Engineering Department
**                    Copyright 1991 Peter L. Romine. All rights reserved.
*/

#include <stdio.h>
#include "menu.h"
#include "cyro.h"

#pragma    check_stack (off)

void cyro_message_to_robot(CYRO *msg)
{
       UCHAR   lrc_in,ack;

   msg->seq = 0;
   msg->tc = 133;
   strcpy(msg->data,"Message to the robot\r");
   msg->len = strlen(msg->data) + 1;

   compute_lrc(msg);

   ack = send_msg(msg);
}
```

```
/* INIT.C  --- Communications Program For The CYRO750 ROBOT */

/* Written by          Peter L. Romine   1990,91
**                     University of Alabama, Huntsville
**                     Electrical and Computer Engineering Department
**                     Copyright 1991 Peter L. Romine. All rights reserved.
*/

#include <graph.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <stddef.h>
#include <time.h>
#include <sys\types.h>
#include <sys\timeb.h>
#include <string.h>
#include "menu.h"
#include "cyro.h"

#pragma    check_stack (off)

static CYRO   msg;

int main( int argc,char **argv)
{
        register        i;
        int             n=100;
        UCHAR   len,lrc,seq,tc,lrc_in,ack,data[35];

        /* Init robo structure */
   sio.uart_base = peek(0,(unsigned)(UART_PTR+(2*COMM1)));
   sio.commport = 0;
   sio.int_on = 1;

        strcpy( robo.stat.program,"Not Running");
        strcpy( robo.stat.welding,"Not Welding");
        strcpy( robo.device.status,"On");

        robo.prog.number = 1;
        robo.prog.size_l = 0;
        robo.prog.size_h = 0;
        strcpy( robo.prog.name,"untitled.ncb");
        strcpy( robo.prog.comment,"No Comment");

   grab_int( );
   inp_flush( );

        printf( "Waiting for the init message from the CYRO\n");
        len = 29;
        while( s_getch( ) != len )
                ;
        /* Got the 29, now get rest of msg */

        seq = s_getch( );
        tc = s_getch( );
        for( i=0; i<(len-1); i++)
               data[i] = s_getch( );

        lrc_in = s_getch( );

        s_putch( 1);

        printf( "Got the init, sending the device ID\n");
        send_device_id( &msg);

    restore_int( );
}
```

```
/* LOADP.C  --- Communications Program For The CYRO750 ROBOT */

/* Written by          Peter L. Romine   1990,91
**                     University of Alabama, Huntsville
**                     Electrical and Computer Engineering Department
**                     Copyright 1991 Peter L. Romine. All rights reserved.
*/

#include <graph.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <stddef.h>
#include <time.h>
#include <sys\types.h>
#include <sys\timeb.h>
#include <string.h>
#include "menu.h"
#include "cyro.h"

static CYRO   msg;
static char   str1[4096],str2[128];

int main(int argc,char **argv)
{
       UCHAR   num;
       FILE    *fp;

       if( argc < 3 )
       {
              printf("USAGE:  loadp <prog#> <file.nc>\n\n");
              printf("Loads the program in file.nc from the PC to the CYRO in location prog#\n");
              exit();
       }

       num = (UCHAR) atoi(argv[1]);
       if( num<1 || num>9 )
       {
              printf("ERROR: prog# = [%d] is invalid.  Must be 1 to 9 \n",(int)num);
              exit();
       }

       if( !(fp=fopen(argv[2],"rb")) )
       {
              printf("ERROR: opening the requested file [%s]!\n",argv[2]);
              exit();
       }
```

```
          /* Init robo structure */
     sio.uart_base = peek(0,(unsigned)(UART_PTR+(2*COMM1)));
     sio.commport = 0;
          strcpy(robo.stat.program,"Not Running");
          strcpy(robo.stat.welding,"Not Welding");
          strcpy(robo.device.status,"On");
          robo.prog.number = 1;
          robo.prog.size_l = 0;
          robo.prog.size_h = 0;
          strcpy(robo.prog.name,"untitled.ncb");
          strcpy(robo.prog.comment,"No Comment");

     grab_int();
     sio.int_on = 1;
     inp_flush();
     send_device_id(&msg);

          strcpy(robo.prog.name,argv[2]);
          robo.prog.number = num;
          printf("Loading %s to CYRO in program slot #%d\n",robo.prog.name,robo.prog.number);

          cyro_load_prog(&msg);

     restore_int();
}


void cyro_load_prog(CYRO *msg)
{
        UCHAR  lrc_in,ack,thisblock,more,block_cnt;
        FILE   *fp;
        unsigned char        id;
        register       i,j;
        long   bytes;

        fp = fopen(robo.prog.name,"rb");
        bytes = filelength(fileno(fp));
        robo.prog.size_h = (UCHAR) (bytes/256L);
        robo.prog.size_l = (UCHAR) (bytes - ((long)robo.prog.size_h * 256L));
        printf("\nbytes = %ld   lb = %d  hb =
%d\n",bytes,(int)robo.prog.size_l,(int)robo.prog.size_h);
        fflush(stdout);

     msg->len = 4;
     msg->seq = 0;
     msg->tc = 194;       /* Request for load program from PC */
     msg->data[0] = robo.prog.number;
          msg->data[1] = robo.prog.size_l;
          msg->data[2] = robo.prog.size_h;

     compute_lrc(msg);

     ack = send_msg(msg);
```

```
            block_cnt = 0;
            more = 1;
            while( more )
            {
                    while(1)
                    {
                            lrc_in = get_msg(msg);
                            s_putch(1);
                            if(msg->data[0] == 1)
                                    break;
                    }

                    /* the message should be a load acknowledge (TC=65)*/

                    if( bytes < 250 )
                            thisblock = (UCHAR)bytes;
                    else
                            thisblock = 250;

                    bytes -= thisblock;

               msg->len = thisblock + 2;
               msg->seq = 0;
               msg->tc = 195;        /* Request for save program from robot */
               msg->data[0] = block_cnt++;

               fread(&(msg->data[1]),1,(size_t)thisblock,fp);

               compute_lrc(msg);
               ack = send_msg(msg);

               if( bytes <= 0 )
                            break;
            }
            fclose(fp);
    }
```

```
/* LOADP.C  --- Communications Program For The CYRO750 ROBOT */

/* Written by           Peter L. Romine   1990,91
**                      University of Alabama, Huntsville
**                      Electrical and Computer Engineering Department
**                      Copyright 1991 Peter L. Romine. All rights reserved.
*/

#include <graph.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <stddef.h>
#include <time.h>
#include <sys\types.h>
#include <sys\timeb.h>
#include <string.h>
#include "menu.h"
#include "cyro.h"

static CYRO   msg;
static char    str1[4096],str2[128];

int main(int argc,char **argv)
{
        UCHAR   num;
        FILE    *fp;

        if( argc < 3 )
        {
                printf("USAGE:  loadp <prog#> <file.nc>\n\n");
                printf("Loads the program in file.nc from the PC to the CYRO in location prog#\n");
                exit( );
        }

        num = (UCHAR) atoi(argv[1]);
        if( num<1 || num>9 )
        {
                printf("ERROR: prog# = [%d] is invalid.  Must be 1 to 9 \n",(int)num);
                exit( );
        }

        if( !(fp=fopen(argv[2],"rb")) )
        {
                printf("ERROR: opening the requested file [%s]!\n",argv[2]);
                exit( );
        }
        /* Init robo structure */
    sio.uart_base = peek(0,(unsigned)(UART_PTR+(2*COMM1)));
    sio.commport = 0,
        strcpy(robo.stat.program,"Not Running");
        strcpy(robo.stat.welding,"Not Welding");
        strcpy(robo.device.status,"On");
        robo.prog.number = 1;
        robo.prog.size_l = 0;
        robo.prog.size_h = 0;
        strcpy(robo.prog.name,"untitled.ncb");
        strcpy(robo.prog.comment,"No Comment");

    grab_int();
    sio.int_on = 1;
    inp_flush();
    send_device_id(&msg);

        strcpy(robo.prog.name,argv[2]);
        robo.prog.number = num;
        printf("Loading %s to CYRO in program slot #%d\n",robo.prog.name,robo.prog.number);

        cyro_load_prog(&msg);

    restore_int();
}


void cyro_load_prog(CYRO *msg)
{
        UCHAR   lrc_in,ack,thisblock,more,block_cnt;
        FILE    *fp;
        unsigned char         id;
        register       i,j;
        long    bytes;

        fp = fopen(robo.prog.name,"rb");
        bytes = filelength(fileno(fp));
        robo.prog.size_h = (UCHAR) (bytes/256L);
        robo.prog.size_l = (UCHAR) (bytes - ((long)robo.prog.size_h * 256L));
        printf("\nbytes = %ld   lb = %d  hb =
%d\n",bytes,(int)robo.prog.size_l,(int)robo.prog.size_h);
        fflush(stdout);

    msg->len = 4;
    msg->seq = 0;
    msg->tc = 194;       /* Request for load program from PC */
    msg->data[0] = robo.prog.number;
        msg->data[1] = robo.prog.size_l;
        msg->data[2] = robo.prog.size_h;

    compute_lrc(msg);

    ack = send_msg(msg);
```

```
        block_cnt = 0;
        more = 1;
        while( more )
        {
                while(1)
                {
                        lrc_in = get_msg(msg);
                        s_putch(1);
                        if(msg->data[0] == 1)
                                break;
                }

                /* the message should be a load acknowledge (TC=65)*/

                if( bytes < 250 )
                        thisblock = (UCHAR)bytes;
                else
                        thisblock = 250;

                bytes -= thisblock;

           msg->len = thisblock + 2;
           msg->seq = 0;
           msg->tc = 195;        /* Request for save program from robot */
           msg->data[0] = block_cnt++;

           fread(&(msg->data[1]),1,(size_t)thisblock,fp);

           compute_lrc(msg);
           ack = send_msg(msg);

           if( bytes <= 0 )
                        break;
        }
        fclose(fp);
}
```

```
/* RUNP.C  --- Communications Program For The CYRO750 ROBOT */

/* Written by         Peter L. Romine   1990,91
**                    University of Alabama, Huntsville
**                    Electrical and Computer Engineering Department
**                    Copyright 1991 Peter L. Romine. All rights reserved.
*/

#include <graph.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <stddef.h>
#include <time.h>
#include <sys\types.h>
#include <sys\timeb.h>
#include <string.h>
#include "menu.h"
#include "cyro.h"

static CYRO   msg;
static char    str1[4096],str2[128];

int main(int argc,char **argv)
{
       UCHAR  num;
       FILE   *fp;

       if( argc < 2 )
       {
              printf("USAGE:  runp <prog#> \n\n");
              printf("Runs the CYRO program in location prog#\n");
              exit( );
       }

       num = (UCHAR) atoi(argv[1]);
       if( num<1 || num>9 )
       {
              printf("ERROR: prog# = [%d] is invalid.  Must be 1 to 9 \n",(int)num);
              exit( );
       }
```

```
         /* Init robo structure */
    sio.uart_base = peek(0,(unsigned)(UART_PTR+(2*COMM1)));
    sio.commport = 0;
         strcpy(robo.stat.program,"Not Running");
         strcpy(robo.stat.welding,"Not Welding");
         strcpy(robo.device.status,"On");
         robo.prog.number = 1;
         robo.prog.size_l = 0;
         robo.prog.size_h = 0;
         strcpy(robo.prog.name,"untitled.ncb");
         strcpy(robo.prog.comment,"No Comment");

    grab_int();
    sio.int_on = 1;
    inp_flush();

    send_device_id(&msg);

         robo.prog.number = num;
         printf("Starting CYRO program slot #%d\n",robo.prog.number);
         cyro_run_prog(&msg);

    restore_int();
}


void cyro_run_prog(CYRO *msg)
{
        UCHAR   ack;

    msg->len = 3;
    msg->seq = 0;
    msg->tc = 130;          /* set program mode */
    msg->data[0] = 1;    /* 1=start, 2=stop */
        msg->data[1] = robo.prog.number;      /* Program # (1-9) */

    compute_lrc(msg);

    ack = send_msg(msg);
}
```

```
/* HALTP.C  --- Communications Program For The CYRO750 ROBOT */

/* Written by          Peter L. Romine   1990,91
**                     University of Alabama, Huntsville
**                     Electrical and Computer Engineering Department
**                     Copyright 1991 Peter L. Romine. All rights reserved.
*/

#include <graph.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <stddef.h>
#include <time.h>
#include <sys\types.h>
#include <sys\timeb.h>
#include <string.h>
#include "menu.h"
#include "cyro.h"

static CYRO   msg;

int main(int argc,char **argv)
{
       UCHAR   num;

       if( argc < 2 )
       {
              printf("USAGE:  haltp <prog#>\n\n");
              printf("Stops the CYRO program in location prog#\n");
              exit( );
       }

       num = (UCHAR) atoi(argv[1]);
       if( num<1 || num>9 )
       {
              printf("ERROR: prog# = [%d] is invalid.  Must be 1 to 9 \n",(int)num);
              exit( );
       }
```

```
            /* Init robo structure */
        sio.uart_base = peek(0,(unsigned)(UART_PTR+(2*COMM1)));
        sio.commport = 0;
        sio.int_on = 1;

            strcpy(robo.stat.program,"Not Running");
            strcpy(robo.stat.welding,"Not Welding");
            strcpy(robo.device.status,"On");
            robo.prog.number = 1;
            robo.prog.size_l = 0;
            robo.prog.size_h = 0;
            strcpy(robo.prog.name,"untitled.ncb");
            strcpy(robo.prog.comment,"No Comment");

        grab_int();
        inp_flush();
        send_device_id(&msg);

            robo.prog.number = num;
            printf("Halting CYRO program slot #%d\n",robo.prog.number);

            cyro_stop_prog(&msg);

            restore_int();
    }

    void cyro_stop_prog(CYRO *msg)
    {
            UCHAR   lrc_in,ack;

        msg->len = 3;
        msg->seq = 0;
        msg->tc = 130;       /* set program mode */
        msg->data[0] = 2;   /* 1=start, 2=stop */
            msg->data[1] = robo.prog.number;      /* Program # (1-9) */

        compute_lrc(msg);

        ack = send_msg(msg);

            lrc_in = get_msg(msg);

            if( lrc_in == msg->lrc )
            {
                    s_putch(1);
            }
            else
            {
                    s_putch(2);
            }
    }
```

```
/* SAVEP.C  --- Communications Program For The CYRO750 ROBOT */

/* Written by          Peter L. Romine   1990,91
**                     University of Alabama, Huntsville
**                     Electrical and Computer Engineering Department
**                     Copyright 1991 Peter L. Romine. All rights reserved.
*/

#include <graph.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <stddef.h>
#include <time.h>
#include <sys\types.h>
#include <sys\timeb.h>
#include <string.h>
#include "menu.h"
#include "cyro.h"

static CYRO   msg;

int main(int argc,char **argv)
{
       UCHAR   num;
       FILE    *fp;

       if( argc < 3 )
       {
              printf("USAGE:  savep <prog#> <file.nc>\n\n");
              printf("Saves the CYRO program in location prog# to the PC in file.nc\n");
              exit( );
       }

       num = (UCHAR) atoi(argv[1]);
       if( num<1 || num>9 )
       {
              printf("ERROR: prog# = [%d] is invalid.  Must be 1 to 9 \n",(int)num);
              exit( );
       }
```

```
        if( (fp=fopen(argv[2],"rb")) )
        {
                printf("WARNING: The requested file [%s] already exists!\n",argv[1]);
                printf("        Do you want to write over it (Y/N)?");
                switch(getchar())
                {
                case 'y':
                case 'Y':
                        fclose(fp);
                        break;

                default:
                        fclose(fp);
                        exit();
                }
        }

        /* Init robo structure */
    sio.uart_base = peek(0,(unsigned)(UART_PTR+(2*COMM1)));
    sio.commport = 0;
    sio.int_on = 1;

        strcpy(robo.stat.program,"Not Running");
        strcpy(robo.stat.welding,"Not Welding");
        strcpy(robo.device.status,"On");

        robo.prog.number = 1;
        robo.prog.size_l = 0;
        robo.prog.size_h = 0;
        strcpy(robo.prog.name,"untitled.ncb");
        strcpy(robo.prog.comment,"No Comment");

    grab_int();
    inp_flush();
    send_device_id(&msg);

        robo.prog.number = num;
        strcpy(robo.prog.name,argv[2]);
        printf("Saving CYRO program #%d to PC in file %s\n",robo.prog.number,robo.prog.name);

        cyro_save_prog(&msg);

        restore_int();
}
```

```
void cyro_save_prog(CYRO *msg)
{
        UCHAR  lrc_in,ack,thisblock,more;
        FILE   *fp;
        register       i;
        long   bytes;

   msg->len = 2;
   msg->seq = 0;
   msg->tc = 193;       /* Request for save program from robot */
   msg->data[0] = robo.prog.number;

   compute_lrc(msg);
   ack = send_msg(msg);

        lrc_in = get_msg(msg);
        s_putch(1);

        /* the message should be a save acknowledge (TC=66)*/

        robo.prog.size_l = msg->data[2];
        robo.prog.size_h = msg->data[3];
        bytes = (long)msg->data[2] + 256L * (long)msg->data[3];

        fp = fopen(robo.prog.name,"wb");
        printf("\nbytes = %ld   lb = %d  hb = %d\n",bytes,(int)msg->data[2],(int)msg->data[3]);
        more = 1;
        while( more )
        {
                lrc_in = get_msg(msg);
                s_putch(1);

                thisblock = msg->len - 2;
           fwrite(&(msg->data[1]),1,(size_t)thisblock,fp);

                bytes -= thisblock;
                if( bytes <= 0 )
                        break;

        msg->len = 2;
        msg->seq = 0;
        msg->tc = 193;       /* Request for save program from robot */
        msg->data[0] = robo.prog.number;
        compute_lrc(msg);
        ack = send_msg(msg);
        }

        fclose(fp);
}
```

```
/* LISTP.C  --- Communications Program For The CYRO750 ROBOT */

/* Written by          Peter L. Romine   1990,91
**                     University of Alabama, Huntsville
**                     Electrical and Computer Engineering Department
**                     Copyright 1991 Peter L. Romine. All rights reserved.
*/

#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <stddef.h>
#include <sys\types.h>
#include <string.h>

static char    str1[4096],str2[128];

int main(int argc,char **argv)
{
        FILE    *fp;
        register        i,j;
        unsigned char           id;
        long    bytes;
        char    comment[90],size_l,size_h;

        if( argc < 2 )
        {
                printf("USAGE:  listp <file.nc>\n\n");
                printf("Lists the program in file.nc from on the PC\n");
                exit();
        }

        if( !(fp=fopen(argv[1],"rb")) )
        {
                printf("ERROR: opening the requested file [%s]!\n",argv[1]);
                exit();
        }

        fread(&id,1,1,fp);
        switch(id)
        {
        case 0x12:      /* Binary NC File */
                fread(comment,80,1,fp);
                printf(comment);
                printf("\n\n");
                fread(&(size_l),1,1,fp);
                fread(&(size_h),1,1,fp);
                bytes = (long)size_l + 256L * (long)size_h;
                break;

        default:
                rewind(fp);
                bytes = (long) fread(str1,1,(size_t)4096,fp);
```

```
                        break;
                }

                fread(str1,1,(size_t)bytes,fp);

        j = 0;
                for(i=0; i<bytes; i++)
        {
                str2[j++] = str1[i];
                switch(str1[i])
                {
                        case '/':
                                i++;
                        case 0x0d:
                                i++;
                        case 0x0a:
                    str2[j-1] = 0x00;
                    strcat(str2,"/\n");
                    j = 0;
                                printf(str2);
                }
        }

                fclose(fp);
}
```

```
/* edtonc.C  --- Communications Program For The CYRO750 ROBOT */

/* Written by          Peter L. Romine   1990,91
**                     University of Alabama, Huntsville
**                     Electrical and Computer Engineering Department
**                     Copyright 1991 Peter L. Romine. All rights reserved.
*/


#include <math.h>
#include <io.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <stddef.h>
#include <sys\types.h>
#include <string.h>

static char    str1[4096],str2[128];

int main(int argc,char **argv)
{
       FILE    *fpin,*fpout;
       register        i,j;
       unsigned char          id;
       long    bytes;
       char    comment[90],size_l,size_h;

       if( argc < 3 )
       {
              printf("USAGE:  edtonc <infile.ed> <outfile.nc>\n\n");
              printf("Converts the editor program infile.ed to CYRO format outfile.nc\n");
              exit( );
       }

       if( !(fpin=fopen(argv[1],"rb")) )
       {
              printf("ERROR: opening the requested infile [%s]!\n",argv[1]);
              exit( );
       }

       if( !(fpout=fopen(argv[2],"wb")) )
       {
              printf("ERROR: opening the requested outfile [%s]!\n",argv[2]);
              exit( );
       }

  printf("Converting the editor file [%s] to the CYRO file [%s]\n",argv[1],argv[2]);
```

```
        fread(&id,1,1,fpin);
        switch(id)
        {
        case 0x12:      /* Binary NC File */
                fread(comment,80,1,fpin);
                printf(comment);
                printf("\n\n");
                fread(&(size_l),1,1,fpin);
                fread(&(size_h),1,1,fpin);
                bytes = (long)size_l + 256L * (long)size_h;
                break;

        default:
                rewind(fpin);
                bytes = filelength(fileno(fpin));
                break;
        }

        fread(str1,1,(size_t)bytes,fpin);

    j = 0;
        for(i=0; i<bytes; i++)
    {
        switch(str1[i])
        {
                case '/':
                        break;

                case 0x0d:
                        break;

                case 0x0a:
           str2[j++] = 0x0a;
                        fwrite(str2,j,1,fpout);
           j = 0;
                        break;

                default:
        str2[j++] = str1[i];
                        break;
        }
    }

        fclose(fpin);
        fclose(fpout);
}
```