RELTIM

A Library of Routines Used in Real Time Machine Control Applications

by
Fred Rasmussen
L. K. Louderbaugh
and
A. G. Ulsoy

Technical Report Number UM-MEAM-84-24

Department of Mechanical Engineering
and Applied Mechanics
University of Michigan
Ann Arbor, MI 48109-2125

May 1984

TABLE OF CONTENTS

CHAPTER 1

Introduction

This document describes a library of FORTRAN callable routines, called REL-TIM.LIB, that were developed for use in real time machine control applications. The four general categories of routines, and a brief description are given below. A more detailed description of each catagory of routines is given in the succeeding chapters. Chapter 2 discusses the Digital to Analog (DAC) and Analog to Digital Conversion (ADC) Routines. Chapter 3 discusses the Parallel Input/Output (I/O) Routines. Chapter 4 discusses the Programmable Real Time Clock Routines. And finally, Chapter 5 discusses the Peek and Poke Routines.

DAC/ADC Routines

These routines operate the Digital Equipment Corporation (DEC) AAV11-A Digital to Analog Converter, and the ADV11-A Analog to Digital Converter. The routines allow the user to send a voltage out through a given channel of the DAC, or read a voltage in from a given channel of the ADC.

Parallel Input/Output Routines

These routines operate the DEC DRV11-J High Density Parallel Interface. The routines allow the user to write out, or read in, a 16 bit binary number to, or from, one of four channels.

Programmable Real Time Clock Routines

These routines are used to operate the DEC KWV11-C Programmable Real Time Clock. The routines allow the user to start and stop the clock, keep track of, and reset the time, in hours, minutes, seconds, and milliseconds.

Peek and Poke Routines

These routines allow the user to write a number to, or read a number from, a given address in memory.

CHAPTER 2

Digital to Analog and Analog to Digital Conversion Routines

2.1 Introduction

This chapter discusses two routines that are used to operate the DEC AAV11-A 4 Channel 12-Bit D/A Converter and the ADV11-A Analog to Digital Converter. A general discussion of the routines is given is this introduction. A more detailed description is contained in the following sections. Section 2.2 discusses DAC, and Section 2.3 discusses ADC.

2.1.1 General Capabilities

The general capabilities of the two routines are:

DAC

Writes a digital number to a user selected channel of the AAV11-A, which is then converted into a proportional analog voltage.

ADC

Reads a digital number from a user selected channel of the ADV11-A, which is proportional to the voltage applied to the channel.

2.1.2 Routine Arguments

The arguments used by DAC and ADC are:

CHANNO

The channel that the user wants to write to or read from. In routine DAC, CHANNO can assume the values 0, 1, 2, or 3, corresponding to channels 0, 1, 2, and 3 respectively. In routine ADC, CHANNO can assume the values 0 through 7, corresponding to channels 0 through 7 respectively. No error checking is done to ensure that CHANNO has an acceptable value. CHANNO is an INTEGER*2 variable.

DACVAL

The digital number that is sent to the AAV11-A for conversion into an analog voltage. The analog voltage is directly proportional to the digital number. DACVAL can range from 0 to 4095, which will cause voltages of -5.12 v and +5.12 v, respectively, to appear on the selected channel. The chart below shows the range of digital values, and the corresponding analog voltages. DACVAL is an INTEGER*2 variable.

Digital Number	0000	2048	4095
Proportional Voltage	- 5.12 v	0.00 v	+ 5.12 v

Two useful FORTRAN formulas for use with DACVAL are given below. The first formula calculates the output voltage (OUTVOL) that is generated by a given value of DACVAL. The second formula calculates the value of DACVAL required to generate a given output voltage.

```
OUTVOL = ( DACVAL - 2048 ) / 400.                    (1)

DACVAL = IFIX ( ( OUTVOL * 400. ) + 2048 )           (2)
```

ADCVAL

The digital value read from the ADV11-A Analog to Digital Converter. ADCVAL is directly proportional to the voltage appearing at the selected channel. The number to voltage conversion chart shown above for DACVAL also applies for ADCVAL. Two useful FORTRAN formulas used with ADCVAL are given below. The first formula calculates the input voltage (INVOLT) that generated the given value of ADCVAL. The second formula calculates the value of ADCVAL that would be generated by a given input voltage. ADCVAL is an INTEGER*2 variable.

```
INVOLT = ( DACVAL - 2048 ) / 400.                    (3)

ADCVAL = IFIX ( ( INVOLT * 400. ) + 2048 )           (4)
```

2.1.3 Source Code

The source code for the routine DAC is LKLDAC.MAC. The source code for the routine ADC is LKLADC.MAC.

2.1.4 Installation Notes

The addresses of the registers of the AAV11-A and the ADV11-A should be at the factory configuration for the routines to work properly. If the register addresses have been changed, the routines must also be changed. The factory configuration for the register addresses is:

AAV11-A Digital to Analog Converter
Factory Configuration of Register Addresses

Register Mnemonic Symbol	Register Octal Address
DAC 0	170440
DAC 1	170442
DAC 2	170444
DAC 3	170446

ADV11-A Analog to Digital Converter
Factory Configuration of Register Addresses

Register Mnemonic Symbol	Register Octal Address
CSR	170400
DBR	170402

2.1.5 Further Information

Further information on the AAV11-A, and the ADV11-A is available in the following publication:

DIGITAL - Microcomputer Interfaces Handbook

2.2 DAC

The FORTRAN callable MACRO-11 subroutine DAC is used to write a digital number (DACVAL) to a user selected channel (CHANNO) of the AAV11-A. At the AAV11-A, an output voltage is generated which is directly proportional to DACVAL. DACVAL can range in value from 0 to 4095, corresponding to voltage values of - 5.12 v and + 5.12 v respectively. The voltage value corresponding to any digital number can be calculated by linear interpolation (see formulas 1 and 2 above). The channel number, CHANNO, can range in value from 0 to 3, corresponding to channels 0 through 3 of the AAV11-A respectively.

The syntax of a call to DAC is:

```
    .
    .
INTEGER CHANNO, DACVAL
    .
    .
CALL DAC ( CHANNO, DACVAL )
    .
    .
```

Some sample calls are:

```
CALL DAC ( 0, 0 )
```

This call causes a voltage of - 5.12 volts to appear on channel 0 of the AAV11-A.

```
CALL DAC ( 3, 1024 )
```

This call causes a voltage of - 2.56 volts to appear on channel 3 of the ADV11-A.

2.3 ADC

The FORTRAN callable MACRO-11 subroutine ADC is used to read a digital number (ADCVAL) from a user selected channel (CHANNO) of the ADV11-A Analog to Digital Converter. ADCVAL is directly proportional to the voltage appearing on the channel of the ADV11-A. ADCVAL can range in value from 0 to 4095, corresponding to voltages of - 5.12 v and + 5.12 v respectively. The voltage value corresponding to any ADCVAL can be calculated by linear interpolation (see formulas 3 and 4 above). The channel number, CHANNO, can range in value from 0 to 7, corresponding to channels 0 through 7 of the AAV11-A respectively.

The syntax of a call to ADC is:

```
    .
    .
INTEGER CHANNO, ADCVAL
    .
    .
CALL ADC ( CHANNO, ADCVAL )
    .
    .
```

Some sample calls are:

```
CALL ADC ( 4, ADCVAL )
```

This call reads the digital number corresponding to the voltage appearing on channel 4, and assigns the number to ADCVAL. If the voltage was + 2.56 v, ADCVAL would be equal to 3072.

```
CALL ADC ( 7, NUMBER )
```

This call reads the digital number corresponding to the voltage appearing on channel 7, and assigns the value to NUMBER. If the voltage was + 3.84 v, NUMBER would be equal to 3584.

CHAPTER 3

Parallel Input/Output Routines

3.1 Introduction

This chapter discusses the two routines used to operate the DEC DRV11-J High Density Parallel Interface. A general discussion of the routines is given in this introduction. A more detailed description of the two routines is given in the following sections. Section 3.1 discusses DRVWRT, and Section 3.2 discusses DRVRED.

3.1.1 General Capabilities

The general capabilities of the two routines are:

DRVWRT
: Writes a 16 bit binary number out to a user selected channel of the DRV11-J.

DRVRED
: Reads a 16 bit binary number in from a user selected channel of the DRV11-J.

3.1.2 Routine Arguments

The two formal arguments for both DRVWRT and DRVRED are:

CHANEL

The channel that the user wants to write to, or read from. CHANEL can assume the values of 1, 2, 3, or 4, which represent channels A, B, C, or D respectively. No error checking is done by the routines to ensure that CHANEL has an acceptable value. CHANEL is an INTEGER*2 variable.

NUMBER

The number that the user wants to write to the selected channel, or the number that is read from the selected channel. NUMBER is an INTEGER*2 variable, and is represented by the LSI-11/23 in two's complement form if its value is negative. The table below shows the pattern of conversion between a decimal integer value, and its two's complement binary and octal representations. The two's complement binary form of the number is what appears at the parallel I/O port during a read or write operation.

SIGNED	TWO'S COMPLEMENT	
DECIMAL VALUE	BINARY REPRESENTATION	OCTAL REPRESENTATION
0	0 000 000 000 000 000	000000
1	0 000 000 000 000 001	000001
2	0 000 000 000 000 010	000002
.	.	.
.	.	.
7	0 000 000 000 000 111	000007
8	0 000 000 000 001 000	000010
9	0 000 000 000 001 001	000011
.	.	.
.	.	.
32764	0 111 111 111 111 100	077774
32765	0 111 111 111 111 101	077775
32766	0 111 111 111 111 110	077776
32767	0 111 111 111 111 111	077777
- 32768	1 000 000 000 000 000	100000
- 32767	1 000 000 000 000 001	100001
- 32766	1 000 000 000 000 010	100002
- 32765	1 000 000 000 000 011	100003
.	.	.
.	.	.
- 4	1 111 111 111 111 100	177774
- 3	1 111 111 111 111 101	177775
- 2	1 111 111 111 111 110	177776
- 1	1 111 111 111 111 111	177777

Table 1 Decimal/Binary/Octal Conversion

3.1.3 Protocol

The parallel I/O routines allow the LSI-11 to communicate with exterior hardware devices through the DRV11-J High Density Parallel Interface. Since the DRV11-J is transceiving electrical signals to and from exterior hardware devices, the timing of those signals is very important. It would be prudent to make sure that the device is "listening" when the DRV11-J is "talking" and vice versa. The DRV11-J comes with 4 protocol signals to ensure that signal synchronization.

Since DRVWRT and DRVRED were not written with any specific hardware device in mind, they make virtually no use of the protocol signals. The only protocol signal used is the USER RDY (user device ready) signal. The USER RDY pin (pin 23) of the channel in use must be grounded (by the user/user's device) for the DRV11-J to be able to WRITE out a number. Note that the DRV11-J can READ whether or not the USER RDY signal is grounded.

Another protocol signal which is readily available for use by a user device is the DRV11J RDY signal. The DRV11J RDY signal (pin 24) of the selected channel goes high (+ 3 v) when a write operation is performed. The DRV11J RDY signal remains high until a read operation is performed, at which point it is goes low (0 v).

In general, the user is responsible for signal synchronization. The user's device must be ready to receive or send a number when the DRV11-J is writing or reading a number respectively.

Further information on signal protocol is available in the DRV11-J User's Guide listed below.

3.1.4 Signal Levels

The voltage values that must appear on the parallel I/O port pins for proper operation are:

Input/Output Signals

1 = + 3 v   0 = ground

Protocol Signals

1 = ground   0 = + 3 v

3.1.5 Source Code

The source code for the routines DRVWRT and DRVRED is located in the file DRV11J.MAC.

3.1.6 Installation Notes

The addresses of the registers of the DRV11-J should be installed at the factory configuration for the routines to work properly. If the register addresses have been changed, the routines must also be changed. The factory configuration for the register addresses is:

DRV11-J High Density Parallel Interface
Factory Configuration of Register Addresses

Register Mnemonic Symbol	Register Octal Address
CSR A	764160
DBR A	764162
CSR B	764164
DBR B	764166
CSR C	764170
DBR C	764172
CSR D	764174
DBR D	764176

3.1.7 Further Information

Further information on the DRV11-J is available from the following publications:

DIGITAL - Microcomputer Interfaces Handbook
DIGITAL - DRV11-J Parallel Line Interface User's Guide

3.2 DRVWRT

The FORTRAN callable MACRO-11 subroutine DRVWRT is used to write a number (NUMBER) to a user selected channel (CHANEL) of the DRV11-J parallel I/O unit. The 16 bit binary number sent out must be a valid INTEGER*2 value (-32768 <= NUMBER <= 32767). Negative numbers are represented in two's complement form as shown in Table 1 above. CHANEL, an INTEGER*2 variable, can take on the values of 1, 2, 3, or 4 to represent channels A, B, C, or D respectively.

The USER RDY bit (bit 23) of the selected port must be grounded if the write is to occur. If the USER RDY bit is not grounded, an error message will be printed out on the terminal.

The syntax of a call to DRVWRT is:

```
    .
    .
INTEGER CHANEL, NUMBER
    .
    .
CALL  DRVWRT ( CHANEL, NUMBER )
    .
    .
```

Some sample calls are:

```
CALL DRVWRT ( 1, 7 )
```

This call sends the binary number 0 000 000 000 000 111 to channel A.

```
CALL DRVWRT ( 2, 32766 )
```

This call sends the binary number 0 111 111 111 111 110 to channel B.

```
CALL DRVWRT ( 3, -32767 )
```

This call sends the binary number 1 000 000 000 000 001 to channel C.

```
CALL DRVWRT ( 4, -1 )
```

This call sends the binary number 1 111 111 111 111 111 to channel D.

3.3 DRVRED

The FORTRAN callable MACRO-11 subroutine DRVRED is used to read a number (NUMBER) from a user selected channel (CHANEL) of the DRV11-J parallel I/O unit. A 16 bit binary number is read from the selected channel, and assigned an INTEGER*2 value. If the most significant bit (bit 15) is high, the number is assumed to be negative, and is interpreted as a two's complement number (as in Table 1 above). CHANEL, an INTEGER*2 variable, can take on the values of 1, 2, 3, or 4 to represent channels A, B, C, or D respectively.

The syntax of a call to DRVRED is:

```
    .
    .
INTEGER CHANEL, NUMBER
    .
    .
CALL  DRVRED ( CHANEL, NUMBER )
    .
    .
```

Some sample calls are:

```
CALL DRVRED ( 2, NUMBER )
```

This call reads the binary word that is presently on channel B, and assigns the equivalent INTEGER*2 value to NUMBER.

```
CALL DRVRED ( 3, IVALUE )
```

This call reads the binary word that is presently on channel C, and assigns the equivalent INTEGER*2 value to IVALUE.

CHAPTER 4

Programmable Real-Time Clock Routines

4.1 Introduction

This chapter discusses the four routines used to operate the DEC KWV11-C Programmable Real-Time Clock. A general discussion of the clock routines is given in this introduction. A more detailed description of the four routines is given in the following sections. Section 4.1 discusses STRCLK, Section 4.2 discusses STPCLK, Section 4.3 discusses KEPTIM, and Section 4.4 discusses SETTIM.

4.1.1 General Capabilities

The general capabilities of the four routines are:

STRCLK - Starts the clock
STPCLK - Stops the clock
KEPTIM - Keeps the time in hours, minutes, seconds, and milliseconds
SETTIM - Sets the time in hours, minutes, seconds, and milliseconds

4.1.2 Routine Arguments

The routines STRCLK and STPCLK have no arguments. The routines KEPTIM and SETTIM use the FORTRAN common block CLKBLK. CLKBLK must appear in the calling program, and is described below:

The FORTRAN syntax for the common block is:

```
      .
      .
INTEGER  HOUR, MINUTE, SECOND, MILSEC
      .
      .
COMMON / CLKBLK / HOUR, MINUTE, SECOND, MILSEC
      .
      .
```

The common block variables are described below:

HOUR:	INTEGER*2.	The present value of the time in hours.
MINUTE:	INTEGER*2.	The present value of the time in minutes.
SECOND:	INTEGER*2.	The present value of the time in seconds.
MILSEC:	INTEGER*2.	The present value of the time in millisecondss.

4.1.3 Alternative Interrupt Service Routines

The clock, when started by calling STRCLK, will generate an interrupt every 1 millisecond. At each interrupt, program control is transferred to an interrupt service routine. In our case, the interrupt service routine is KEPTIM, which keeps track of the time in hours, minutes, seconds, and milliseconds (note that KEPTIM is <u>NOT</u> a FORTRAN callable subroutine).

It is possible that the user might want to use an interrupt service routine other than KEPTIM. For example, the user might use a routine, called SAMPLE, to sample data from some external hardware device every 1 millisecond. To implement SAMPLE, the user must put the address of the first instruction of SAMPLE in the first word of the interrupt on overflow vector. This is done by changing the first executable statement of the routine STRCLK. See the source code for an example.

4.1.4 Source Code

The source code for the routines STRCLK, STPCLK, and KEPTIM is located in the file CLOCK.MAC. The source for the routine SETTIM is located in the file SETTIM.FOR.

4.1.5 Installation Notes

The addresses of the registers of the KWV11-C should be at the factory configuration for the routines to work properly. If the register addresses have been changed, the value of CSRADR in STRCLK, STPCLK, and KEPTIM, and the value of BPRADR in STRCLK must be changed accordingly. The factory configuration for the KWV11-C register addresses is:

KWV11-C Programmable Real-Time Clock
Factory Configuration of Register Addresses

Register Mnemonic Symbol	Register Octal Address
CSR	170420
BPR	170422
CLK OV	440

4.1.6 Further Information

Further information on the KWV11-C Programmable Real-Time Clock is available in the following publication:

DIGITAL - Microcomputer Interfaces Handbook

4.2 STRCLK

The FORTRAN callable MACRO-11 subroutine STRCLK is used to start the KWV11-C Programmable Real-Time Clock. Calling STRCLK causes the clock to begin operating in Mode 1 (an interrupt is generated for every clock overflow until the clock is stopped), at a frequency of 1 MHz, with a clock interrupt being generated once per every 1000 clock cycles (every 1 millisecond).

The frequency of the clock interrupts can be changed by altering either the control status register number (CSRNUM), or the buffer preset register number (BPRNUM). The clock cycle frequency can be reduced from its 1 MHz value by changing CSRNUM (which will slow down the interrupts). Or the overflow frequency can be increased or decreased by increasing or decreasing BPRNUM respectively. BPRNUM is loaded with the number of clock cycles before an interrupt on overflow is generated (in two's complement form).

STRCLK has one other key feature; it lowers the priority of the processor when it services a keyboard generated interrupt. It does this by changing the value of the processor status word which is stored as the second word of the keyboard interrupt vector. It was found necessary to lower the keyboard priority because typing at the keyboard was causing some of the clock overflow interrupts to go by unserviced. This caused the clock to "lose" a millisecond here and there. No adverse effects on the keyboard performance have been observed.

A sample call to STRCLK is:

```
CALL STRCLK
```

4.3 STPCLK

The FORTRAN callable MACRO-11 subroutine STPCLK is used to shut off the KWV11-C Programmable Real-Time Clock. After STPCLK is called, no further interrupts will be generated, the value of time will remain at the value it had when STPCLK was called, and the priority of the Keyboard interrupt is returned to the value it had before STRCLK was called.

A sample call to STPCLK is:

```
CALL STPCLK
```

4.4 KEPTIM

KEPTIM is a MACRO-11 interrupt service routine that is used to keep track of time. Every 1 millisecond, the KWV11-C Programmable Real-Time Clock generates an interrupt, and program control is passed to KEPTIM. KEPTIM performs three major tasks. First, KEPTIM checks the flag overrun bit of the KWV11-C CSR to see whether more than one clock interrupt has occurred since the last clock interrupt was serviced. If more than one overflow has occurred, an error message will be printed out. Some amount of time will have been "lost".

Next, KEPTIM increments the values of HOUR, MINUTE, SECOND, and MILSEC as approriate. Finally, KEPTIM clears the CSR's overflow flag and overflow overrun flag to tell the clock that its interrupt has been serviced.

In order for the value of the time to be passed back to a main program, the main program must contain the common block CLKBLK.

There is no sample FORTRAN call to KEPTIM since it is not a FORTRAN call-able subroutine.

4.5 SETTIM

SETTIM is a FORTRAN subroutine that provides the user with a convenient way to set the value of the time. SETTIM simply assigns the values of the subroutine arguments to the equivalent common block variables. Thus the common block CLKBLK must appear in the calling program. Note that the subroutine does no error checking (say to make sure that the values of time are positive). Note also that SETTIM could be replaced by four FORTRAN assignment statements of the form:

```
    .
    .
HOUR   = ____
MINUTE = __
SECOND = __
MILSEC = ____
    .
    .
```

The syntax of a call to SETTIM is:

```
    .
    .
INTEGER  HOUR, MINUTE, SECOND, MILSEC
    .
    .
COMMON / CLKBLK / HOUR, MINUTE, SECOND, MILSEC
    .
    .
CALL  SETTIM ( HOUR, MINUTE, SECOND, MILSEC )
    .
    .
```

A sample calls is:

```
CALL SETTIM ( 1, 14, 51, 763 )
```

This call sets the value of HOUR to 1, MINUTE to 14, SECOND to 51, and MILSEC to 763.

CHAPTER 5

Peek and Poke Routines

5.1 Introduction

This chapter discusses the two routines PEEK and POKE. A general discussion of the routines is given in this introduction. A more detailed description of the two routines is given in the following sections. Section 5.1 discusses PEEK, and Section 5.2 discusses POKE.

5.1.1 General Capabilities

The general capabilities of the two routines are:

PEEK	Reads a number from a user selected memory location.
POKE	Writes a number to a user selected memory location.

5.1.2 Routine Arguments

The two formal arguments for both PEEK and POKE are:

ADDRES
: The address of the memory location that the user wants to read from or write to. ADDRES is an INTEGER*2 variable, and may be assigned values in either decimal or octal form. Examples of addressing in decimal and octal form are given below. The user is responsible for making sure that ADDRES is a valid machine address. The user is warned that poking a number to an unknown memory location can have substantial affects on the system's software performance.

NUMBER
: The number that the user wants to write to the memory location, or the value of the number read from the memory location. NUMBER is an INTEGER*2 variable.

The user is reminded that negative numbers are stored in two's complement form (see section 3.1.2). Thus, it may be useful to PEEK to, or POKE from, a negative valued address. For instance, the address of the CSR of the KWV11-C Programmable Real-Time Clock is 170420 in octal form, which translates to the signed decimal integer value -3824.

The user is also reminded that he can assign octal values to integers directly in FORTRAN. FORTRAN uses the quotation mark symbol, " , to signify that the following number is to be interpreted as an octal value. Thus, the statement:

```
ADDRES = "170422
```

is equivalent to:

```
ADDRES = - 3822
```

where "170422 is the address of the buffer preset register of the clock.

5.1.3 Source Code

The source code for the routines PEEK and POKE is located in the file PEK-POK.MAC.

5.2 PEEK

The FORTRAN callable MACRO-11 subroutine PEEK is used to read a number (NUMBER) from a user selected memory location.

The syntax of a call to PEEK is:

```
    .
    .
INTEGER ADDRES, NUMBER
    .
    .
CALL  PEEK ( ADDRES, NUMBER )
    .
    .
```

Some sample calls to PEEK are:

```
CALL PEEK ( 0, NUMBER )
```

This call reads the contents of memory location 0 and stores the value in NUMBER.

```
CALL PEEK ( "60, KEYVEC )
```

This call reads the contents of memory location 60 octal (48 decimal, which is the first word of the keyboard interrupt vector) and stores the value in KEYVEC.

5.3 POKE

The FORTRAN callable MACRO-11 subroutine POKE is used to write a number (NUMBER) to a user selected memory location.

The syntax of a call to POKE is:

```
    .
    .
INTEGER ADDRES, NUMBER
    .
    .
CALL  POKE ( ADDRES, NUMBER )
    .
    .
```

Some sample calls to POKE are:

```
CALL POKE ( 0, "177777 )
```

This call writes the octal number 177777 (- 1 decimal is stored as 177777 in two's complement octal form) to the memory location whose address is 0.

```
CALL POKE ( -3824, "113 )
```

This call writes the octal number 113 to the memory location whose address is -3824 (-3824 decimal is stored as 170420 in two's complement octal form, and is the address of the CSR of the clock). This call causes the clock to begin operating in Mode 1, at 1 MHz, with interrupt on overflow.

APPENDIX A

Brief Description of Library Routines

A brief description of all the routines contained in this library is given below:

Digital to Analog and Analog to Digital Converter Routines:

DAC

Writes a digital number to a user selected channel of the AAV11-A Digital to Analog Converter. The DAC converts the number into a proportional analog voltage.

ADC

Reads a digital number from a user selected channel of the ADV11-A Analog to Digital Converter. The digital number is proportional to the analog voltage applied to the channel.

Parallel Input/Output Routines

DRVWRT

Writes a 16 bit binary number out to a user selected channel of the DRV11-J High Density Parallel Interface.

DRVRED

Reads a 16 bit binary number in from a user selected channel of the DRV11-J High Density Parallel Interface.

Programmable Real Time Clock Routines

STRCLK

Starts the KWV11-C Programmable Real-Time Clock operating at 1 MHz, in Mode 1, with interrupts on overflow occurring every 1 millisecond.

STPCLK

Stops the operation of the KWV11-C Programmable Real-Time Clock.

KEPTIM

Keeps track of the passage of time in hours, minutes, seconds, and milliseconds.

SETTIM

An easy way to reset the value of time in hours, minutes, seconds, and milliseconds.

Peek and Poke Routines

PEEK

Reads a number from a user selected memory location.

POKE

Writes a number to a user selected memory location.

APPENDIX B

Hardware Installation Notes

The following paragraphs list the addresses of the hardware registers used in the RELTIM library. The addresses have all been left at the DEC factory configurations. If any of the register addresses have been changed, the corresponding software in the library must also be changed for the routines to operate properly.

The following abbreviations are used below:

BPR	-	Buffer Preset Register
CLK OV	-	Clock Overflow Vector
CSR	-	Control Status Register
DAC	-	Digital to Analog Converter Holding Register
DBR	-	Data Buffer Register

Hardware used with the DAC/ADC routines

AAV11-A Digital to Analog Converter
Factory Configuration of Register Addresses

Register Mnemonic Symbol	Register Octal Address
DAC 0	170440
DAC 1	170442
DAC 2	170444
DAC 3	170446

ADV11-A Analog to Digital Converter
Factory Configuration of Register Addresses

Register Mnemonic Symbol	Register Octal Address
CSR	170400
DBR	170402

Hardware used with the Parallel I/O routines

DRV11-J High Density Parallel Interface
Factory Configuration of Register Addresses

Register Mnemonic Symbol	Register Octal Address
CSR A	764160
DBR A	764162
CSR B	764164
DBR B	764166
CSR C	764170
DBR C	764172
CSR D	764174
DBR D	764176

Hardware used with the Clock routines

KWV11-C Programmable Real-Time Clock
Factory Configuration of Register Addresses

Register Mnemonic Symbol	Register Octal Address
CSR	170420
BPR	170422
CLK OV	440

APPENDIX C

Hardware Information References

The Digital Equipment Corporation publications listed below contain a more detailed discussion of the specifications and programming details of the hardware used with this library.

AAV11-A 4-Channel 12-Bit D/A Converter

Microcomputer Interfaces Handbook

ADV11-A Analog to Digital Converter

Microcomputer Interfaces Handbook

DRV11-J High-Density Parallel Interface

Microcomputer Interfaces Handbook
DRV11-J Parallel Line Interface User's Guide

KWV11-C Programmable Real-Time Clock

Microcomputer Interfaces Handbook

APPENDIX D

Library Development History

LIBRARY NAME:	RELTIM.LIB
LIBRARY CONTENTS:	General purpose routines for use in real time machine control applications. The routines are used to operate a digital to analog converter, an analog to digital converter, a parallel line unit, and a programmable real time clock. Routines to peek and poke are also included.
SOURCE CODE LANGUAGES:	MACRO-11, FORTRAN
COMPUTER USED:	Digital Equipment Corporation LSI-11/23
OPERATING SYSTEM:	RT-11 FB (S) V04.00G
HARDWARE REQUIREMENTS:	Digital Equipment Corporation's: AAV11-A 4-Channel 12-Bit D/A Converter ADV11-A Analog to Digital Converter DRV11-J High Density Parallel Interface KWV11-C Programmable Real Time Clock
LIBRARY CREATION DATE:	May 5, 1984
LIBRARY DEVELOPMENT DATE:	February through March, 1984
LIBRARY DEVELOPMENT SITE:	Manufacturing Systems Modeling and Control Laboratory Department of Mechanical Engineering University of Michigan Ann Arbor, Michigan 48104
LIBRARY PROGRAMMERS:	Fred Rasmussen and L. K. Louderbaugh
LIBRARY PROJECT SUPERVISOR:	Professor A. Galip Ulsoy
LIBRARY PROBLEM CONTACT:	Professor A. Galip Ulsoy
LIBRARY FUNDING SOURCE:	The library was developed primarily for use with the Variable Gain Adaptive Control project: NSF Grant MEAM-8112629

APPENDIX E

Library Source Code Listings

This appendix contains a listing of all the source code used to create the library RELTIM.LIB. The source code file names are given below:

Digital to Analog and Analog to Digital Converter Routines:

```
        LKLDAC.MAC
        LKLADC.MAC
```

Parallel Input/Output Routines

```
        DRV11J.MAC
```

Programmable Real Time Clock Routines

```
        CLOCK.MAC
        SETTIM.FOR
```

Peek and Poke Routines

```
        PEKPOK.MAC
```

DIGITAL TO ANALOG AND ANALOG TO DIGITAL CONVERTER ROUTINES

The following pages contain
the source code for the routines:

DAC
ADC

```
        .TITLE  DAC
        .GLOBL  DAC
;
;       General Description: This real time routine is used to initiate a
;               digital to analog conversion.  It is assumed that the
;               AAV11-A board is installed at the standard adresses:
;
;                       DAC0    170440
;                       DAC1    170442
;                       DAC2    170444
;                       DAC3    170446
;
;       Calling example:
;
;               CALL DAC(CHANNO,DACVAL)
;
;       Arguments:
;               Common blocks: None
;               Formal arguments:
;                       CHANNO: An intiger*2 variable containing the DAC
;                               channel number
;                       DACVAL: The integer*2 value that is to be converted
;
;       External References: None
;
;                       by Leal Lauderbaugh
;
;
DAC:    MOV     (R5)+,R0
        MOV     @(R5)+,R0               ;GET CHANNEL #
        MOV     @(R5)+,R1               ;GET VALUE
        MOV     R0,R2                   ;MULTIPLY THE CHANNEL NUMBER BY 2
        ADD     R2,R0                   ;BY ADDING IT TO ITSELF
        ADD     #170440,R0              ;SET UP ADRESS FOR CONVERTER
        MOV     R1,(R0)                 ;MOVE THE VALUE INTO THE CONVERTER
        RTS     PC
        .END
        .TITLE ADC
        .GLOBL ADC
;
;       General Description:  this is a fortran callable subroutine that will
;               preform analog to digital conversions.
;
;       Calling example: CALL ADC(CHANNO,ADCVAL)
;
;       Arguments:
;
;               Formal arguments:
;                       CHANNO: The channel number(0-7) of the adc channel
;                               where the conversion is to occur.
;
;                       ADCVAL: the integer result of the analog to digital
;                               conversion.
;
;                       by Leal Lauderbaugh
;
;
ADC:    MOV     (R5)+,R0        ;GET THE NUMBER OF ARGUMENTS
        CLR     @#170400        ;CLEAR THE CONTROL REGISTER
        MOVB    @(R5)+,@#170401 ;MOVE THE CHANNEL NUMBER TO THE HIGH ADDRESS
;                               ;OF THE CONTROL REGISTER.
        INCB    @#170400                ;START THE CONVERSION
3$:     TSTB    @#170400                ;TEST TO SEE IF DONE
        BPL     3$
        MOV     @#170402,@(R5)+ ;RETURN THE RESULT
        RTS     PC
        .END
```

PARALLEL INPUT/OUTPUT ROUTINES

The following pages contain
the source code for the routines:

DRVWRT
DRVRED

```
                              *************
                              * DRVWRT.MAC *
                              *************
```

The following MACRO-11 code is designed to allow a user to work with the Digital Equipment Corporation DRV11-J High Density Parallel Line Interface from a FORTRAN program.

Written by Fred Rasmussen

Spring 1984

Manufacturing Systems
Modeling and Control Laboratory
Department of Mechanical Engineering
University of Michigan
Ann Arbor, Michigan 48104-2125

```
 TITLE  PARALLEL I/O ROUTINES
 IDENT  /VERSN1/
 SBTTL  DATA OUTPUT SEQUENCE
 MCALL .PRINT
```

The following program segment, DRVWRT, writes out a user selected number to a user selected channel of the DRV11-J parallel I/O unit.

```
 PSECT DRVWRT,RW,I,GBL,REL,CON
 GLOBL DRVWRT

CSRBAS = 764154                 ;The base address that an index will be added
                                ;to, to give the address of the control status
                                ;register for the desired I/O port.
DBRBAS = 764156                 ;The base address that an index will be added
                                ;to, to give the address of the data buffer
                                ;register for the desired I/O port.
CSRWRT = 000400                 ;A number which, when loaded into the desired
                                ;CSR, will cause that port to operate in the
                                ;output mode, with no interrupts allowed. Note
                                ;that the user's device still has to set bit
                                ;15 of the desired port's CSR (ground it) for
                                ;the port to operate properly as an output
                                ;port.


DRVWRT: MOV @2(R5), R0          ;Move the desired channel number into R0

        ASL R0                  ;Multiply the channel number by 2
        ASL R0                  ;Multiply the channel number by 2 again

        MOV CSRBAS(R0), R1      ;Move the contents of the desired CSR into
                                ;R1. Since the user ready bit of the CSR is
```

```
                                    ;bit 15, the N bit of the processor will be
                                    ;set if the user ready bit is set.
;
        BMI WRTDAT                  ;If the user ready bit is set, branch to
                                    ;write out the desired number.
;
        JMP ERROR                   ;Since the user ready bit of the CSR was
                                    ;not set, warn the user and return to the
                                    ;calling program.
;
WRTDAT: MOV #CSRWRT,CSRBAS(R0)      ;Load the desired CSR with the number to
                                    ;allow it to write out the data.
;
        MOV @4(R5),DBRBAS(R0)       ;Write the desired number to the data
;                                   ;buffer register.
;
        RTS PC                      ;Return to the calling program.
;
ERROR:  JSR PC, ERRMSG              ;Warn the user that his device had not
                                    ;enabled the user ready bit of the desired
                                    ;CSR, and therefore no output data can be
                                    ;written to that port.
;
        RTS PC                      ;Return to the calling program.
;
;
;
;
;
;
;
.SBTTL  DATA INPUT SEQUENCE
;
;
; The following program segment, DRVRED, reads in a number from a user
; selected channel of the DRV11-J parallel I/O unit.
;
;
.PSECT DRVRED,RW,I,GBL,REL,CON
.GLOBL DRVRED
;
CSRBAS = 764154                     ;The base address that an index will be added
                                    ;to, to give the address of the control status
                                    ;register for the desired I/O port.
DBRBAS = 764156                     ;The base address that an index will be added
                                    ;to, to give the address of the data buffer
                                    ;register for the desired I/O port.
CSRRED = 000000                     ;A number which, when loaded into the desired
                                    ;CSR, will cause that port to operate in the
                                    ;input mode, with no interrupts allowed.
;
;
DRVRED: MOV @2(R5), R0              ;Move the desired channel number into R0
;
        ASL R0                      ;Multiply the channel number by 2
        ASL R0                      ;Multiply the channel number by 2 again
;
        MOV #CSRRED,CSRBAS(R0)      ;Load the desired CSR with the number to
                                    ;allow it to read in the data.
;
        MOV DBRBAS(R0),@4(R5)       ;Read the number from the desired channel's
                                    ;data buffer register, and put it in the
```

```
                                       ;memory location that is reserved for it.

         RTS PC                        ;Return to the calling program.
```

The following program segment, ERRMSG, prints out a message to the user, warning him that the user ready bit of the desired port was not set. The hardware of the DRV11-J requires that the user ready bit be set in order for data to be output. Note that the user ready bit is set by grounding it.

```
 PSECT ERRMSG,RW,I,GBL,REL,CON
 GLOBL ERRMSG
 MCALL .PRINT



ERRMSG: .PRINT #MESSAG                 ;Print a warning to the user.

         RTS    PC                     ;Return to the calling program.

MESSAG: .ASCIZ /WARNING. THE USER READY BIT WAS NOT GROUNDED. NO OUTPUT GENERAT

 EVEN                                  ;This macro statement is needed because the
                                       ;last statement filled up an odd number of
                                       ;bytes, and every instruction must begin on
                                       ;an even numbered address.
 END
```

PROGRAMMABLE REAL TIME CLOCK ROUTINES

The following pages contain
the source code for the routines:

STRCLK
STPCLK
KEPTIM
SETTIM

```
;                               ************
;                               * CLOCK.MAC *
;                               ************
;
;
;
; The following MACRO-11 code is designed to allow a user to work with the
; Digital Equipment Corporation's KWV11-C Programmable Real-Time Clock
; from a FORTRAN program.
;
; Written by Fred Rasmussen
;
; Spring 1984
;
; Manufacturing Systems
; Modeling and Control Laboratory
; Department of Mechanical Engineering
; University of Michigan
; Ann Arbor, Michigan  48104-2125
;
;
;
;
.TITLE  CLOCK DRIVING ROUTINES
.MCALL .PRINT
.GLOBL STRCLK,KEPTIM,STPCLK,FORMSG,CLKBLK,HOUR,MINUTE,SECOND,MILSEC,CSRADR
;
; The following program section (.PSECT) sets up a common block of data named
; CLKBLK that must also be present in the FORTRAN calling program for proper
; operation of this code as Fortran callable subroutines.
;
.PSECT  CLKBLK,RW,D,GBL,REL,OVR
;
CLKBLK::
;
HOUR::          .BLKW 1         ;These lines will reserve one word of memory
MINUTE::        .BLKW 1         ;for each of the listed variables.  Note that
SECOND::        .BLKW 1         ;the order of the variables should be the same
MILSEC::        .BLKW 1         ;as the order that they are listed in the
                                ;COMMON statement of the calling program.
;
;
; The following .PSECT defines the subroutine STRCLK which is used to start
; the clock operating at a frequency of 1MHz, overflowing at a frequency of
; 1kHz, and jumping to the interrupt service routine KEPTIM at each clock
; overflow.
;
;
.PSECT  STRCLK,RW,I,GBL,REL,CON
.GLOBL  STRCLK,KEPTIM,CSRADR
;
OVVEC = 440                     ;The clock overflow vector first word is
                                ;located at address 440 (octal).
OVPSW = 340                     ;The clock overflow vector second word is
                                ;located at address 442 (octal), and will
                                ;be loaded with the processor status word
                                ;that will be used during the clock overflow
                                ;service routine. The value 340 (octal) will
                                ;clear the T,N,Z,V, and C bits, and prevent
                                ;other interrupts during the servicing of the
                                ;clock overflow (sets the processor priority
                                        ;level to 7 (the highest priority level)).
CSRADR = 170420                         ;The address of the control status register
                                        ;(CSR) of the KWV11-A clock is 170420 (octal).
BPRADR = 170422                         ;The address of the buffer preset register
                                        ;(BPR) of the KWV11-A clock is 170422 (octal).
CSRNUM = 000113                         ;The number 000113, when inserted into the
                                        ;CSR, will cause the clock to operate in mode
                                        ;1, at a frequency of 1MHz, with the interrupt
                                        ;on clock overflow enabled.
BPRNUM = 176030                         ;The number 176030, which is the two's comple-
                                        ;ment of 1000 (decimal) expressed as an octal
                                        ;number, will cause the clock to overflow once
                                        ;every 1000 clock cycles.
KEYVEC = 000060                         ;The address of the first word of the inter-
                                        ;rupt vector for the Keyboard. The second word
                                        ;of the Keyboard interrupt vector contains the
                                        ;processor status word (PSW) that will be used
                                        ;when a Keyboard interrupt is being serviced.
                                        ;The PSW priority must be lowered from its
                                        ;normal level 7 (the highest priority possi-
                                        ;ble) to level 3 in order to prevent the Key-
                                        ;board from interfering with the clock.  If
                                        ;the PSW is left at level 7, a clock overflow
                                        ;overrun can occur when the user types data
                                        ;into the Keyboard from the FORTRAN calling
                                        ;program
                                        ;


STRCLK: MOV #KEPTIM,OVVEC               ;Put the address of the interrupt on overflow
                                        ;service routine (KEPTIM) in the first word of
                                        ;the interrupt on overflow vector.
        MOV #OVPSW,OVVEC+2              ;Put the overflow processor status word into
                                        ;the second word of the interrupt on overflow
                                        ;vector.
        MOV #000140,KEYVEC+2            ;Set the Keyboard interrupt processor status
                                        ;word to 140 (octal) (set the Keyboard inter-
                                        ;rupt priority level to 3).
        MOV #BPRNUM,BPRADR              ;Load the buffer preset register.
        MOV #CSRNUM,CSRADR              ;Load the control status register. This com-
                                        ;mand actually starts the clock operating.
        RTS PC                          ;Return to calling program.
```

The following program section, KEPTIM, is the clock's interrupt on overflow service routine. The function of KEPTIM is to keep track of the time that has passed since the clock was started. The counters for the hours, minutes, seconds, and milliseconds are adjusted as need be after each clock overflow. KEPTIM also checks the flag overrun (FOR) bit of the CSR to see if more than one overflow has occurred since the last time it serviced the clock. Lastly, KEPTIM clears the overflow flag (OVFLO FLAG) and FOR bits before it returns control to the main program.

```
 PSECT KEPTIM,RW,I,GBL,REL,CON
 GLOBL KEPTIM,FORMSG,CLKBLK,HOUR,MINUTE,SECOND,MILSEC,CSRADR


 CSRADR= 170420                         ;The address of the control status register of
```

```
                                        ;the clock is 170420 (octal).
CHKFOR = 010000                         ;This "mask" number is used to check whether
                                        ;the FOR bit is set.
CLRFOR = 010200                         ;This "mask" number is used to clear the FOR
                                        ;and the OVFLO FLAG bits.
;
KEPTIM: BIT #CHKFOR,CSRADR              ;If the FOR bit is clear, branch to continue
        BEQ NOFOR                       ;the routine. Otherwise, print an error mes-
        JSR PC,FORMSG                   ;sage saying that a flag overrrun occurred.
;
;
NOFOR:  INC MILSEC                      ;Increment the millisecond counter.
        CMP MILSEC,#001750              ;Compare the present millisecond count to
        BLO CLEAR                       ;1000 (decimal). If MILSEC is lower than
                                        ;1000, branch to CLEAR.
        CLR MILSEC                      ;Since MILSEC is 1000 or greater, clear it,

        INC SECOND                      ;and increment the second counter.
        CMP SECOND,#74                  ;Compare the present second count to 60
        BLO CLEAR                       ;(decimal). If SECOND is lower than 60,
                                        ;branch to CLEAR.
        CLR SECOND                      ;Since SECOND is 60 or greater, clear it,

        INC MINUTE                      ;and increment the minute count.
        CMP MINUTE,#74                  ;Compare the present minute count to 60
        BLO CLEAR                       ;(decimal). If MINUTE is lower than 60
                                        ;branch to CLEAR.
        CLR MINUTE                      ;Since MINUTE is 60 or greater, clear it,

        INC HOUR                        ;and increment the hour count.

CLEAR:  BIC #CLRFOR,CSRADR              ;Clear the OVFLO FLAG and FOR bits if they
                                        ;are set.
        RTI                             ;Return from this interrupt service routine.


;
;
;
; The following program segment, STPCLK, turns off the clock by sending a
; "shut off code" to the control status register of the clock.
;
;
;
.PSECT STPCLK,RW,I,GBL,REL,CON
.GLOBL STPCLK,CSRADR
;
CSRADR = 170420                         ;The address of the control status register
                                        ;of the clock is 170420 (octal).
KILCLK = 000002                         ;This number, when inserted into the control
                                        ;status register, will stop the clock.
KEYVEC = 000060                         ;The address of the first word of the inter-
                                        ;rupt vector for the keyboard. The second word
                                        ;of the keyboard interrupt vector contains the
                                        ;processor status word (PSW) that will be used
                                        ;when a keyboard interrupt is being serviced.
                                        ;The current PSW value (140) should be reset
                                        ;to the value that it had before it was alter-
                                        ;ed by the STRCLK routine (340). This will
                                        ;reset the keyboard interrupt priority level
                                        ;back to 7 (highest priority) from its cur-
```

```
                                            ;rent level 3.


STPCLK: MOV #KILCLK,CSRADR                  ;Stop the clock.
        MOV #000340,KEYVEC+2                ;Reset the keyboard interrupt processor status
                                            ;word back to its original value.
        RTS PC                              ;Return to the calling program.
```

The following program segment, FORMSG, prints out a message to the user, warning him that a flag overrun has occurred. A flag overrrun occurs if the clock overflows when the overflow flag is still set from the last over-flow, that is, the interrupt service routine, which clears the overflow flag, has not finished servicing the previous overflow.

```
 PSECT FORMSG,RW,I,GBL,REL,CON
 GLOBL FORMSG
 MCALL .PRINT

FORMSG: .PRINT #MESSAG                      ;Print a warning to the user.
        RTS    PC                           ;Return to the calling program.
MESSAG: .ASCIZ /A CLOCK OVERFLOW OVERRUN HAS OCCURRED/
.EVEN                                       ;This macro statement is needed because the
                                            ;last statement filled up an odd number of
                                            ;bytes, and every instruction must begin on
                                            ;an even numbered address.
 END
```

```
      SUBROUTINE  SETTIM ( H, M, S, MS )
C
C General Description:
C
C Subroutine SETTIM simply provides an easy means for the programmer to reset
C the COMMON block variables: HOUR, MINUTE, SECOND, MILSEC.  SETTIM is simply
C a series of assignment statements that equate the dummy argument to the
C equivalent COMMON block variable.
C
C Calling Example:
C
C       CALL SETTIM ( 1, 2, 3, 4 )
C
C This call will set the value of HOUR to 1, MINUTE to 2, SECOND to 3, and
C MILSEC to 4.  The arguments should be integer constants or variables.
C
C Arguments:
C
C       Formal:
C
C               H:      INTEGER*2. Present value of time in hours.
C               M:      INTEGER*2. Present value of time in minutes.
C               S:      INTEGER*2. Present value of time in seconds.
C               MS:     INTEGER*2. Present value of time in milliseconds.
C
C
C       Common blocks:
C
C               COMMON  /CLKBLK/   HOUR, MINUTE, SECOND, MILSEC
C
C               HOUR:   INTEGER*2. Present value of time in hours.
C               MINUTE: INTEGER*2. Present value of time in minutes.
C               SECOND: INTEGER*2. Present value of time in seconds.
C               MILSEC: INTEGER*2. Present value of time in milliseconds.
C
C
C External References: None
C
C Written by Fred Rasmussen
C
C Spring 1984
C
C Manufacturing Systems
C Modeling and Control Laboratory
C Department of Mechanical Engineering
C University of Michigan
C Ann Arbor, Michigan  48104-2125
C
C
C
C
C Declarations
C
      INTEGER HOUR,MINUTE,SECOND,MILSEC,H,M,S,MS
C
C Common blocks
C
      COMMON /CLKBLK/ HOUR, MINUTE, SECOND. MILSEC
C
C Set the time.
C
```

```
      HOUR   = H
      MINUTE = M
      SECOND = S
      MILSEC = MS
C
C Return to the calling program.
C
      RETURN
      END
```

PEEK AND POKE ROUTINES

The following pages contain
the source code for the routines:

PEEK
POKE

```
;                               **************
;                               * PEKPOK.MAC *
;                               **************
;
;
;
;
;
; General Description:
;
; Subroutines PEEK and POKE are FORTRAN callable assembly language routines
; that read the contents of a particular memory location and return that
; value, and write a certain value to a given memory location, respectively.
; The values and addresses entered in the FORTRAN calling program should be
; given in decimal form.
;
;
; Arguments:
;
;       Common blocks:  None
;
;       Formal arguments:
;
;                         ADDRES - The FORTRAN INTEGER*2 variable whose value i
;                                  the address that is to be PEEKED or POKED.
;                                  In assembly language, the address of ADDRES
;                                  is passed in location 2(R5) from FORTRAN.
;
;                         VALUE  - The FORTRAN INTEGER*2 variable whose value i
;                                  the number that is to be PEEKED or POKED.
;                                  In assembly language, the address of VALUE
;                                  is passed in location 4(R5) from FORTRAN.
;
; Calling syntax:
;
;       CALL PEEK ( ADDRES, VALUE )
;       CALL POKE ( ADDRES, VALUE )
;
; Calling example:
;
;       CALL PEEK ( 0, IVALUE )
;
;               This call causes the contents of memory location 0 to be
;               assigned to the variable IVALUE.
;
;
;       CALL PEEK ( 1, 30 )
;
;               This call causes the contents of memory location 1 to be
;               loaded with the value 30.
;
;
; External References:
;
;               None
;
;
; Written by Fred Rasmussen
;
; Spring 1984
;
```

```
; Manufacturing Systems
; Modeling and Control Laboratory
; Department of Mechanical Engineering
; University of Michigan
; Ann Arbor, Michigan  48104-2125
;
;
;
;
        .TITLE SUBROUTINES PEEK AND POKE
        .PSECT PEEK,RW,I,GBL,REL,CON
        .GLOBL PEEK
        .SBTTL SUBROUTINE PEEK
;
;
;
PEEK::  MOV R0, -(SP)               ;Store the contents of register 0 on stack.
        MOV @2(R5), R0              ;Move the desired address (contents of
                                    ;ADDRES) into register 0.
        MOV (R0), @4(R5)            ;Move the contents of the desired address
                                    ;into the contents of VALUE.
        MOV (SP)+, R0               ;Restore the value of register 0 from the
                                    ;stack.
        RTS PC                      ;Return to the calling program.
;
;
;
;
        .SBTTL SUBROUTINE POKE
        .PSECT POKE,RW,I,GBL,REL,CON
        .GLOBL POKE
;
;
POKE::  MOV R0, -(SP)               ;Store register 0 on the stack for later
                                    ;retrieval.
        MOV @2(R5), R0              ;Move the desired address (contents of
                                    ;ADDRES) into R0.
        MOV @4(R5), (R0)            ;Move the contents of VALUE into the
                                    ;desired address.
        MOV (SP)+, R0               ;Restore register 0 to its original value
                                    ;from the stack.
        RTS PC                      ;Return to the calling program.
        .END
```