ON-LINE RANGE PREDICTION SYSTEM (II)

September 1988
ON-LINE RANGE PREDICTION SYSTEM*

(II)

* Research supported by NASA-Ames Dryden under Grant NAG 2-303, "Survey and Analysis of X-Band Radar for Flight Research in the Western Aeronautical Test Range", Principal Investigator: Nhan Levan, Research Assistant: Tayfun Cataltepe, Department of Electrical Engineering, 7732 Boelter Hall, UCLA.

SEPTEMBER 1988
Table of contents

I. INTRODUCTION.........................................................1

II. TEST RESULTS.........................................................1

III. PROGRAM TERMINATION.............................................2

IV. TIME LAG BETWEEN INPUT AND OUTPUT..............................3

V. CONTINUOUS PROPAGATION...........................................4

VI. CONCLUSION..........................................................6

VII. REFERENCES.........................................................7

VIII. FIGURES..............................................................8

Fig. 1: Input data configuration.........................................9

Fig. 2: Input-output time lag...........................................10

Fig. 3: Continuous propagation.........................................10

Fig. 4: System block diagram...........................................11

IX. APPENDIX: PROGRAM LISTINGS.......................................12
I. INTRODUCTION:

This report is a follow-up to the interim technical report of July 1987 regarding the on-line range prediction system for the laser ranger at Crows Landing [1]. The on-line range prediction system is designed for providing a prediction of the target range in the case of a laser data dropout. It consists of real time implementation of a Kalman filter on an IBM PC/AT equipped with necessary hardware. The system was set up and tested at Crows Landing in the Fall of 1987.

This is a report of the improvements made on the on-line range prediction system during 1988. It is organized as follows. We begin by discussing and proposing solutions to the several problems encountered during system tests. Then, we explain the improvements made on the filter software, namely, accounting for the time lag and providing data continuously. Finally, we mention the ideas that can be considered in the future.

II. TEST RESULTS:

During the initial tests at Crows Landing, although the range output was stable, a jittering in the velocity output of the filter was observed. This problem was resolved during further tests performed at UCLA in Spring 1988 using an identical computer system and additional circuitry provided by Fred Shigemoto. The circuitry consisted of a 21 bit serial data generator simulating the incoming laser range data, and a serial to parallel converter that is also used in the original system at Crows Landing. Range and range velocity
outputs of the system were displayed on a simple digital display circuit.

It was found that the jittering problem was caused by the "data ready" signal which was a square wave with period \( T \), the sampling period of the filter. Thus, the "on" time of the ready pulse was also \( T \) seconds. But the sampling period is (and must be) longer than the time it takes to process a single data sample (2 milliseconds). Hence, when the system started checking for the ready pulse immediately after outputting the previously processed sample, it found a "high" signal even though no new sample had arrived. Consequently, the synchronization of the system with the arriving data samples was lost and this has resulted in unusual velocity estimates, or jittering.

For best performance, then, the ready signal should be a pulse having a width greater than 4 microseconds but less then the 2 milliseconds program execution time. Only in this case, each "data ready" pulse will correspond to a new incoming data sample and the synchronization of the system will be maintained.

III. PROGRAM TERMINATION:

The second problem is the termination of the program. At present, once it starts running it is not possible to stop the Kalman filter program using the keyboard commands except by system reset. Automatic stopping of the program occurs only when the range input is greater than 50 miles.

The assembly language routines and the main program are updated in a way that will enable normal termination of the execution upon request. The method is that an externally supplied termination request signal is recognized by the software and the program stops running. Bit 21, which is one
of the unused bits of the 32 bit input data is employed for this purpose (see Fig.1).

A minimal external hardware consisting of a toggle switch between 0 and 5 volts is required. Bit 21 is already connected to bit 5 of port 2 on the input board (pin 43 on the input board connector). A high value set manually on this bit via the switch causes the input subroutine to return a special number to the main program which detects it, resulting in the program termination. The number sent is $2^{21}$ (counts), which corresponds to a range greater than 50 miles and since the filter software is already set to stop whenever the input range data is greater than 50 miles, this causes normal termination of the program.

Since the input subroutine is periodically called by the main program, a stop request is recognized by the upcoming subroutine call, hence it is always possible to stop the program. When program termination is requested through the switch, the data at the input port is ignored.

IV. TIME LAG BETWEEN INPUT AND OUTPUT:

There are two major improvements which have been made on the existing Kalman filter. The first one regards the output and its corresponding input. Since it takes a finite amount of time to process a sample (around 2 milliseconds), the range output corresponds to the input sample that has arrived 2 milliseconds earlier than the output instant (see Fig.2).

One way of accounting for this time lag is by propagating the range estimate using the velocity estimate. That is, at the instant $nT+\Delta$, $\hat{R}_{n+\Delta}$ is sent out to the port instead of $\hat{R}_n$, where
\[ \hat{R}_{n+\Delta} = \hat{R}_n + \hat{V}_n \Delta, \]

and \( \Delta \) is the amount of time lag, which is around 2 milliseconds execution time plus the time it takes to do the above calculation. \( \hat{R}_n \) and \( \hat{V}_n \) are the estimates of range and velocity, respectively, calculated by the Kalman filter using the input data that has arrived at the instant \( nT \). The main program has been updated to do the propagation described. A listing can be found in the appendix.

V. CONTINUOUS PROPAGATION:

In its present form, after outputting the range and velocity estimates, the Kalman filter program waits for the next input sample. During this time period between the output and the arrival of the next sample, the system is idle in the sense that no data processing is performed. But, at the same time, the computer is not free to do any other task unless this task is integrated in the filter software.

This time period can be filled with range data updated by way of propagation. Currently, a multiplier and adder circuit is being designed at Moffett to perform this task. But, the propagation can be done in software rather than in hardware. The main program outputs the range and velocity estimates \( \hat{R}_n \) (or \( \hat{R}_{n+\Delta} \) as explained above) and \( \hat{V}_n \) by passing them to an assembly language subroutine which configures a 32 bit data out of them and sends it to the output port. After this operation, the range \( \hat{R}_n \) can be propagated using \( \hat{V}_n \) within the assembly language subroutine, and then it can be sent to the port again. As in the previous case,

\[ \hat{R}_{n+\tau} = \hat{R}_n + \hat{V}_n \tau \]
where \( \tau \) is the time it takes to do the above calculation. This operation is repeated until the next sample (or the data ready pulse) arrives as follows:

\[
\hat{R}_{n+kt} = \hat{R}_{n+(k-1)\tau} + \hat{V}_n \tau,
\]

where \( k \) is the repetition count (see Fig. 3). Note that the velocity is assumed to be constant at \( \hat{V}_n \) during this period of repeated propagation.

Statistically, the propagated range is the best possible estimate at that instant. Since the most recent observation (range input) is already utilized to obtain the Kalman filter estimates \( \hat{R}_n \) and \( \hat{V}_n \), and there is no new information available, the only way to estimate the range at any subsequent instant is by way of propagation [2].

The purpose of propagation is only to provide range data until the next measurement arrives, and it does not affect the operation of the Kalman filter. When the new sample arrives, the filter uses the sample to update the estimate \( \hat{R}_n \), not the propagated value. Also, since a linear time invariant model is assumed, the sampling interval \( T \) has to stay constant. If a sample arrives earlier than \( T \) seconds, the Kalman filter recognizes it as the next input and proceeds with the estimation, but this estimate is wrong since the calculations are based on constant sampling interval assumption.

The assembly language routines for handling input and output have been revised to do the tasks described above. In order to avoid too many subroutine calls and minimize the execution time, both of them are combined in a single assembly language subroutine. A listing is included in the appendix. The subroutine parameters are the output data to be configured, the range and velocity estimates, and the
input data. The range and velocity estimates are passed onto the I/O subroutine in both real and 32 bit integer formats. Integer versions are used for configuring the output data. Real numbers are used in the propagation calculations. The IBM PC/AT is equipped with a 80287 math co-processor which is capable of handling high accuracy real number arithmetic with simple instructions in the assembly language level.

VI. CONCLUSION:

In this report, we proposed solutions to the problems encountered during the initial system tests, namely, jittering in the velocity output and the program termination. Also, we explained several ideas for improving the system performance. These are regarding the input-output time lag and more effective utilization of the system by providing more range estimates between the input samples.

As a result of these changes, the software had to be revised and a minimal external hardware had to be added. As before, either on-site tests or tests at at UCLA with the test circuitry are necessary to verify the functionality of the system.

In the future, more improvements can be made on the system. Incorporating an atmospheric refraction correction scheme has been on the agenda from the beginning. Once the proposed changes in this report are tested, a suitable refraction correction algorithm may be integrated into the Kalman filtering software. Also, since it is essentially a programmable real time data processing system, other applications of the system are possible.
VII. REFERENCES:


VIII. FIGURES:
Fig. 1: Input data configuration
Fig. 2: Input -output time lag

Fig. 3: Continuous propagation
Fig. 4: System block diagram
IX. APPENDIX: PROGRAM LISTINGS
**rangepr2.for**

C******************************************************************************
C VERSION II
C THIS IS A TWO STATE KALMAN FILTER FOR LASER RANGE PREDICTION.
C BY TAKING ADVANTAGE OF THE SYSTEM DYNAMICS, MATRIX CALCULATIONS
C ARE ELIMINATED TO MINIMIZE EXECUTION TIME. SOME CONSTANTS THAT
C ARE USED IN THE PROGRAM ARE INITIALIZED AT THE BEGINNING IN ORDER
C TO SAVE CALCULATION TIME IN THE MAIN LOOP. ASSEMBLY LANGUAGE
C ROUTINES ARE USED FOR INPUT/OUTPUT FOR HIGH SPEED. SOME PARAMETERS
C ARE ENTERED INTERACTIVELY BY THE USER.
C
C DEFINITIONS OF SYMBOLS
C T : SAMPLING INTERVAL
C SCALE, OFFSET: PARAMETERS FOR CONVERTING COUNTS INTO FT
C XL1.S1, XL2.S2: ELEMENTS OF STATE ESTIMATE VECTOR
C XMNS1 : ONE STEP PREDICTION FOR RANGE
C V, IV : OBSERVED RANGE (FEET, COUNTS)
C IV0 : OBSERVED RANGE AT THE PREVIOUS INSTANT (CNTS)
C XL1.S1, IXPLS2, KOUT : INTEGER VARIABLES USED FOR I/O ROUTINE CALLS
C P11, P12, P22 : ELEMENTS OF ERROR COVARIANCE MATRIX
C H11, H12, H22 : ELEMENTS OF PREDICTION ERROR COVARIANCE
C FF11,
C FF12, FF22 : ELEMENTS OF STATE NOISE COVARIANCE
C GG : OBSERVATION NOISE VARIANCE
C SIGMA : STRAIGHT LINE FLIGHT VARIANCE
C MINVAR : MINIMUM STATE NOISE VARIANCE
C MAXVAR : MAXIMUM STATE NOISE VARIANCE
C Z1 : MAXIMUM INNOVATION AT WHICH MINVAR IS USED
C Z2 : MINIMUM INNOVATION AT WHICH MAXVAR IS USED
C
C ASSEMBLY LANGUAGE SUBROUTINES
C INOUT : READS THE 24 BIT DATA FROM THE INPUT PORT
C WHEN THE DATA READY SIGNAL COMES. WHEN THE READY
C SIGNAL IS NOT AVAILABLE, PROPAGATES THE RANGE
C (SECOND ARGUMENT) USING THE VELOCITY (THIRD ARG.).
C PROPAGATION LOOP TIME IS PLACED IN THE LAST
C ARGUMENT AND MUST NOT BE CHANGED.
C SENDS THE PROPAGATION TO OUTPUT PORT. READY
C SIGNAL IS CONNECTED TO BIT 24. THE INPUT DATA IS
C PLACED IN THE RETURN ARGUMENT. THE ARGUMENT MUST BE
C DECLARED AS 32 BIT INTEGER (SIXTH ARG.). ALSO,
C CONFIGURES A 32 BIT OUTPUT DATA (FIRST ARGUMENT)
C BY COMBINING THE 18 BITS OF THE FOURTH AND 12 BITS
C (MSB=SIGN) OF THE THIRD ARGUMENT. SENDS THE
C CONFIGURED ARGUMENT TO THE OUTPUT PORT. ALL INTEGER
C ARGUMENTS MUST BE DECLARED AS 32 BIT INTEGERS.
C THE SEVENTH ARGUMENT IS ONLY NEEDED IN THE ASSEMBLY
C ROUTINE AND NOTHING MUST BE PASSED TO SUBROUTINE
C IN THIS VARIABLE.
C
C INDAT2 : READS THE FIRST TWO CONSECUTIVE SAMPLES FOR
C INITIALIZATION DURING START UP, IF REQUESTED.
C******************************************************************************
INITIALIZATIONS

INTEGER*4 IV, IV0, IXPLS1, IXPLS2, KOUT, IDUMMY
REAL MINVAR, MAXVAR, MINVARA, MAXVARA
TWOTAU=0.0002

1 PRINT*, 'SCALE FACTOR = ?(FT/COUNT)'
READ*, SCALE
IF (SCALE.LE.0.0) GO TO 1
PRINT*, 'OFFSET = ?(FT)'
READ*, OFFSET
PRINT*, 'SAMPLING INTERVAL = ?(SEC)'
READ*, T

IF (T.LT.0.003) THEN
PRINT*, 'SAMPLING INTERVAL MUST BE >= 3 MILLISEC'
GO TO 2
ELSE
ENDIF
PRINT*, 'DO YOU WANT TO INITIALIZE THE RANGE AND RANGE VELOCITY'
PRINT*, 'OR USE THE FIRST TWO VALID SAMPLES FOR INITIALIZATION?'
PRINT*, '
3 PRINT*, '0 = USE THE FIRST TWO SAMPLES'
PRINT*, '1 = INITIALIZE MANUALLY
READ*, KTEST1
IF (KTEST1.LT.0.OR.KTEST1.GT.1) GO TO 3
IF (KTEST1.EQ.1) THEN
4 PRINT*, 'INITIAL VALUE OF RANGE = ?(FT)'
READ*, XPLS1
IF (XPLS1.LE.0.0) GO TO 4
PRINT*, 'INITIAL VALUE OF RANGE VELOCITY = ?(FT/SEC)'
READ*, XPLS2
IF (ABS(XPLS2).GT.1000.0) GO TO 5
IV=(XPLS1-OFFSET)/SCALE
IV0=-1
ELSE
ENDIF
PRINT*, '
PRINT*, 'THE PARAMETERS ARE :
PRINT*, '
PRINT 100, SCALE
100 FORMAT (1X, 'SCALE FACTOR = ',F11.3,' (FT/COUNT)')
PRINT 101, OFFSET
101 FORMAT (1X, 'OFFSET = ',F11.3,' (FT)')
PRINT 102, T
102 FORMAT (1X, 'SAMPLING INTERVAL = ',F11.3,' (SEC)')

GG=20.0
SIGMA=31.68
MINVAR=0.005*SIGMA
MAXVAR=2.0*MINVAR
Z1=0.01*SQRT(SIGMA)
Z2=100.0*Z1
P11=10.0
P12=0.0
P22=10.0
PRINT*, '
PRINT*, '
THE NOISE PARAMETERS ARE PRESET AS:

\[ \text{Observation Noise Variance (GG)} \]
\[ \text{Minimum State Noise Variance (MINVAR)} \]
\[ \text{Maximum State Noise Variance (MAXVAR)} \]
\[ \text{Maximum Innovation at which Minvar is used (Z1)} \]
\[ \text{Minimum Innovation at which Maxvar is used (Z2)} \]
\[ \text{Initial Value of Error Covariance Matrix (P)} \]

DO YOU WANT TO CHANGE THE NOISE PARAMETERS?

0 = NO, 1 = YES

OBSERVATION NOISE VARIANCE (GG) = ?

MINIMUM STATE NOISE VARIANCE (MINVAR) = ?

MAXIMUM STATE NOISE VARIANCE (MAXVAR) = ?

MAXVAR >= MINVAR

MINVAR MUST BE < MAXVAR, ENTER MINVAR AND MAXVAR AGAIN

ELSE

ENDIF
PRINT*, 'MAXIMUM INNOVATION AT WHICH MINVAR IS USED (Z1) = ?'
READ*, Z1A
IF (Z1A.LT.0.0) GO TO 12
Z1=Z1A
12 PRINT*, 'MINIMUM INNOVATION AT WHICH MAXVAR IS USED (Z2) = ?'
READ*, Z2A
IF (Z2A.LT.0.0) GO TO 13
Z2=Z2A
13 IF (Z2.LT.Z1) THEN
   PRINT*, 'Z2 MUST BE > Z1, ENTER Z1 AND Z2 AGAIN'
   PRINT*, '
   GO TO 11
ELSE
ENDIF
PRINT*, 'INITIAL VALUE OF ERROR COVARIANCE MATRIX :
PRINT*, 'P(1,1) = ?'
READ*, P11A
IF (P11A.LT.0.0) GO TO 14
P11=P11A
14 PRINT*, 'P(1,2) = ?'
READ*, P12A
IF (P12A.LT.0.0) GO TO 15
P12=P12A
15 PRINT*, 'P(2,2) = ?'
READ*, P22A
IF (P22A.LT.0.0) GO TO 16
P22=P22A
16 IF ((P11*P22-P12**2).LT.0.0) THEN
   PRINT*, 'ERROR COVARIANCE NOT POSITIVE DEFINITE, ENTER P AGAIN'
   PRINT*, '
   GO TO 13
ELSE
ENDIF
TSQ=T**2
TSQ2=TSQ/2.0
TCUB=T**3/3.0
TWOT=T*2.0
Q0=MINVAR/TCUB
Q1=MAXVAR/TCUB
SLOPE=(Q1-Q0)/(Z2-Z1)
ZINT=Q0-Z1*SLOPE
IF (KTEST1.EQ.1) GO TO 30
CALL INDAT2(IV0)
17 CALL INDAT2(IV)
IF (IV0.LT.0) GO TO 17
CALL INDAT2(IV)
IF (IV.LT.0) GO TO 17
IF (ABS(IV-IV0).GT.20) GO TO 17
XPLSI=IV*SCALE+OFFSET
XP_S2=(IV-IV0)*SCALE/T
IV0=IV
GO TO 30
MAIN LOOP STARTS

UPDATE STATE NOISE COVARIANCE (VARY LINEARLY WITH THE INNOVATION Z)

\( Z = \text{ABS}(\text{XMNS1} - V) \)

\begin{align*}
&\text{IF (Z.GE.22) GO TO 20} \\
&\text{IF (Z.LE.21) GO TO 30} \\
&\text{QR = Z*SLOPE+ZINT} \\
&\text{GO TO 40} \\
&\text{QR = Q1} \\
&\text{GO TO 40} \\
&\text{QR = Q0} \\
&\text{GO TO 40} \\
&\text{FF11 = QR*TCUB} \\
&\text{FF12 = QR*TSQ2} \\
&\text{FF22 = QR*T} \\
&\text{QR = Q1} \\
&\text{GO TO 40} \\
&\text{QR = Q0} \\
&\text{GO TO 40} \\
&\text{FF11 = QR*TCUB} \\
&\text{FF12 = QR*TSQ2} \\
&\text{FF22 = QR*T} \\
\end{align*}

CALCULATE PREDICTION ERROR COVARIANCE

\( H_1 = P_{11} + \text{TWO}_{1} P_{12} + P_{22} \text{TSQ} + FF_{11} \)
\( H_2 = P_{12} + T P_{22} + FF_{12} \)
\( H_2 = P_{22} + FF_{22} \)

STATE PROPAGATION (PREDICTION)

\( \text{XMNS1} = \text{XPLS1} + T \text{XPLS2} \)
\( E = H_1 + G G \)
\( D_1 = G G / E \)
\( D_3 = -H_2 / E \)

CONVERT COUNTS INTO FEET, CHECK FOR RUN/STOP,
CHECK FOR DATA VALID/INVALID, CHECK ONE STEP DIFFERENCE

\( V = IV * \text{SCALE} + \text{OFFSET} \)
\( \text{IF (V.GT.260000.0) GO TO 99} \)
\( \text{IF (IV.LT.0) GO TO 55} \)
\( \text{IF (IV0.GE.0.AND.ABS(IV-IV0).GT.20) GO TO 55} \)

STATE UPDATE (ESTIMATE)

\( \text{XPLS1} = D_1 \text{XMNS1} + H_1 V / E \)
\( \text{XPLS2} = D_3 * (\text{XMNS1} - V) + \text{XPLS2} \)

CALCULATE ERROR COVARIANCE

\( P_{11} = D_1 H_1 \)
\( P_{12} = D_1 H_1 \)
\( P_{22} = D_3 H_1 + H_2 \)
\( \text{GO TO 56} \)

IF OBSERVATION INVALID, USE ONE STEP PREDICTION

\( \text{XPLS1} = \text{XMNS1} \)
\( P_{11} = H_1 \)
\( P_{12} = H_1 \)
\( P_{22} = H_2 \)

SEND THE RANGE AND VELOCITY ESTIMATES TO THE OUTPUT PORT
READ THE NEW SAMPLE IF DATA IS READY
IF NOT READY, PROPAGATE THE RANGE ESTIMATE USING VELOCITY EST.
TAU IS THE TIME IT TAKES TO PROPAGATE (MULT. BY 2 SINCE VEL IS DIVIDED BY 2)

\( IV0 = IV \)
\( \text{PLS1} = (\text{XPLS1} + \text{XPLS2} * .002 - \text{OFFSET}) / \text{SCALE} \)
\( \text{IXPLS1} = \text{PLS1} \)
\( \text{PLS2} = \text{XPLS2} / (2.0 * \text{SCALE}) \)
\( \text{IXPLS2} = \text{PLS2} \)
\( \text{CALL INOUT(KOUT,PLS1,PLS2,IXPLS1,IXPLS2,IV,IDUMMY,\text{TWOTAU})} \)

GO BACK FOR THE NEXT DATA
GO TO 50

STOP

END
TITLE FORTRAN SUBROUTINE

; DATA TO BE SENT TO THE PORT
; LOCATION OF RANGE (SHORT REAL)
; LOCATION OF VELOCITY (SHORT REAL)
; RANGE (32 BIT INTEGER)
; VELOCITY (32 BIT INTEGER)
; INPUT DATA
; VELOCITY IN SIGN/MAG. 16 BIT INT. FORM
; 2 * PROPAGATION LOOP TIME

; CONFIGURE INPUT AND OUTPUT PORTS
MOV DX, 230H
MOV AL, 0FH
OUT DX, AL
MOV DX, 228H
SUB AX, AX
OUT DX, AL

; STORE VELOCITY IN SIGN MAGNITUDE FORM FOR LATER USE
LDS SI, ES: VEL[BX]
MOV AL, [SI] + 3
CMP AL, 00H
JNS CONFIG
MOV AX, [SI]
NOT AX
ADD AX, 1
OR AX, 8000H
LDS SI, ES: Y[BX]
MOV [SI], AX
JMP MANIP

CONFIG: MOV AL, [SI]
LDS SI, ES: Y[BX]
MOV [SI], AL
LDS SI, ES: VEL[BX]
MOV AL, [SI] + 1
AND AL, 7FH
LDS SI, ES: Y[BX]
MOV [SI] + 1, AL
; CONFIGURE THE 32 BIT OUTPUT (DOUT)
MANIP:  LDS SI, ES: RANGE[BX] ; LOAD ADDRESS OF RANGE
        MOV AX, [SI] ; GET THE FIRST TWO BYTES OF RANGE
        LDS SI, ES: DOUT[BX] ; LOAD ADDRESS OF DATA TO BE SENT OUT
        MOV [SI], AX ; STORE THE FIRST TWO BYTES OF RANGE IN DOUT
        LDS SI, ES: RANGE[BX] ; RANGE ADDRESS AGAIN
        MOV AL, [SI] + 2 ; GET THE THIRD BYTE OF RANGE
        MOV CL, 2 ; ROTATION COUNT
        ROR AL, CL
        LDS SI, ES: Y[BX] ; LOAD THE ADDRESS OF VELOCITY
        MOV AH, [SI] ; GET THE FIRST BYTE OF VELOCITY
        ROL AX, CL ; PUT THE CONFIGURED BYTE IN AH
        LDS SI, ES: DOUT[BX] ; ADDRESS OF DOUT
        MOV [SI] + 2, AH ; STORE THE THIRD BYTE TO GO OUT
        LDS SI, ES: Y[BX] ; ADDRESS OF VELOCITY
        MOV AX, [SI] ; GET VELOCITY IN AX
        MOV CL, 3 ; CONFIGURE FOURTH BYTE
        SAR AX, CL
        MOV CL, 4
        SAR AH, CL
        MOV CL, 3
        SAR AX, CL
        AND AL, 3FH
        LDS SI, ES: DOUT[BX] ; STORE THE FOURTH BYTE TO GO OUT
        MOV [SI] + 3, AL
; SEND THE DATA TO THE OUTPUT PORTS
        MOV DX, 231H ; POINT TO PORT 0
        MOV AL, [SI] ; GET THE FIRST BYTE IN AL
        OUT DX, AL ; OUT FIRST BYTE TO PORT 0
        INC DX ; POINT TO PORT 1
        MOV AL, [SI] + 1 ; GET THE SECOND BYTE IN AL
        OUT DX, AL ; OUT SECOND BYTE TO PORT 1
        INC DX
        MOV AL, [SI] + 2 ; THIRD BYTE IN AL
        OUT DX, AL ; OUT THIRD BYTE TO PORT 2
        INC DX
        OUT DX, AL ; LAST BYTE TO PORT 3
        MOV AL, [SI] + 3
; READ THE DATA READY PULSE
        MOV DX, 22CH ; POINT TO INPUT PORT 3
        IN AL, DX
        RCR AL, 1 ; GET THE READY PULSE IN CARRY
        JC READY ; IF DATA READY, GO READ IT
; IF DATA NOT READY, PROPAGATE THE RANGE
        FINIT ; INITIALIZE THE CO-PROCESSOR
        LDS SI, ES: RVEL[BX] ; VELOCITY TO TOP OF STACK
        FLD DWORD PTR [SI]
        LDS SI, ES: TOTAU[BX]
        FMUL DWORD PTR [SI] ; MULTIPLY BY TIME
        LDS SI, ES: RRANGE[BX]
        FADD DWORD PTR [SI] ; ADD TO THE RANGE
        FST DWORD PTR [SI] ; STORE THE REAL VALUE
        LDS SI, ES: RANGE[BX]
        FISTP DWORD PTR [SI] ; POP AND STORE THE 16 BIT INTEGER VERSION
        FWAIT
        JMP MANIP ; CHECK DATA READY PULSE AGAIN
; If data ready, read it from the input ports

READY: LDS SI, ES: KV[BX] ; Location of the input
       MOV DX, 22BH ; Point to port 2
       IN AL, DX
       AND AL, 3FH
       CMP AL, 20H ; Is there a stop request?
       JNS TERM ; If stop requested, set up for termination
       AND AL, 1FH
       CMP AL, 10H ; Is the data valid?
       JS PRED ; If not, go to predictor set-up
       AND AL, 0FH ; If valid, mask the valid/invalid bit
       MOV [SI] + 2, AL
       DEC DX
       IN AL, DX
       MOV [SI] + 1, AL
       DEC DX
       IN AL, DX
       MOV [SI], AL
       MOV AL, 00H
       MOV [SI] + 3, AL
       JMP FIN

; Input data configuration for invalid samples

PRED: MOV AL, 0FFH
       MOV [SI] + 3, AL
       JMP FIN

; Input data configuration for stop requests

TERM: SUB AX, AX
       MOV [SI], AX
       MOV AX, 0020H
       MOV [SI] + 2, AX

FIN: POP SP ; Restore the registers
       POP DX
       POP CX
       POP BX
       POP AX
       POP SP

INOUT ENDP
MYPRO ENDS
END
inports2.asm

PAGE 132
TITLE FORTRAN SUBROUTINE

FRAME STRUC
NV DD ?
FRAME ENDS
STACK SEGMENT WORD STACK 'STACK'
DB 64 DUP('MYSTACK')
STACK ENDS
MYPROG SEGMENT 'CODE'
ASSUME CS:MYPROG, SS:STACK
INDAT PROC FAR
PUBLIC INDAT
PUSH AX ;SAVE THE REGISTERS
PUSH BX
PUSH CX
PUSH DX
PUSH SP
MOV DX, 228H ;POINT TO CONTROL REG
SUB AX, AX ;0 TO AX
OUT DX, AL ;CONFIGURE PORTS FOR INPUT
ADD DX, 4H ;POINT TO PORT 3
REDY: IN AL, DX ;READ THE DATA READY BIT
RCR AL, 1 ;GET THE READY BIT IN CF FOR CHECKING
JNC REDY ;IF DATA NOT READY, CHECK AGAIN
LDS SI, ES:NV[BX] ;LOCATION OF THE RANGE
DEC DX ;POINT TO PORT 2
IN AL, DX ;GET THE MSB OF THE DATA, BITS 16-23
AND AL, 1FH ;MASK THE UNUSED BITS (23,22,21)
CMP AL, 10H ;IS THE DATA VALID? (AL-10)
JS PRD ;IF NOT VALID, GO TO PREDICTOR SET UP
AND AL, 0FH ;IF VALID, MASK THE VALID/INVALID BIT
MOV [SI]+2, AL ;STORE THE THIRD BYTE
DEC DX ;POINT TO PORT 1
IN AL, DX ;GET THE SECOND BYTE
MOV [SI]+1, AL ;STORE THE SECOND BYTE
DEC DX ;POINT TO PORT 0
IN AL, DX ;GET THE FIRST BYTE
MOV [SI], AL ;STORE THE FIRST BYTE
MOV AL, 00H
MOV [SI]+3, AL ;COMPLETE TO 32 BIT +VE INTEGER
JMP FINAL ;GO TO RETURN
PRD: MOV AL, 0FFH
MOV [SI]+3, AL ;MAKE KV A NEGATIVE INTEGER
FINAL: POP SP ;RESTORE THE REGISTERS
POP DX
POP CX
POP BX
POP AX
RET
INDAT ENDP
MYPROG ENDS
END