IMPLEMENTATION OF A SELF-TUNING REGULATOR ON A MICROCOMPUTER

GEORGE KIZIROGLU

RE-182 June 1976
Department of Automatic Control
Lund Institute of Technology

IMPLEMENTATION OF A SELF-TUNING REGULATOR ON A MICROCOMPUTER

BY

GEORGE KIZIROGLU

Master Thesis in Automatic Control.

Department of Automatic Control,
Lund Institute of Technology,
Lund, Sweden

Supervisors: L Andersson
B Wittenmark

ABSTRACT

The main purpose with this master thesis is to show how a microcomputer, Intel 8080, is used to implement a special kind of adaptive regulators, called self-tuning regulators. The report presents some theory, software solution and some experiments.
The programmemory requirement is about 1.4K Bytes, and the datamemory needed is 0.5K Bytes.
The self-tuning regulator have been implemented on different processes simulated by a analog computer.

SAMMANFATTNING

Ändamålet med detta examensarbete, är att visa hur en mikrodator, Intel 8080, används för att implementera vissa typer av adaptiva regulatorer, s.k. självinställande regulatorer.
I rapporten framförs en del teori,mjukvarulösningen, och en del gjorda experiment.
Programminnesuttrymmet är ungefär 1.4K Bytes och dataminnesbehovet är 0.5K Bytes.
Självinställande regulatorn har implementerats på olika processer som har simulerats på en analogimaskin.

CONTENTS

1. INTRODUCTION

Τά καλά τοῖς κόπoις κτῶντε
Greek Proverb

The department of Automatic Control at Lund Institute of Technology have for many years done researches on Adaptive controllers.Extensive research have been done on the theory and application of self-tuning regulators {4 }.
Self-tuning regulators have been applied on different kind of industrial processes, paper-machines, ore crushers, autopilot for tankers using minicomputers.
The microcomputers have now been developed so much,that it would be interesting to implement a self-tuning regulator on a microcomputer.
In Chapter 2 is given a theoretical background for the further discussions in this report.Chapter 3 describes the assembly language of Intel 8080.A description of the program- and datastructure is given in Chapter 4.
In this master thesis also some hardware was developed. A cooperation was done with P-O Sjöberg for the construction of the microcomputer system, in which the self-tuning regulator was implemented. A short description is given in chapter 5. Chapter 6 describes,the way the user have to do for test the regulator on different processes.
Chapter 7 describes some results of the implementation of the regulator on some processes,simulated on the analog computer.
Appendix A containes the Instruction Set for Intel 8080, and in Appendix B the whole program is given,which takes about 1.4 K Bytes and is programmed in PROM-memories.

2. SELF-TUNING REGULATORS

I think,therefore I am.
RENE DESCARTES

2.1 The Basic Algorithm

The self-tuning regulators are controllers which are used on processes with constant but unknown parameters.
The basic algorithm described in (4) is derived to control minimum phase single-input single-output systems.The algorithm can be divided into two parts.Firstly,the parameters in a prediction model of the process are estimated using the method of least squares.Secondly,the minimum variance regulator is computed,as if the values of the estimated parameters were the true ones.Thus it is not taken into consideration that the values of the estimated parameters are uncertain.

Consider the system

$$A(q^{-1})y(t) = B(q^{-1})u(t-k-1) + C(q^{-1})e(t) \qquad (2.1)$$

where e(t) is a sequence of independent $N(0,\sigma)$ random variables and

$$A(q^{-1}) = 1+a_1q^{-1}+a_2q^{-2}+\ldots\ldots+a_nq^{-n}$$
$$B(q^{-1}) = b_1+b_2q^{-2}+\ldots\ldots\ldots+b_nq^{-(n-1)}$$
$$C(q^{-1}) = 1+c_1q^{-1}+c_2q^{-2}+\ldots\ldots+c_nq^{-n}$$

The purpose of the control is to minimize the loss function

$$E\Sigma(y_r(t)-y(t))^2 \qquad (2.2)$$

where $y_r(t)$ is a known time-varying reference value.
Introduce the reference input, $u_r(t)$, defined as

$$u_r(t) = \frac{A(q^{-1})}{B(q^{-1})}\, y_r(t+k)$$

Equation (2.1) can now be written as

$$A(q^{-1})(y(t)-y_r(t)) = B(q^{-1})(u(t-k-1)-u_r(t-k-1))+C(q^{-1})e(t)$$

The loss function (2.2) is minimized by the control strategy

$$u(t) - u_r(t) = -\frac{G(q^{-1})}{B(q^{-1})F(q^{-1})}\,(y(t) - y_r(t)) \qquad (2.3)$$

where F and G are polynamials given by the identity

$$C(q^{-1}) = A(q^{-1})F(q^{-1}) + q^{-k-1}G(q^{-1})$$

The control law (2.3) can now be written as

$$u(t) = -\frac{G(q^{-1})}{B(q^{-1})F(q^{-1})}\,y(t) + \frac{C(q^{-1})}{B(q^{-1})F(q^{-1})}\,y_r(t+k+1)$$

To compute the optimal control law it is thus necessary to know the reference value in at least k+1 steps ahead.

Now introduce the prediction model of the process

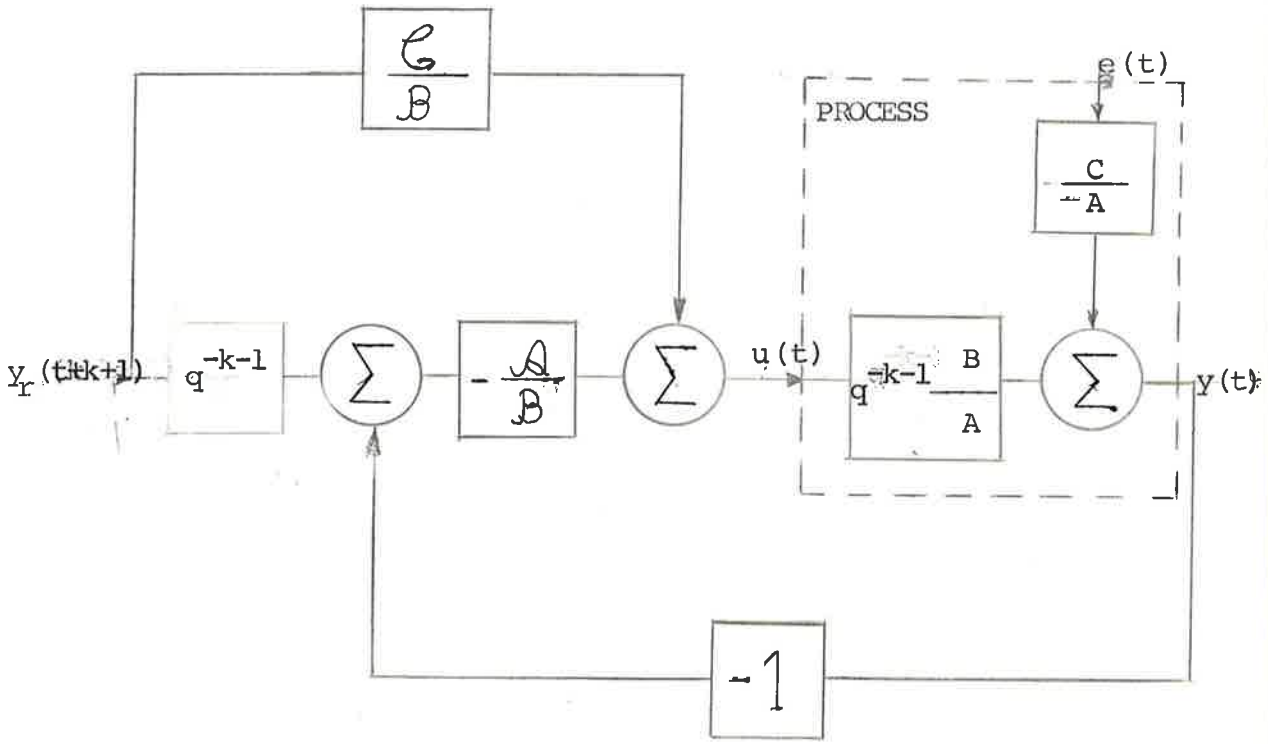


Figure 2.1

$$y(t+k+1) - y_r(t+k+1) + \mathcal{A}(y(t) - y_r(t)) = \beta_0 \mathcal{B}u(t) + \mathcal{C}y_r(t+k+1) + \varepsilon(t+k+1) \quad (2.4)$$

where

$$\mathcal{A} = \alpha_1 + \alpha_2 q^{-1} + \ldots\ldots\ldots + \alpha_{na} q^{-(na-1)}$$
$$\mathcal{B} = 1 + \beta_1 q^{-1} + \ldots\ldots\ldots + \beta_{nb} q^{-nb}$$
$$\mathcal{C} = \gamma_1 + \gamma_2 q^{-1} + \ldots\ldots\ldots + \gamma_{nc} q^{-(nc-1)}$$

The minimum variance control law, for considering this model will be

$$u(t) = (\mathcal{A}(y(t)-y_r(t))/\beta_0 - \mathcal{C}y_r(t+k+1)/\beta_0)/\mathcal{B} \quad (2.5)$$

The structure of the process and the controller is shown in Figure 2.1. Notice that the block $\frac{\mathcal{C}}{\mathcal{B}}$ represents a feedforward regulator and the block $-\frac{\mathcal{A}}{\mathcal{B}}$ a feedback regulator.

Introduce

$$VS(t+k+1) = y_r(t+k+1)/\beta_0$$

$$YS(t) = (y(t)-y_r(t))/\beta_0 = y(t)/\beta_0 - VS(t)$$

We can rewrite (2.4) and (2.5) as follows:

$$YS(t) + \mathcal{A}YS(t-k-1) = \mathcal{B}u(t-k-1) + \mathcal{C}VS(t) + \varepsilon(t+k+1)$$

$$u(t) = (\mathcal{A}YS(t) - \mathcal{C}VS(t+k+1))/\mathcal{B}$$

2.2 Selection of Parameters in the Algorithm

The basic self-tuning regulator is specified by the following parameters:

a) Sampling time, T
b) Scale factor in the regulator, β_o
c) Exponential forgetting factor, λ
d) Number of pure time-delays in the model, k
e) Allowed maximum value of the control signal,ULIM
f) Number of regulator parameters, NA,NB,NC
g) Initial values for the least squares estimator

A thorough discussion is given in { 4 }.

a) Some aspects that must be considered when determining the sample intervall are follows:

- The dynamics of the process
- The dead time in the process
- The complexity of the regulator
- The computation times

b) If the sign of the scale factor is assumed known its value is not crucial for the convergence if the control signal is limited.The transient behaviour seems to be best if β_o and b_1 are of the same magnitude.
The number of times the control signal hits the limit can be used to determine if a good value of β_o has been used.If β_o is too small,then the control signal use to reach the limit more often,than if too large a β_o is used. Too large a β_o will on the other hand give very small input signals in the beginning.Further the α_i parameters use to be large if β_o is too large.

c) The forgetting factor is used to determine how many old values should be remembered.About 200 old values are remembered for λ=0.99, while only 40 are remembered for λ=0.95. A rule of thumb to determine how many values that are remembered, n, is given by $n = 2/(1-\lambda)$.
The parameter λ is best chosen by inspecting the variations in the parameter estimates.Too much fluctuations implies either that too many parameters are used in the regulator

or that a too small a value of λ is used.
If the parameter estimates are almost constant it is advisable to make a small decrease in the value of λ occasionally.This should be done in order to make sure that the estimates really have converged and not only are slowly changing.

d) The number of pure time-delays in the model, k, is the most crucial parameter to choose.Too large a value of, k,can give an output variance which is larger than necessary.Sometimes it is also difficult to get the estimates to converge.Too small k can make very difficult to get good control and even make it impossible to stabilize the system.

e) A limitation of the control signal has in many cases good influence on the transient behaviour of the self-tuning regulators.This is the case if the initial estimates of the parameters are poor.The regulator might then send large signals into the process which can cause large errors initially.The control signal can thus be limited to get a smooth start-up.The limit can be small in the beginning and later increased when better parameter estimates are obtained.Another way is to have a fixed limit which should not be reached under normal control of the process.If the maximum value is reached often it is not possible for the regulator to converge to the minimum variance regulator.

f) If NA or NB or NC is too large then it can be shown that the resulting controller is equivalent to the minimum variance regulator,but there might be a common factor in the regulator.No indication is given of how much the output variance is increased if the number of regulator parameters is decreased.

g) Initial values of the parameters α_i and β_i and the covariance matrix, P(t), must be given when starting up the least squares estimator.
If the parameters are unknown a standard choise can be

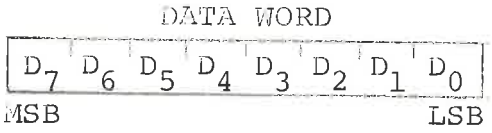
to put them equal to zero.In this case the covariance matrix shall be given a rather large value,say (10-100)×I, where I is a unit matrix. A too small value on P(0) can, however, give rise to slow rate of convergence.In order to avoid this the exponential forgetting factor can be given a value somewhat less than one.

3. SHORT DESCRIPTION OF INTEL 8080 ASSEMBLER

Se non é vero, é bon trovato

Instruction and Data Formats:

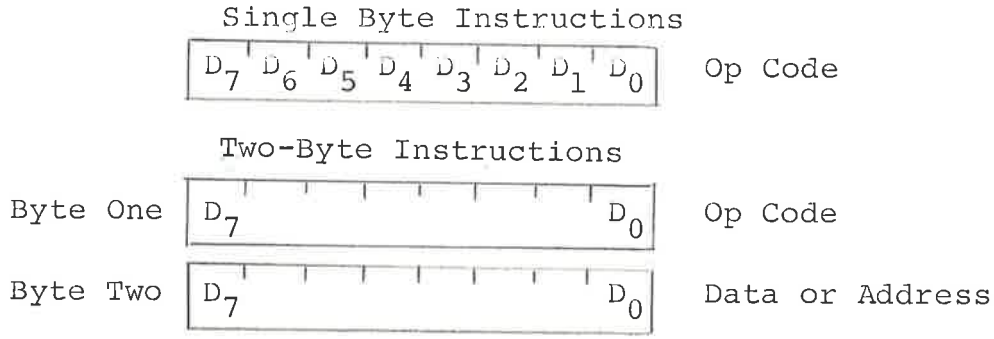
Memory for the Intel 8080 is organized into 8-bit quantities,called Bytes.Each Byte has a unique 16-bit binary address corresponding to its sequential position in memory. Data in the 8080 is stored in the form of 8-bit binary integers:



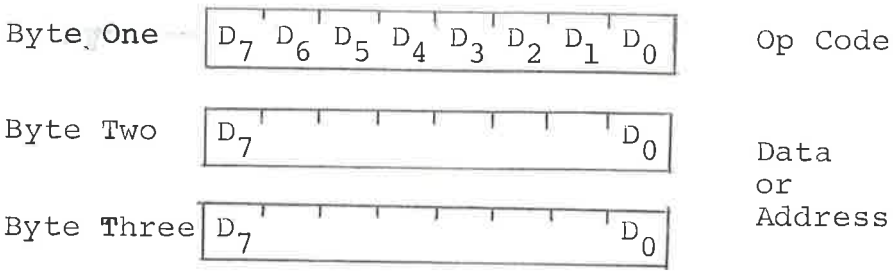
MSB= Most significant Bit

LSB= Least significant Bit

The 8080 program instructions may be one,two or three bytes in length.Multiple byte instructions must be stored in succesive memory locations; the address of the first byte is always used as the address of the instructions. The exact instruction format will depend on the particular operation to be executed.



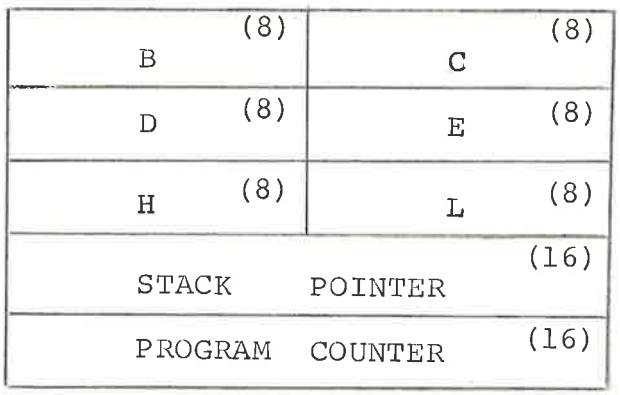
Three-Byte Instructions

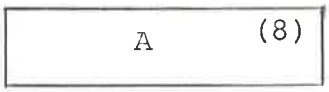


Registers:

The program-addressable registers in 8080 are:
Six general purpose registers addressed singly or in pairs
B,C, D,E, H,L
The 8-bit accumalator,also known as register, A.
The 16-bit stack point register, SP.
The 16-bit program counter register, PC.

One of the registers that are employed internally in the 8080 chip, but cannot be addressed from a computer program include
a 5-bit flag register (Z, S, P, CY, AC)
They are Zero,Sign,Parity,Carry and Auxiliary Carry.





A flag is "set" by forcing the bit to 1 ; "reset" by forcing the bit to 0. When an instruction affects a flag, it affects it in the following manner:

Zero: If the result of an instruction has the value 0, this flag is set; otherwise it is reset.

Sign: If the most significant bit of the result of the operation has the value 1,this flag is set; otherwise it is reset.

Parity:If the result has even parity, this flag is set; otherwise it is reset.

Carry:If the instruction resulted in a carry (from addition) or a borrow (from subtraction or comparison) out of the highorder bit this flag is set; otherwise it is reset.

Auxiliary
Carry:Used in connection with computing decimal numbers (is seldom used)

Instruction set:

The 8080 instruction set includes five different types of instructions:
Data transfer group: The 8080 has four different modes for addressing data stored in memory or in registers:

Direct- Bytes 2 and 3 of the instruction contain the exact memory address of the data item, e.g.
LDA addr (Load Accumulator direct)

Register- The instruction specifies the register or register-pair in which the data is located,e.g.
MOV A,B (Move B to A)

Register
indirect- The register pair H,L contains the memory address where the data is located, e.g.
MOV A,M (The content of the memory location whose address is in registers H and L is moved to A)

Condition flags are not affected by any instruction in this group.

Immediate- The instruction contains the data itself.

Arithmetic group: All instructions in this group affect at least one of the five flags according to the standard rules, e.g.

ADD B ((A)←(A)+(B))

Logical Group: This group of instructions performs logical (Boolean) operations,AND,OR,EXCLUSIVE-OR,COMPARE,ROTATE or COMPLEMENT on data in registers or in memory.Besides the instructions CMC and STC , A register is involved in all instructions.At least one of the five flags are affected by all instructions, e.g.

ANA B ((A)←(A) AND (B))

Branch Group: The program counter register is involved in all instructions.Consider flags are not affected by any instruction.

Stack, I/O, and Machine Control Group: Except the instruction POP PSW, no other instruction affects the condition flags.

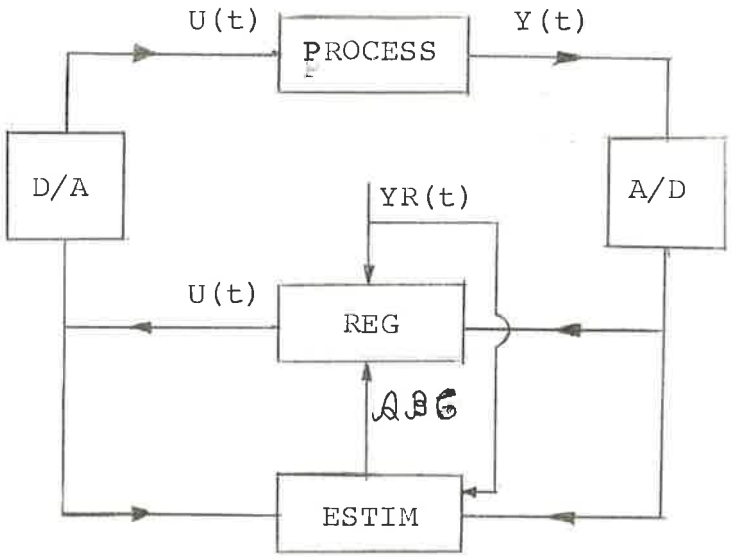
The execution time of all the instructions is varying from 4 (e.g. ADD B) up to 18 states (XTHL).For a detailed description of the instruction set of the Intel 8080, see Appendix.

4. SOFTWARE SOLUTION

Slow and steady wins the race.
DAVID LLOYD,FABLES

4.1 System Solution

The structure of the system is given in the following figure.



At each sample the microcomputer reads following values from the A/D - converter:

The output signal of the process, Y

The reference signal of Y, YR

and outputs the control signal to the process via an D/A - converter.Normally the self-tuning regulator,STURE, is divided into two parts.

In the first part the parameters of the process are estimated by the estimator subroutine,ESTIM.The algorithm is based on the model

$$Y(t+k+1)-YR(t+k+1)+AE(1)\times(Y(t)-YR(t))+....+AE(NA)\times(Y(t-NA+1)-YR(t-NA+1))= B0\times(U(t)+BE(1)\times U(t-1)+....+BE(NB)\times U(t-NB)+ +CE(1)\times YR(t+k+1)+.....+CE(NC)\times YR(t+k-NC+2)+E(t+k+1)$$

In the second part the regulator subroutine,REG,computes the control signal to be applied at time,t, to the process from

U(t)=(AE(1)×(Y(t)-YR(t))+..+AE(NA)×(Y(t-NA+1)-YR(t-NA+1)))/B0
-BE(1)×U(t-1)-....-BE(NB)×U(t-NB)-(CE(1)×YR(t+k+1)+...
...+CE(NC)×YR(t-NC+k+2))/B0

AE(I),BE(I),CE(I) are the least squares estimates from the subroutine,ESTIM.
Assuming that the parameters changes slowly between the samples,this kind of scheduling is uneffective,because it takes long time until U(t) can be computed.Instead,we can divide the subroutine REG into REG1 and REG2. In REG1 is U(t) computed by the formula:

U(t)=AE(1)×(Y(t)-YR(t))/B0 - CE(1)×YR(t+k+1)/B0 + Fscal(t-1)

where AE(1) and CE(1) are the estimated parameters from the sampleinterval,t-1,and Y(t),YR(t),YR(t+k+1) are the input values at time t.
At the rest of the sampleinterval ,t, the new parameters are estimated and on the base of these Fscal(t) is computed by

Fscal(t)=AE(2)×(Y(t)-YR(t))+..+AE(NA)×(Y(t-NA+2) -
YR(t-NA+2))/B0 - BE(1)×U(t) - ...-BE(NB)×U(t-NB) -
(CE(2)×YR(t+k+1)+....+CE(NC)×YR(t-NC+k+3)/B0

4.2 Programstructure

The mainprogram, STURE, is divided into the following subroutines:

The high-level subroutines:	INIT,ADIN,DAOUT,FMOVE, FLIM,SCAPR,REG1,ESTIM, REG2
The low-level subroutines:	FMUL,FADD,FDIV,LOAD, STORE,BINF,FBIN,FMAGN NEGDE,OUFLW,OFLW,UFLW,UFLWE

The low-level subroutines are explained in 4.2.2. Using the mainprogram and the high-level subroutines the whole program can be written in Algol.

4.2.1 STURE in Algol

```
Begin  Integer  k;
       Real  YR,Y,B0;
       Array VS(1:14),YS(1:14);
       Procedure  ADIN,REG1,ESTIM,REG2;
       Comment
       The vectors YS,U,VS,T and the variables NA,NB,
       NC,k appears in the procedures REG1,ESTIM, and
       REG2 and they have following meaning:
       YS- vector of scaled process outputs of dimension
       NA+k+2. YS is organized as follows.
           YS(1)=(Y(t)-YR(t))/B0 = Y(t)/B0 - VS(t)
           YS(2)=(Y(t-1)-YR(t-1))/B0 = Y(t-1)/B0 - VS(t-1)
             ...
             ...
       U- vector of process inputs of dimension NB+k+2.
       U is organized as follows.
           U(1)=U(t-1)
           U(2)=U(t-2)
             ..
             ..
       VS-scaled vector of reference signal of dimension
       NC+k+2. VS is organized as follows.
           VS(1)=YR(t+k+1)/B0
           VS(2)=YR(t+k)/B0
             ..
           VS(k+2)=YR(t)/B0
             ..
       T-vector of estimated parameters of dimension
       NA+NB+NC and organized as follows.
           T(1)=AE(1)
           T(2)=AE(2)
             ..
```

```
        T(NA)=AE(NA)
        T(NA+1)=BE(1)
          ..
        T(NA+NB)=BE(NB)
        T(NA+NB+1)=CE(1)
          ..
        T(NA+NB+NC)=CE(NC)
NA- number of A- parameters in (4.1)
NB- number of B- parameters in (4.1)
NC- number of C-parameters in (4.1)
k-  number of pure time delays in the process.
Dimension restrictions
MAX(NA+NB+NC)=9
MAX(NA+k+2)=14
MAX(NB+k+2)=14
MAX(NC+k+2)=14
End of comments;

INIT;
STURE:   YR:=ADIN(0);
         Y:= ADIN(1);
         VS(1):=YR/B0;
         YS(1):=Y/B0 - VS(k+2);
         REG1;
         ESTIM;
         REG2;
         GO TO STURE
END
```

```
Procedure  INIT;
Begin  comment
       This procedure initialize STURE;
       Real FSCAL, 1/B0, 1/RL, ULIM, A;
       Integer  NA, NB, NC, k, I, J;
       Read (1/B0); Read (1/RL); Read (ULIM);
       Read (NA); Read (NB); Read (NC);
       Read (k); Read (A); Fscal:= 0 ;
       For  I:=1  step  1  until  14  do
       YS(I):=U(I):=VS(I):=0 ;
       For  I:=1  step  1  until  9  do
       Begin   T(I):=S(I):=FI(I):=0 ;
               For  J:=1  step  1  until 9 do P(I,J):=0;
               P(I,I):=A;
       End
End;
```

```
Real Procedure  ADIN (I);
Integer I;
Begin  comment
       Entry:   I=channel number
       This procedure converts an input value from
       channel I to a normalised floating-point number
       with the exponent in Excess-64 form and the
       mantissa in two-complement representation;
       Real Procedure    BINF;
       Begin   ADIN:=BINF
       End
End;
```

```
Real Procedure  DAOUT (I,U);
Integer I;  Real U;
Begin  comment
       Entry:    I = channel number
                 U = the control signal
       This procedure converts the limited controlsignal
       from a normalised floating-point number to a
       fix number and outputs it in channel number I;
       Integer procedure FBIN (A);  Real A;
       DAOUT:= FBIN(U)
End;
```

```
Procedure  FMOVE(A,L);
Integer L;  Array A(1:L);
Begin  comment
       Entry:    A = the vector
                 L = the vectorlength
       This procedure moves the vector A one step
       upwards in the memory;
       Integer  I;
       For  I:= 1  step  1  until  L do A(L+2-I):=A(L+1-I);
       A(1):= 0;
End;
```

```
Real Procedure  FLIM(U,UL);  Real U,UL;
Begin  comment
       Entry:    U = signal to be limited
                 UL = limit of signal
       This procedure limits the magnitude of signal U
       to UL volts;
       If  ABS (U) ≤ UL
       then  FLIM:=U  else
       if U> UL
       then FLIM:=UL  else FLIM:= -UL
End;
```

```
Real Procedure  SCAPR(A,B,N);
Integer N;   Array  A,B(1:N);
Begin  comment
       Scalar product of the vectors A and B
       of length  N ;
       Integer  I;  Real  R;
       R:=0 ;
       If  N = 0  then  go to  out;
       For  I = 1  step 1  until  N do R:=R+A(I)×B(I);
       out:   SCAPR := R ;
End;
```

```
Procedure   REG1;
Begin  comment
       This procedure outputs the control signal
       to the process. Fscal is computed from the
       sampleintervall t-1.
       U-vector is moved one step upwards in the memory;
       Array YS,U,VS(1:14);
       Integer  NA,NB,NC,k,NS;
       Real Fscal,U,A,U1;
       Real Procedure FLIM,DAOUT,BINF;
       NS:= NB+k+1;
       FMOVE(U,NS);
       U1:= T(1)×YS(1)+Fscal ;
       If NC=/=0 then U1:=U1-T(NA+NB+1)×VS(1);
       U:= FLIM(U1,ULIM);
       A:= DAOUT(0,U);
       U(1):= BINF(A);
End;
```

```
Procedure  ESTIM;
Begin  comment
       Performs the tuning in the self-tuning regulator.
       The tuning is done with a repeated least squares
       estimation of the model parameters.
       YS-U- and VS- vectors are not changed in ESTIM.
       P- covariance matrix of the parameter estimates
       of order (NA+NB+NC)×(NA+NB+NC)
       RL- the base of the exponential forgetting function;

       Array YS,U,VS(1:14), T,FI,S,PI,RK(1:9),P(1:9,1:9);
       Integer NA,NB,NC,k,I,J,N1,NS;
       Real  RL,Denom,Res;

       N1:= NA+NB+NC ;
       FI(1):= -YS(k+2) ;
       If NB=/=0 then FI(NA+1):= U(k+2) ;
       If NC=/=0 then FI(NA+NB+1):= VS(k+2) ;
       For I:=1 step 1 until N1 do
       Begin   for J:=1 step 1 until N1 do
               PI(J):= P(I,J) ;
               S(I):= SCAPR(PI,FI,N1)
       End;
       Denom:= SCAPR(FI,S,N1) ;
       Denom:= 1.0 + Denom ;
       Res:= SCAPR(FI,T,N1) ;
       Res:= YS(1)-U(k+1)-Res ;
       NS:= N1-1 ;
       FMOVE(FI,NS) ;
       For I:= 1 step 1 until N1 do
       Begin    RK(I):= S(I)/Denom ;
                T(I):= T(I)+RK(I)×Res ;
                For J:=1 step 1 until I do
                Begin  P(I,J):=(P(I,J)-RK(I)×S(J))/RL ;
                       P(J,I):= P(I,J) ;
                End;
       End;
End;
```

```
Procedure  REG2
Begin  comment
       This procedure moves YS- and VS-vectors one
       step upwards in the memory and computes Fscal
       in the sampleintervall t.

       Array YS,U,VS(1:14) ,  T,A(1:9);
       Integer  NA,NB,NC,NS,I,k;
       Real  Fscal,F1,F2,F3 ;

       NS:= NA+k+1 ;
       FMOVE(YS,NS) ;
       NS:= NC+k+1 ;
       FMOVE(VS,NS) ;
       F1:= SCAPR(YS,T,NA) ;
       For I:=1 step 1 until NB do A(I):=T(NA+I) ;
       F2:= SCAPR(U,A,NB) ;
       For I:=1 step 1 until NC do A(I):=T(NA+NB+I) ;
       F3:= SCAPR(VS,A,NC);
       Fscal:= F1-F3-F2 ;
End;
```

4.2.2 The low-level subroutines

The floating-point arithmetic routines FMUL,FADD,FDIV use a floating-point number representation.This representation uses two 8-bit words holding the magnitude of the number in 2's-complement form,normalised within the range $\pm(0.5$ to $1.0-2^{-15})$,together with one 8-bit word holding the 7-bit exponent in Excess-64 form. Excess-64 form is chosen,since when used in conjuction with the 8 th bit (MSB) of the exponent word,it is easy to detect and distinguish between overflow and underflow of the exponent.The exponent range is -64 to +63 decimal.

The range of numbers represented by these three words is 0.5×2^{-64} to $(1-2^{-15})\times2^{63}$,which is 2.7105×10^{-19} to 0.9223×10^{19} and accuracy is 1 part in 2^{15} (approx. 0.003%) giving 4.5 decimal digits.

Overflow results are set to 0.9223×10^{19} with the appropriate sign; underflows are set to zero,which is represented here by three words all containing zero,which is 0×2^{-64}, in order to make zero less significant than the smallest non-zero number.

The choise of register allocation convention is influenced by the special functions associated with certain registers. H and L are used together to address memory,so should not be used to hold the first operand or the result.H and L can be transferred directly to and from consecutive memory locations,storing the contents of L before H; the accumalator,A, can also be transferred directly to and from memory.D and E can be exchanged with H and L by a single instruction.

The arithemetic routines are pressed for register space and therefore they first calculate the result exponent and move it out of the way.An operand exponent can therefore be loaded initially into the accumalator,ready for the exponent calculation.To chain aruthemetic routines

it is best if the result of one operation is left in the correct registers to be the first operand of the next operation.Taking into account all these considerations, registers (B DE) are used for the first operand exponent, more significant half,and less significant half respectively, and registers (A HL) are similarly allocated for the second operand (if any).The remaining register,C, is used for storing a count of the negations performed,and a copy of the carry bit for determining over or underflow.All routines leave the result in registers (B DE).

Floating-point numbers in memory are allocated three consecutive words containing the exponent,less significant half,and more significant half respectively.This order allows use of the INTEL 8080's direct double load and store instructions (LHLD and SHLD). The stack is used for temporary storage of numbers.

All arithmetic routines operate on normalised numbers and expect the operands to be correctly normalised and in the correct registers.The routines give invalid results if used with unnormalised operands.

Basic routines used by the arithmetic routines

Negation routine,NEGDE

NEGDE negates the contents of registers D and E, trated as a 2's complement number.There are two entry points,one of which increments register C containing the negation count; the order does not.

In two's complement the largest negative number,NEGMAX, has no positive equivalent,and if used as an operand of NEGDE the result still equals NEGMAX. NEGMAX is not counted as a correctly normalised number,and can only occur as the result of an addition of two negative numbers. The floating-point addition routine detects the result of NEGMAX by finding that the number is still negative after NEGDE has been used to try and make it positive. NEGMAX is then shifted to its correctly normalised form.

Overflow and Underflow Handler, OUFLW

OUFLW consists of several routines and entry points, which detect which of exponent overflow or underflow has occured and set the result accordingly.

OUFLW: uses the carry produced during exponent calculation, and stored in the MSB of register C, to determine the condition prevailing. It then uses OFLW or UFLW to set the correct result.

OFLW: sets the result to the largest floating-point number, 0.9223×10^{19} and sets the sign according to the negation count in C.

UFLW: sets the result to zero.

UFLWE: sets the result exponent to zero. Used to correct the exponent if the result of a calculation is truly zero. It is not reported as an overflow.

Floating-Point arithmetic routines FMUL, FADD, FDIV

For a detailed description of these routines, see {2 }. A modification has been done of the routine FADD and it is listed in Appendix B.

Execution times are listed in Table 1 for typical and maximum time cases for the floating-point routines. If a fast return is made because of a zero operand or an over or under flow then the time taken is much less than the times given.

FADD	typical	0.25 mS	maximum	0.53 mS
FMUL		0.75 mS		0.861 mS
FDIV		0.75 mS		0.845 mS

Table 1 Execution times

LOAD,STORE,BINF,FMAGN: See Appendix

FBIN: For the assembler program with the comments, see Appendix. Here follows the flow diagram:

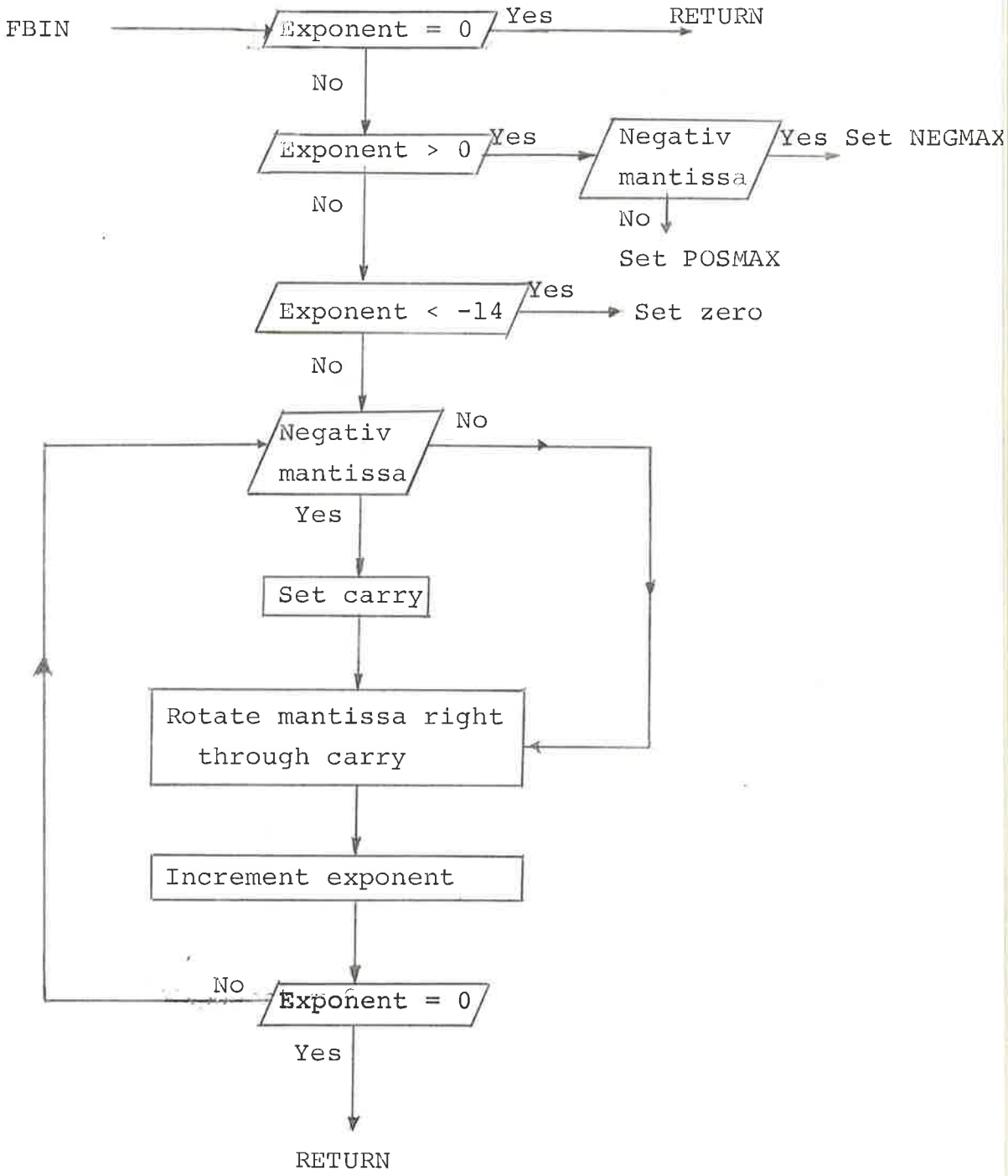


Fig 4.5 Flow diagram for the conversion from a floating-point number to a fix number.

The amount of memory needed for the subroutines used in the Self-Tuning regulator are as follows:

	Bytes
UTIL	60
FADD	93
FMUL	79
FDIV	74
FBIN	48
FLIM	23
SCAPR	74
ADIN	23
DAOUT	8
FMOVE	36
INIT	76
STURE	91
REG1	97
ESTIM	432
REG2	107

The routines above begin at address 000:071.
The following instructions are stored in the memory locations 000:000-000:070

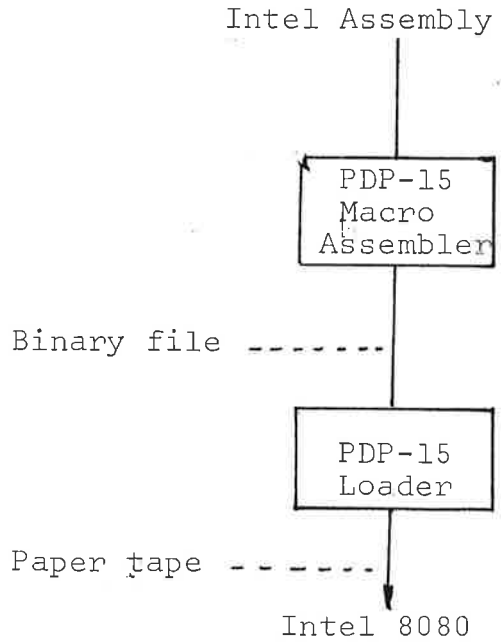
```
000:    LXI SP,10×400+400
003:    CALL INIT
006:    JMP STURE
011:    NOP
..       .
..       .
067:    NOP
070:    RET
```

Including this the total amount of memory requirement is 1378 Bytes, meaning that 6 PROM are needed.

The execution time of the program depends on the number of parameters.The experiments have resulted in following execution times.(Instruction cycle-time of 1 µs were used)

Number of parameters NA+NB+NC	Time (ms)
1	typical 20
2	50
3	100
5	210
7	400
9	550

For the compilation of the program for the microcomputer, the host computer PDP-15 was used.
Using the macrodefinition file ASS80,the assembler of PDP can generate code for Intel 8080.The program LOAD8 take then care of the of the work of the assembler.The programs ASS80 and LOAD8 are described in {1}.Load8 relocates and link together the subroutines and it produces absolute code on a paper tape,which,via a teletype, directly can be read into the RAM- and PROM memories.



4.3 Datastructure

All the data needed to STURE, are stored in RAM-memory beginning at bank 010 (A bank has 0400=256 bytes. 1 Byte = 1 word = 8 bits). See fig. 4.6 and fig. 4.7

Every data is represented as a floating-point number, except for the integers NA,NB,NC,k,N1,Slask3,Slask4,N and for the fractional number ULIM.

A floating-point number has the 8-bits exponent in Excess-64 form and the 16 bits mantissa in two's complement form.Each number has its exponent stored in the first byte,the low-order mantissa in the second byte and the high-order mantissa in the third byte.

The meaning of the vectors YS,U,VS,T are explained in the comments to STURE.The adress of the first element of the vectors U,VS,T,S,FI are fixed by the following dimension restrictions:

MAX(NA+NB+NC)=9
MAX(NA+k+2) = 14
MAX(NB+k+2) = 14
MAX(NC+k+2) = 14

Dynamical allocation has not been used for these vectors since we can save more programmemory than datamemory. It is also naturally to use a temporary storage for the computed values of S(I) since these are used several times in the subroutine ESTIM,and it is impossible to save all the values in the stack.

On the contrary the need of a seperate FI-vector stored in the memory can be discussed. The FI-vector is organized as follows:

FI(1)= -YS(k+2)
. .
FI(NA)= -YS(k+NA+1)
FI(NA+1)= U(k+2)
. .

FI(NA+NB) = U(k+NB+1)

..
FI(NA+NB+NC) = VS(k+NC+1)

The advantages of having a separate FI-vector are

a) The subroutine SCAPR needs to be called only once instead of three times for computing each element of the S-vector,also for computing the variables Denom and Res.

b) The execution time is shorter.

The drawbacks of having a separate FI-vector are

a) More data memory space.

b) One extra call of the routine FMOVE is needed.

As the memory requirement is only 033 Bytes of a FI-vector, we see that the advantages are superior the drawbacks.

1/B0, 1/RL -	See the comments to the Algol program.
NA,NB,NC,k -	These variables are explained in the comments to STURE:
N1 -	The subroutine INIT computes N1=NA+NB+NC.
P0 -	At the beginning of the simulations the subroutine INIT updates the P-matrix according to P(0) = P0×I ,where I is a unit matrix.
Denom,Res,RKI-	These variables are used in ESTIM.It is not necessary to store the RK-vector.In this case the stack is used.
Fscal -	Is used in REG1 and REG2.
ULIM -	The limit of control signal is stored with the less significant half on the first byte and the most significant half on the second byte.Considering that $\|U\| \leq$ ULIM , Ulim is varying from 00000 to 077777 (2-complement form) corresponding to 0 V respectively 9.984 V.

The rest of the bank 010, 030 Bytes,is used as temporary storage space by the stackregister,which gets the value 0400 at the beginning of the program. SP is then decremented by the first CALL-instruction.

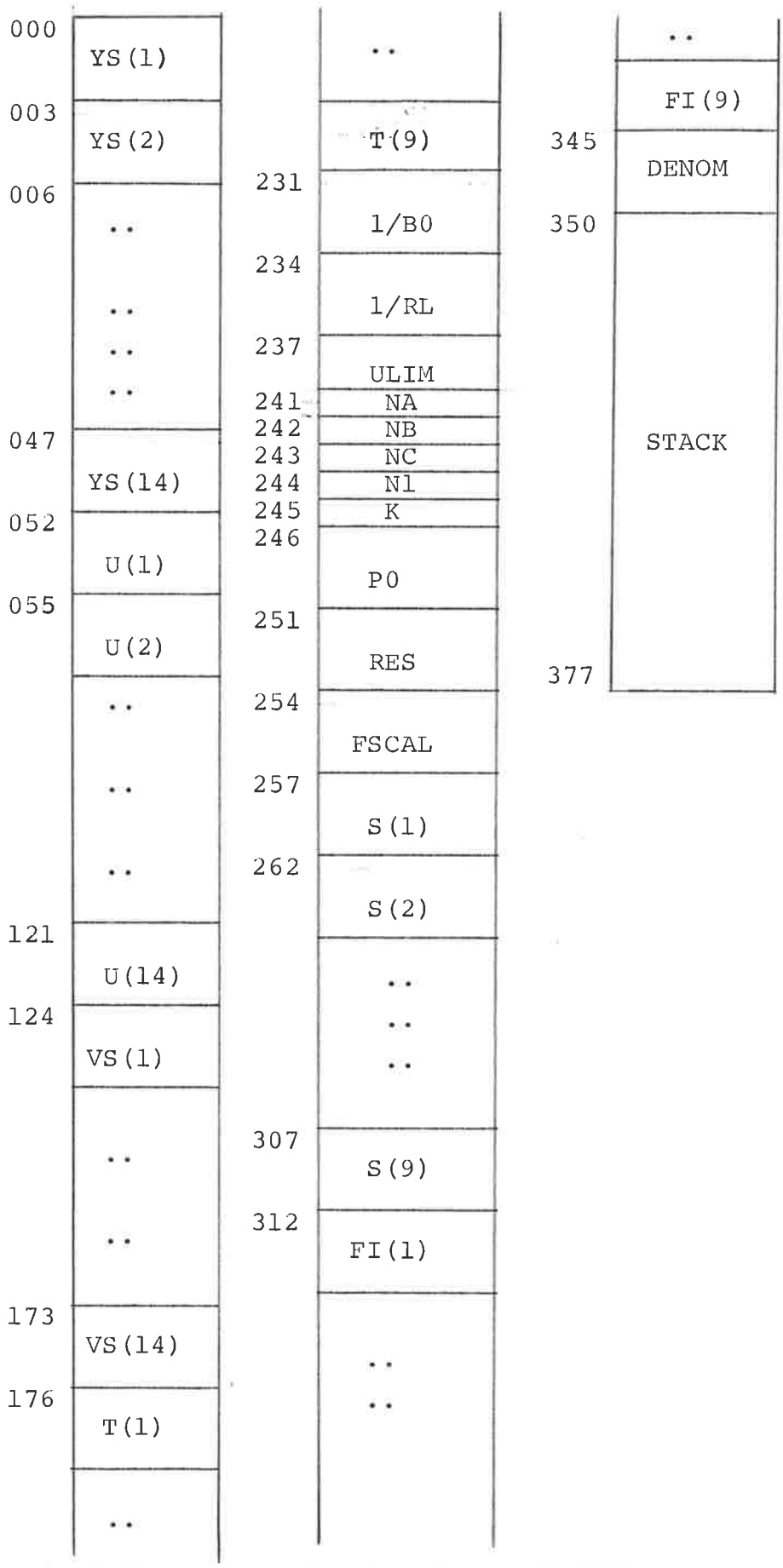


Fig. 4.6 Dataorganization for Bank 010

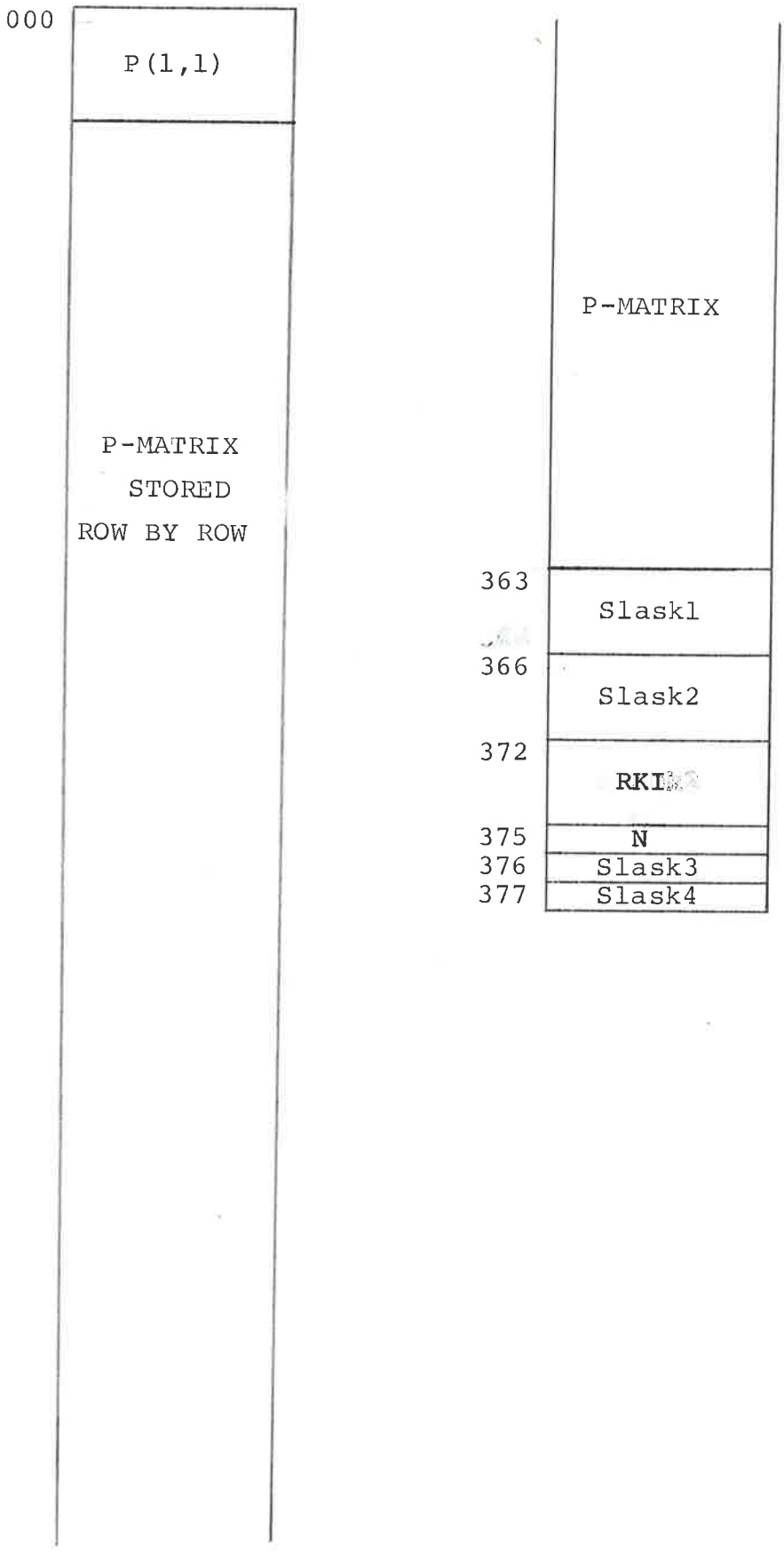


Fig. 4.7 Dataorganization for Bank 011

In Bank 011 the P-matrix and some temporary cells are stored.

P-matrix- The P-covariance matrix of the parameter estimates of order (NA+NB+NC)×(NA+NB+NC) is symmetric. Advantages with symmetric matrixes are:

a) Memory space can be saved. A n×n matrix needs only to store n×(n+1)/2 elements.
In our case,assuming that n=9,using the symmetric property only 0107 bytes has to be stored as maximum,otherwise 0363 bytes are stored as maximum.

b) The computation P(I,J)=P(J,I) is done automatically.

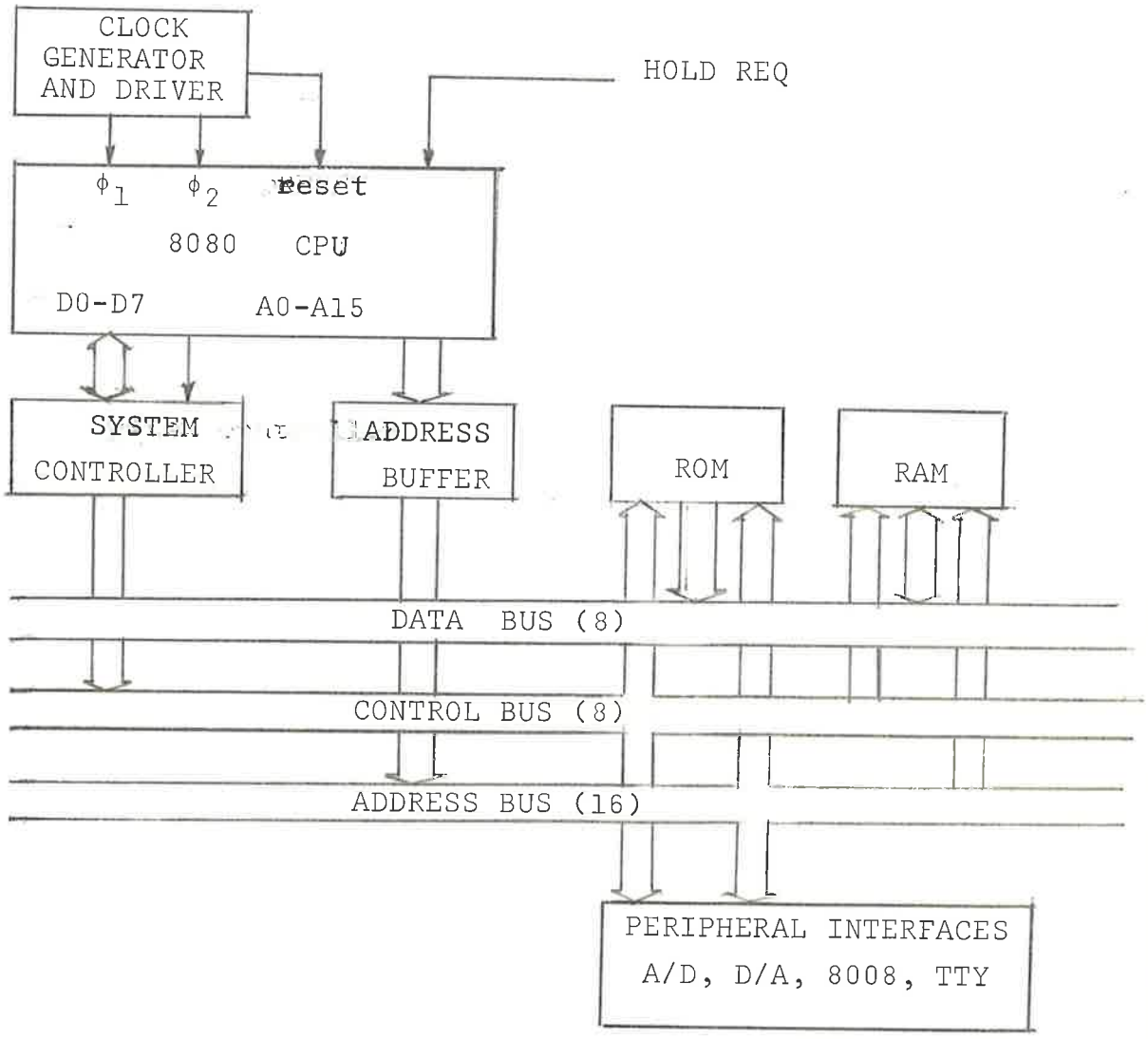
If the array is stored lexicographically as follows:
P(1,1),P(2,1),P(2,2),P(3,1),....,P(n,1),....P(n,n)
we get a big problem when computing the S-vector,for calling the subroutine SCAPR, (see ESTIM).We will need more space for programmemory for each test of the indixes of elements.Consider,the execution time and the programmemory space,the P-matrix is ordered by rows and would be stored with dynamically allocation in the order:
P(1,1),P(1,2),...P(1,n),P(2,1),..P(2,n),...P(n,1),..P(n,n).

Slask1- Three temporary cells,used in ESTIM.
Slask2- Three temporary cells,used in SCAPR.
N- One temporary cell,used in SCAPR.
Slask3- One temporary cell,used in ESTIM.
Slask4- One temporary cell,used in ESTIM.

5. A HARDWARE SUMMARY

> Houses are built to live in and not to look on.
>
> FRANCIS BACON

In this chapter follows a summary of the hardware. For a detailed description see { 3 }.
In the figure below is shown the microcomputer system, where STURE,is implemented.



For this implementation we need six modules; CPU,ROM,RAM, A/D, D/A, Interface to 8008.

CPU module - Contains the Clock generator,the System controller,the Central Processing Unit,and some extra logic for interrupt,display,etc.

Clock generator contains a crystal-controlled oscillator, a"divide by nine" counter, two high-level drivers and several auxilliary logic functions.
It generates two non-overlapping clock pulses (ϕ_1 and ϕ_2) who drive the 8080.
Each clock period marks a state; three to five states constitute a machine cycle,and one to five machine cycles comprise an instruction cycle.A full instruction cycle requires anywhere from 4 to 18 states for its completion, depending on the kind of instruction involved.With a crystal frequency of 9 MHz,we get an instruction cycle time of 1 μs.
The System controller generates all control signals required to directly interface MCS-80 family RAM,ROM, and I/O- components.
The Central Processing Unit consists of the following functional units:

- Registers array and address logic
- Arithmetic and logic unit (ALU)
- Instruction register and control section
- Bi-directional, 3-state data bus buffer.

When power is applied initially to the 8080,the processor begin operaing immediately.The contents of its program counter, stack pointer,and the other working registers are naturally subject to random factors and cannot be specified.For this reason, it will be necessary to begin the power-up sequence with RESET,where the program execution begins with memory location zero.
Systems which require the processor to wait for an explicit start-up signal will store a halt instruction (EI,HLT).
A manual or an automatic INTERRUPT will be used for starting.

One-byte call instruction RESTART (RST) facilitates the processing of interrupts.RST enables the interrupting device to direct a call to memory location 070. Since the RST-instruction is a CALL-instruction,a RETURN (RET) instruction has to be put in cell 070.
The 8080 CPU contains provisions for Direct Memory Access (DMA) operations.By applying a HOLD to the appropriate Control pin on the processor,an external device such as the Intel 8008,can cause the CPU to suspend its normal operations and relunguish control of the address and data busses.
The processor responds to a request of this kind by floating its address to other devices sharing the busses.

ROM module - A ROM is a device that stores data in the form of program and it is non-volatible.
This module contains 2K Bytes of PROM, with addresses 000:000 - 007:377.
The Self-Tuning regulator needs less than 1,5K.

RAM module - RAMs do not hold their data when power is removed.
The RAM module also contains 2K with addresses 010:000 - 017:377.

Interface to 8008 - The problem of operator's communication in this first experimental setup is solved by using another microcomputer based on an Intel 8008,which has some communication programs.
A part of the 8008 memory is shared between the 8008 and 8080,thus enabling the operator to enter parameters to the 8008 and then switch the shared memory to the 8080.

A/D - converter-It is an eight channel,eight-bit converter with a conversion time of appr. 75 µs.
The input value to the CPU is in offset binary form.The conversion to 2-complement is done by software.

D/A-converter- It has four eight-channels.
The input value to the converter is in offset

binary form.

Both the A/D - and D/A - converters work between 0-9.98 V with a resolution of 39 mV.

Analog signal (V)	Binary representation D/A - and A/D - converters
9.984	11111111
9.945	11111110
.	.
.	.
.	.
4.992	10000000
.	.
.	.
0.039	00000001
0	00000000

6. MANUAL FOR STURE

Science is organized knowledge.
HERBERT SPENCER,EDUCATION

1. Connect the interface module to the memory slot.
2. Switch on the power supplies to both Intel 8080, and to the TRANSINTRO.
3. Set the switch EXT/INT on the interface module to EXT, and push the switch RUN/STOP/SINGLE STEP to the SINGLE STEP once.
   Now, TRANSINTRO has the command over the datamemory belonging to 8080.
4. Set the switchregister in TRANSINTRO to

   0 0 0 0 0 0 0 0 0 0 1 1 1 0 1

   and push on RUN:
   Now,to change your data, you can use the operatorconsol and the program which has been developed at the department.
5. A memory address may be selected using a thumbwheel switch. The content of the selected location, which is shown on the display, may be changed using another thumbwheel switch.There are possibilities to choose if the input value will be an integer or real,decimal or octal.When the IN-button is pushed the computer reads the new value. The floating-point numbers 1/RL, 1/B0, P0 have the mantissa in 2-complement and the exponent in Excess-64 form, $a*2^b$, where $b=(x-100)_8$ and x is the value on the display in octal form.
   All the vectors are initialized by the subroutine INIT.

The input parameters have following organization in the memory:

memory location		content	
010:	231	1/B0	Exponent
	232	1/B0	Less significant half of mantissa
	233	1/B0	Most significant half of mantissa
	234	1/RL	Exponent
	235	1/RL	Less significant half of mantissa
	236	1/RL	Most significant half of mantissa
	237	ULIM	Less significant half of ULIM (=0, using an 8-bit D/A -converter
	240	ULIM	Most significant half of ULIM
	241	NA	Integer
	242	NB	Integer
	243	NC	Integer
	244	N1	INIT computes N1=NA+NB+NC
	245	k	Integer
	246	P0	Exponent
	247	P0	Less significant half of mantissa
	250	P0	Most significant half of mantissa

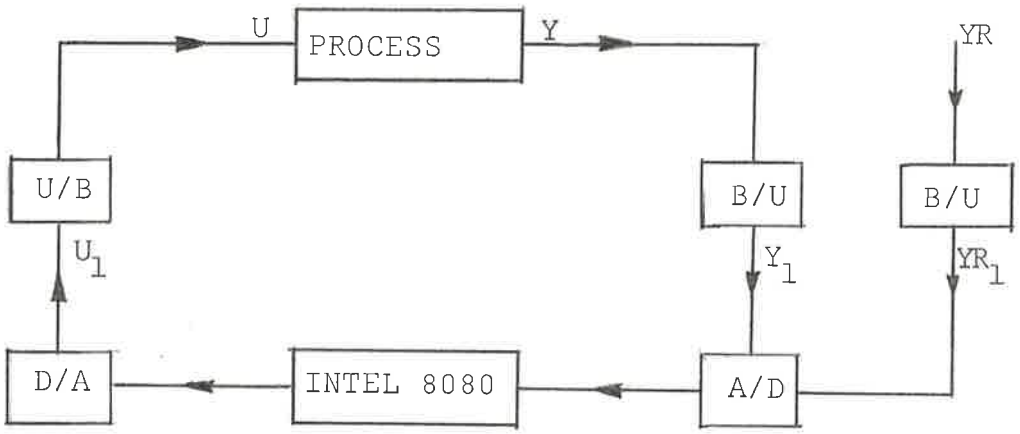
6. Push STOP on the TRANSINTRO.
7. Switch the button EXT/INT to INIT.
8. Connect the reference signal to channel 0 on the A/D -converter.
9. Connect the output signal from the process to channel 1 on the D/A -converter.
10. Connect the output from channel 0 on the D/A -converter to the process.
11. Connect the interrupt signal to INT on the CPU-module, and choose a sampling-interval.
12. Push RESET
13. Push RUN (voilà)

If you want to stop,for example change any parameter push STOP,and GO TO 3.

7. EXPERIMENTS

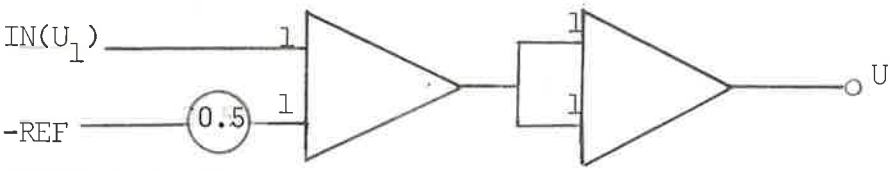
Experience is the best of schoolmasters.
PROVERT

Different processes have been simulated on the analog computer.The output value from the process as well as the input value to the process can vary between -10 V to +10 V. Since the A/D - and D/A - converters works only for 0-10 V we have to use an Bipolar-Unipolar converter and an Unipolar-Bipolar converter respectively.

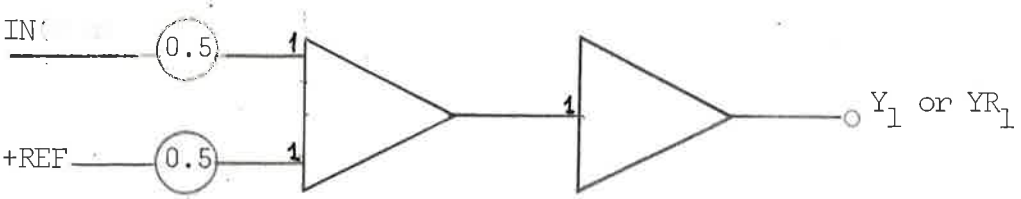


The schemes for the blocks U/B and B/U on the analog computer are as follows:

UNIPOLAR/BIPOLAR:



BIPOLAR/UNIPOLAR:



In all the following experiments, the number of pure time-delays, k ,have been chosen equal to zero.

1. A simple test of the algorithm.

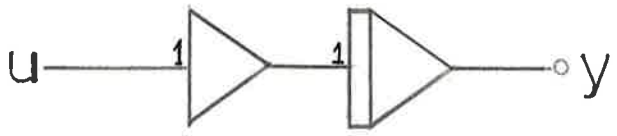
A registration of the output signal from the process (A single integrator) and the control signal to the process has been done when applicating a referencesignal change of 5 V.
A change of 5 V is for the computer a change of 0.5.
Figure 7.1 a shows the simulation of the process on the PDP-15.The reference signal change was entered after 15 samples.The second plot (Fig. 7.1 b) is from the analog computer and the microcomputer.
After the change of reference signal the output signals behave appropriate in the same manner in both figures.

2. Single-Integrator

Consider the system

$$(1-q^{-1})y(t) = T\times u(t-1) + e(t) \qquad (7.1)$$

The scheme of the process on the analog computer is as follows:

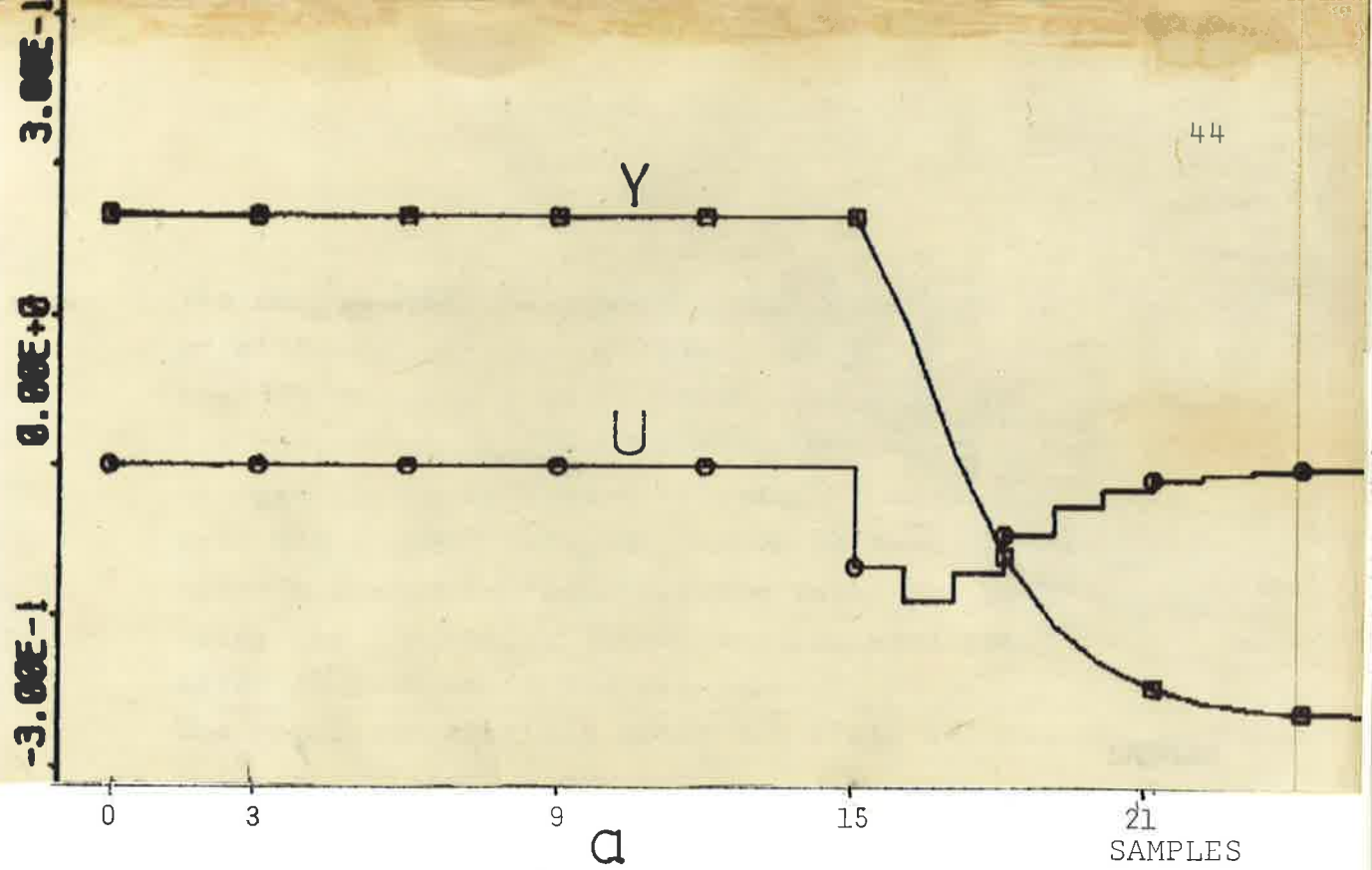


The optimal minimum-variance regulator is defined by the polynamials

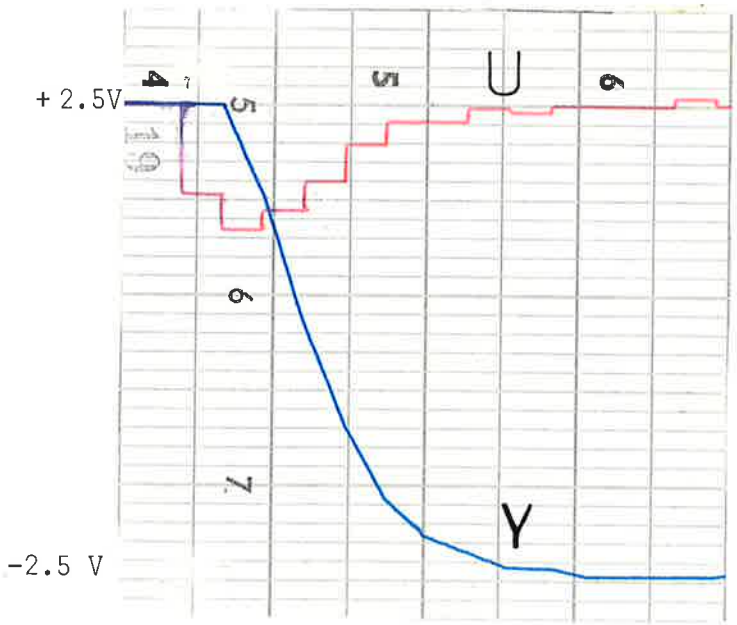
$$\mathcal{A} = 1 \quad \mathcal{B} = 1 \quad \mathcal{C} = 1-q^{-1}$$

Let T=1 (the samplinginterval is one second).

a) Let the reference value be a square wave with amplitude 1 V and a period of 20 steps. Further we assume that e(t)=0. The system is controlled by a self-tuning regulator with NA=1, NB=0, NC=0. The initial covariance matrix was P(0)=5×I, the exponential forgetting factor was λ=0.98, and the scale factor β_0 was 1.



a



b

Fig. 7.1 - The output and the control signal when the process is a single integrator.
The simulations were done on
a. A Minicomputer, PDP-15
b. A Microcomputer and an Analogcomputer

The output from the system and the input to the system at different periods of time is shown in figure 7.2.From the figure it is seen, ignoring for some disturbances on the converters,that the self-tuning regulator,after 10-12 changes in the reference values,converges to a regulator with the property that the error is zero after one step after a change in the reference value has occured. When using the forgetting factor $\lambda=1$ the same result was obtained after 30 changes in the set-point.
The regulator obtained after 300 steps (30 changes in the reference signal) was

$$u(t) = -0.87(y(t)-y_r(t))$$

The parameter α_1 converge to the value -1.
The optimal control law is $u(t) = -1(y(t)-y_r(t))$

The closed loop system will then be

$$y(t) = y_r(t-1) + e(t) \qquad (7.2)$$

For the system (7.1) the self-tuning algorithm converges to the same controller indepedent of the amplitude and period of the square wave signal as long as the system has time to settle between the changes.

b) Let the initial parameters be the same as in a) except for NC. Using feedforward control with NC=2, the simulation gave the result that the delay of the response was one sampling interval shorter than without feedforward control. In this case the regulator obtained after 300 steps was

$$u(t)=-0.93(y(t)-y_r(t))-(0.98y_r(t+1)-0.99y_r(t))$$

The optimal control law is

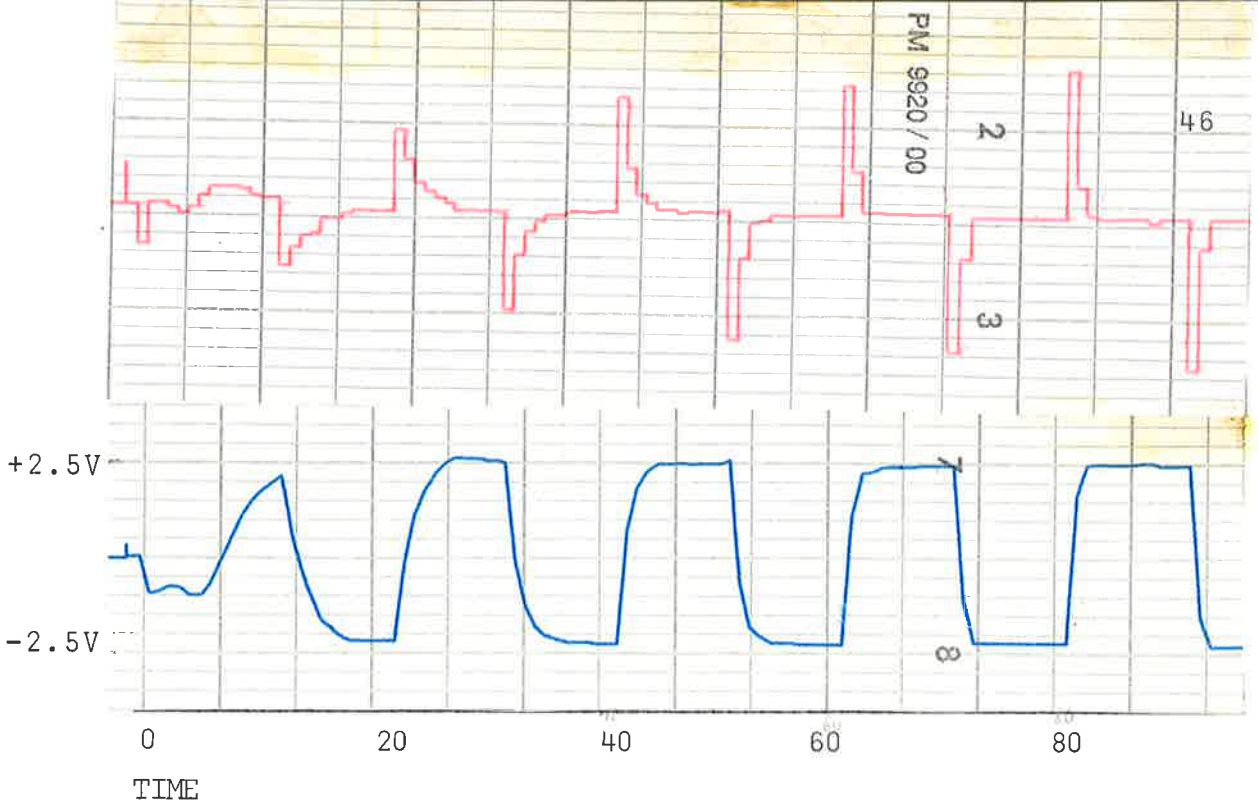
$$u(t)=-1(y(t)-y_r(t))-(y_r(t+1)-y_r(t))$$

The closed loop system will then be

$$y(t)=y_r(t)+e(t)$$

Compare with (7.2)

With the same initial parameters as above,a test of the regulator was done by changing the process during the control.



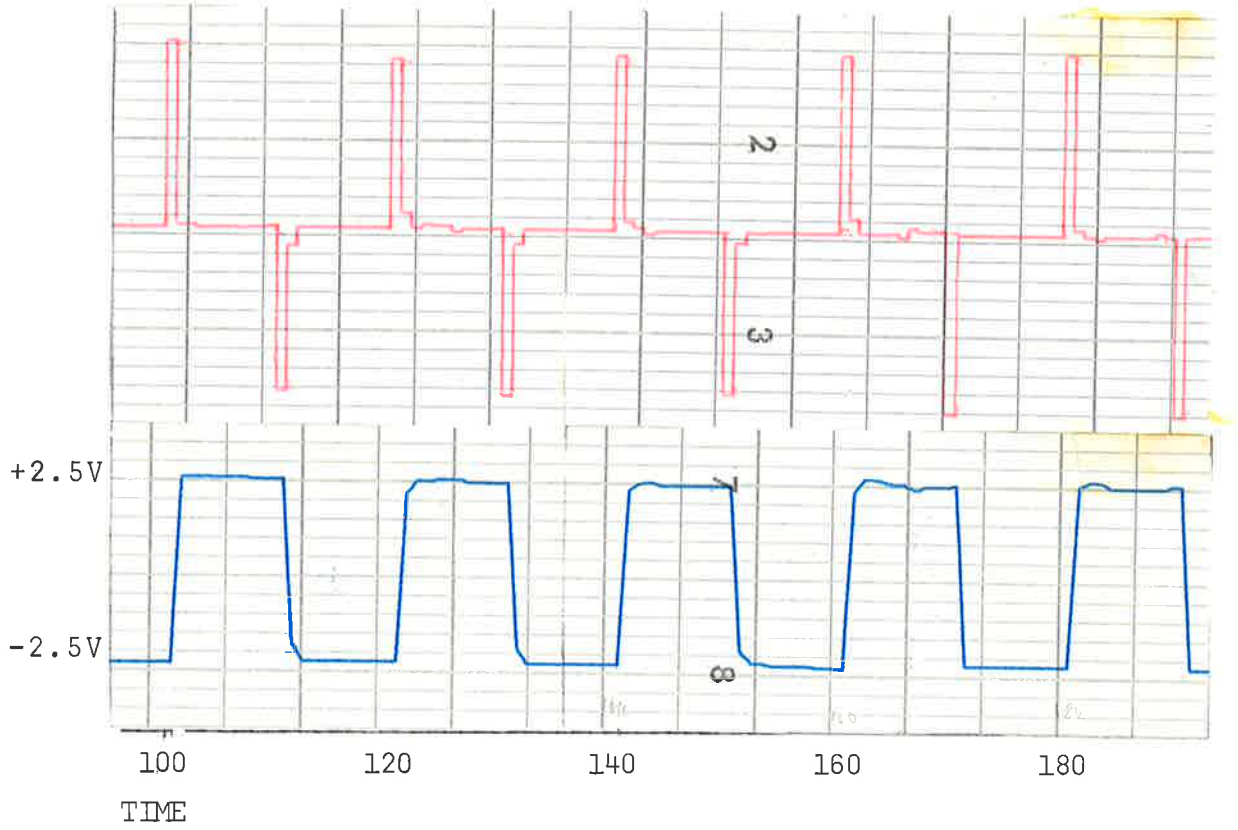


Fig. 7.2 The output and the input of system (7.1). The reference signal is a square wave.

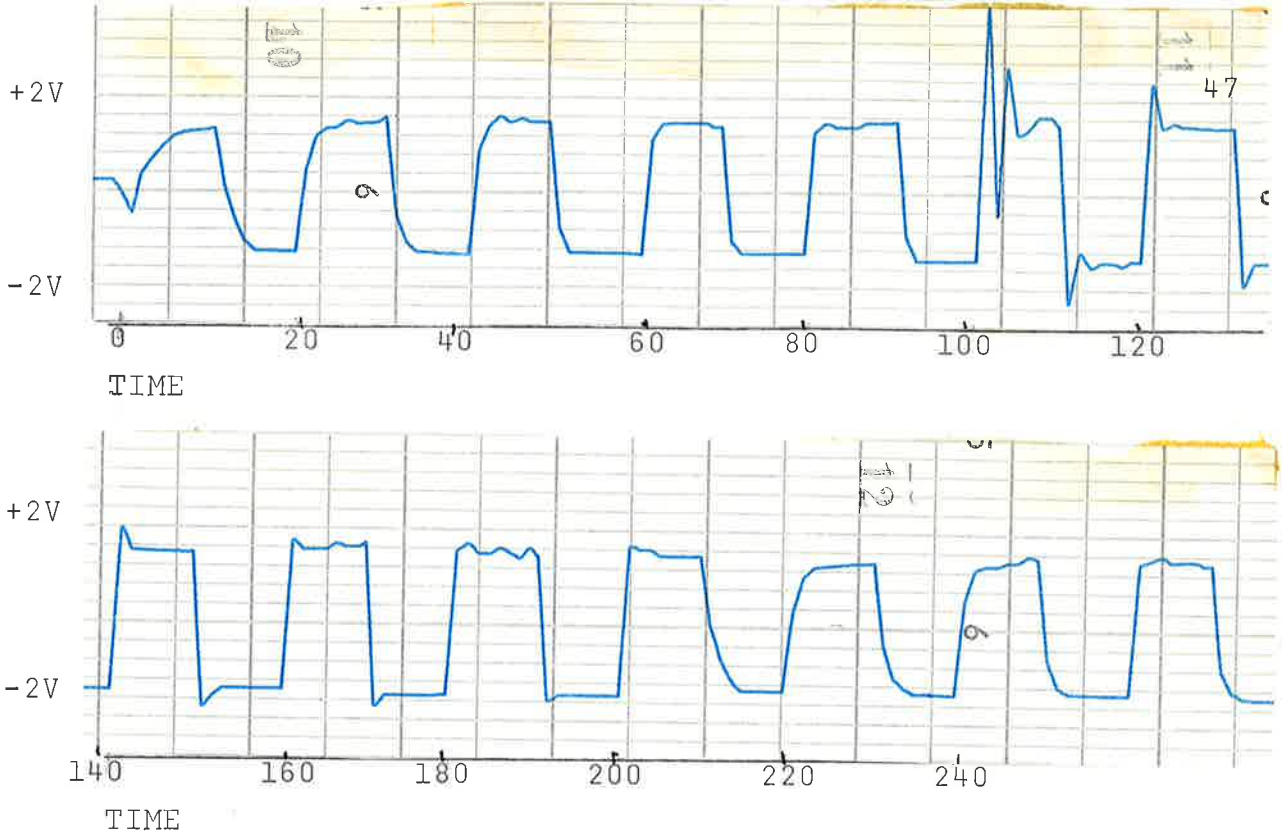


Fig. 7.3 The output of the system 7.1,when a change of the process occures.

Figure 7.3 shows the step responses at different times. After 11 changes of the reference value, the gain of the process was doubled.The overshot became of the same magnitude as the disturbances already at time 160.At time 220 the gain was set to its initial value.

c) Let the reference value be a triangular wave with amplitude 1.8 V and a period of 60 steps.The forgetting factor, λ, was chosen to 0.98. The system has beencontrolled by the self-tuning regulator with NA=2, NB=1, NC=0. The output from the system is shown in figure 7.4.
The regulator obtained after 488 steps was

$$u(t)=-0.31(y(t)-y_r(t))-0.225(y(t-1)-y_r(t-1))-0.875u(t-1)$$

The closed loop system in this case will be

$$y(t)=(2-q^{-1})y_r(t-1)$$

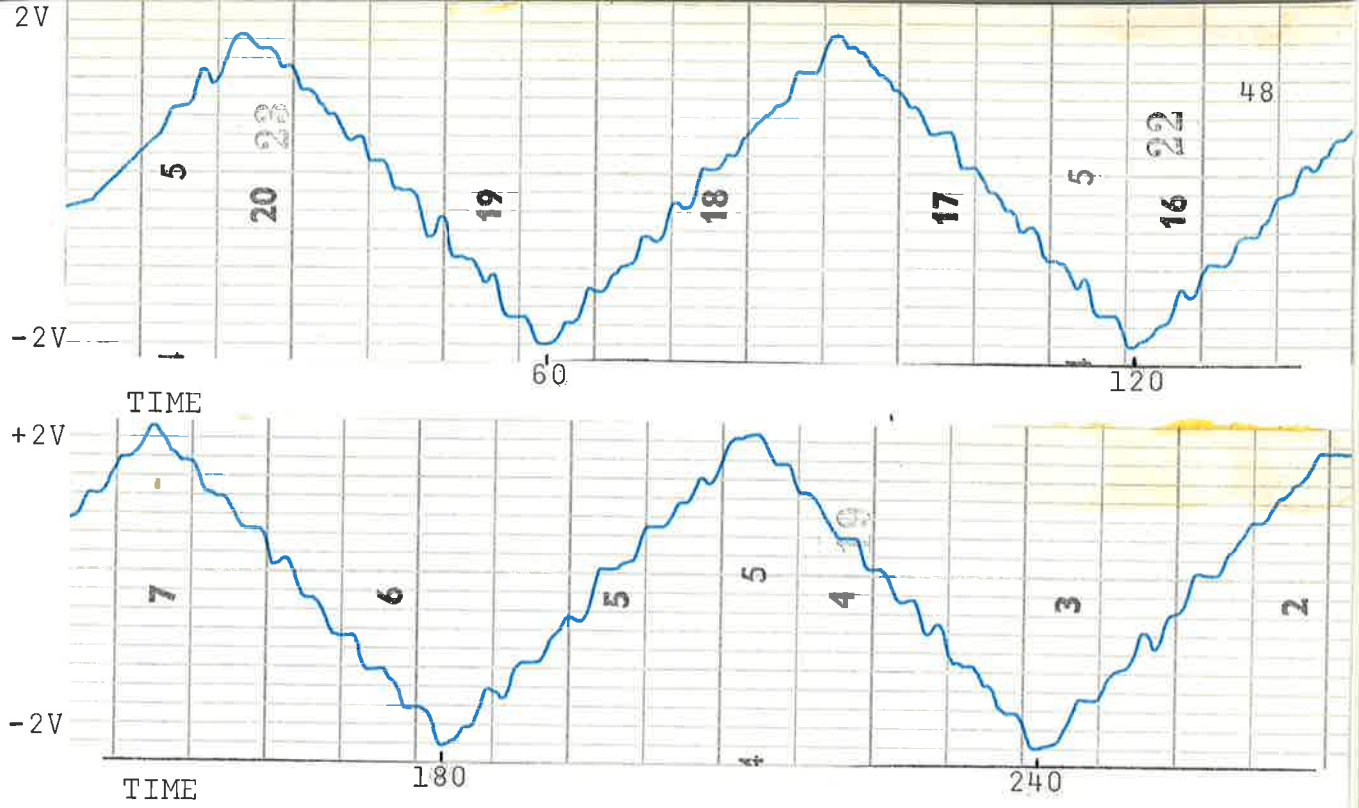


Fig. 7.4 The output of system 7.1, without any feedforward control. The reference signal is a triangular wave.

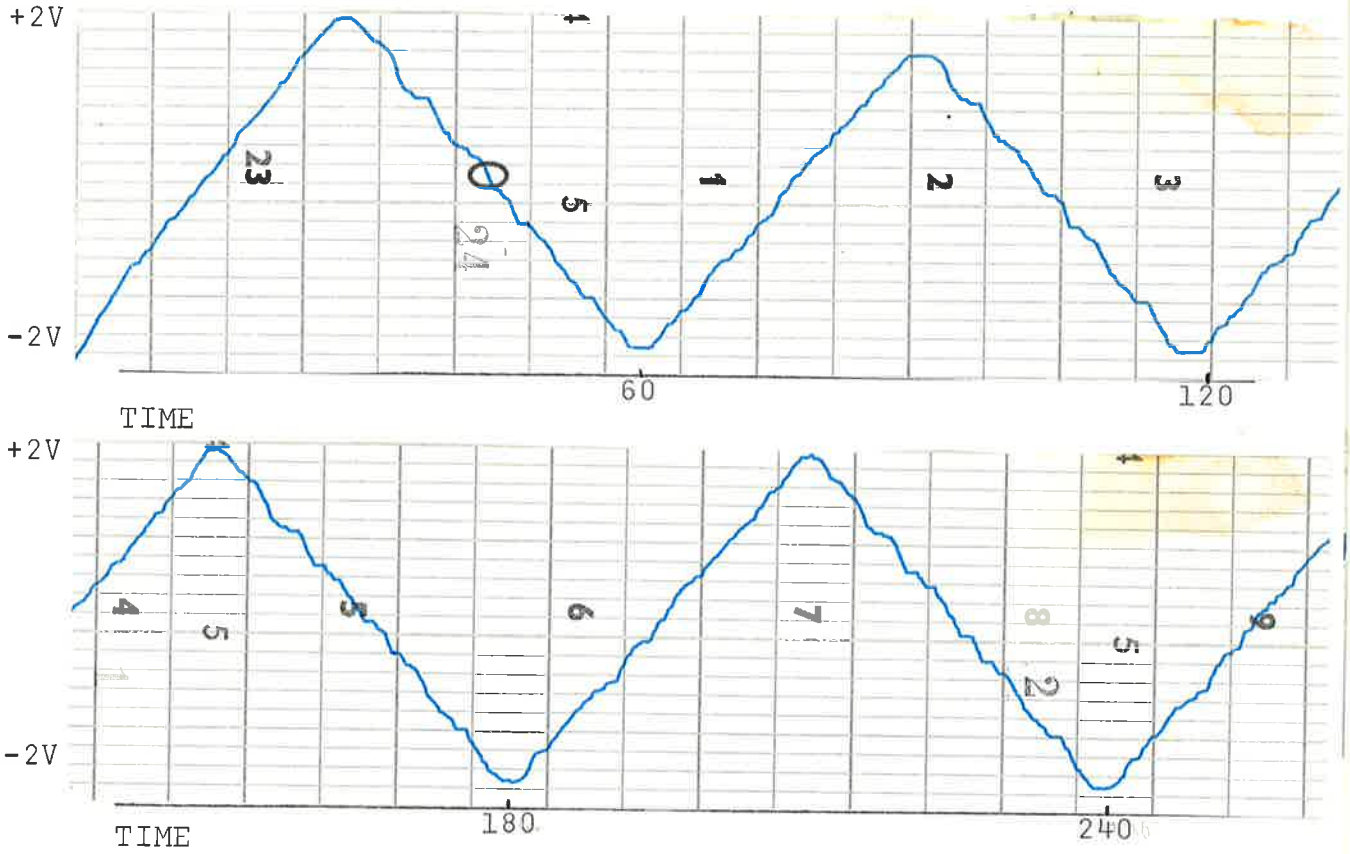


Fig. 7.5 The output of system 7.1, with feedforward control. The reference signal is a triangular wave.

d) Let NA=1, NB=0, NC=2. Using the optimal feedforward controller the closed loop system will be

$$y(t) = y_r(t) + e(t)$$

The result when using the self-tuning regulator is shown in figure 7.5.The regulator obtained, in this case, after 360 steps was

$u(t)=-1.64(y(t)-y_r(t))-(-1.4y_r(t+1)+1.25y_r(t))$

The optimal control law is

$u(t)=-1(y(t)-y_r(t))-(-1y_r(t+1)+1y_r(t))$

e) All the experiments above have been done with the noise equal to zero.

Figure 7.6 shows the output of the system when adding noise.In this case a constant reference signal is used, and NA=1,NB=0,NC=0 has been chosen.

The output is varying within $\pm$ 1V.The regulator converged to

$$u(t) = -0.97(y(t)-y_r(t)$$

Normally the regulator converge faster with a noise.

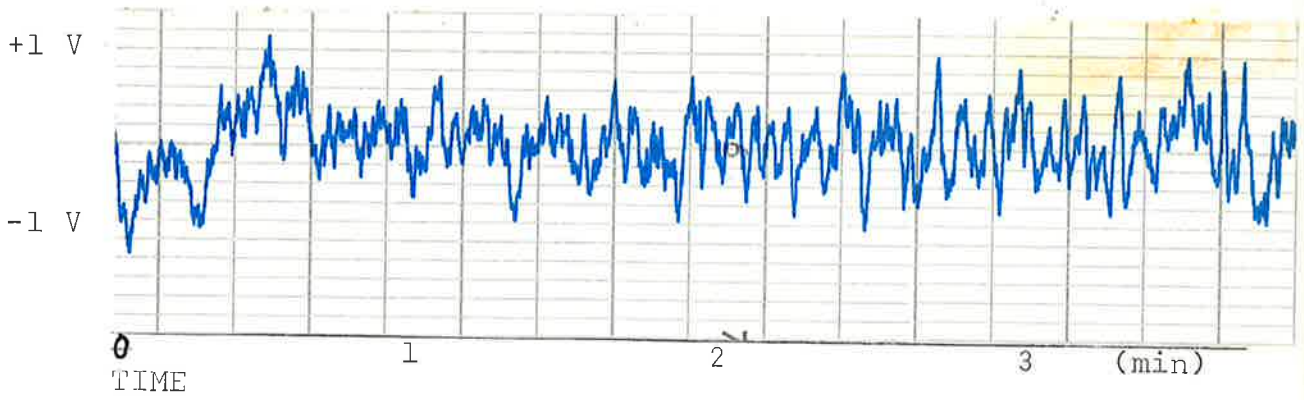
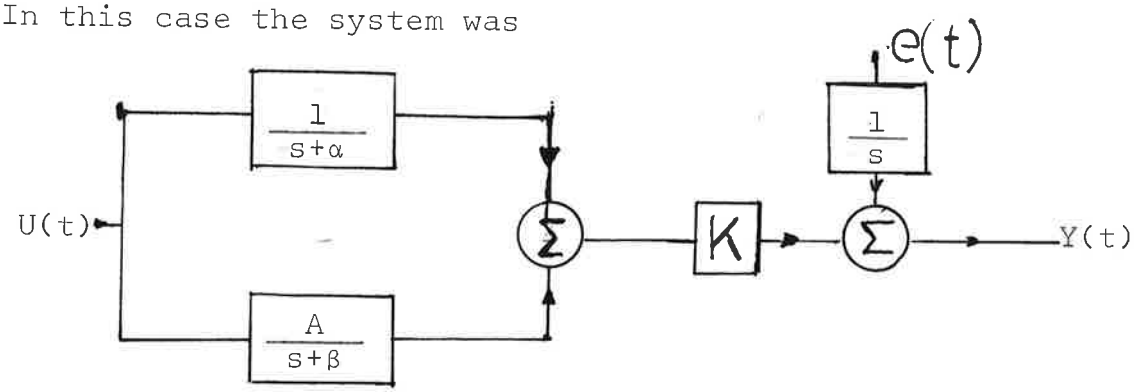


Fig. 7.6 The output of system 7.1,with noise and constant reference signal.

3. Third order system.

In this case the system was



where $\alpha=0.211$, $\beta=0.713$, $A=0.877$ and $K=4.744$

The sampled system will then be

$$A(q^{-1})y(t)=B(q^{-1})u(t-1)+C(q^{-1})e(t) \qquad (7.2)$$

where $A(q^{-1})= 1 - 2.6q^{-1} + 2.23q^{-2} - 0.63q^{-3}$

$B(q^{-1})= 0.5(1-q^{-1})$

$C(q^{-1})= 1 - 1.6q^{-1} + 0.63q^{-2}$

The optimal minimum variance regulator is determined by

$$\mathcal{A} = -1 + 1.6q^{-1} - 0.63q^{-2}$$
$$\mathcal{B} = 1 - q^{-1} \qquad \beta_o=0.5$$
$$\mathcal{C} = 1 - 2.6q^{-1} + 2.23q^{-2} - 0.63q^{-3} \qquad (7.3)$$

The sample interval for this system has been chosen to 0.5s. The scale factor $\beta_o=0.5$.

a) Parameter convergence.

Let the reference signal be constant and intoduce noise on the system. Let the forgetting factor be $\lambda=1$. The convergence matrix has been chosen to $P(0)=10\times I$.
In figure 7.7 the parameter estimates are shown.

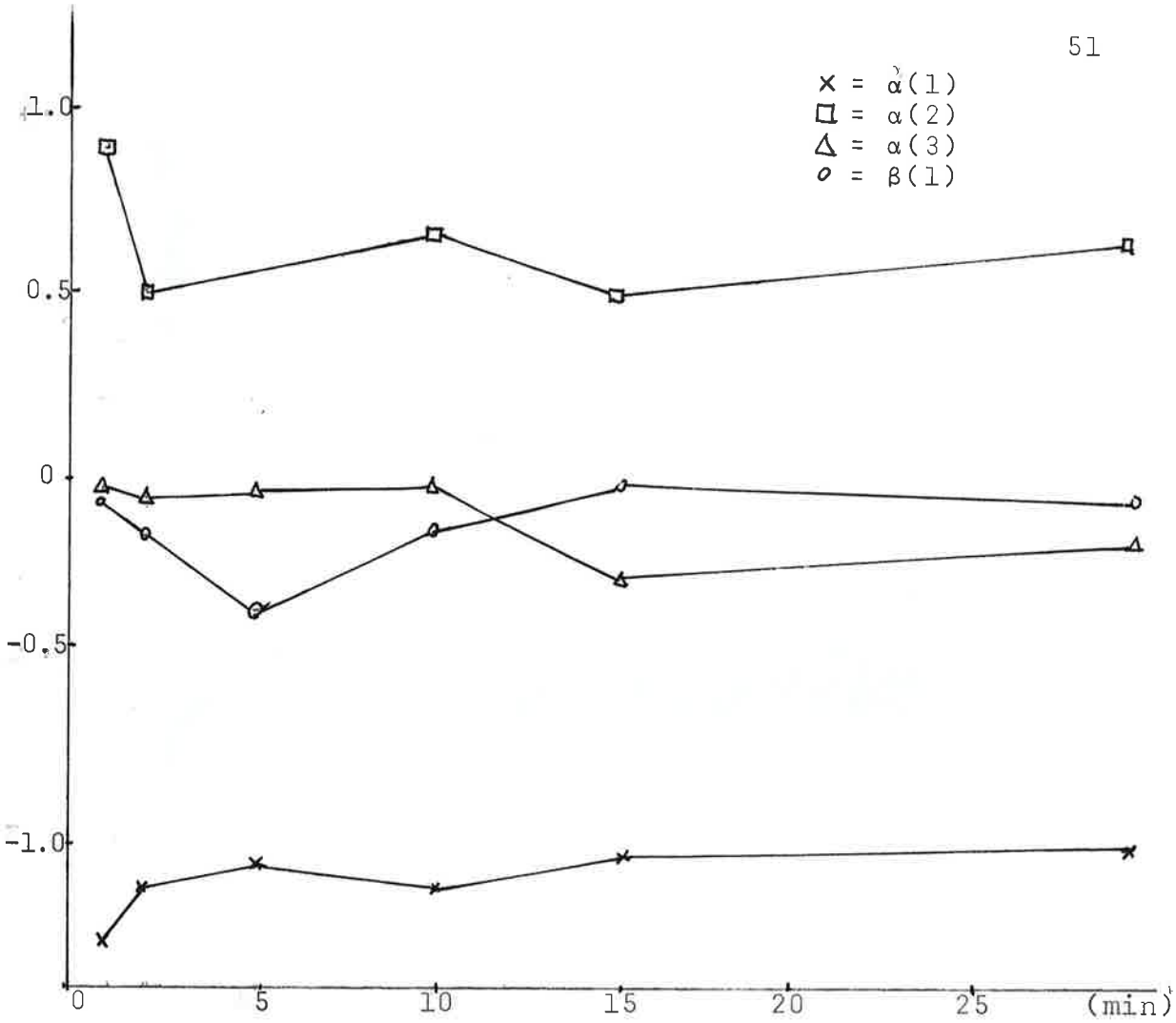


Fig. 7.7 Parameter estimates of system 7.2.

Even if the parameters have not converged to the optimal values after 30 min.,the control of the system was very good which will be shown in the next section.

b) Comparison between

Controlling the system with STURE. (Fig. 7.8 a)
Controlling the system with the theoretical minimum variance regulator. (Fig. 7.8 b)

The reference signal is constant and noise is introduced on the system.The initial parameters are:
$\lambda=1$, $P(0)=5\times I$, NA=3, NB=1, NC=0.
Although the parameters in the selftuning regulator had not converged to the theoretical values (7.3) the behavior of the closed loop system was very good.By just looking at the recording of output when using the two regulators it can be seen that the selftuning regulator is at least as good as the theoretical regulator.

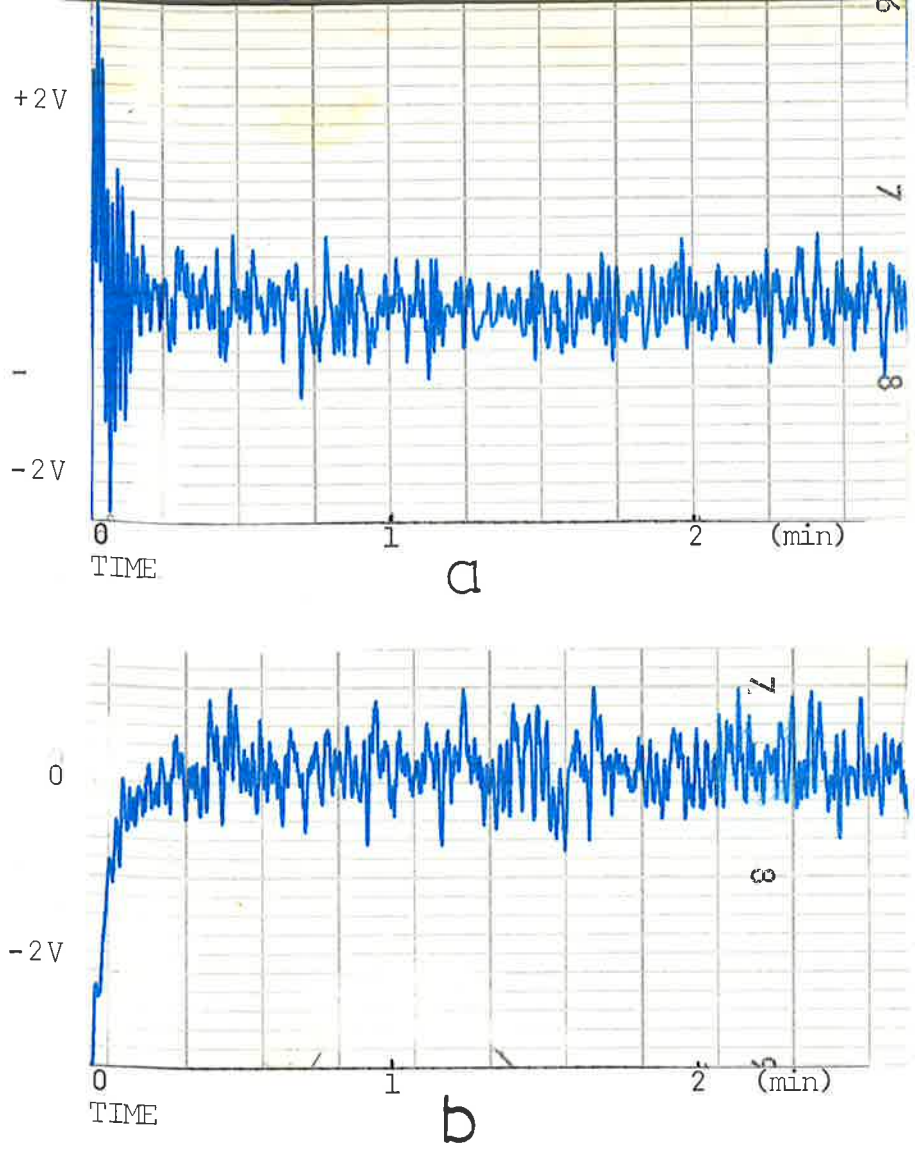


Fig. 7.8 The output of system 7.2 with
a) the self-tuning regulator
b) the minimum variance regulator

c) Let the reference signal be a square wave with amplitude 2V and period about 50 steps.
Noise is also added to the process.A better perfomance can be obtained if the controller also gets direct information about changes in the reference signal. This can be taken care of by a feedforward signal from the reference value.Let therefore NA=3,NB=1,NC=4.
The covariance matrix $P(0)=5\times I$.
The forgetting factor was at the beginning chosen to $\lambda=0.98$ and when the overshoot become less, the forgetting factor was changed to $\lambda=1$. (Fig. 7.9 a,b)
The change of λ occured after 5 minutes.

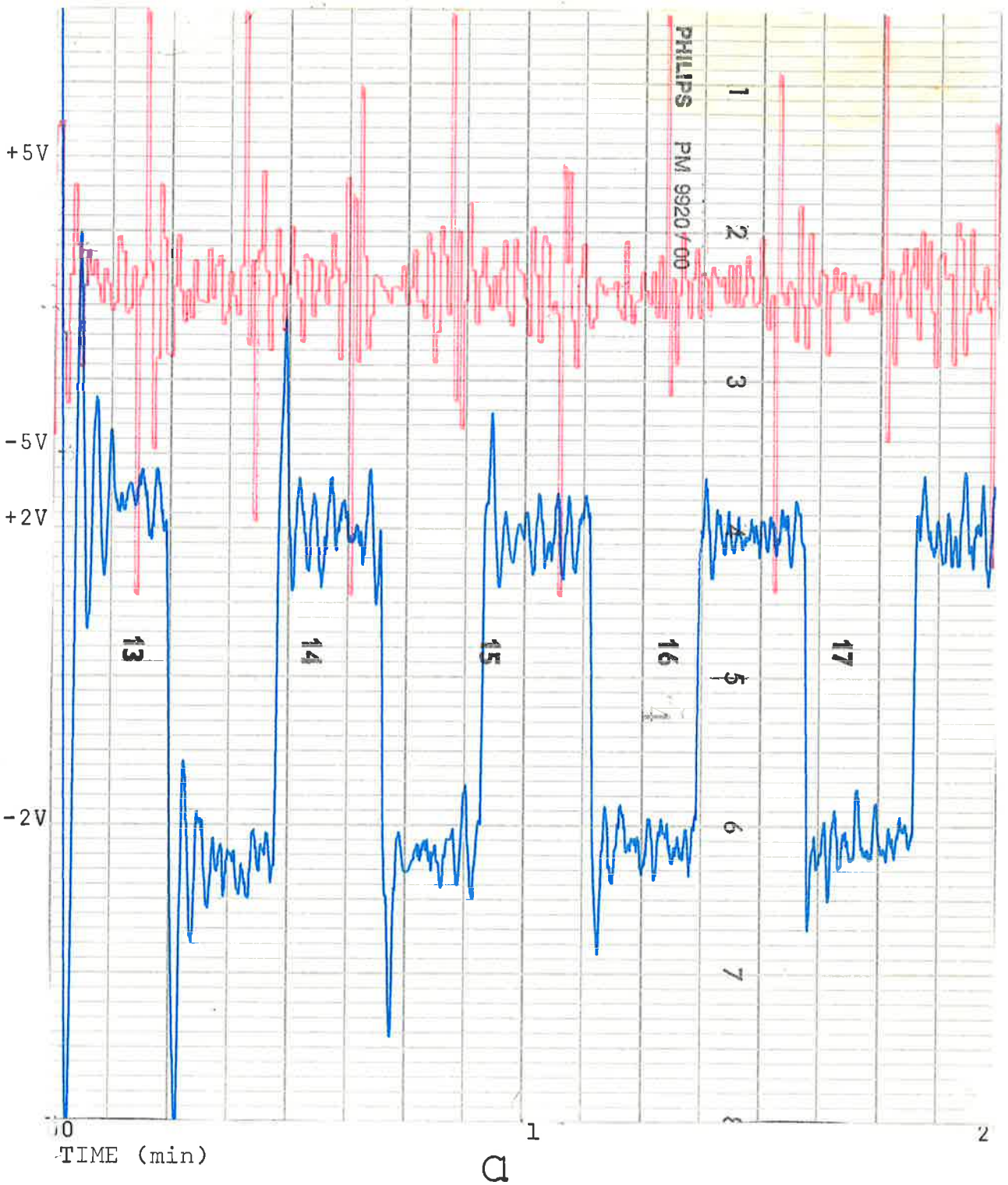


Fig. 7.9 The output and the control signal of system 7.2 with $\lambda=0.98$

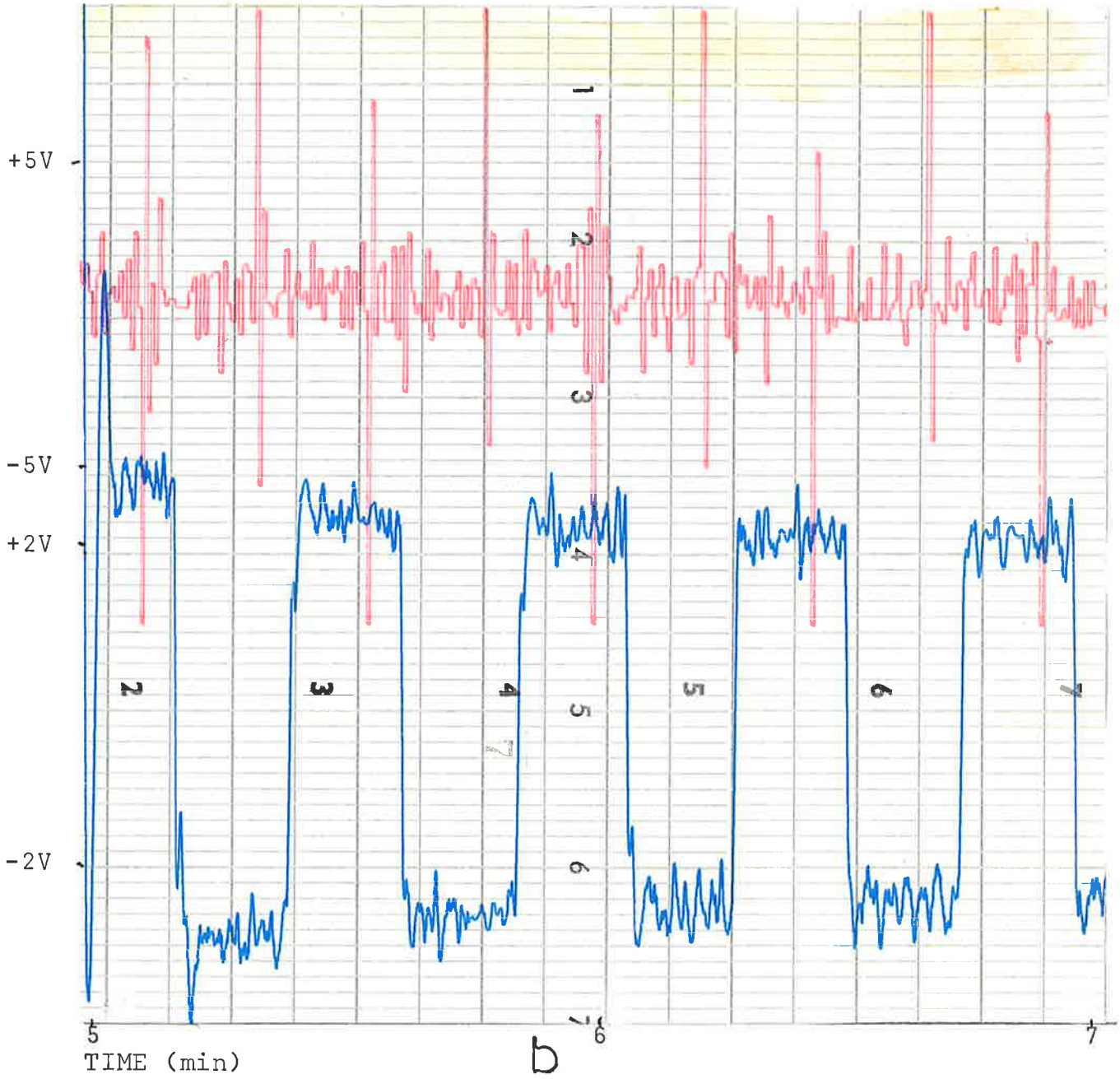


Fig. 7.9 The output and the control signal of system 7.2 with $\lambda = 1$.

The regulator obtained after 1150 steps (48 changes in the reference signal) was

$$U(t)=(-1.68+1.25q^{-1}-0.38q^{-2})(y(t)-y_r(t))/0.5 - (-0.6u(t-1) -$$
$$-(-1.2q+2.5-1.75q^{-1}+0.4q^{-2})y_r(t)/0.5$$

8. REFERENCES.

I recommend you to take care of the minutes for the hours will take care of themselves.

LORD CHESTERFIELD

{1} Andersson, L : Cross Assembly and relocation of programs for the Intel microprocessors using a PDP-15 as host computer, Report 7602,Division of Automatic Control, Lund Institute of Technology.

{2} Clarke, F,D : Floating-point arithmetic routines and Macros for an Intel 8080 microprocessor,Department of Engineering Science,University of Oxford.

{3} Sjöberg, P-O : Hardware to Intel 8080,Department of Automatic Control,Lund Institute of Technology.

{4} Wittenmark,B : A self-tuning regulator, Report 7311, April 1973,Division of Automatic Control, Lund Institute of Technology.

INSTRUCTION SET

Summary of Processor Instructions

Mnemonic	Description	Instruction Code[1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock[2] Cycles
$MOV_{r1,r2}$	Move register to register	0	1	D	D	D	S	S	S	5
MOV M, r	Move register to memory	0	1	1	1	0	S	S	S	7
MOV r, M	Move memory to register	0	1	D	D	D	1	1	0	7
HLT	Halt	0	1	1	1	0	1	1	0	7
MVI r	Move immediate register	0	0	D	D	D	1	1	0	7
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10
INR r	Increment register	0	0	D	D	D	1	0	0	5
DCR r	Decrement register	0	0	D	D	D	1	0	1	5
INR M	Increment memory	0	0	1	1	0	1	0	0	10
DCR M	Decrement memory	0	0	1	1	0	1	0	1	10
ADD r	Add register to A	1	0	0	0	0	S	S	S	4
ADC r	Add register to A with carry	1	0	0	0	1	S	S	S	4
SUB r	Subtract register from A	1	0	0	1	0	S	S	S	4
SBB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4
ANA r	And register with A	1	0	1	0	0	S	S	S	4
XRA r	Exclusive Or register with A	1	0	1	0	1	S	S	S	4
ORA r	Or register with A	1	0	1	1	0	S	S	S	4
CMP r	Compare register with A	1	0	1	1	1	S	S	S	4
ADD M	Add memory to A	1	0	0	0	0	1	1	0	7
ADC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7
SUB M	Subtract memory from A	1	0	0	1	0	1	1	0	7
SBB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7
ANA M	And memory with A	1	0	1	0	0	1	1	0	7
XRA M	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7
ORA M	Or memory with A	1	0	1	1	0	1	1	0	7
CMP M	Compare memory with A	1	0	1	1	1	1	1	0	7
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7
ANI	And immediate with A	1	1	1	0	0	1	1	0	7
XRI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7
ORI	Or immediate with A	1	1	1	1	0	1	1	0	7
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7
RLC	Rotate A left	0	0	0	0	0	1	1	1	4
RRC	Rotate A right	0	0	0	0	1	1	1	1	4
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4
RAR	Rotate A right through carry	0	0	0	1	1	1	1	1	4
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10
JC	Jump on carry	1	1	0	1	1	0	1	0	10
JNC	Jump on no carry	1	1	0	1	0	0	1	0	10
JZ	Jump on zero	1	1	0	0	1	0	1	0	10
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	10
JP	Jump on positive	1	1	1	1	0	0	1	0	10
JM	Jump on minus	1	1	1	1	1	0	1	0	10
JPE	Jump on parity even	1	1	1	0	1	0	1	0	10
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	10
CALL	Call unconditional	1	1	0	0	1	1	0	1	17
CC	Call on carry	1	1	0	1	1	1	0	0	11/17
CNC	Call on no carry	1	1	0	1	0	1	0	0	11/17
CZ	Call on zero	1	1	0	0	1	1	0	0	11/17
CNZ	Call on no zero	1	1	0	0	0	1	0	0	11/17
CP	Call on positive	1	1	1	1	0	1	0	0	11/17
CM	Call on minus	1	1	1	1	1	1	0	0	11/17
CPE	Call on parity even	1	1	1	0	1	1	0	0	11/17
CPO	Call on parity odd	1	1	1	0	0	1	0	0	11/17
RET	Return	1	1	0	0	1	0	0	1	10
RC	Return on carry	1	1	0	1	1	0	0	0	5/11
RNC	Return on no carry	1	1	0	1	0	0	0	0	5/11
RZ	Return on zero	1	1	0	0	1	0	0	0	5/11
RNZ	Return on no zero	1	1	0	0	0	0	0	0	5/11
RP	Return on positive	1	1	1	1	0	0	0	0	5/11
RM	Return on minus	1	1	1	1	1	0	0	0	5/11
RPE	Return on parity even	1	1	1	0	1	0	0	0	5/11
RPO	Return on parity odd	1	1	1	0	0	0	0	0	5/11
RST	Restart	1	1	A	A	A	1	1	1	11
IN	Input	1	1	0	1	1	0	1	1	10
OUT	Output	1	1	0	1	0	0	1	1	10
LXI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
LXI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
LXI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
LXI SP	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	11
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	11
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	11
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	11
POP B	Pop register pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
STA	Store A direct	0	0	1	1	0	0	1	0	13
LDA	Load A direct	0	0	1	1	1	0	1	0	13
XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
XTHL	Exchange top of stack, H & L	1	1	1	0	0	0	1	1	18
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	5
PCHL	H & L to program counter	1	1	1	0	1	0	0	1	5
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	5
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	5
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	5
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	5
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	5
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	5
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	5
DCX SP	Decrement stack pointer	0	0	1	1	1	0	1	1	5
CMA	Complement A	0	0	1	0	1	1	1	1	4
STC	Set carry	0	0	1	1	0	1	1	1	4
CMC	Complement carry	0	0	1	1	1	1	1	1	4
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
EI	Enable Interrupts	1	1	1	1	1	0	1	1	4
DI	Disable interrupt	1	1	1	1	0	0	1	1	4
NOP	No-operation	0	0	0	0	0	0	0	0	4

NOTES: 1. DDD or SSS – 000 B – 001 C – 010 D – 011 E – 100 H – 101 L – 110 Memory – 111 A.
2. Two possible cycle times, (5/11) indicate instruction cycles dependent on condition flags.

```
FILE NAMES
     INIT
     STURE
     REG1
     REG2
     ESTIM
     SCAMU
     FBIN
     FLIM
     FMOVE
     FDIV


LIBRARY:  NONE


MEMORY REQUIRED:  000:100 - 005:216


LOAD MAP
     INIT        000:100
     STURE       000:220
     REG1        000:353
     REG2        001:136
     ESTIM       001:312
     SCAMU       003:210
     FBIN        004:273
     FLIM        004:353
     FMOVE       005:001
     FDIV        005:045
     ADDA        005:160


GLOBAL SYMBOL TABLE
     INIT        000:100
     STURE       000:220
     REG1        000:353
     REG2        001:136
     ESTIM       001:312
     BINF        004:263
     FADD        004:042
     FMAGN       004:237
     FMUL        003:323
     LOAD        004:247
     NEGDE       004:177
     NEGDE1      004:200
     OFLW        004:215
     OUFLW       004:210
     SCAPR       003:210
     STORE       004:255
     STSGN       004:222
     UFLW        004:231
     UFLWE       004:234
     FBIN        004:273
     FLIM        004:353
     FMOVE       005:001
     MOVE        005:014
     FDIV        005:045
     ADIN        005:160
     DAOUT       005:207
```

```
/       SUBROUTINE INIT
/       INITIALIZE STURE
/       THE VECTORS YS-U-VS- AND T ARE GETTING THEIR
/       INITIALISE VALUES ZERO.
/       P(0)=A*I THERE A IS AN ARBITARY REAL NUMBER
/       STORED IN THE FIRST THREE BYTES IN BANK11 AND
/       I IS A UNIT MATRIX.
        AUTHOR GEORGE KIZIROGLU 1976-05-23
/       SUBROUTINES REQUIRED
/         ZERO, LOAD, STORE
BANK10=10
BANK11=11
N1=BANK10*400+244
NA=241
SLASK3=BANK11*400+377
        .GLOBL LOAD,STORE,INIT
INIT    MVI B,230
        LXI H,BANK10*400
        CALL ZERO
/
/COMPUTE N1=NA+NB+NC AND STORE IT
/
        LXI H,BANK10*400+NA
        MOV A,M                     /A=NA
        INX H
        ADD M                       /A=NA+NB
        INX H
        ADD M                       /A=NA+NB+NC
        INX H
        MOV M,A
/
/SET ZERO TO THE REST OF BANK10
/
        MVI B,110
        LXI H,BANK10*400+251
        CALL ZERO
/
/INITIALIZE THE P-MATRIX
/
        LXI H,BANK10*400+246
        CALL LOAD
        PUSH PSW
        MVI B,373
        LXI H,BANK11*400
        CALL ZERO
        LXI H,BANK11*400
        MVI C,1
NEXT    POP PSW
        CALL STORE
        STA SLASK3
        LDA N1
        CMP C
        RZ
        INR C
        MOV B,A
        ADD B
        ADD B
        INR A
        ADD L
        MOV L,A
        LDA SLASK3
```

```
        PUSH PSW
        JMP NEXT
/
/SETS ZERO TO B NUMBER OF BYTES BEGINNING
/ON ADDRESS GIVING BY THE REGISTERS H L
/
ZERO    XRA A
NEXT1   MOV M,A
        INX H
        DCR B
        JNZ NEXT1
        RET
        .END
```

```
/       MAINPROGRAM STURE
/       SAMPLE Y(T) AND V(T) AND CALL DIFFERENT SUBROUTINES
/       FOR COMPUTING U(T)
/       AUTHOR GERORGE KIZIROGLU 1976-03-19
BANK10=10
B0=BANK10*400+231
VS1=124
YS1=BANK10*400
NC=BANK10*400+243
K=245
        .GLOBL ADIN,NEGDE1,FQDD,LOAD,FMUL,STORE,REG1
        .GLOBL REG2,STURE,ESTIM
STURE   EI
        HLT
/
/READ YREF FROM CHANNEL NUMBER 0 AND
/CONVERT IT TO FLOATING NUMBER
/
        MVI A,0
        CALL ADIN
        PUSH D
/
/COMPUTE VS(T+K+1)=YREF(T+K+1)/B0
/
        LXI H,B0
        CALL LOAD
        XCHG                        /HL A=B0
        POP D
        CALL FMUL                   /DE B=VS(T+K+1)
        MOV A,B
        LXI H,BANK10*400+VS1
        CALL STORE
/
/READ Y FROM CHANNEL NUMBER 1 AND
/COMPUTE YS(1)=Y(T)/B0-V(T)
/
        MVI A,1
        CALL ADIN
        PUSH D
        LXI H,B0
        CALL LOAD
        XCHG
        POP D
        CALL FMUL                   /DE B=Y(T)/B0
        PUSH D
        PUSH B
        LDA BANK10*400+K
        ADI 1
        MOV C,A
        ADD C
        ADD C
        ADI VS1
        MOV L,A
        MVI H,BANK10                /HL=VS(K+2) WHERE V(T)=VS(K+2)
        CALL LOAD
        MOV B,A
        CALL NEGDE1
        MOV A,B
        XCHG                        /HL A=-VS(T)
        POP B
        POP D
```

```
        CALL FADD               /DE B=YS(T)
        MOV A,B
        LXI H,YS1
        CALL STORE
/
/BEGIN THE ESTIMATION AND
/THE TUNING
/
        CALL REG1
        CALL ESTIM
        CALL REG2
        JMP STURE
        .END
```

```
/       ADDA
/       SUBROUTINE ADIN
/       READS THE INPUT VALUE FROM A CHANNEL
/       AND CONVERTS IT A FLOATING NUMBER
/       ENTRY:    A=CHANNEL NUMBER
/       EXIT:      B=EXPONENT OF THE INPUT VALUE
/                 DE=MANTISSA OF THE INPUT VALUE
/       AUTHOR GEORGE KIZIROGLU 1976-05-01
/       SUBROUTINES REQUIRED
/            BINF
/
        .GLOBL ADIN,DAOUT,BINF,FBIN
ADIN    OUT 376
        XRA A
WAIT    IN 377
        ANI 1
        CPI 0
        JNZ WAIT
        IN 376
        XRI 200                    /CONVERTS TO TWO-COMPLEMENT
        MOV D,A
        MVI E,0
        CALL BINF
        RET
/
/       SUBROUTINE DAOUT
/       OUTPUTS THE INPUT FLOATING NUMBER
/       IN A FIX FORM TO A CHANNEL
/       ENTRY:    A=CHANNEL NUMBER
/                DE B=THE LIMITED OUTPUT FLOATING NUMBER
/       EXIT:    A=THE FIX OUTPUT VALUE
/       AUTHOR GEORGE KIZIROGLU 1976-05-01
/       SUBROUTINES REQUIRED
/         FBIN
/
DAOUT   MOV A,D
        XRI 200                    /CONVERTS TO BINARY OFFSET
        OUT 377
        OUT 375
        RET
        .END
```

```
/       SUBROUTINE REG1
/       COMPUTES FSUM=AE(1)*YS(1)-CE(1)*VS(1)
/       THE U-VECTOR IS THEN SHIFTED
/       U(T)=FSUM+FSCAL IS THEN COMPUTED AND STORES IN MEMORY
/       FSCAL IS COMPUTED FROM THE LATEST SAMPLE IN REG2
/       REG1 WORKS ONLY IN BANK 10
/       AUTHOR   GEORGE KIZIROGLU 1976-03-19
/       SUBROUTINES REQUIRED
/             LOAD
/             STORE
/             FMUL
/             FADD
/             FMOVE
BANK= 10
AE1=176
YS1=0
VS1=124
U1=52
NA=BANK*400+241
FSCAL=254
ULIM=BANK*400+237
        .GLOBL LOAD,FMUL,NEGDE1,FADD,FMOVE,STORE
        .GLOBL REG1,FMAGN,FLIM,DAOUT,BINF,FBIN
/
/COMPUTE AE(1)*YS(1)
/
REG1    LXI H,BANK*400+AE1
        CALL LOAD
        PUSH D
        PUSH PSW                    /AE(1) IS SAVED
        MVI L,YS1
        CALL LOAD
        MOV B,A                     /DE B =YS(1)
        POP PSW
        POP H                       /HL A =AE(1)
        CALL FMUL
        LDA NA+2
        CPI 0
        JZ NOCE1
        PUSH D
        PUSH B                      /AE(1)*YS(1) IS SAVED
/
/SEARCH CE(1) AND  COMPUTE CE(1)*VS(1)
/
        LXI H,NA
        MOV A,M
        INX H
        ADD M                       /A=NA+NB
        MOV L,A
        ADD L
        ADD L
        ADI AE1
        MOV L,A                     /L↑CE(1)
        CALL LOAD
        PUSH D
        PUSH PSW                    /DE A =CE(1)
        MVI L,VS1
        CALL LOAD
        MOV B,A                     /DE B =VS(1)
        POP PSW
        POP H                       /HL A =CE(1)
```

```
        CALL FMUL                   /DE B =CE(1)*VS(1)
/
/COMPUTE FSUM
/
        CALL NEGDE1
        MOV A,B
        XCHG
        POP B
        POP D
        CALL FADD
NOCE1   MOV A,B
        PUSH D
        PUSH PSW                            /FSUM IS SAVED
/
/SHIFT U-VECTOR ONE STEP
/
        LXI H,NA+1                  /HL=ADDRESS TO NB
        MVI E,U1                    /E=ADDRESS TO U(1)
        CALL FMOVE
/
/COMPUTE U(T)=FSCAL+FSUM
/
        MVI L,FSCAL
        CALL LOAD                   /DE A =FSCAL
        MOV B,A
        POP PSW
        POP H
        CALL FADD
        CALL FMAGN
        CALL FBIN
        LXI H,ULIM
        CALL FLIM
        CALL DAOUT
        CALL BINF
        MOV A,B
        LXI H,BANK*400+U1
        CALL STORE
        RET
        .END
```

```
/       SUBROUTINE ESTIM
/       ESTIMATES THE PARAMETERS AE(1).....AE(NA),
/       BE(1).....BE(NB),CE(1).....CE(NC) AND UPDATES
/       P-MATRIX.
/       A SEPARATE FI-VECTOR IS STORED BEGINNING AT ADDRESS 275
/       WITH NA+NB+NC ELEMENTS.
/       AT EACH SAMPLE THE VALUES -YS(K+2),U(K+2),VS(K+2) ARE
/       MOVED TO FI(1),FI(NA+1) AND FI(NA+NB+1) RESPECTIVELY.
/       THE P-MATRIX IS IN BANK 11,AND STORED BY ROWS.
/       AUTHOR   GEORGE KIZIROGLU 1976-03-19
/       SUBROUTINS REQUIRED  LOAD,STORE,NEGDE,SCAPR,FSUB
/                FADD,FDIV,FMUL,MOVE(IN FSHIFT)
BANK10=10
BANK11=11
NA=241
FI1=312
U2=55
VS1=124
S1=257
DEN=345
RES=251
T1=176
RK1=372
SLASK1=363
RL=234
SLASK3=BANK11*400+376
N1=BANK10*400+244
        .GLOBL ESTIM,LOAD,NEGDE1,STORE,SCAPR,FADD,MOVE
        .GLOBL FDIV,FMUL
/
/UPDATE FI-VECTOR
/
ESTIM   LXI H,BANK10*400+NA+4
        MOV A,M
        INR A
        MOV B,A                     /B=K+1
        ADD B
        ADD B
        STA SLASK3+1
        MOV L,A                     /L↑ YS(K+2)
        CALL LOAD
        MOV B,A
        CALL NEGDE1                 /NEGATES YS(K+2)
        MOV A,B
        LXI H,BANK10*400+FI1
        CALL STORE                  /-YS(K+2) IS PLACED IN FI(1)
        LDA BANK10*400+NA+1
        CPI 0
        JZ NC
        LDA SLASK3+1
        ADI U2
        MOV L,A                     /L↑U(K+2)
        CALL LOAD
        MOV C,A                     /DE C=U(K+2)
        MVI L,NA
        MOV A,M
        MOV B,A
        ADD B
        ADD B
        ADI FI1
        MOV L,A                     /L↑FI(NA+1)
```

```
        MOV A,C
        CALL STORE              / U(K+2) IS PLACED IN FI(NA+1)
NC      LDA BANK10*400+NA+2
        CPI 0
        JZ SV
        LDA SLASK3+1
        ADI VS1
        MOV L,A                 /L↑VS(K+2)
        CALL LOAD
        MOV B,A
        LDA N1                  /A=NA+NB+NC
        MVI L,NA+2
        SUB M
        MOV C,A                 /C=NA+NB
        ADD C
        ADD C
        ADI FI1
        MOV L,A                 /L↑FI(NA+NB+1)
        MOV A,B
        CALL STORE              /FI(NA+NB+1)=VS(K+2)
/
/COMPUTE THE S-VECTOR AND STORE IN BANK 10
/BEGINNING AT ADDRESS 242
/DE=ADDRESS TO P(I,J)
/HL=ADDRESS TO S(I)
/
SV      LDA N1
        STA SLASK3              /SLASK3 TEST IF S(N1) COMPUTED
        MVI L,S1
        PUSH H
        LXI D,BANK11*400        /DE↑P(1,1)
SI      LXI B,BANK10*400+FI1    /BC↑FI(1)
        MVI L,N1                /HL↑N1
        CALL SCAPR
        XTHL                    /HL↑S(I) AND SP=ADDRESS TO P(I+1,1)
        MOV A,B
        CALL STORE
        INX H
        XTHL                    /HL↑P(I+1,1) AND SP=ADDRESS TO S(I+1)
        XCHG                    /DE↑P(I+1,1)
        LXI H,SLASK3
        DCR M
        MVI H,BANK10
        JNZ SI
        POP D
/
/COMPUTE DENOM =1.0+FI(I)*S(I)
/
        LXI B,BANK10*400+FI1    /L↑FI(1)
        LXI D,BANK10*400+S1     /DE↑S(1)
        MVI L,N1
        CALL SCAPR
        LXI H,40000
        MVI A,101               /HL A =1.0
        CALL FADD
        MOV A,B
        LXI H,BANK10*400+DEN
        CALL STORE              /DENOM IS STORED
/
/COMPUTE RES=YS(1)-U(K+1)-FI(I)*T(I)
/
```

```
        LXI B,BANK10*400+FI1
        LXI D,BANK10*400+T1
        MVI L,N1
        CALL SCAPR
        CALL NEGDE1
        PUSH D
        MOV A,B
        PUSH PSW                    /DE A =FI(I)*T(I) IS SAVED
        LXI H,BANK10*400
        CALL LOAD
        MOV B,A                     /DE B=YS(1)
        POP PSW
        POP H                       /HL A =FI(I)*T(I)
        CALL FADD
        PUSH D
        PUSH B
        LDA SLASK3+1
        ADI U2-3
        MVI H,BANK10
        MOV L,A                     /L↑U(K+1)
        CALL LOAD
        MOV B,A
        CALL NEGDE1
        MOV A,B
        XCHG
        POP B
        POP D
        CALL FADD
        MOV A,B
        LXI H,BANK10*400+RES
        CALL STORE                  / RES IS STORED
/
/SHIFT FI-VECTOR
/
        LDA N1
        SUI 1
        MOV B,A
        ADD B
        ADD B
        ADI FI1
        MOV L,A                     /L↑FIRST SHIFTING ELEMENT
        INR B                       /B GIVES THE NUMBER OF SHIFTS
        CALL MOVE                   /MOVE IS WITHIN SUBROUTINE FSHIFT
/
/UPDATE THE PARAMETERS AND THE P-MATRIX
/D=3N1
/E=THE ADDRESS TO P(I,1)
/DE IS ON THE BOTTOM OF THE STACK THE WHOLE TIME
/B C=I J  IS MOST OF TIME BEFORE DE
/
        LDA N1
        MOV D,A                     /D=NA+NB+NC=N1
        ADD D
        ADD D
        MOV D,A                     /D=3N1
        MVI E,0                     /E↑P(1,1)
        MVI B,1                     /B=I
/
/LOOP1 IS PASSING THROUGH FOR EACH I-VALUE
/COMPUTE RK(I)=S(I)/DENOM
/         T(I)=T(I)+RK(I)*RES
```

```
/
LOOP1   MVI C,1                 /C=J
        MOV A,B
        ADD B
        ADD B
        PUSH D
        PUSH B
        PUSH PSW                /A GIVES THE I-ELEMENT OF A VECTOR
        ADI S1-3                /A↑S(I)
        MOV L,A
        CALL LOAD
        MOV B,A
        PUSH D
        PUSH B                  /S(I) IS SAVED
        MVI L,DEN
        CALL LOAD
        XCHG                    /HL A =DENOM
        POP B
        POP D
        CALL FDIV               /DE B =S(I)/DENOM
        PUSH D
        MOV A,B
        LXI H,BANK11*400+RKI
        CALL STORE              /RK(I) IS STORED
        MOV B,A
        MVI L,RES
        MVI H,BANK10
        CALL LOAD
        XCHG                    /HL A =RES
        POP D
        CALL FMUL               /DE B =RK(I)*RES
        POP PSW
        ADI T1-3
        MOV L,A
        MVI H,BANK10            /HL↑T(I)
        PUSH H
        PUSH D
        PUSH B
        CALL LOAD
        XCHG                    /HL A =T(I)
        POP B
        POP D
        CALL FADD               /T(I) IS UPDATED
        MOV A,B
        POP H
        CALL STORE
        POP B                   /B,C=I,J
        POP D                   /D,E=3N1,ADDRESS TO P(I,1)
        MOV L,E
        MVI H,BANK11            /HL=ADDRESS TO P(I,J)
        MOV A,D
        ADD E
        MOV E,A
/
/LOOP2 IS PASSING THROUGH FOR EACH VALUE OF J
/COMPUTE P(I,J)=(P(I,J)-RK(I)*S(J))/RL
/
LOOP2   PUSH D                  /E↑P(I+1,1) IS SAVED FOR NEXT LOOP1
        CALL LOAD
        DCR L
        DCR L
```

```
        PUSH H                  /HL↑P(I,J)
        PUSH B
        PUSH D
        PUSH PSW
        MOV A,C
        ADD C
        ADD C
        ADI S1-3
        MOV L,A
        MVI H,BANK10            /HL↑S(J)
        CALL LOAD
        PUSH D
        MOV B,A
        LXI H,BANK11*400+RKI
        CALL LOAD
        XCHG                    /HL A =RK(I)
        POP D
        CALL FMUL               /DE B =RK(I)*S(J)
        CALL NEGDE1
        XCHG
        POP PSW
        POP D                   /THE OLD P(I,J) IS FETCHED
        MOV C,A
        MOV A,B
        MOV B,C
        CALL FADD               /P(I,J)-RK(I)*S(J)
        PUSH D
        LXI H,BANK10*400+RL
        CALL LOAD               /1/RL IS FETCHED
        XCHG                    /HL A =1/RL
        POP D
        CALL FMUL
        MOV A,B                 /DE A =THE UPDATED P(I,J)
        LXI H,BANK11*400+SLASK1
        CALL STORE              /P(I,J) IS SAVED IN SLASK1
        POP B
        POP H                   /L↑P(I,J)
        CALL STORE              /THE UPDATED P(I,J) IS SAVED IN MEMORY
        INR L                   /L↑P(I,J+1)
        POP D                   /D=3N1
        MOV A,B
        CMP C
        JZ NEWI                 /TEST IF LOOP2 READY, IF I=J
/
/SEARCHES THE SYMMETRIX ELEMENT
/P(I,J)=P(J,I)
/
        PUSH D
        PUSH H
        XRA A
        MOV E,C
OM      DCR E
        JZ UT
        ADD D
        JMP OM
UT      MOV E,A                 /E=ADDRESS TO P(J,1)
        MOV A,B
        ADD B
        ADD B
        ADD E
        SUI 3
```

```
        MOV L,A                 /HL↑P(J,I)
        PUSH H
        LXI H,BANK11*400+SLASK1
        CALL LOAD
        POP H
        CALL STORE              /THE SYMMETRIX ELEMENT IS STORED
        INR C
        POP H                   /HL↑P(I,J+1)
        POP D                   /D=3N1
        JMP LOOP2
NEWI    LDA N1
        CMP B
        RZ                      /TESTS IF B=N1
        INR B
        MVI H,BANK10
        JMP LOOP1
        .END
```

```
/       SUBROUTINE REG2
/       SHIFT THE TWO VECTORS YS AND VS ONE STEP
/       COMPUTE FSCAL=AE(1)*YS(1)+....+AE(NA)*YS(NA)-
/       -..BE(1)*U(1)-..BE(NB)*U(NB)-CE(1)*VS(1)-...-CE(NC)*VS(NC)
/       YS(1) IN THIS SAMPLE IS YS(2) FOR THE NEXT SAMPLE
/       AND FSCAL IS USED IN THE NEXT SAMPLE
/       REG2 WORKS IN BANK 10
/       AUTHOR   GEORGE KIZIROGLU  1976-03-19
/       SUBROUTINES REQUIRED
/              FMOVE
/              SCAPR
/              FADD
/               STORE
BANK10=10
AE1=176
VS1=BANK10*400+124
NA=BANK10*400+241
U1=BANK10*400+52
FSCAL=BANK10*400+254
        .GLOBL FMOVE,SCAPR,FADD,NEGDE1,STORE,REG2
/
/SHIFT YS- AND VS VECTORS ONE STEP UPWARDS IN MEMORY
/
REG2    LXI H,NA                    /ADDRESS TO NA
        MVI E,0                     /ADDRESS TO YS(1)
        CALL FMOVE
        LXI H,NA+2                  /ADDRESS TO NC
        MVI E,VS1                   /ADDRESS TO VS(1)
        CALL FMOVE
/
/COMPUTE F1=YS(2)*AE(2)+....+YS(NA)*AE(NA)
/
        LXI H,NA
        LXI D,BANK10*400+AE1
        LXI B,BANK10*400
        CALL SCAPR
        PUSH D
        PUSH B                      /DE B =F1
/
/SEARCH BE(1)
/
        LDA NA
        MOV D,A
        ADD D
        ADD D
        ADI AE1
        MOV E,A                     /E↑BE(1)
/
/COMPUTE F2=BE(1)*U(1)+...+BE(NB)*U(NB)
/
        MVI D,BANK10
        LXI B,U1
        LXI H,NA+1                  /HL↑NB
        CALL SCAPR
        LDA NA+2
        CPI 0           /TEST IF NC=0
        JZ F1
        PUSH D
        PUSH B
/
/SEARCH CE(2)
```

```
/
        LXI H,NA
        MOV A,M
        INX H
        ADD M                           /A=NA+NB
        MOV D,A
        ADD D
        ADD D
        ADI AE1
        MOV E,A                         /E+CE(2)
/
/COMPUTE F3=CE(2)*VS(2)+....CE(NC)*VS(NC)
/
        MVI D,BANK10
        LXI B,VS1
        LXI H,NA+2
        CALL SCAPR
/
/COMPUTE F3+F2
/
        MOV A,B
        POP B
        POP H
        XCHG
        CALL FADD
/
/COMPUTE F1-(F3+F2)
/
F1      CALL NEGDE1
        XCHG
        MOV A,B
        POP B
        POP D
        CALL FADD
/
/SAVE FSCAL FOR NEXT SAMPLE
/
        MOV A,B
        LXI H,FSCAL
        CALL STORE
        RET
        .END
```

```
/       SUBROUTINE SCAPR
/       SCALAR PRODUCT OF TWO VECTORS WITH ELEMENTS
/       ORGANIZED IN THREE BYTES.
/       THE ACCUMULATED SUM IS STORED IN SLASK2 IN BANK 11
/       ENTRY   DE=ADDRESS OF FIRST ELEMENT IN VECTOR 1
/               BC=ADDRESS OF FIRST ELEMENT OF VECTOR 2
/               HL=ADDRESS TO THE BYTE GIVING THE VECTORLENGTH
/       EXIT    DE=MANTISSA OF THE RESULT
/                 B=EXPONENT OF THE RESULT
/               HL=ADDRESS TO THE FIRST BYTE AFTER VECTOR 1
/       AUTHOR GEORGE KIZIROGLU 1976-03-28
/
/       SUBROUTINES REQUIRED
/             STORE
/             LOAD
/             FMUL
/             FADD
BANK11=11
SLASK2=BANK11*400+333
BANK10=10
N=BANK10*400+353
/
        .GLOBL SCAPR,FMUL,FADD,LOAD,STORE
SCAPR   MOV A,M
        MOV L,A
        ANI 1
        JZ EVEN
        PUSH B          /BC↑V2(1)
        PUSH D          /DE↑V1(1)
        JMP ODD
EVEN    PUSH D
        PUSH B
ODD     MOV A,L
        LXI D,0
        MVI B,0
        CPI 0
        JZ FINIT                /TEST IF THE VECTORLENGTH IS ZERO
NEXTEL  STA N
        LXI H,SLASK2
        MOV A,B
        CALL STORE
        POP H
        CALL LOAD               /DE A =V2(I)
        INX H           /HL↑V2(I+1)
        XTHL            /HL↑V1(I) AND ↑V2(I+1) IS STORED IN STACK
        PUSH D
        PUSH PSW
        CALL LOAD
        MOV B,A         /DE B =V1(I)
        INX H           /HL↑V1(I+1)
        POP PSW         /A=EXPONENT TO V2(I)
        XTHL            /HL=V2(I) AND ↑V1(I+1) IS STORED IN STACK
        CALL FMUL
        PUSH D
        LXI H,SLASK2
        CALL LOAD
        XCHG            /HL A =THE ACCUMULATED SUM
        POP D
        CALL FADD               /DE B =THE UPDATED SUM
        LDA N
        DCR A
```

```
        JNZ NEXTEL              /TEST IF ANY ELEMENT LEFT
FINIT   POP H
        POP H                   /HL↑V1(N+1)
        RET
        .END
```

```
/       SUBROUTINE FMOVE
/       MOVES A VECTOR ONE STEP UPWARDS IN MEMORY
/       AUTHOR  GEORGE KIZIROGLU 1976-03-19
/       ENTRY   HL=ADDRESS OF ONE OF THE VARIABLES NA,NB,NC
/                E=ADDRESS OF FIRST ELEMENT OF THE VECTOR
/       THE NUMBER OF SHIFTING ELEMENTS GIVES BY
/                  K+1+(NA OR NB OR NC)
/       SUBROUTINES REQUIRED
/              LOAD
/              STORE
K=245
        .GLOBL LOAD,STORE,FMOVE,MOVE
FMOVE   MOV A,M          /A=NA OR NB OR NC
        MVI L,K
        ADD M                / A=K+A
        MOV B,A
        ADD B
        ADD B
        INR B            /B=K+1+(NA OR NB OR NC)
        MOV L,E
        ADD L
        MOV L,A
/
/ L POINTS ON THE FIRST SHIFTING ELEMENT
/
MOVE    CALL LOAD
        INR L
        CALL STORE
        MOV A,L
        SUI 10
        MOV L,A          /L POINTS ON THE NEXT ELEMENT
        DCR B
        JNZ MOVE                 /TEST FOR MORE SHIFT
        ADI 3
        MOV L,A          /HL↑FIRST ELEMENT OF THE VECTOR
        LXI D,0          /ONLY THE MANTISSA NEEDS TO BE ZERO
        CALL STORE
        RET
        .END
```

```
/       SUBROUTINE FBIN
/       CONVERTS A NORMALISED FLOATING NUMBER TO A
/       FRACTIONAL TWO BYTES NUMBER
/
/       ENTRY    B=EXPONENT
/                DE=MANTISSA
/       EXIT    DE=THE FRACTIONAL NUMBER
/       REGISTERS AFFECTED A, B, D, E
/       AUTHOR GEORGE KIZIROGLU 1976-04-07
/
/
/       SUBROUTINES REQUIRED
/             NONE
/
FBIN    MOV A,B
        CPI 100
        RZ                          /RETURN IF EXPONENT IS ZERO
        JP OWF
        CPI 62                      /TEST IF EXPONENT > -14
        JP FBIN1
        LXI D,0                     /D E =IF SMALL NUMBER
        RET
FBIN1   XRA A
        ORA D                       /TEST OF SIGN
        JP FBIN2
        STC
FBIN2   RAR
        MOV D,A                     /SHIFT TO RIGHT
        MOV A,E
        RAR
        MOV E,A
        INR B
        MOV A,B
        CPI 100
        RZ                          /RETURN IF EXPONENT IS ZERO
        JMP FBIN1
OWF     XRA A
        ORA D
        JP OWFP
        LXI D,100000                    /D E =NEGMAX
        RET
OWFP    LXI D,077777                    /D E =POSMAX
        RET
        .END
```

```
/       SUBROUTINE FLIM
/       LIMITS THE MAGNITUDE OF U TO ULIM
/       U       SIGNAL TO BE LIMITED
/       ULIM    LIMIT OF SIGNAL
/       ENTRY   DE=THE MAGNITUDE OF U
/                  C=SIGN OF U
/                 HL=THE ADDRESS TO ULIM
/       EXIT    DE=SIGNAL WHOSE MAGNITUDE IS LESS THAN OR EQUAL WITH ULI
/       REGISTERS AFFECTED A C D E H L
/       AUTHOR GEORGE KIZIROGLU 1976-04-07
/
/       SUBROUTINES REQUIRED
/            NEGDE1 (WITHIN FADD)
/
        .GLOBL FLIM,NEGDE1
FLIM    PUSH D
        MOV E,M
        INX H
        MOV D,M                     /DE=ULIM
        POP H                       /HL=THE MAGNITUDE OF U(T)
        MOV A,D
        SUB H
        JNZ DIFF                            /JUMP IF D=/=H
        MOV A,E
        SUB L
DIFF    JM SIGN                     /JUMP IF MAGNITUDE OF U(T) > ULIM
        XCHG                        /HL=ULIM AND DE=MAGNITUDE OF U(T)
SIGN    MOV A,C
        RAR
        CC NEGDE1                           /NEGATE DE IF  C=1
        RET
        .END
```

```
/       UTILITY SUBROUTINES
/       IMPLEMENTER   GEORGE KIZIROGLU
/       SUBROUTINE NEGDE
/       NEGATES THE INPUT NUMBER
/       ENTRY   DE=INPUT MANTISSA
/                 C=NEGATIONCOUNTER
/       EXIT    DE=THE NEGATED NUMBER
/                 C=INCREMENTED NEGATIONCOUNTER
/
        .GLOBL NEGDE,NEGDE1,OUFLW,OFLW,STSGN,UFLW,UFLWE,LOAD
        .GLOBL FMAGN,STORE,DISP
NEGDE   INR C
NEGDE1  XRA A
        SUB E
        MOV E,A
        MVI A,0
        SBB D
        MOV D,A
        RET
/
/       SUBROUTINESS OUFLW,UFLW,UFLWE,STSGN
/
OUFLW   XRA A
        ADD C
        JM UFLW
OFLW    LXI H,77777
        MVI B,177
STSGN   MOV A,C
        RAR
        XCHG
        CC NEGDE1
        RET
UFLW    LXI D,0
UFLWE   MVI B,0
        RET
/       SUBROUTINE FMAGN
/       TAKES THE MAGNITUDE OF THE INPUT NUMBER
/       ENTRY   DE=THE MANTISSA
/                 B=THE EXPONENT
/       EXIT     C=THE SIGN OF THE NUMBER
/                DE=THE MAGNITUDE OF THE NUMBER
/                 B=THE EXPONENT IS UNCHANGED
/       AUTHOR     GEORGE KIZIROGLU     07-04-76
/
/       SUBROUTINES REQUIRED
/            NEGDE  (WITHIN FADD)
/
FMAGN   MVI C,0                 /THE NEGATION COUNTER=0
        XRA A
        ORA D
        JM NEGDE                /JUMP IF NEGATIVE
        RET
/       SUBROUTINES LOAD, STORE
/       LOADS OR STORES A D E
/            A=EXPONENT
/               D=MOST SIGNIFICANT HALF OF MANTISSA
/               E =LEAST SIGNIFICANT HALF OF MANTISSA
/       AUTHOR  GEORGE KIZIROGLU 1976-03-19
/       ENTRY=ADDRESS OF FIRST WORD IN H,L
/       NO FLAGS ARE AFFECTED
/       SUBROUTINES REQUIRED
```

```
/               NONE
LOAD    MOV A,M
        INR L
        MOV E,M
        INR L
        MOV D,M
        RET
STORE   MOV M,A
        INR L
        MOV M,E
        INR L
        MOV M,D
        RET
/       SUBROUTINE BINF
/       CONVERTS A FRACTIONAL TWO BYTES NUMBER TO
/       A NORMALISED NUMBER
/       ENTRY:  DE=THE FRACTIONAL NUMBER
/       EXIT:   DE=MANTISSA
/                B=EXPONENT
/       AUTHOR   GEORGE KIZIROGLU  1976-04-07
/
/       SUBROUTINES REQUIRED
/          FADD
/
BINF    MVI B,100          /EXPONENT=0
        MVI C,0            /NEGATION COUNTER=0
        MOV H,D            /FOR NEGMAX
        JMP FNORMO
        .END
```

```
/       SUBROUTINE FADD
/       IMPLEMENTER   GEORGE KIZIROGLU
        .GLOBL NEGDE,OFLW,UFLWE,STSGN,UFLW
        .GLOBL FADD
FADD    MOV C,A
        MOV A,B
        SUB C
        JZ FADD3
        JP FADD1
        CMA
        INR A           /FOR A CORRECT TEST OF SHIFT>16
        XCHG
        MOV B,C
FADD1   CPI 20
        RP
        MOV C,A
FADD2   MOV A,H
        RLC
        RAR
        RAR
        MOV H,A
        MOV A,L
        RAR
        MOV L,A
        DCR C
        JNZ FADD2
        RAR
        ORA H
        XRA H
        JP FADD3
        INX H
FADD3   MOV C,H
        DAD D
        SBB A
        XRA C
        XRA D
        XRA H
        MVI C,0
        XCHG
        JM FADD4
FNORMO  XRA A
        ORA D
        CM NEGDE
        JP FNORM
FADD4   MOV A,H
        RLC
        MOV C,A
        INR B
        JM OFLW
        MOV A,D
        RAR
        MOV D,A
        MOV A,E
        RAR
        MOV E,A
        RAR
        ORA D
        XRA D
        RP
        INX D
        RET
```

```
FNORM   ORA E
        JZ UFLWE
        XCHG
FNORM1  MOV A,H
        ADD H
        JM STSGN
        DCR B
        JM UFLW
        DAD H
        JMP FNORM1
        .END
```