# USER MANUAL

# LINKAGE

PDP-9 Facilities

Carnegie-Mellon University
Hybrid Computation Laboratory
September, 1968

## PREFACE

The linkage routines described in this section are those which relate to PDP-9 facilities. These routines may be called by PDP-9 stand-alone FORTRAN IV programs, or may be called for additional facilities in hybrid FORTRAN IV programs. Linkage routines relating to EAI 693 (hybrid interface) facilities are described in the EAI 693 LINKAGE section of this manual.

TABLE OF CONTENTS

## INDEX OF TABLES

## 1.0 OPERATOR COMMUNICATION

The routines in this section enable the programmer/operator to communicate with a running program from the PDP-9 console in an efficient and convenient manner.

### 1.1 ACCUMULATOR (SENSE) SWITCHES

There are 18 toggle switches located in the right middle of the PDP-9 operator console (under the REGISTER display lights). These are the accumulator switches, numbered from Ø to 17. The state of these switches (up or down, 1 or Ø, respectively) can be monitored by the PDP-9 under program control. The PDP-9 (or analog/hybrid) user thus has 18 sense switches at his disposal.

1.2 PROGRAM: Read Single AC Switch (Function)

NAME: RSAC

PURPOSE: 1. Read the state of an AC switch on the PDP-9 console.

AUTHOR: C. L. Cross

CALLING SEQUENCE: RSAC (ISWITCH)

ARGUMENTS: ISWITCH specifies the bit position (0-17) of the AC switches to be read.

RETURN: RSAC is TRUE if the specified AC switch was in the UP position.
RSAC is FALSE if the specified AC switch was in the DOWN position, or if an illegal switch was requested.

EXECUTION TIME: 32 to 41 μsec

STORAGE REQUIREMENTS: 27 locations

PROGRAMMING HINTS: RSAC must be declared LOGICAL in the FORTRAN IV calling program.

1.3 PROGRAM: Read Bank of AC Switches

NAME: RBAC

PURPOSE: 1. Read the state of all the AC switches on the PDP-9 console.

AUTHOR: C. L. Cross

CALLING SEQUENCE: CALL RBAC (IPATTERN)

ARGUMENTS: Bits 0-17 specify the state of the corresponding AC switches: if bit i of IPATTERN is "1", then AC switch i was in the UP position; if bit i of IPATTERN is "0", then AC switch i was in the DOWN position.

EXECUTION TIME: 18 μsec

STORAGE REQUIREMENTS: 12 locations

## 2.0 AUTOMATIC PRIORITY INTERRUPT CONTROL

The CMU Hybrid Computation Laboratory PDP-9 configuration includes the Automatic Priority Interrupt (API) option. This option facilitates those applications which require a sophisticated interrupt system.

### 2.1 API HARDWARE DESCRIPTION

The multilevel automatic priority interrupt option (API) affords immediate access to data handling subroutines on a ranked priority basis. Of the eight priority levels added by this option the four higher levels are assigned to device (hardware) use, and the lower four are assigned to software use. The priority levels are fully nested; i.e., a higher priority request can interrupt in-process servicing of a lower priority. The API identifies the source of an interrupt directly, through distinct channel assignments. Also included in this option are provisions for programmed raising of the active program segment to a priority level higher than the normal assignment, when the situation requires exclusion of interrupt requests at specific priority levels. All API levels are ranked above the standard program interrupt (PI - does not automatically distinguish the cause of the program interrupt), which is in turned ranked above normal program segments of null priority (see Table 1).

The API operates in the following manner. A hardware device requests service by transmitting an interrupt request signal to the central processor on a line corresponding to its specific, preassigned priority level. If this priority level is higher than the priority of the device which requested the currently active program segment, an interrupt is granted to the new device. Upon receipt of the grant signal, the device transmits its channel-address back to the processor. The processor then executes the instruction in this address, which is always a jump-to-subroutine to the device service subroutine. The new priority level is remembered and no further servicing of this or lower priority levels is permitted until the device service subroutine is exited.

This priority network insures that high data rate or critical devices will always interrupt slower device service routines while holding still lower priority interrupt requests off line until they can be serviced.

TABLE 1. CMU Hybrid Lab API Utilization

| increasing priority | | PRIORITY | DEVICE |
|---|---|---|---|
| highest | API | Ø | hybrid interface - EAI 68Ø overload interrupt |
| | API | 1 | DEC tape<br>real-time clock |
| | API | 2 | paper tape reader<br>hybrid interface - general purpose interrupts |
| | API | 3 | (reserved for dataphone to UNIVAC 11Ø8) |
| | API | 4<br>5<br>6<br>7 | software levels |
| | | PI | paper tape punch<br>teletype |
| lowest | | null<br>(no priority) | normal program level |

## 2.2 API SOFTWARE UTILIZATION

The chief advantage of this API system lies in the proper use of the software levels. In the real-time environment, it is necessary to maintain data input/output flow, but it is not possible to perform long, complex calculations at priority levels which shut out these data transfers. With the API, a high priority data input routine which recognizes the need for the complex calculation can call for it with a software level interrupt. Since the calculation is performed at a lower priority than the data handling, the latter can go on undisturbed. Further, there is no need to interface the data collection routine with the lowest priority (background) program which may run independently of the real-time system.

The API also offers programmed priority changes. In order for an interruptible program to change parameters in an interrupt service subroutine, the priority interrupt system is normally turned off while the changes are effected. Unfortunately, all interrupts are shut out during this time including those that indicate machine errors or are vital to control real time processes. Thus, the API has been designed so that a program segment may raise its priority only high enough to shut out those devices whose service routines require changes.

Since the Keyboard Monitor I/O handlers operate at API software level 4, program segments of priority 4 or higher cannot initiate I/O operations. Also, no program segment can initiate I/O on a device assigned to a lower priority than the program segment.

EXAMPLE 1:

Suppose a hybrid problem has the following digital requirements:

1. Periodic (say 10 cps) interrupting of the digital program to sample data being generated by the analog program.
2. Re-initialization of both the analog and digital programs whenever the analog overloads.

3. Updating of analog program parameters by the digital computer upon occurrence of some special situation in the analog program.

Suppose further that #2 has the highest priority and #1 has the lowest priority. An outline of a digital program which would satisfy these requirements is as follows:

```
      SUBROUTINE FAULT
C     THIS SUBROUTINE WILL BE ASSOCIATED WITH
C         EAI 68Ø OVERLOAD INTERRUPTS
C     WHEN THE 68Ø OVERLOADS, THE RUNNING PROGRAM
C         SEGMENT WILL BE INTERRUPTED AND CONTROL
C         WILL BE PASSED TO THIS SUBROUTINE WITH API
C         LEVEL Ø ACTIVATED, WHICH WILL EFFECTIVELY
C         SHUT OUT ANY OTHER INTERRUPTS
      (re-initialize analog/digital programs)
C     EXECUTION OF THE FOLLOWING STATEMENT RELEASES
C         API LEVEL Ø AND RETURNS CONTROL TO THE
C         INTERRUPTED PROGRAM SEGMENT
      RETURN
      END


      SUBROUTINE TIME
C     THIS SUBROUTINE WILL BE ENTERED ONCE EVERY
C         1ØØ MILLISECONDS UPON OCCURENCE OF A
C         REAL-TIME CLOCK OVERFLOW ON API LEVEL 1
      EXTERNAL SAMPLE
C     THE FOLLOWING STATEMENT ENTERS A REQUEST
C         FOR EXECUTION OF SUBROUTINE SAMPLE
C         AT API LEVEL 5 (SOFTWARE) WHENEVER
C         THE DIGITAL HARDWARE CAN GRANT AN
C         INTERRUPT AT API LEVEL 5
      CALL ISLI (5, SAMPLE, IERROR)
C     THE FOLLOWING STATEMENT RELEASES API LEVEL 1
      RETURN
      END


      SUBROUTINE GPIØ
C     THIS SUBROUTINE WILL BE ASSOCIATED WITH GENERAL
C         PURPOSE INTERRUPT Ø FROM THE ANALOG PATCHBOARD
C     THE SPECIAL SITUATION IN THE ANALOG PROGRAM WILL BE
C         PATCHED TO CAUSE THE GPIØ HOLE TO GO HIGH ON THE
C         ANALOG PATCHBOARD WHICH WILL CAUSE AN API LEVEL
C         2 INTERRUPT
      (update analog program parameters)
C     THE FOLLOWING STATEMENT RELEASES API LEVEL 2
      RETURN
      END
      SUBROUTINE SAMPLE
C     THIS SUBROUTINE SAMPLES THE REQUIRED DATA FROM
C         THE ANALOG COMPUTER AND IS INITIATED BY
C         SUBROUTINE TIME
C     NOTE THAT IT IS EXECUTED AT API LEVEL 5
C         WHICH IS A LOWER PRIORITY THAN
C         THAN #2 OR #3 CONDITION PROCESSING
      (sample data from the analog program)
C     THE FOLLOWING STATEMENT RELEASES API LEVEL 5
      RETURN
      END


C     MAIN PROGRAM
      EXTERNAL FAULT, TIME, GPIØ
C     INITIALIZE THE INTERFACE AND LINKAGE ROUTINES
      CALL INIT (IERROR)
C     ASSOCIATE SUBROUTINE FAULT WITH OVERLOAD INTERRUPTS
      CALL IOVL (FAULT)
C     ASSOCIATE SUBROUTINE TIME WITH REAL TIME CLOCK INTERRUPTS
          (6 x 16-2/3 MILLISECONDS = 100 MILLISECONDS = 10 CPS)
      CALL LTCH (6,TIME,IERROR)
C     ASSOCIATE SUBROUTINE GPIØ WITH SPECIAL CONDITION (#3) INTERRUPTS
      CALL IGPI (Ø,GPIØ,IERROR)
      (main program)
      END
```

## 2.3 PROGRAM: Initialize (PDP-9 only)

NAME: INIT9

PURPOSE: 1. To provide the necessary control and linkage for timer and software level interrupt processing.

AUTHOR: C. L. Cross

CALLING SEQUENCE: CALL INIT9

ARGUMENTS: none

EXECUTION TIME: 27 μsec

STORAGE REQUIREMENTS: 122 locations

COMMENTS:

1. INIT9 is provided as an abbreviated version of INIT (see hybrid interface linkage routines write-up) for those FORTRAN IV programs which use the linkage routines in this section only. In such a situation, either INIT or INIT9 (but not both) may be used.
2. A call INIT9 statement is required before execution of LTSH, LTRH, LTCH, DTIM, or ISLI.
3. Havoc results if a user attempts to load both INIT and INIT9 from one FORTRAN IV program.
4. INIT9 may be called more than once: each time it is called,the timer will be disabled and the API system will be re-initialized.

## 2.4 PROGRAM: Restore API

NAME: RAPI

PURPOSE: 1. Enable the Automatic Priority Interrupt (API) system.

AUTHOR: C. L. Cross

CALLING SEQUENCE: CALL RAPI

ARGUMENTS: none

EXECUTION TIME: 8 μsec

STORAGE REQUIREMENTS: 5 locations

COMMENTS: 1. If the API system is already enabled, execution of RAPI will have no effect.

2.5 PROGRAM: Suspend API

NAME: SAPI

PURPOSE: 1. Disable the Automatic Priority Interrupt (API) system.

AUTHOR: C. L. Cross

CALLING SEQUENCE: CALL SAPI

ARGUMENTS: none

EXECUTION TIME: 7 μsec

STORAGE REQUIREMENTS: 4 locations

COMMENTS: 1. If the API system is already disabled, execution of SAPI will have no effect.

2. The timer should be disabled before execution of SAPI: uncoverable monitor errors will result if a timer interrupt occurs when the API system is disabled.

## 2.6 PROGRAM: Read Active API Level (Function)

NAME: LEVL

PURPOSE: 1. Read the number of the currently active API priority level.

AUTHOR: C. L. Cross

CALLING SEQUENCE: LEVL (DUMMY)

ARGUMENTS: DUMMY is any dummy argument - it is never referenced.

RETURN: LEVL = number of currently active API priority level (0-7); or
LEVL = -1 if no API priority level is currently active.

EXECUTION TIME:

| | |
|---|---|
| for LEVL = 0, | time = 16 μsec |
| = 1 | = 26 μsec |
| = 2 | = 36 μsec |
| = 3 | = 46 μsec |
| = 4 | = 56 μsec |
| = 5 | = 66 μsec |
| = 6 | = 76 μsec |
| = 7 | = 86 μsec |
| =-1 | = 89 μsec |

STORAGE REQUIREMENTS: 18 locations

## 2.7 PROGRAM: Raise Priority Level (Function)

NAME: RPRI

PURPOSE: 1. Raise the API priority of the requesting program segment.

AUTHOR: C. L. Cross

CALLING SEQUENCE: RPRI (ILEVEL)

ARGUMENTS: ILEVEL specifies the new priority level desired.

RETURN: RPRI is TRUE and the priority is raised to ILEVEL if ILEVEL < LEVL (current API level).
RPRI is FALSE and the priority is unchanged if ILEVEL ≥ LEVL (current API level), or if an illegal priority level is requested, or if the API system is disabled.

EXECUTION TIME: 22 to 146 μsec

STORAGE REQUIREMENTS: 41 locations

COMMENTS:

1. Program segments which call on RPRI may be nested as long as successive calls of RPRI request successively higher levels.

2. The PDP-9 will hang up (in an infinite loop) if any I/O is attempted within a program segment of priority 4 or higher; or if high-speed punch or teletype I/O is attempted within a program segment of priority 5, 6, or 7.

3. If RPRI is TRUE, then the priority was raised to ILEVEL 4 μsec before RPRI returned to the calling program.

PROGRAMMING HINTS: RPRI must be declared LOGICAL in the FORTRAN IV calling program.

2.8 PROGRAM: Release Priority Level

NAME: RELP

PURPOSE: 1. Restore the priority of the last program segment to raise its priority back to normal (null) program priority.

AUTHOR: C. L. Cross

CALLING SEQUENCE: CALL RELP

ARGUMENTS: none

EXECUTION TIME: 6 μsec

STORAGE REQUIREMENTS: 3 locations

COMMENTS: 1. RELP has no effect if all program segments are currently at normal (null) program priority.

2. RELP should not be used to debreak from the HANDLER of LTSH, LTRH, LTCH, IGPI, or ISLI.

3. The highest active API priority level is released 4 μsec after initiating execution of RELP.

## 2.9 PROGRAM: Initialize Software Level Interrupt

NAME: ISLI

PURPOSE: 1. Request a software level interrupt and associate it with a user's subroutine.

AUTHOR: C. L. Cross

CALLING SEQUENCE: CALL ISLI (ILEVEL, HANDLER, IERROR)

ARGUMENTS: ILEVEL specifies the software level interrupt to be requested (4, 5, 6, or 7).

HANDLER specifies the parameter-less FORTRAN IV user subroutine which is to handle the software level interrupt.

IERROR indicates errors as follows:

```
IERROR = Ø no error
       = 1 illegal software level requested
       = 7 API system is disabled
         (ISLI cannot proceed)
```

EXECUTION TIME: 69(4), 73(6), 74(5), or 76(7) μsec

STORAGE REQUIREMENT: 95 locations

COMMENTS:

1. HANDLER is replaced by the system halt routine after occurence of the software interrupt and before execution of HANDLER.

2. Execution of HANDLER is initiated 21 μsec after the software level interrupt request is granted.

3. Control is returned to the interrupted program segment 12 μsec after HANDLER RETURN's.

4. The PDP-9 will hang up (in an infinite loop) if any I/O is attempted within a program segment of priority 4; or if high-speed punch or teletype I/O is attempted within a program segment of priority 5, 6, or 7.

5. A CALL INIT9 (or CALL INIT) statement must be executed prior to calling ISLI.

PROGRAMMING HINTS: HANDLER must be declared EXTERNAL in any FORTRAN IV program which contains a CALL ISLI statement.

2.10 PROGRAM: Read API Status

NAME: APIS

PURPOSE: 1. Read the Status of the API system.

AUTHOR: Christopher L. Cross

CALLING SEQUENCE: CALL APIS (IPATTERN)

ARGUMENTS: Bits 0 and 2-17 of IPATTERN will specify the status of the API system as follows:

```
    bit Ø=1:  API is enabled
         =Ø:  API is disabled
    bit 2 + i=1:  a device is requesting service on
                  API level i, Ø≤i≤7
             =Ø:  no devices are requesting service on
                  API level i, Ø≤i≤7
    bit 10 +i=1:  API level i is active, Ø≤i≤7
             =Ø:  API level i is inactive, Ø≤i≤7
Bit 1 of IPATTERN is unused.
```

EXECUTION TIME: 21 $\mu$sec

STORAGE REQUIRED: 12 locations

COMMENTS: 1. A priority level is active if interrupt servicing has commenced at that level, or if a raise priority (see RPRI) has been executed to that level.

## 3.0 REAL-TIME CLOCK

The real-time clock generates a clock pulse every 16.7 msec* (60 cps) to increment a time counter stored in system memory. The counter initiates a program interrupt on API level 1 when a programmed preset time interval is completed. The clock can be enabled or disabled under program control. The console CLK switch must be in the down position (or the entire console LOCKed) to permit programmed control of the facility. The clock remains disabled with the switch in the up position. Depressing the I/O RESET console key also disables the facility.

Four linkage routines are provided for control of the clock facility. LTSH provides a single time period for applications such as time delays. LTRH and LTCH provide periodic time interval markers for applications such as regular data transferrals. DTIM disables the timer. Only time periods which are multiples of 16-2/3 msec can be provided.

The PDP-9 will hang up (in an infinite loop) if any I/O is attempted within the HANDLER of LTSH, LTRH, and LTCH.

*NOTE: The first clock pulse after enabling the clock facility occurs anywhere from 0 to 17 msec later.

## 3.1 PROGRAM: Load Timer Once and Set Handler

NAME: LTSH

PURPOSE: 1. To cause an interrupt to occur after the specified time duration.

AUTHOR: C. L. Cross

CALLING SEQUENCE: CALL LTSH (ITIME, HANDLER, IERROR)

ARGUMENTS: ITIME specifies the time duration before the interrupt as follows: $t = (ITIME)*(16\text{-}2/3 \text{ msec})$, where $ITIME > 0$.

HANDLER specifies the parameter-less FORTRAN IV user subroutine which is to handle the clock overflow interrupt when it occurs.

IERROR indicates errors as follows:
IERROR = 0 no error
= 1 illegal ITIME requested ($ITIME \leq 0$)

EXECUTION TIME: 64 μsec

STORAGE REQUIREMENTS: 51 locations

COMMENTS:

1. Calling LTSH, LTRH, or LTCH from subroutine HANDLER (directly or indirectly) will result in unpredictable errors if subroutine HANDLER does not RETURN within one (new) time period after calling LTSH, LTRH, or LTCH.
2. HANDLER is replaced by the system halt routine after occurrence of the clock overflow and execution of HANDLER.
3. Execution of HANDLER is initiated 24 μsec after the clock overflow interrupt request is granted.
4. Control is returned to the interrupted program segment 20 μsec after HANDLER RETURNs.
5. A CALL INIT9 (or CALL INIT) statement must be executed prior to calling LTSH.
6. If a timer interrupt occurs when the API system is disabled an unrecoverable monitor error will result.

PROGRAMMING HINTS: HANDLER must be declared EXTERNAL in any FORTRAN IV program which contains a CALL LTSH statement.

## 3.2 PROGRAM: Load Timer Repetitively and Set Handler

NAME: LTRH

PURPOSE: 1. To cause the occurrence of variable-frequency real-time interrupts whose period is dependent on interrupt processing time.

AUTHOR: C. L. Cross

CALLING SEQUENCE: CALL LTRH (ITIME, HANDLER, ERROR)

ARGUMENTS: ITIME specifies the time period between interrupts as follows: $t = (\text{ITIME})*(16\text{-}2/3 \text{ msec})$, where $\text{ITIME} > 0$.

HANDLER specifies the parameter-less FORTRAN IV user subroutine which is to handle the clock overflow interrupts when they occur.

IERROR indicates errors as follows:

IERROR = 0 no error

= 1 illegal ITIME requested ($\text{ITIME} \leq 0$)

EXECUTION TIME: 67 μsec

STORAGE REQUIREMENTS: 53 locations

COMMENTS:

1. The first interrupt occurs one time period after calling LTRH. Subsequent interrupts occur (14 microseconds + 1 time period) after HANDLER RETURNs.
2. Calling LTSH, LTRH, or LTCH from subroutine HANDLER (directly or indirectly) will result in unpredictable errors if subroutine HANDLER does not RETURN within one (new) time period after calling LTSH, LTRH, or LTCH.
3. Execution of HANDLER is initiated 24 μsec after the clock overflow interrupt request is granted.
4. Control is returned to the interrupted program segment 27 μsec after HANDLER RETURNs.
5. A CALL INIT9 (or CALL INIT ) statement must be executed prior to calling LTRH.
6. If a timer interrupt occurs when the API system is disabled an unrecoverable monitor error will result.

PROGRAMMING HINTS: HANDLER must be declared EXTERNAL in any FORTRAN IV program which contains a CALL LTRH statement.

## 3.3 PROGRAM: Load Timer Continuously and Set Handler

NAME: LTCH

PURPOSE: 1. To cause the occurrence of variable-frequency real-time interrupts whose period is independent of interrupt processing time.

AUTHOR: C. L. Cross

CALLING SEQUENCE: CALL LTCH (ITIME, HANDLER, IERROR)

ARGUMENTS: ITIME specifies the time period between interrupts as follows: $t = (\text{ITIME})*(16\text{-}2/3 \text{ msec})$, where $\text{ITIME} > 0$

HANDLER specifies the parameter-less FORTRAN IV user subroutine which is to handle the clock overflow interrupts when they occur.

IERROR indicates errors as follows:
IERROR = 0 no error
= 1 illegal ITIME requested ($\text{ITIME} \le 0$)

EXECUTION TIME: 66 μsec

STORAGE REQUIREMENTS: 52 locations

COMMENTS:

1. The interrupts will occur with a constant period of t, regardless of the execution time of HANDLER. Thus, results are unpredictable if HANDLER has an execution time greater than t.

2. Calling LTSH, LTRH, or LTCH from subroutine HANDLER (directly or indirectly) will result in unpredictable errors if subroutine HANDLER does not RETURN within one (new) time period after calling LTSH, LTRH, or LTCH.

3. Execution of HANDLER is initiated 32 μsec after the clock overflow interrupt request is granted.

4. Control is returned to the interrupted program segment 13 μsec after HANDLER RETURNs.

5. A CALL INIT9 (or CALL INIT) statement must be executed prior to calling LTCH.

6. If a timer interrupt occurs when the API system is disabled an unrecoverable monitor error will result.

PROGRAMMING HINTS: HANDLER must be declared EXTERNAL in any FORTRAN IV program which contains a CALL LTCH statement.

3.4 PROGRAM: Disable Timer

NAME: DTIM

PURPOSE: 1. To disable the variable-frequency real-time interrupts initiated by LTRH or LTCH.

AUTHOR: C. L. Cross

CALLING SEQUENCE: CALL DTIM

ARGUMENTS: none

EXECUTION TIME: 16 μsec

STORAGE REQUIREMENTS: 10 locations

COMMENTS: 1. If the clock is already disabled, execution of DTIM will have no effect.

2. DTIM may be called (directly or indirectly) from HANDLER of LTSH (redundant), LTRH, or LTCH. The effect of such a call on DTIM will be to prevent any further clock interrupts.

## 4.0 ARITHMETIC ROUTINES

### 4.1 NORMALIZATION

The PDP-9 is an 18 bit word-oriented machine with floating point arithmetic mechanized by system software routines utilizing the standard fixed point (integer) hardware. Due to the emphasis on digital computer speed inherent in hybrid programming, the linkage routines are designed to facilitate use of the full precision of the 18 bit PDP-9 word. This is accomplished by encouraging a normalized mode of operation in which all numerical information transmitted from, or to, the (high-speed) linkage routines is (or is assumed to be) in normalized integer form.

The use of normalized integer form in digital computer programming is closely related to the machine unit scaling of analog computer programming. All variables are scaled between + and - one digital machine unit which is, in the case of the PDP-9, 131,072 or $2^{17}$. For hybrid communication purposes, one digital machine unit is equivalent to one analog machine unit (10 volts for the 680). To preserve the normalized form, all multiplication and division must be accomplished with the MULT and DIVD linkage routines.

4.2 PROGRAM: Normalized Integer Multiply (Function)

NAME: MULT

PURPOSE: 1. To perform a normalized integer multiplication of two integer arguments.

AUTHOR: C. L. Cross

CALLING SEQUENCE: MULT (MPCAND, MPYER)

ARGUMENTS: MPCAND specifies the normalized integer multiplicand.
MPYER specifies the normalized integer multiplier.

RETURN: MULT = normalized integer product of MPCAND and MYPER.
(MULT = (MPCAND*MPYER)/131,072)

EXECUTION TIME: 50 (MULT $\geq$ 0) or 52 (MULT $<$ 0) $\mu$sec.

STORAGE REQUIREMENTS: 23 locations

4.3 PROGRAM: Normalized Integer Divide

NAME: DIVD

PURPOSE: 1. To perform a normalized integer division of two integer arguments.

AUTHOR: C. L. Cross

CALLING SEQUENCE: CALL DIVD (IDIVND, IDIVSR, IQUOT, IERROR)

ARGUMENTS: IDIVND specifies the normalized integer dividend.
IDIVSR specifies the normalized integer divisor.
IQUOT specifies the normalized integer quotient
(IQUOT = (131,072*IDIVND)/IDIVSR)
IERROR indicates errors as follows:
IERROR = 0 no error
= 1 IDIVND $\leq$ IDIVSR

EXECUTION TIME: 87 to 89 μsec

STORAGE REQUIREMENTS: 46 locations