

BALLISTIC MISSILE  
DEFENSE ORGANIZATION  
7100 Defense Pentagon  
Washington, D.C. 20301-7100

**PARALLEL FUNCTION PROCESSOR  
RELIABILITY TEST**

SPECIAL TECHNICAL REPORT  
REPORT NO. STR-0142-90-004

January 23, 1990

---

**GUIDANCE, NAVIGATION AND CONTROL  
DIGITAL EMULATION TECHNOLOGY LABORATORY**

Contract No. DASG60-89-C-0142

Sponsored By

The United States Army Strategic Defense Command

---

**COMPUTER ENGINEERING RESEARCH LABORATORY**

Georgia Institute of Technology

Atlanta, Georgia 30332 - 0540

---

Contract Data Requirements List Item A004

Period Covered: Not Applicable

Type Report: As Required

20010829 012

**DISTRIBUTION STATEMENT A**  
Approved for Public Release  
Distribution Unlimited

UL10301



## DISCLAIMER

DISCLAIMER STATEMENT - The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

## DISTRIBUTION CONTROL

- (1) DISTRIBUTION STATEMENT - Approved for public release; distribution is unlimited.
- (2) This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227 - 7013, October 1988.

**PARALLEL FUNCTION PROCESSOR  
RELIABILITY TEST**

January 23, 1990

---

Authors

Michael B. Woods, Richard M. Pitts, James N. Hardage

**COMPUTER ENGINEERING RESEARCH LABORATORY**

Georgia Institute of Technology

Atlanta, Georgia 30332-0540

---

Eugene L. Sanders

USASDC

Contract Monitor

Cecil O. Alford

Georgia Tech

Project Director

---

Copyright 1990

Georgia Tech Research Corporation

Centennial Research Building

Atlanta, Georgia 30332

UL10301

## Table of Contents

1.	<b>Introduction</b>	1
2.	<b>T3 Test Results</b>	1
3.	<b>FPPMU Test Results</b>	1
4.	<b>T2 Test Results</b>	2
5.	<b>FUNCTION Test Results</b>	2
6.	<b>Error Summary Per Processor</b>	4
7.	<b>Temperature Data</b>	5
8.	<b>Interpretation of Results/Recommendations</b>	5
9.	<b>Source Code</b>	5
10.	<b>Appendices</b>	6
	10.1 Appendix A Temperature Profiles	7
	10.2 Appendix B Test Programs	8

## 1. INTRODUCTION

A week long test was run on the Parallel Function Processor (PFP) Monday June 12, 1989 through Sunday June 18, 1989. Temperature data was collected over a 103 hour period from Tuesday June 13, 1989 through Saturday June 17, 1989 at 32 points on the system. Plots of 31 of these points are included in this report, the 11th thermocouple did not work correctly, giving all negative temperatures.

The system was run with 2 full crossbars, two sequencers, one array interconnect link (i.e., one board on each crossbar) and 45 processors. The remaining 17 processor slots were empty. The right processor bank was fully populated (31 processors, one array interconnect) with the remaining boards (14 processors, 1 array interconnect) on the left side.

Four diagnostic tests, T3, FPPMU, T2, and FUNCTION were run in a continuous loop for the whole week. Results are given in two forms, per test and per processor.

## 2. T3 TEST RESULTS

The T3 test is the comprehensive crossbar test. In it, one processor broadcasts a number over both crossbars (using one array interconnect link) to all other processors in the system. The host then reads the received number from each processor to verify the data that was received. Four 32 bit patterns (00000000, 55555555, AAAAAAAA, FFFFFFFF) are sent by a designated processor. Each processor becomes the sender every 45th cycle.

The total number of transfers was:

$$(45 \text{ processors})(44 \text{ xfers/subcycle})(4 \text{ subcycles/cycle}) \\ (840 \text{ cycles})(11 \text{ loops}) = \underline{73180800 \text{ 32 bit xfers}}$$

The total number of bit errors was:

1 bit

The total bit error rate was:

$$1/(32 \text{ bits/transfer})(73180800) \text{ or } 1/2,341,785,600$$

$$= \mathbf{4.27 \times 10^{-10} \text{ (error bits/transfer bits)}}$$

### 3. FPPMU TEST RESULTS

The FPPMU test is the comprehensive test of the multibus repeater system. The host sends alternating patterns of 00000000, 55555555, AAAAAAAAAA, FFFFFFFF to each processor across the multibus and then reads the values back.

The total number of 32 bit transfers was:

$$(4800 \text{ xfers/cycle})(11 \text{ cycles}) + 15,200 \text{ xfers} = \underline{68000 \text{ transfers}}$$

The total number of bits transferred was:

$$(32)(68000) = \underline{2,176,000 \text{ bits}}$$

The total number of bits in error was:

0 bits

Thus the error rate was:

$$\underline{0.00 \text{ error bits/transfer bits}}$$

### 4. T2 TEST RESULTS

The T2 test was primarily written to aid in finding faulty processor/sequencer handshake cables. One processor acts as a central node sending alternating patterns of 00000000, FFFFFFFF to each processor individually. (Thus, each sequencer cable is checked individually.)

The total number of 32 bit transfers was:

234,000 transfers

The number of words in error was:

21 words

In addition there were two "hang up" errors for a total of:

23 errors

The error rate was:

$$\underline{23/234,000 \text{ or } 9.829 \times 10^{-5} \text{ Transfer Errors/Transfers}}$$

NOTE: 20 of the 23 errors occurred in one bunch. See the error summary table for details.

## 5. FUNCTION TEST RESULTS:

The function test consists of testing all functions that are hardware assisted on the processor by computing them remotely at each processor, as well as locally at the host, and then comparing the results. The following functions were computed at each processor on each cycle:

- 200 sine values
- 180 cosine values
- 180 tangent values
- 200 arcsine values
- 200 arccosine values
- 200 arctangent values
- 64 reciprocal values
- 102 exponential values
- 100 square root values
- 30 natural log values

This is a total of 1456 computed values per processor per cycle.

The total number of computed values was:

$$(250 \text{ iterations})(10 \text{ cycles})(45 \text{ processors})(1456) = \underline{163,800,000}$$

The total number of errors was:

28 values

This gives an error rate of:

$$\frac{28}{163,800,000} = 1.709 \times 10^{-7} \text{ Function errors/Function Computations}$$

NOTE: All errors occurred in two bunches early in the test and then went away.

## 6. ERROR SUMMARY

Table 6.1 gives a summary of all errors by test and by processor. Two errors are not tabulated in Table 6.1. These occurred during the T2 test. These errors "hung up" the system and did not list to the printer. The system was reset after each of these errors and testing continued.

Table 6.1 ERROR SUMMARY PER PROCESSOR

ERRORS	PROCESSOR	FPPMU	T2	T3	FUNCTION	TOTAL
1	2050-15	0	0	0	0	0
2	2050-14	0	0	0	0	0
3	2050-13	0	0	0	0	0
4	2050-12	0	0	0	0	0
5	2050-11	0	0	0	24	24
6	2050-10	0	0	0	0	0
7	2050-09	0	0	0	0	0
8	2052-15	0	0	0	0	0
9	2052-14	0	0	0	0	0
10	2052-13	0	0	0	0	0
11	2052-12	0	0	0	0	0
12	2052-11	0	0	0	0	0
13	2052-10	0	0	0	0	0
14	2052-09	0	0	0	0	0
15	2052-08	0	0	0	0	0
16	2056-15	0	0	0	0	0
17	2056-14	0	0	0	0	0
18	2056-13	0	0	0	0	0
19	2056-12	0	0	0	0	0
20	2056-11	0	0	0	0	0
21	2056-10	0	20	0	4	24
22	2056-09	0	0	0	0	0
23	2056-08	0	0	1	0	1
24	2056-07	0	0	0	0	0
25	2056-06	0	0	0	0	0
26	2056-05	0	0	0	0	0
27	2058-15	0	1	0	0	1
28	2058-14	0	0	0	0	0
29	2058-13	0	0	0	0	0
30	2058-12	0	0	0	0	0
31	2058-11	0	0	0	0	0
32	2058-10	0	0	0	0	0
33	2058-09	0	0	0	0	0
34	2058-08	0	0	0	0	0
35	2058-07	0	0	0	0	0
36	2058-06	0	0	0	0	0
37	2058-05	0	0	0	0	0
38	2060-14	0	0	0	0	0
39	2060-13	0	0	0	0	0
40	2060-12	0	0	0	0	0
41	2060-11	0	0	0	0	0
42	2060-10	0	0	0	0	0
43	2060-09	0	0	0	0	0
44	2060-08	0	0	0	0	0
45	2060-06	0	0	0	0	0
Totals		0	21	1	28	50

## **7. TEMPERATURE DATA**

Plots of all temperature measurements are given in Appendix A. As expected, the hottest point in the system is located on the floating point processor underneath the piggyback memory board. Differences in the temperature plots for the 5 thermocouples glued under memory boards are primarily due to the glue job holding them on the board. The thermocouples that appear coolest were insulated from the chip surface by glue while the warmer ones were in direct contact with the chip surface.

## **8. INTERPRETATION OF RESULTS/RECOMMENDATIONS**

Forty-eight of the 52 errors were located on two processors. One of these (the processor from slot 2056-10) now has a hard failure, is no longer loadable, and has been removed from the system. These errors were all local processor and/or function board errors that do not reflect any system wide problem.

The two times the system hung up could be attributable to several problems. The two most likely candidates are 1) the two marginal processors hung something up, or 2) a power glitch caused by one of several thunderstorms which occurred during the week.

The one bit dropped during a T3 test looks like an honest error, and possibly one bit dropped by processor P13 during a T2 test may be an honest error.

The temperature data looks acceptable "as is" with the possible exception of the chips on the FPP board located under the piggyback memory board. The low 60 degree C range readings are getting very close to the 70 degree C rating on these chips. As previously discussed, I recommend we use military grade chips in this area which are rated at 125 degrees C.

Also, the location of the card cage in the cabinet (high, low, or middle) does not seem to affect the temperature of the processor board too much. The boards in the lower card cages seem to be running just as warm as the ones in the upper card cages.

Thermocouples #20 and #28 do not appear to have the same variance as the others. I have no explanation for this other than they may have experienced more or less noise (maybe some kind of direct metal to metal contact) than the other signals due to the way they were routed.

Thermocouples 14 and 16 are labeled correctly but we may have placed them in the wrong place. Thermocouple 16 follows thermocouple 13 very closely, leading us to believe we placed #16 above the top left chassis instead of the bottom left chassis. Thermocouple 14 looks like room temperature, which could be correct since the bottom left card cage it was located near only had a multibus repeater and an array interconnect in it.

## **9. SOURCE CODE**

Source code for all test programs is given in Appendix B.

## 10. APPENDIX

## 10.1 Appendix A Temperature Profiles

UNDER MEMORY BOARD ON PROCESSOR P31

TC #:

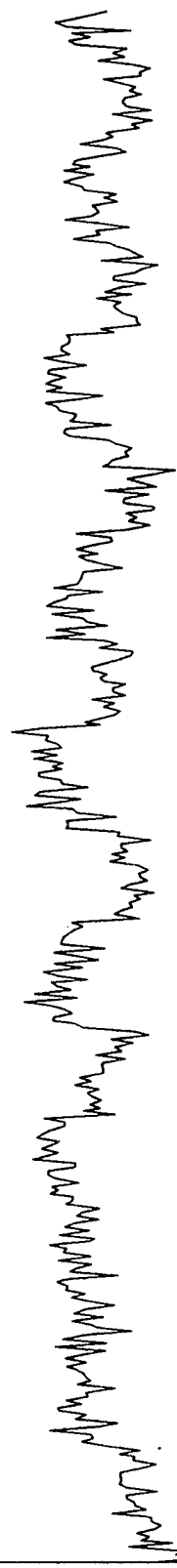
60

TEMPERATURE DEGREES C

40

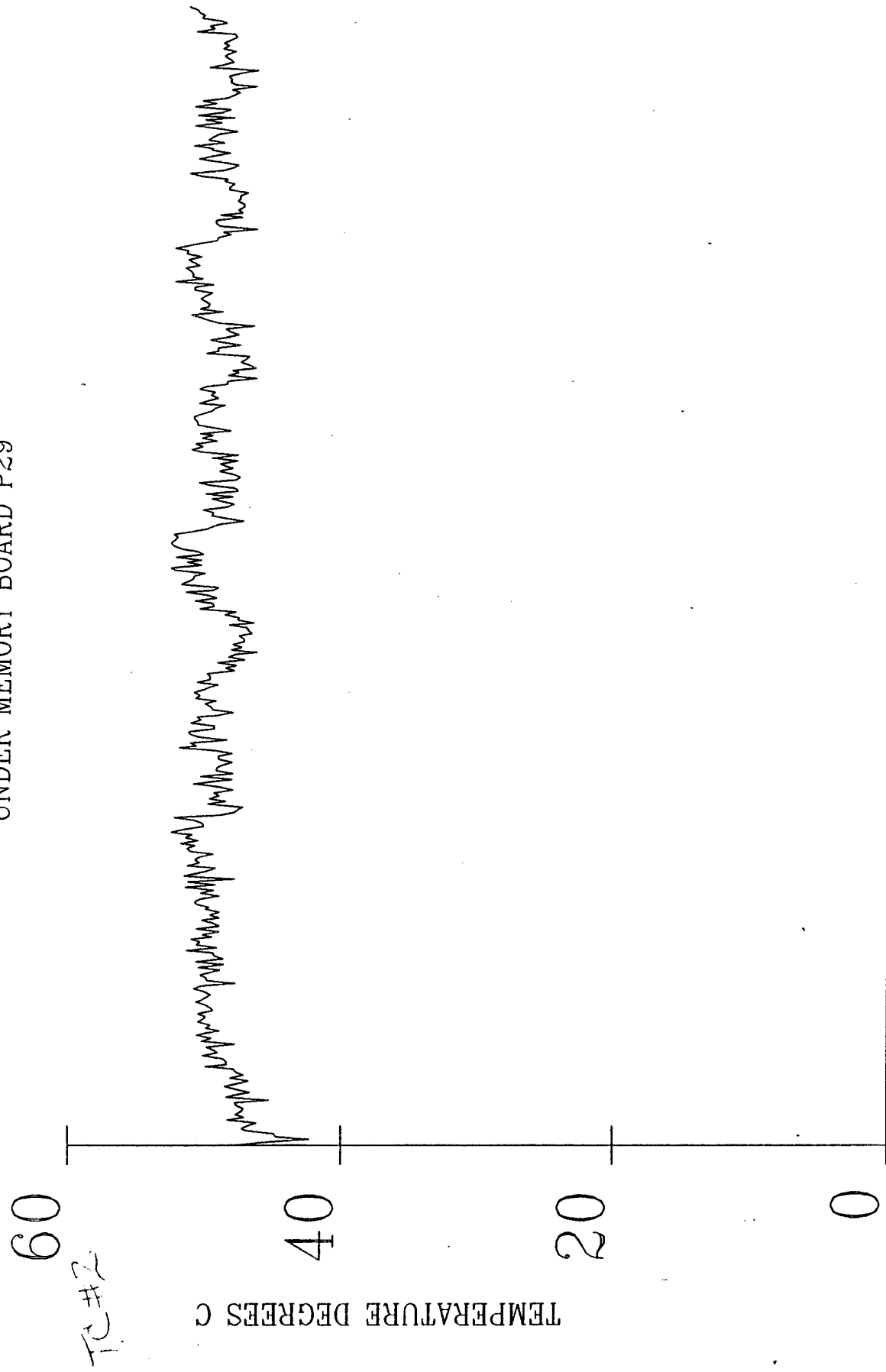
20

0



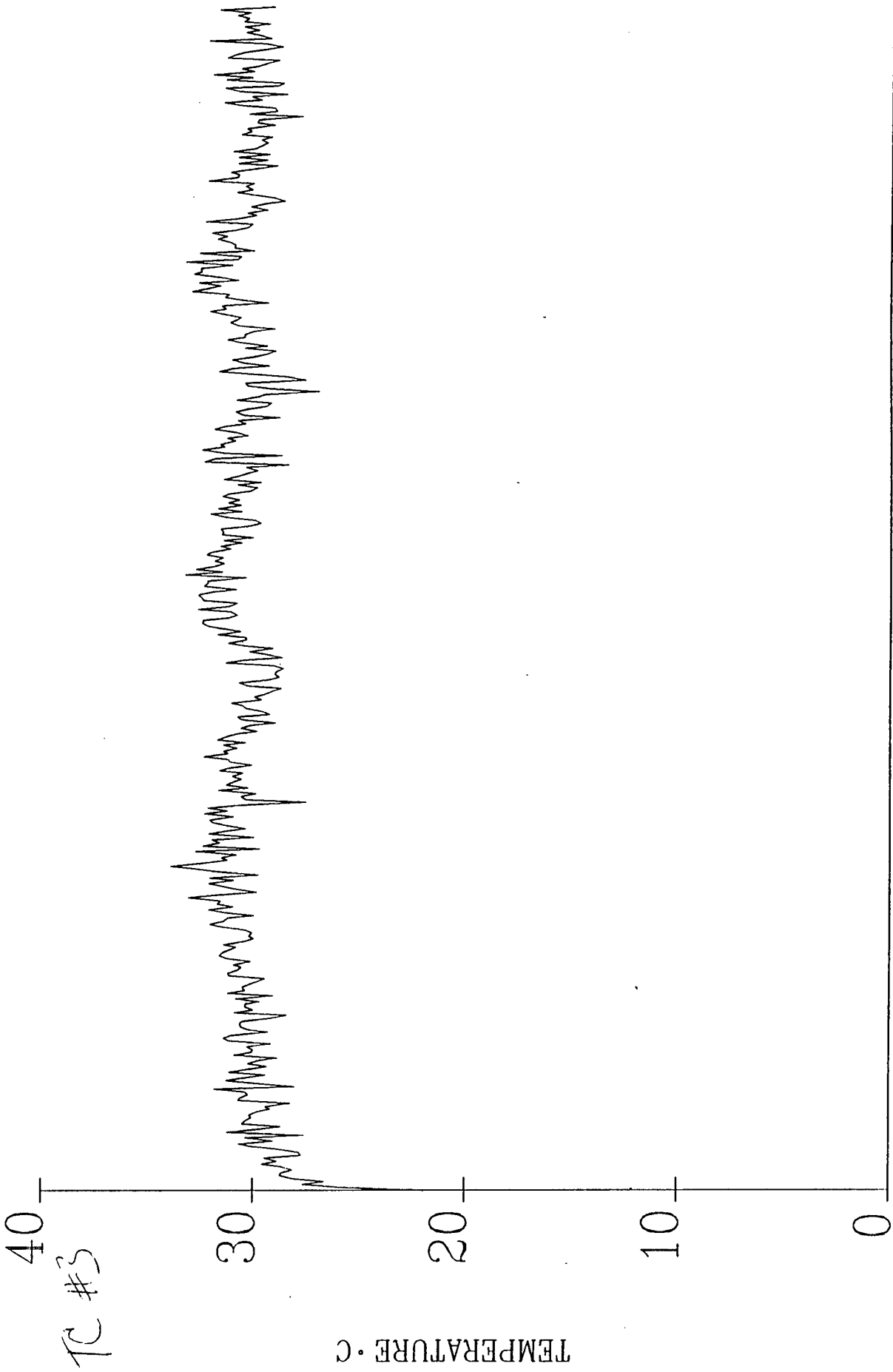
RUNTIME = 103 HOURS

UNDER MEMORY BOARD P29



RUNTIME = 103 HOURS

LOOSE IN LOWER LEFT CARD CAGE



RUNTIME = 103 HOURS

P52 UNDER MEMORY BOARD

80

60

40

20

0

TEMPERATURE · C

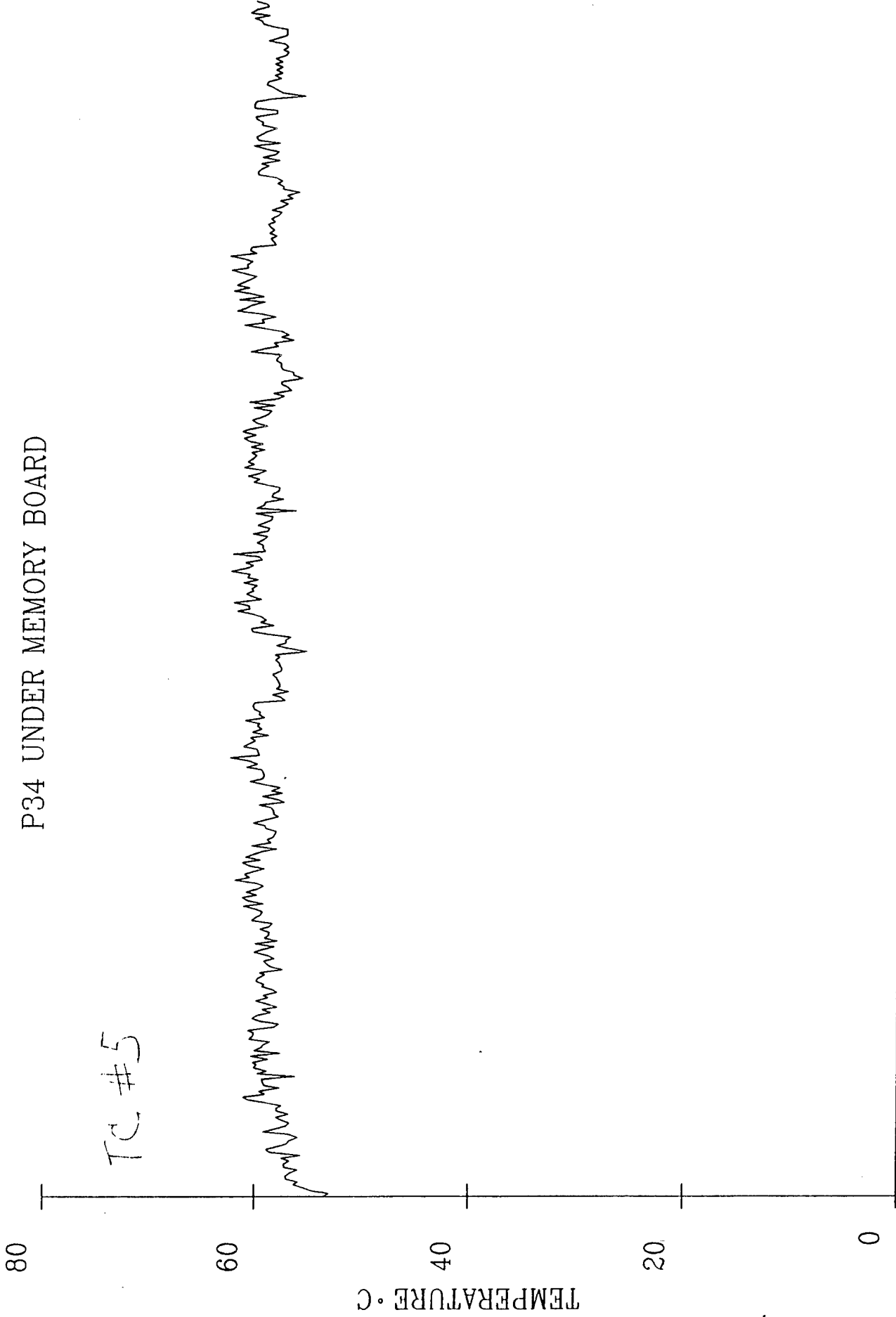
TC #4



RUNTIME = 103 HOURS

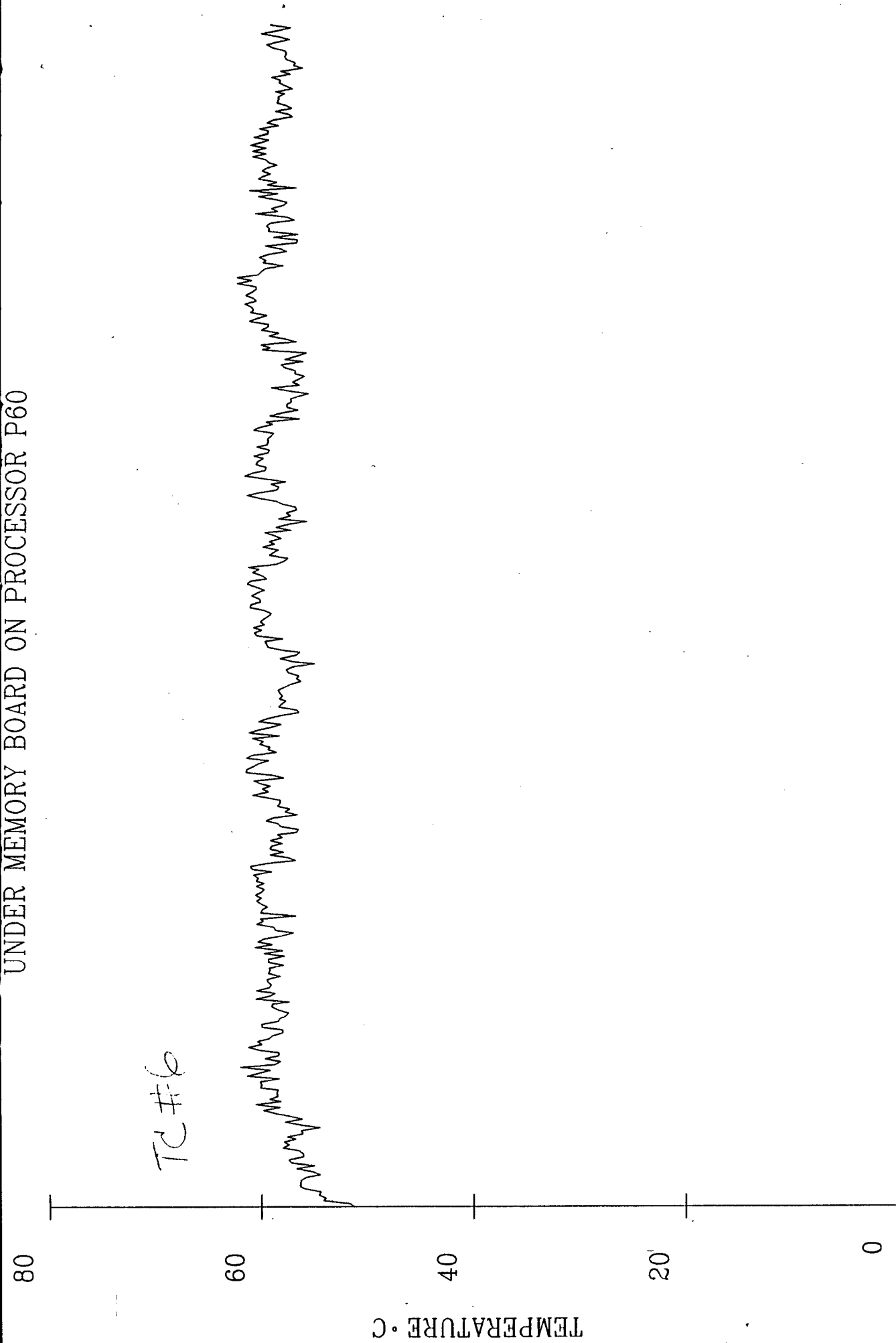
P34 UNDER MEMORY BOARD

TC #5



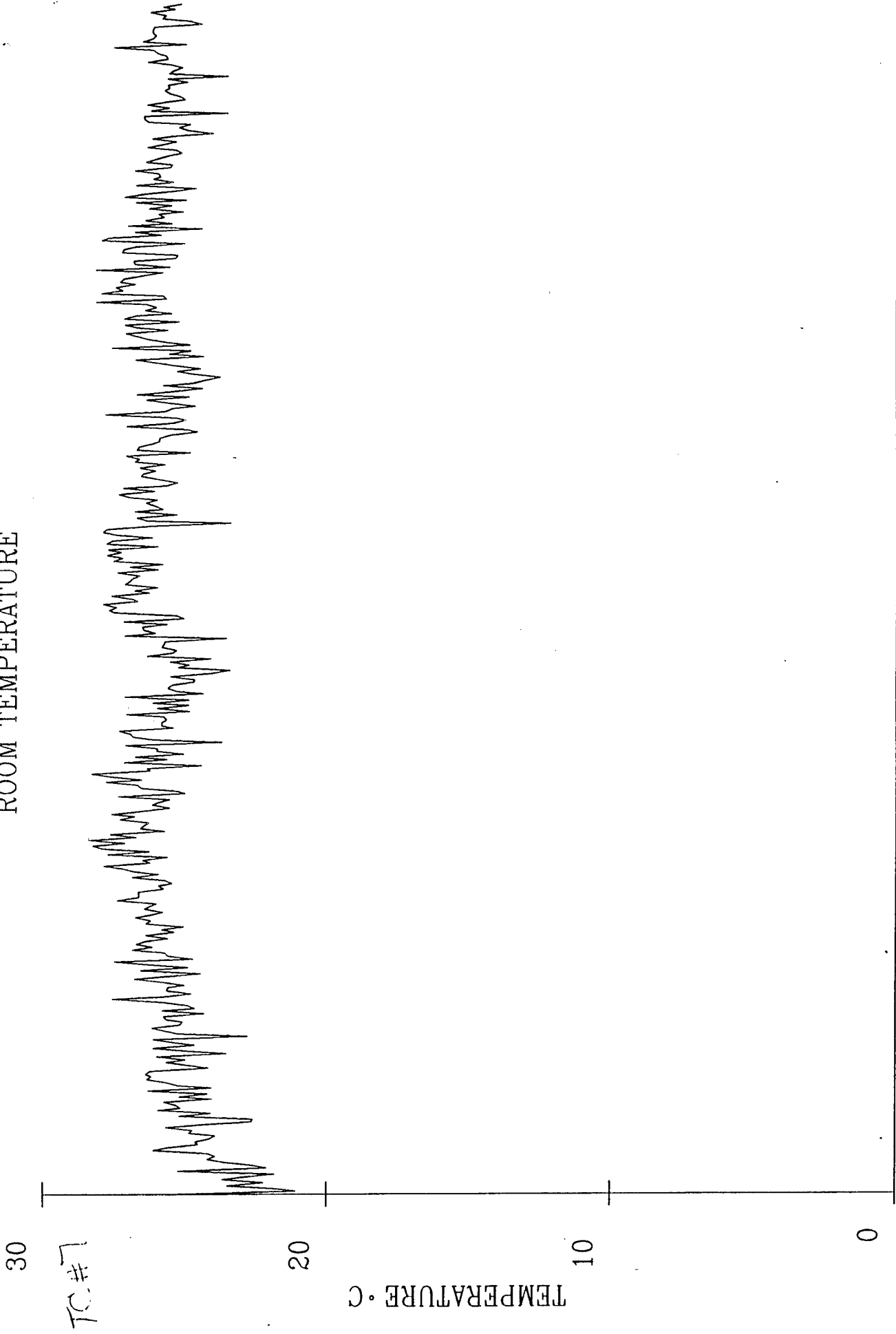
RUNTIME = 103 HOURS

UNDER MEMORY BOARD ON PROCESSOR P60



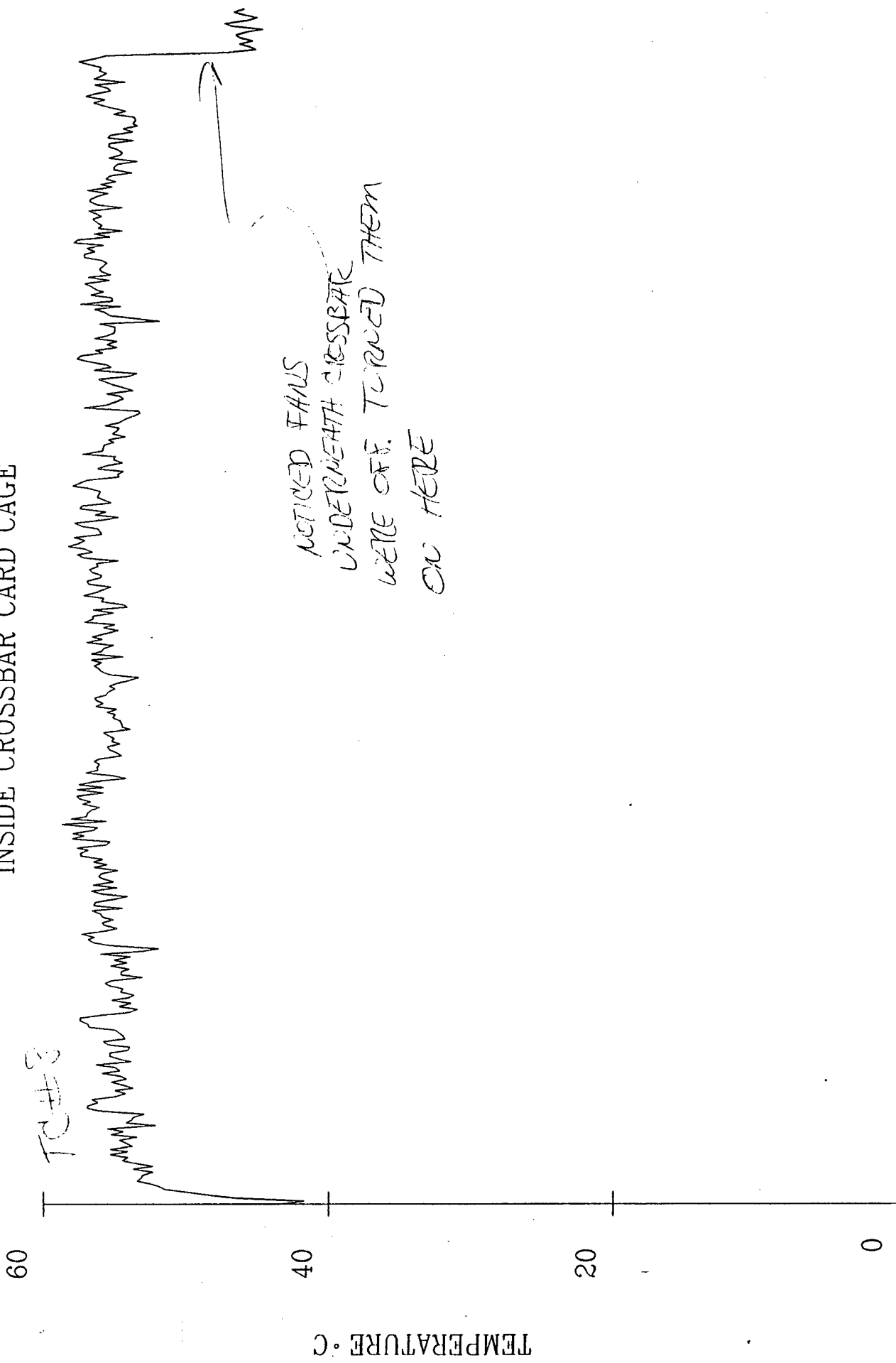
RUNTIME = 103 HOURS

ROOM TEMPERATURE



RUNTIME = 103 HOURS

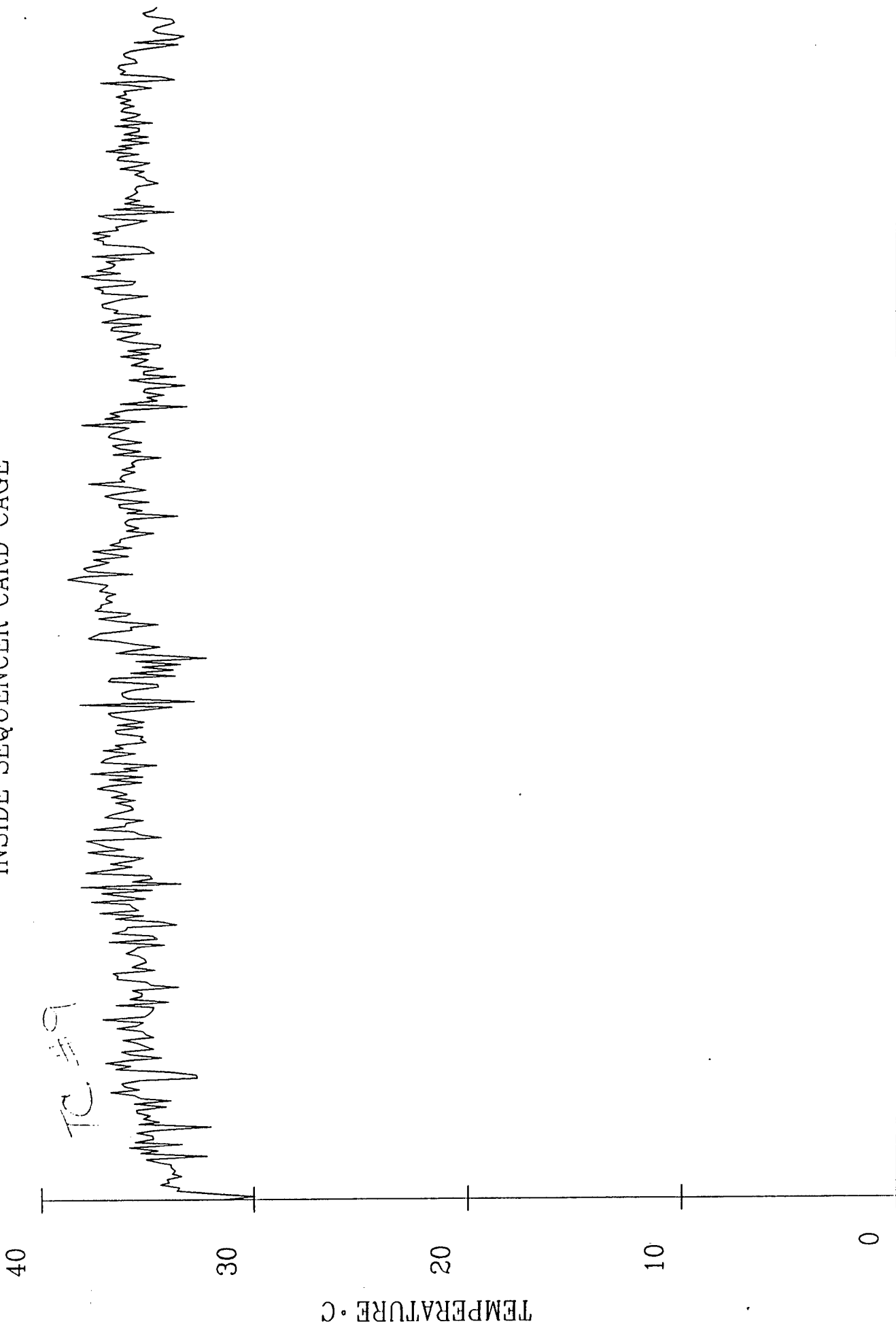
INSIDE CROSSBAR CARD CAGE



NOTICED FANS  
UNDERNEATH CROSSBAR  
WERE OFF. TURNED THEM  
ON HERE

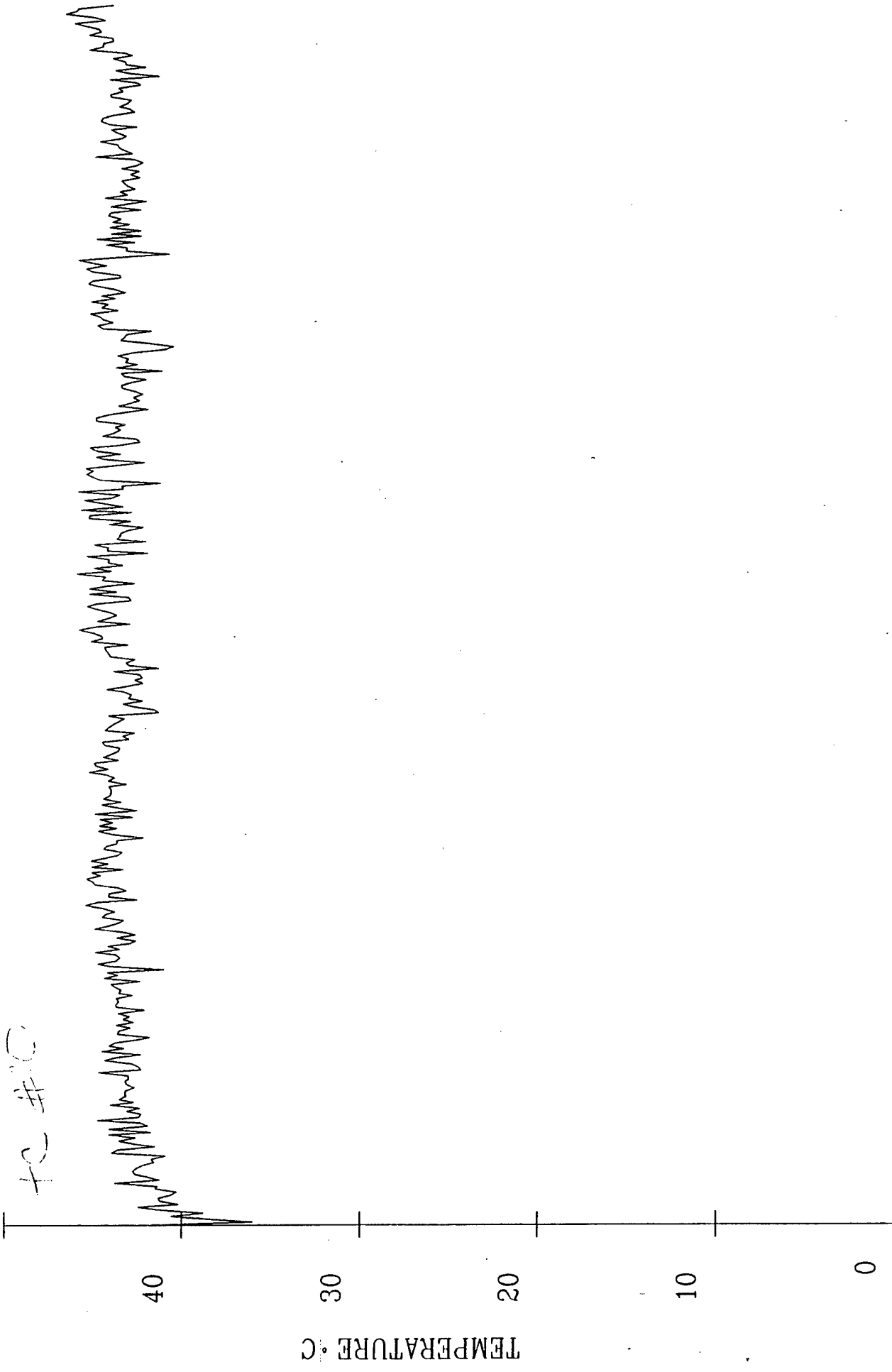
RUNTIME = 103 HOURS

INSIDE SEQUENCER CARD CAGE



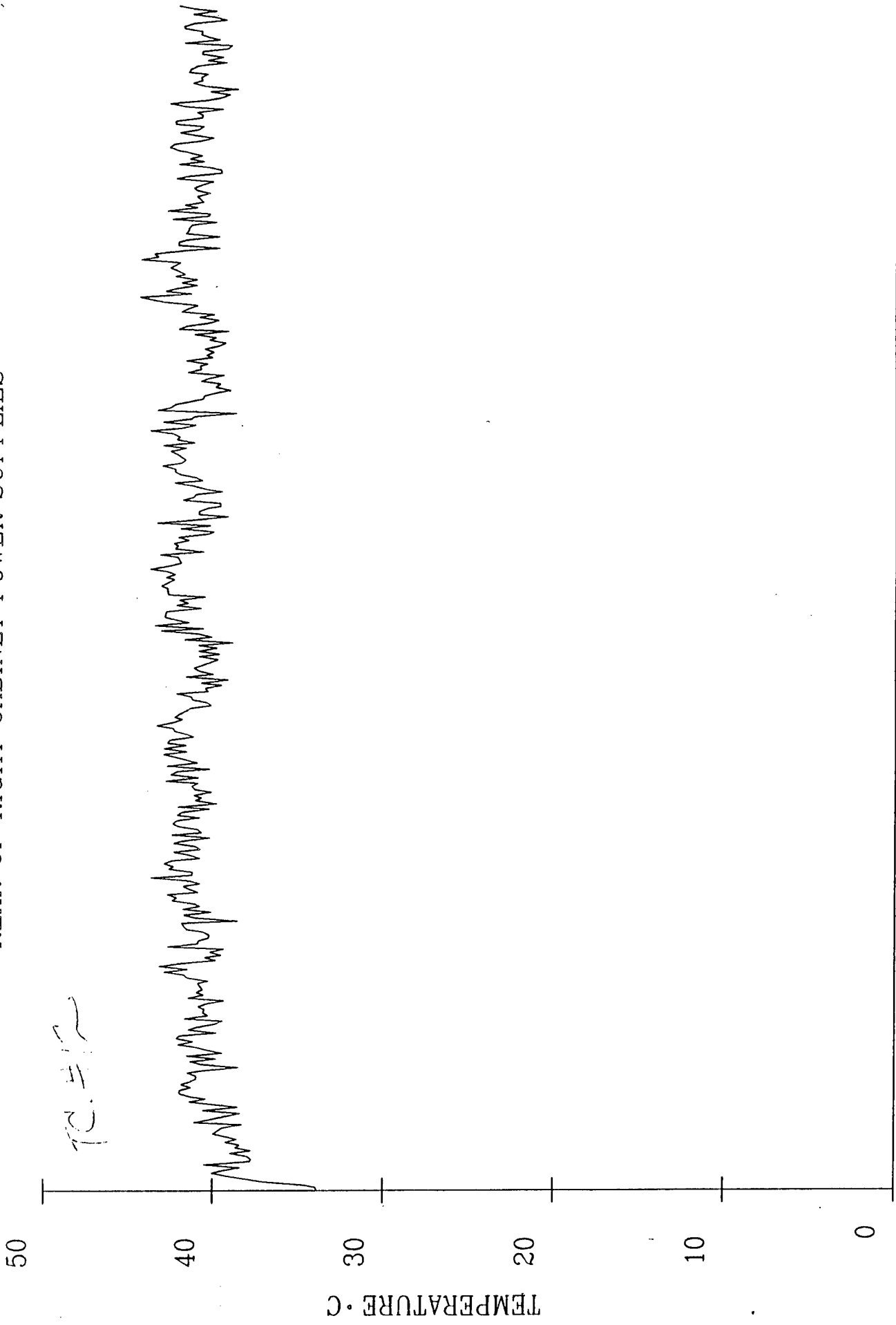
RUNTIME = 103 HOURS

ON CROSSBAR STATUS DISPLAY



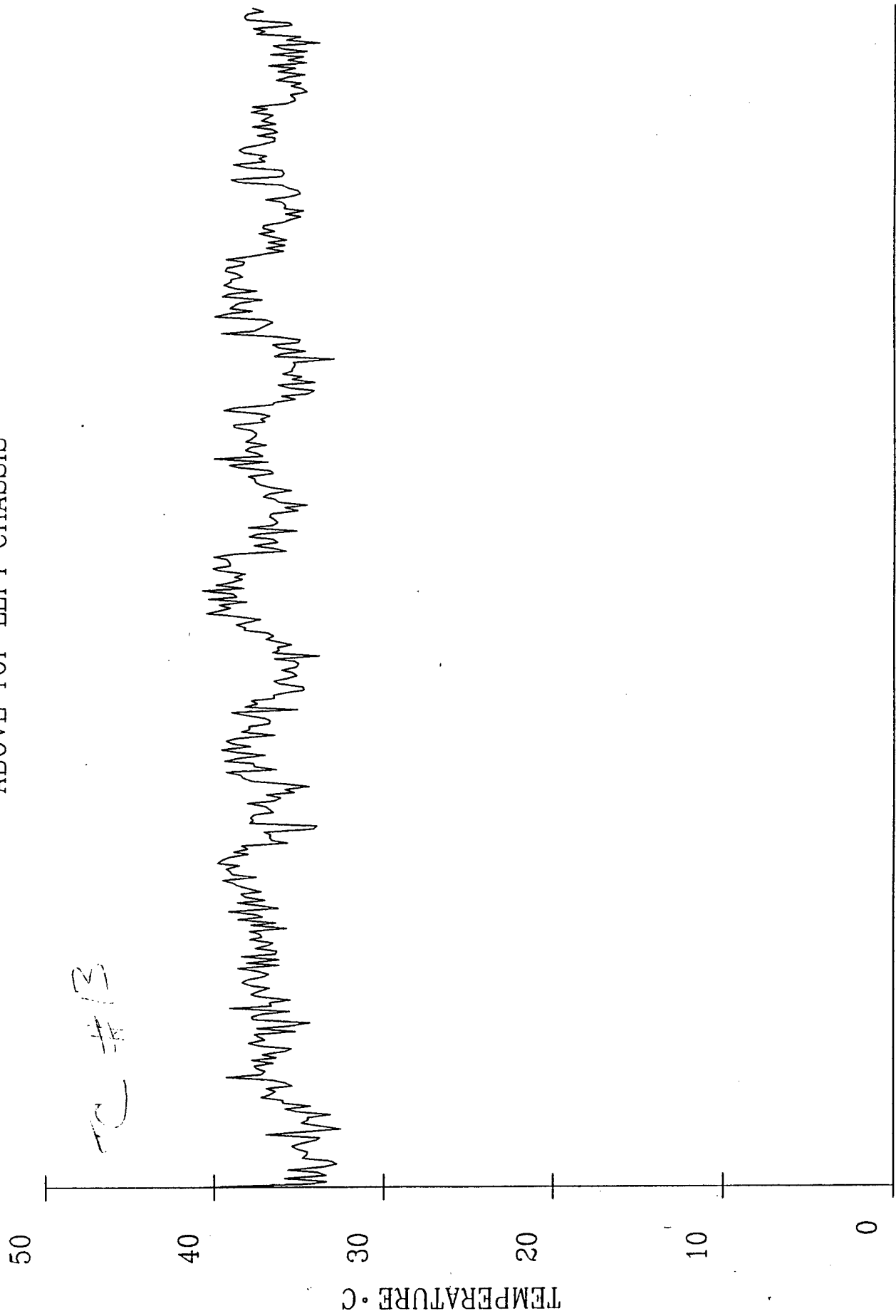
RUNTIME = 103 HOURS

REAR OF RIGHT CABINET POWER SUPPLIES



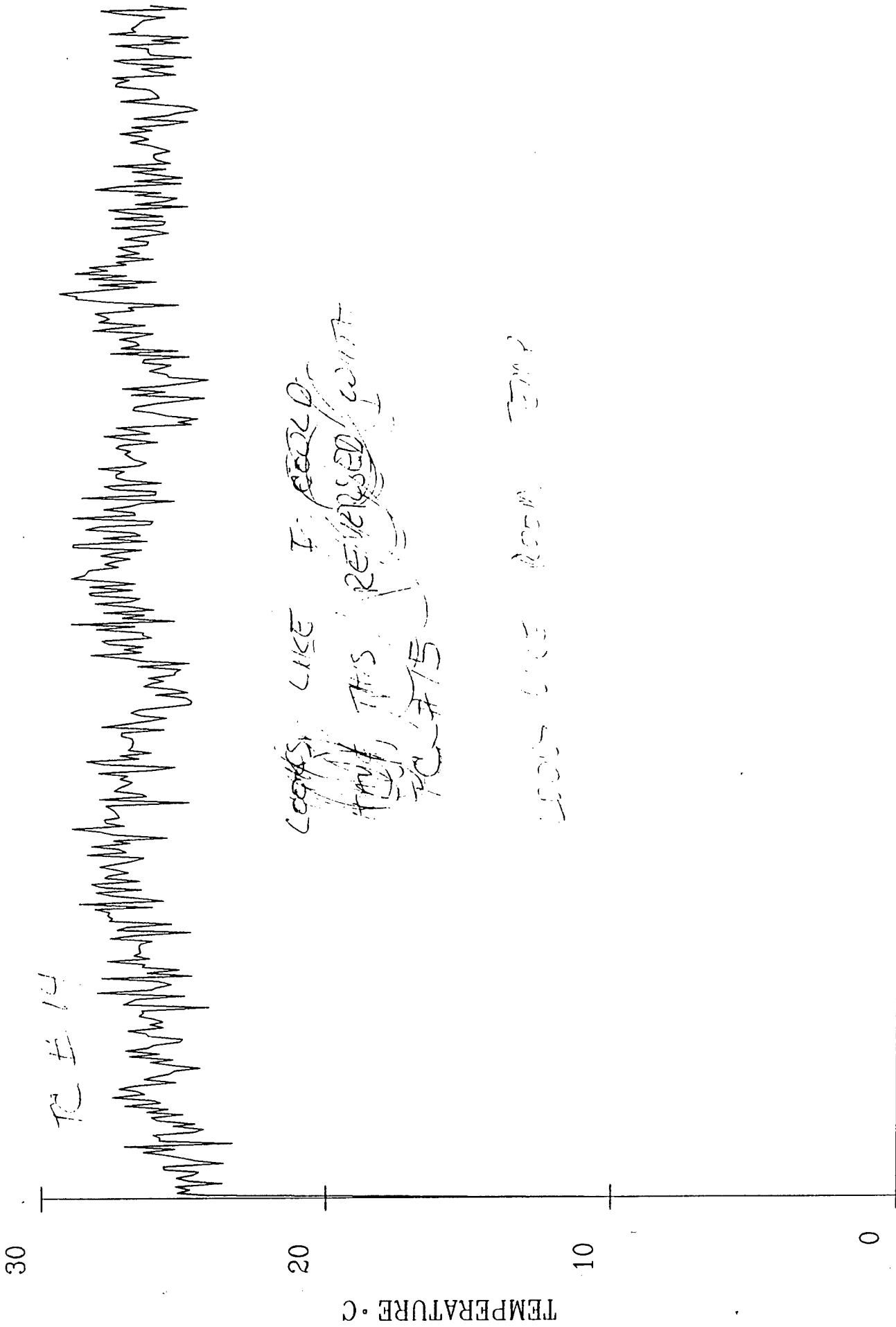
RUNTIME = 103 HOURS

ABOVE TOP LEFT CHASSIS



RUNTIME = 103 HOURS

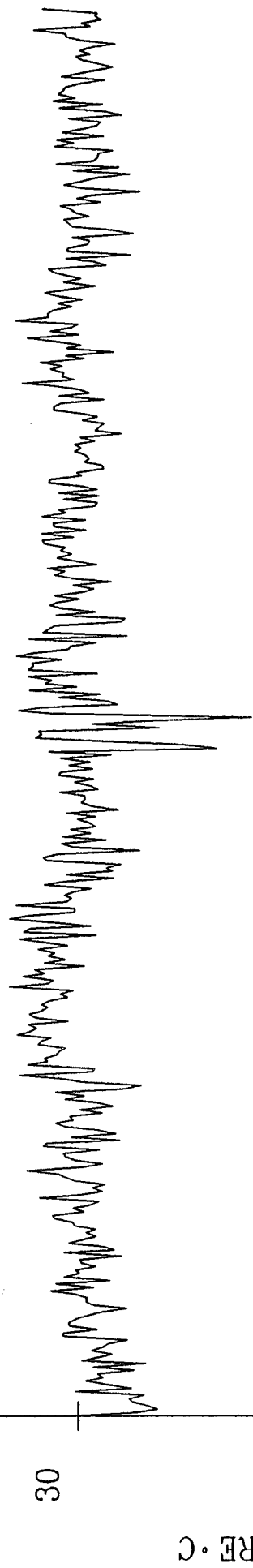
ABOVE MIDDLE LEFT CHASSIS



RUNTIME = 103 HOURS

ABOVE LOWER LEFT CHASSIS

TC #5

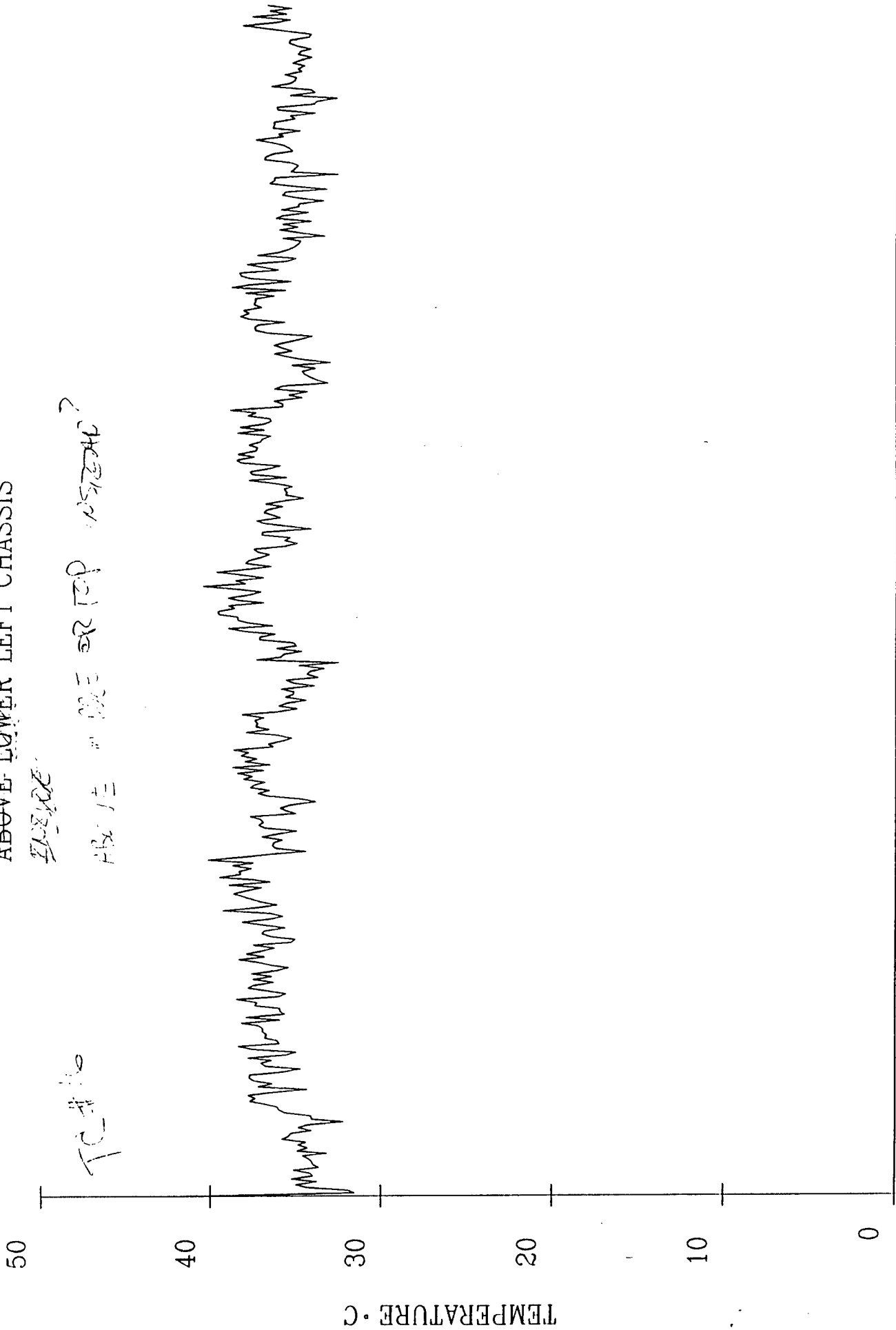


RUNTIME = 103 HOURS

ABOVE LOWER LEFT CHASSIS

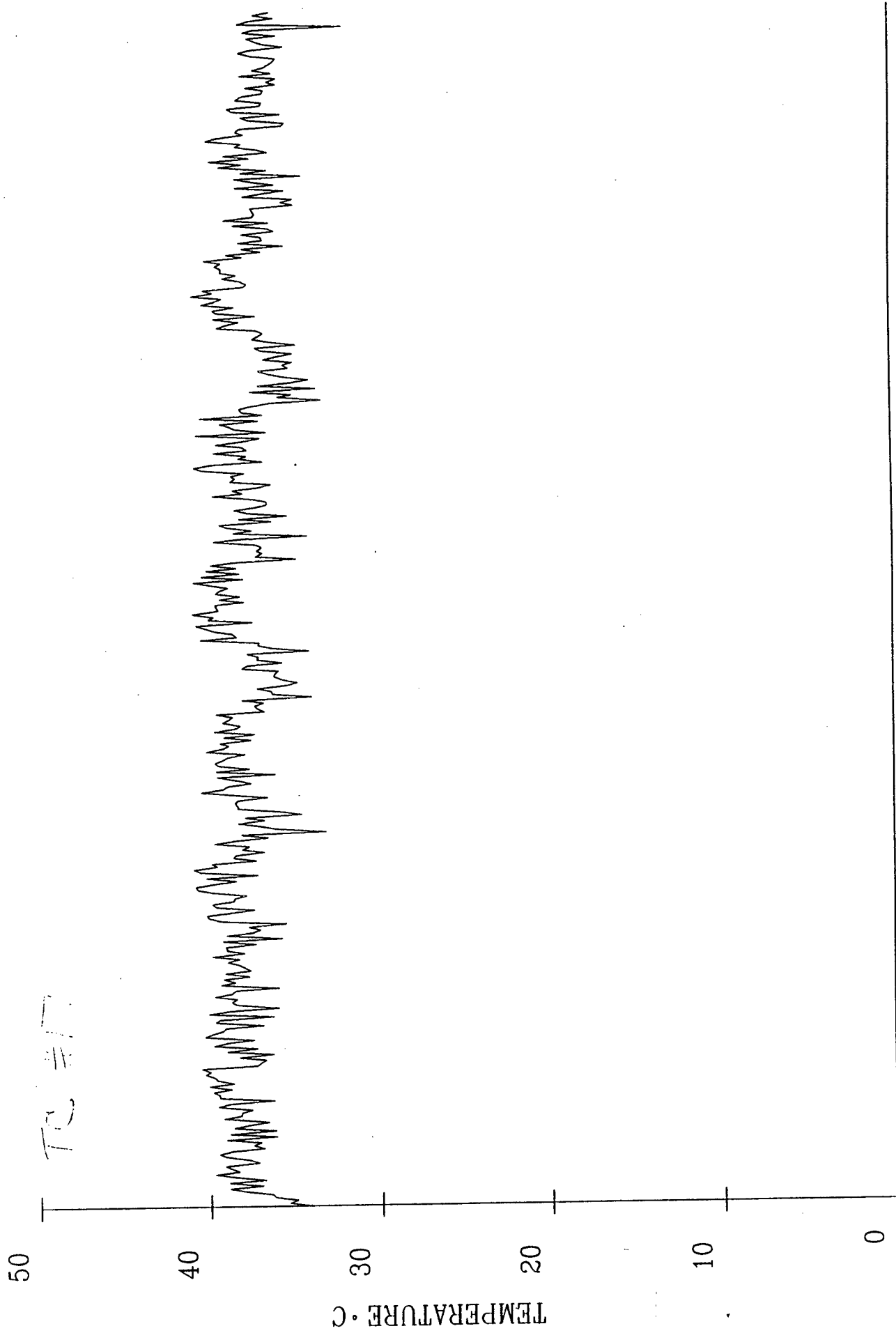
*THREE*

*TC #10*  
*ABOVE WIRE OR TOP WIRE?*

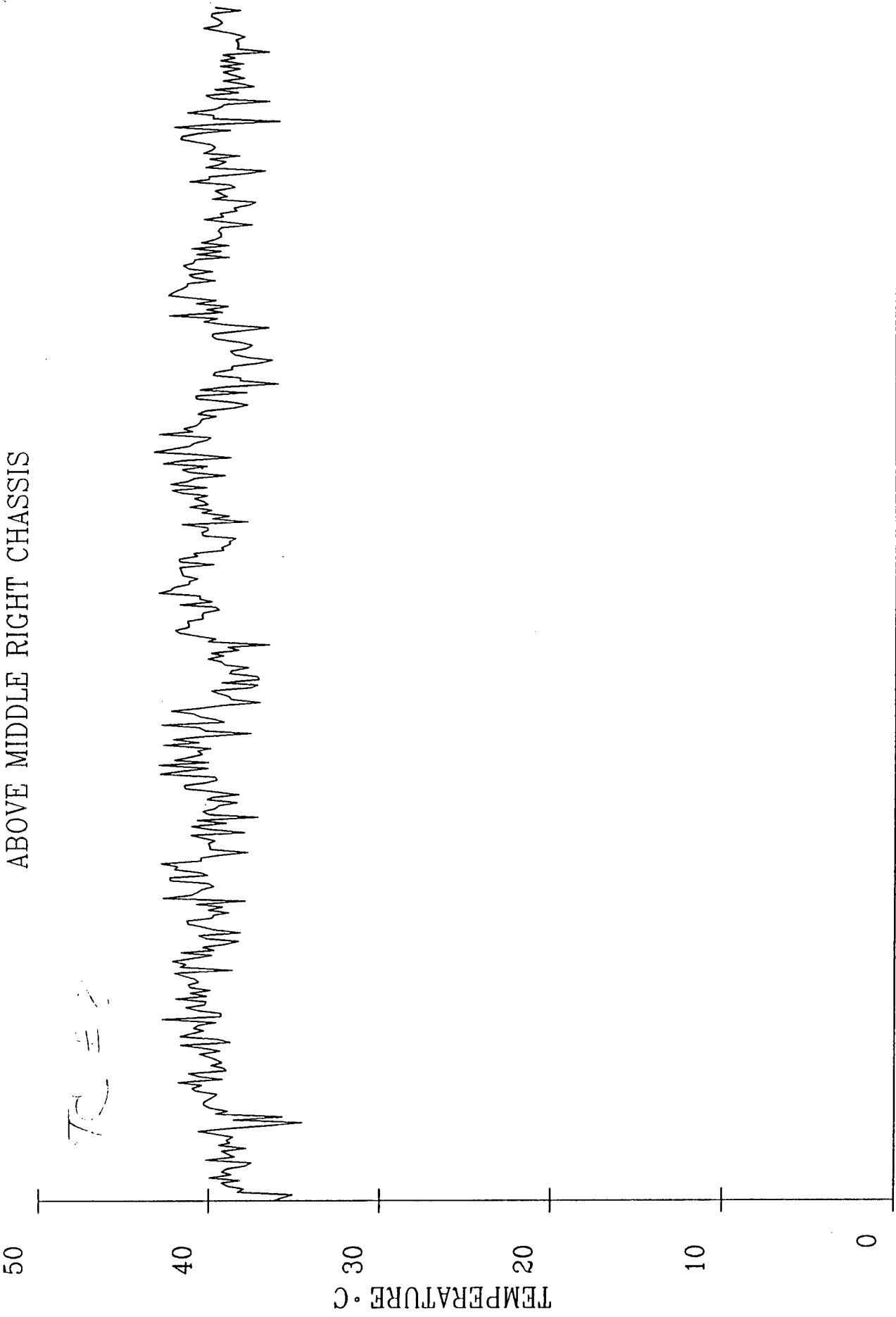


RUNTIME = 103 HOURS

ABOVE TOP RIGHT CHASSIS



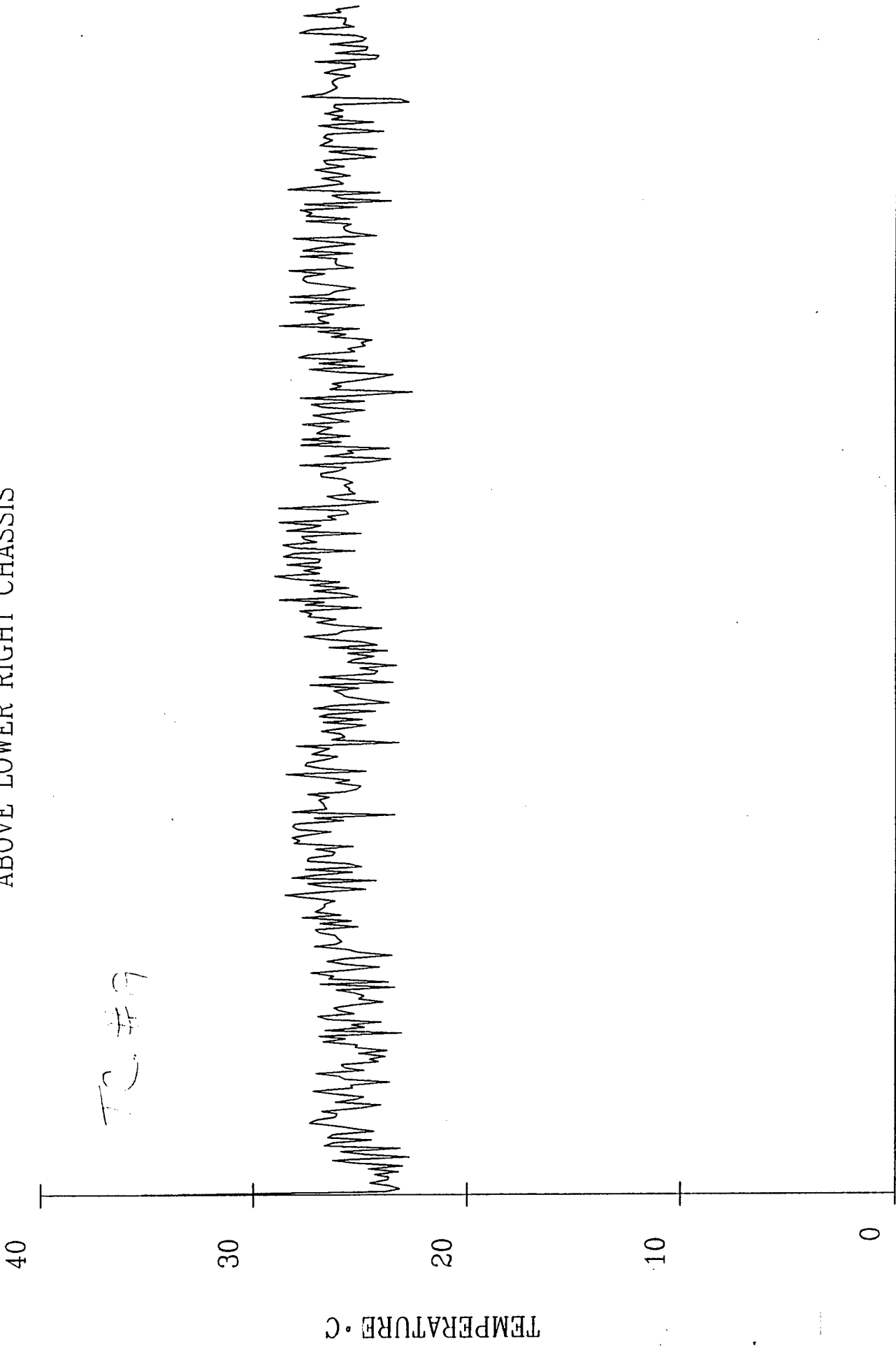
ABOVE MIDDLE RIGHT CHASSIS



RUNTIME = 103 HOURS

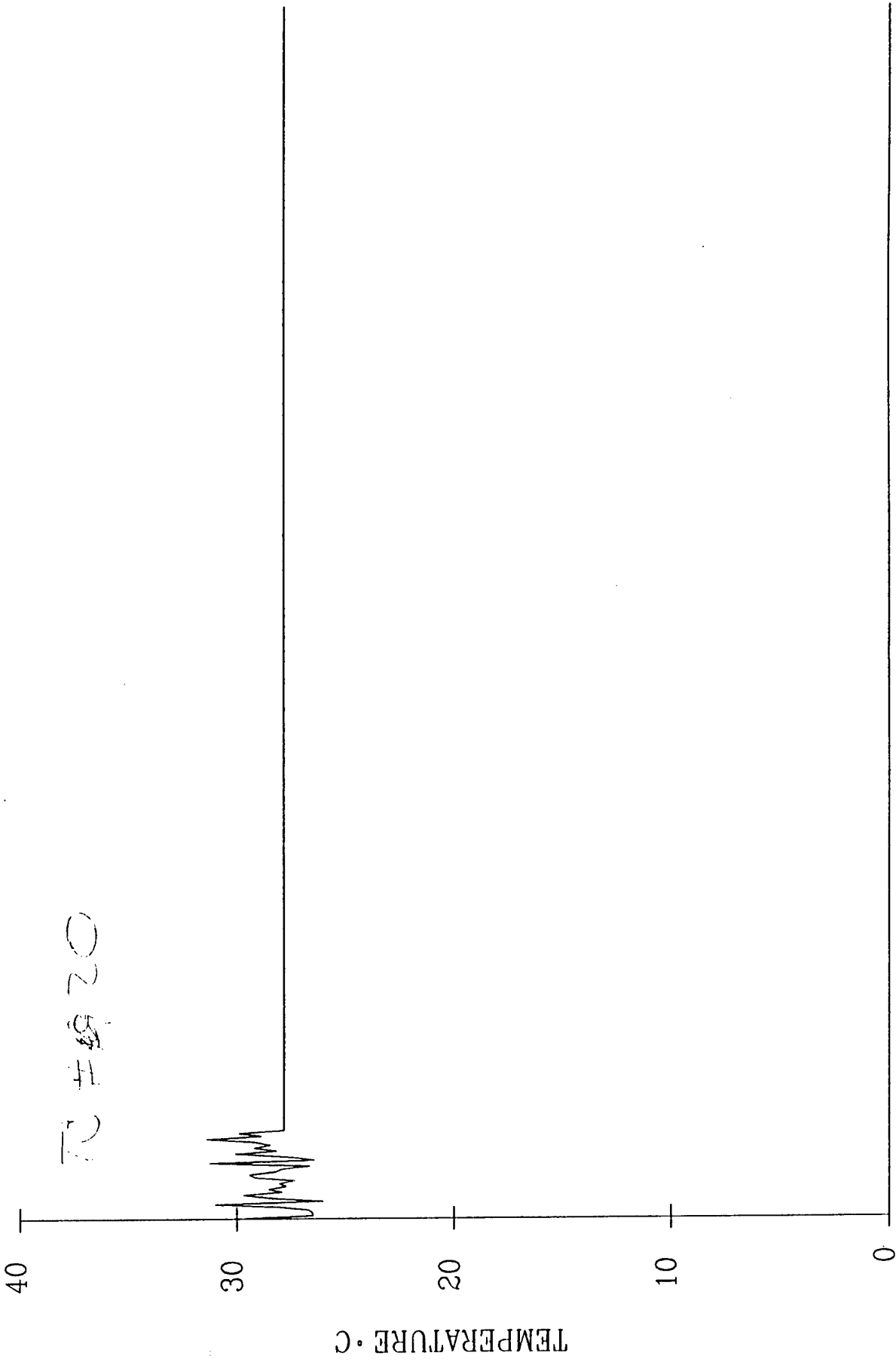
ABOVE LOWER RIGHT CHASSIS

TC #9



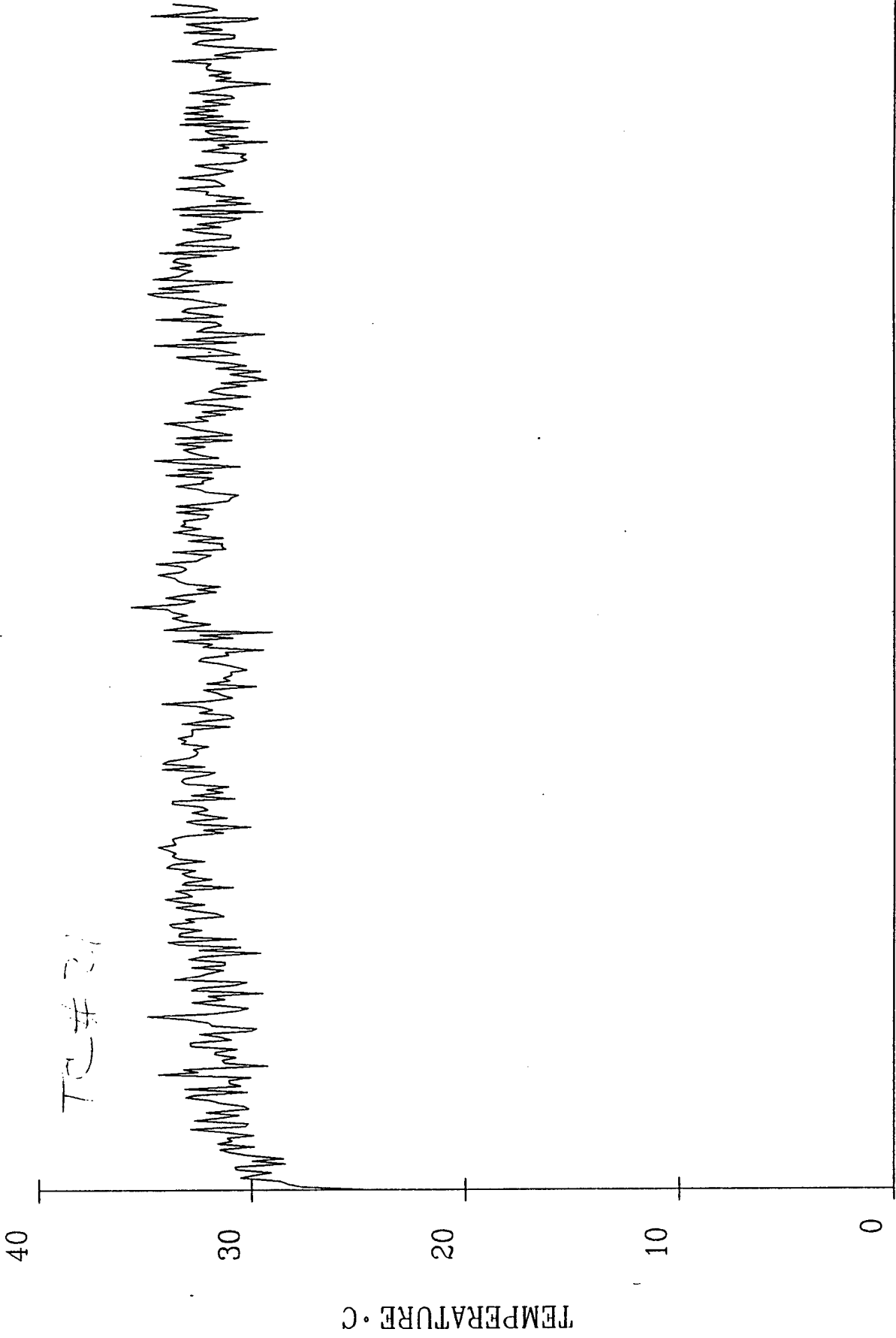
RUNTIME = 103 HOURS

TOP OF LEFT CABINET



RUNTIME = 103 HOURS

TOP OF MIDDLE CABINET



RUNTIME = 103 HOURS

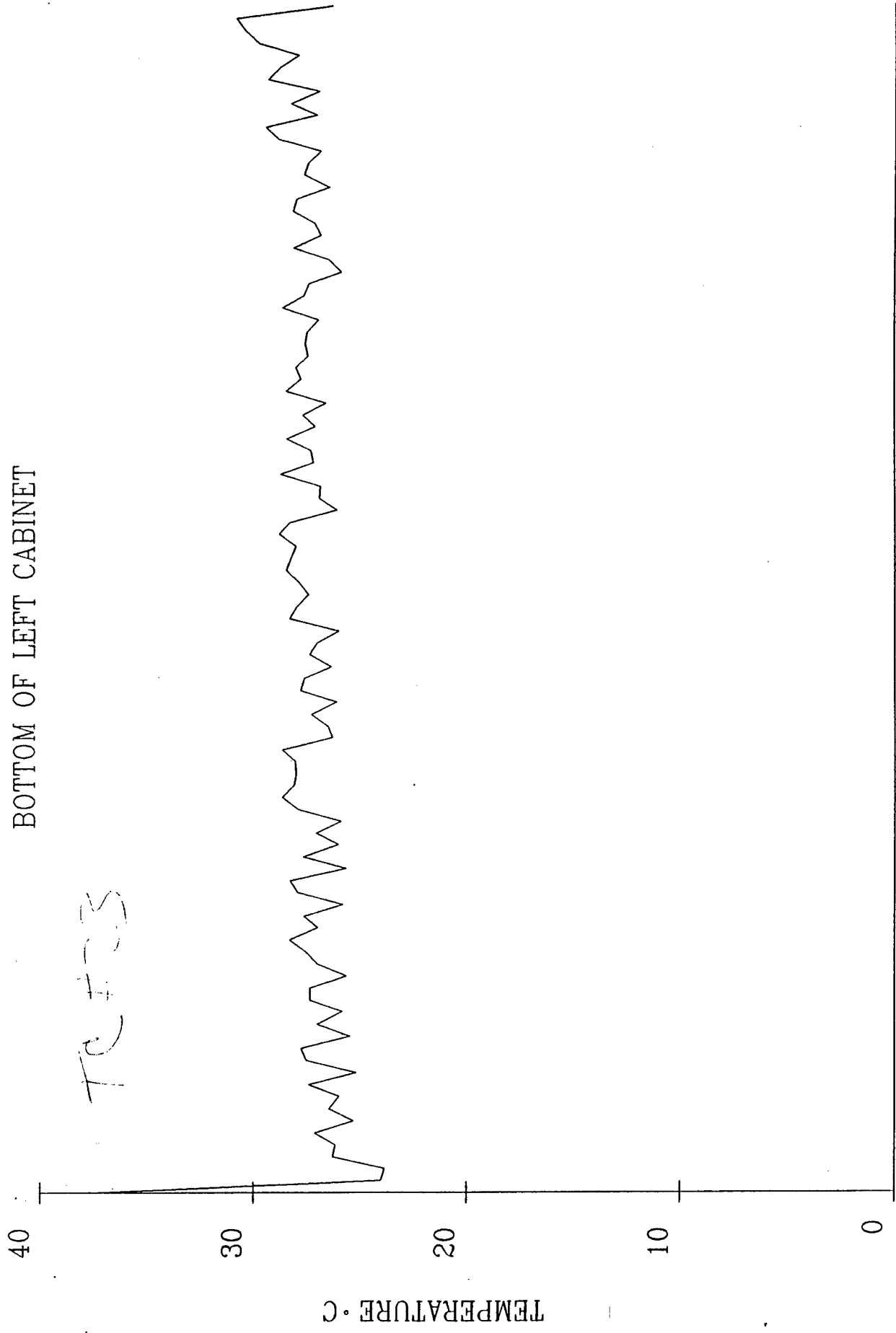
TOP OF RIGHT CABINET



RUNTIME = 103 HOURS

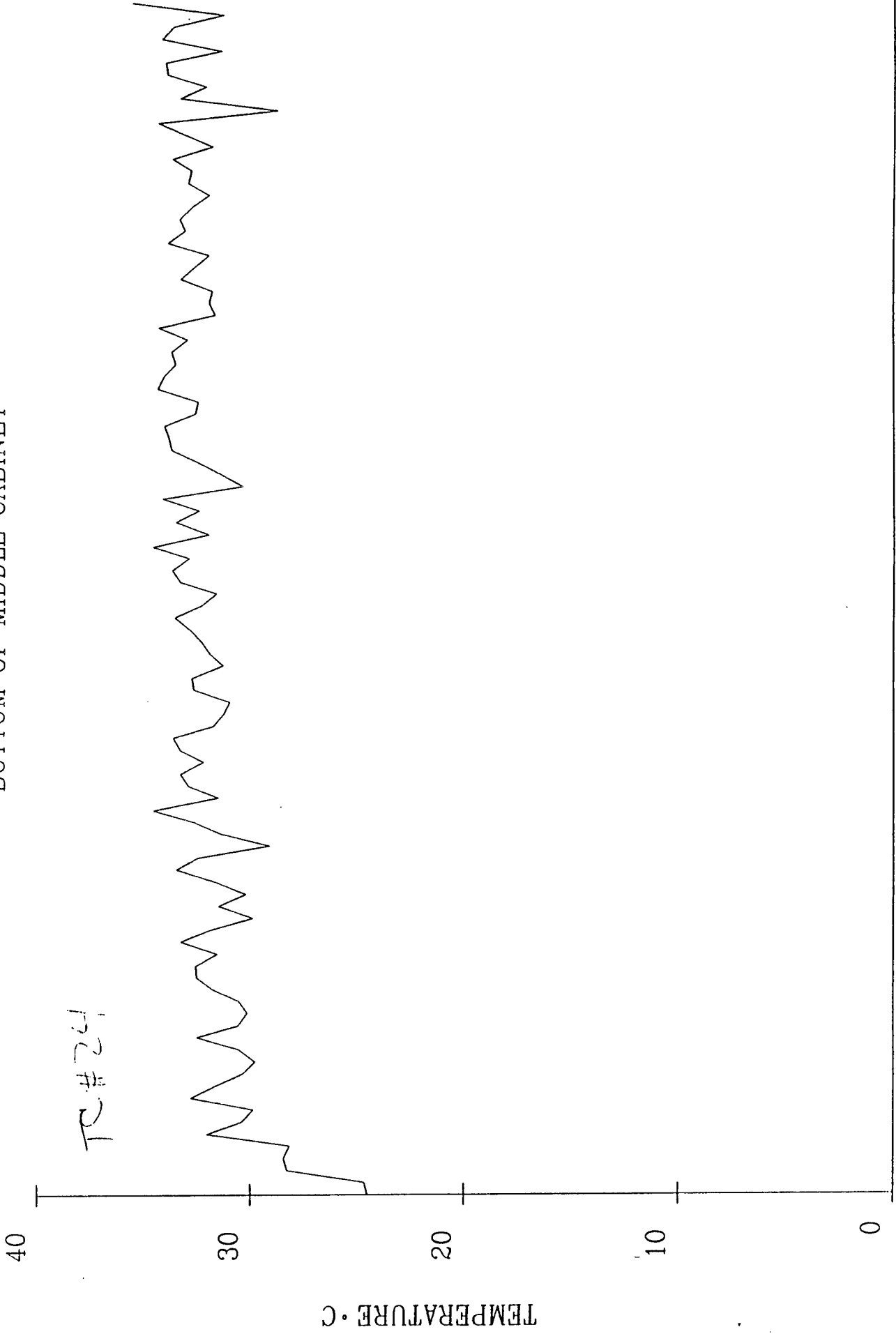
BOTTOM OF LEFT CABINET

TC # 23



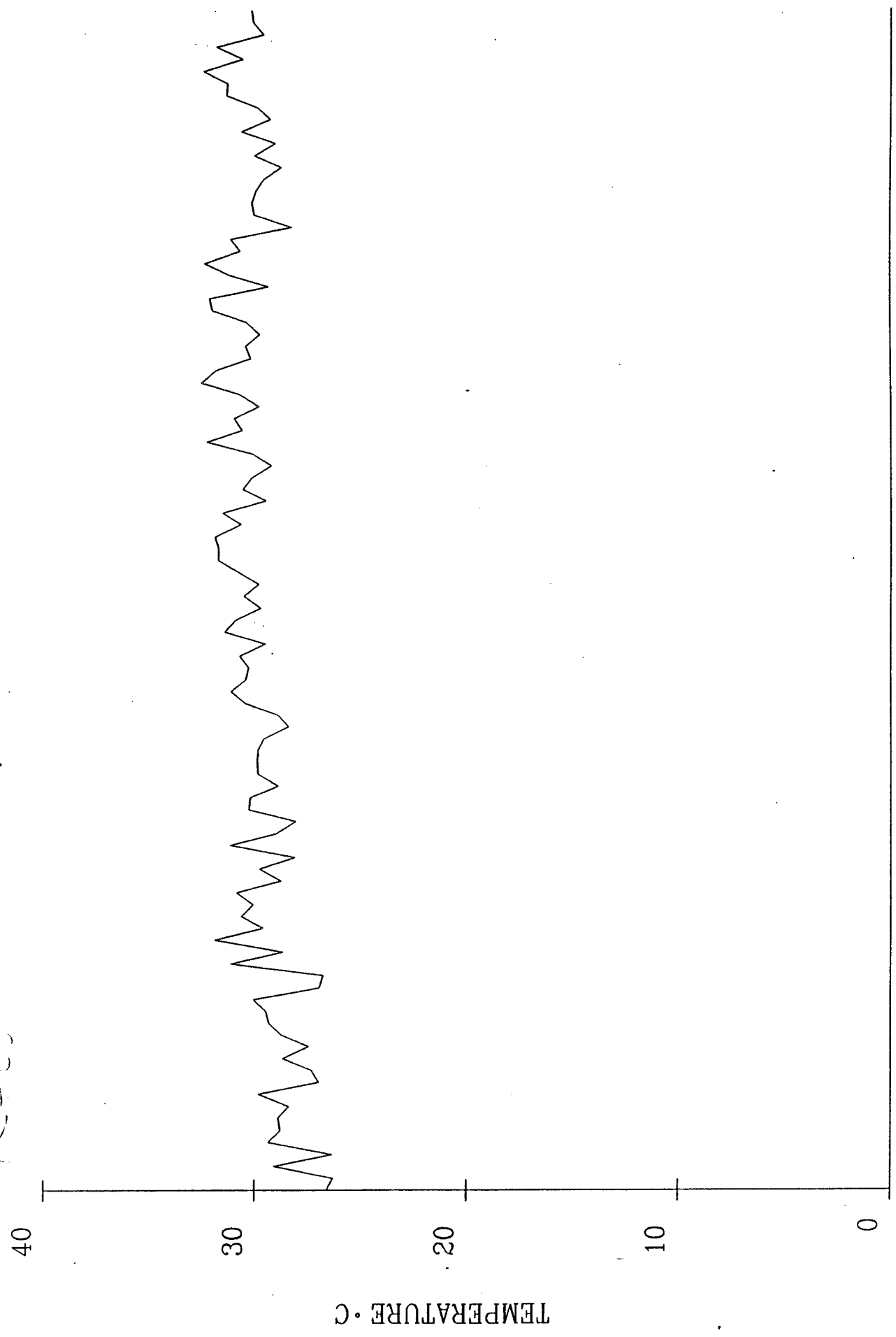
RUNTIME = 103 HOURS

BOTTOM OF MIDDLE CABINET



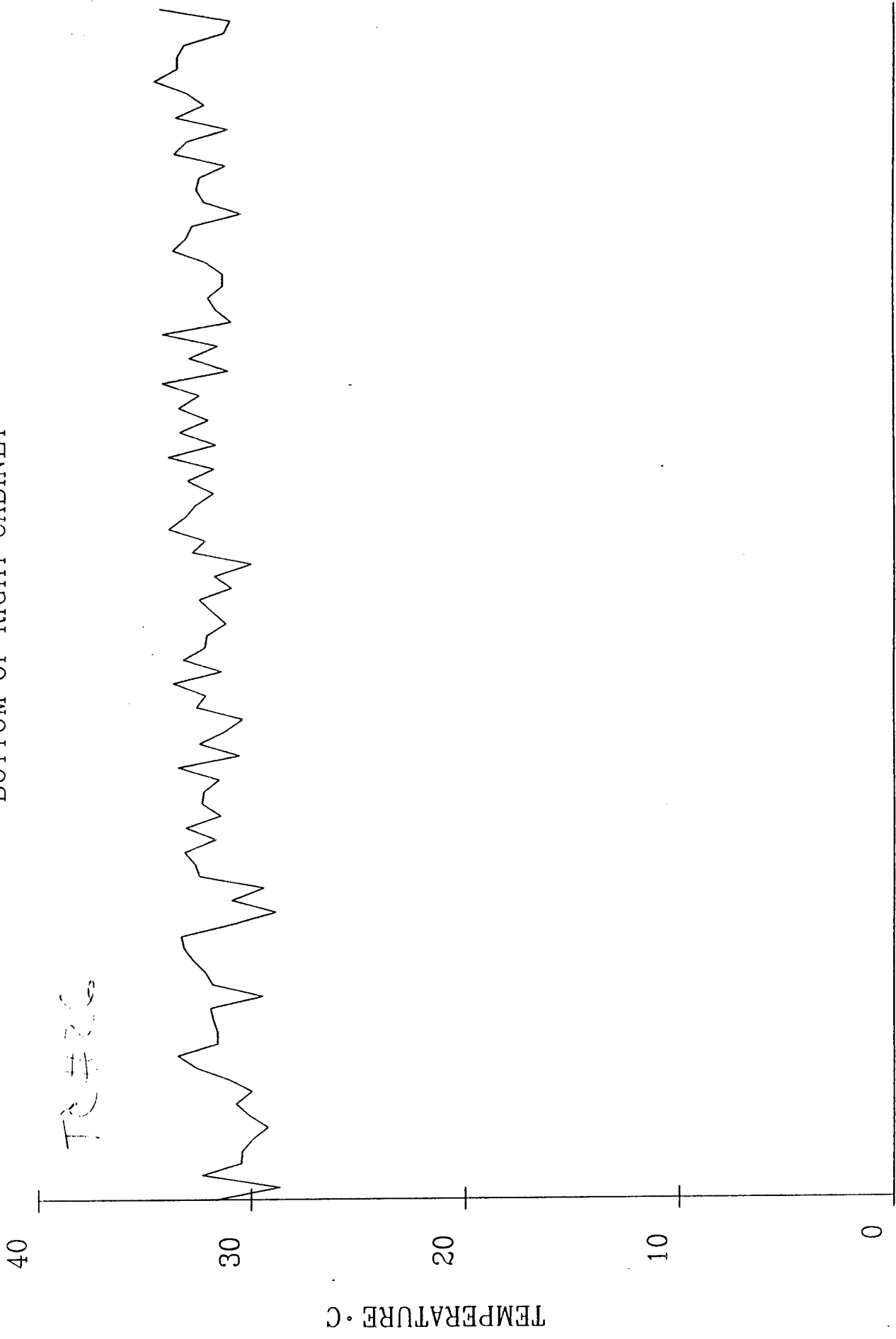
RUNTIME = 103 HOURS

TC#25 LEFT TOP SEQUENCER BREAK OUT BOARD



RUNTIME = 103 HOURS

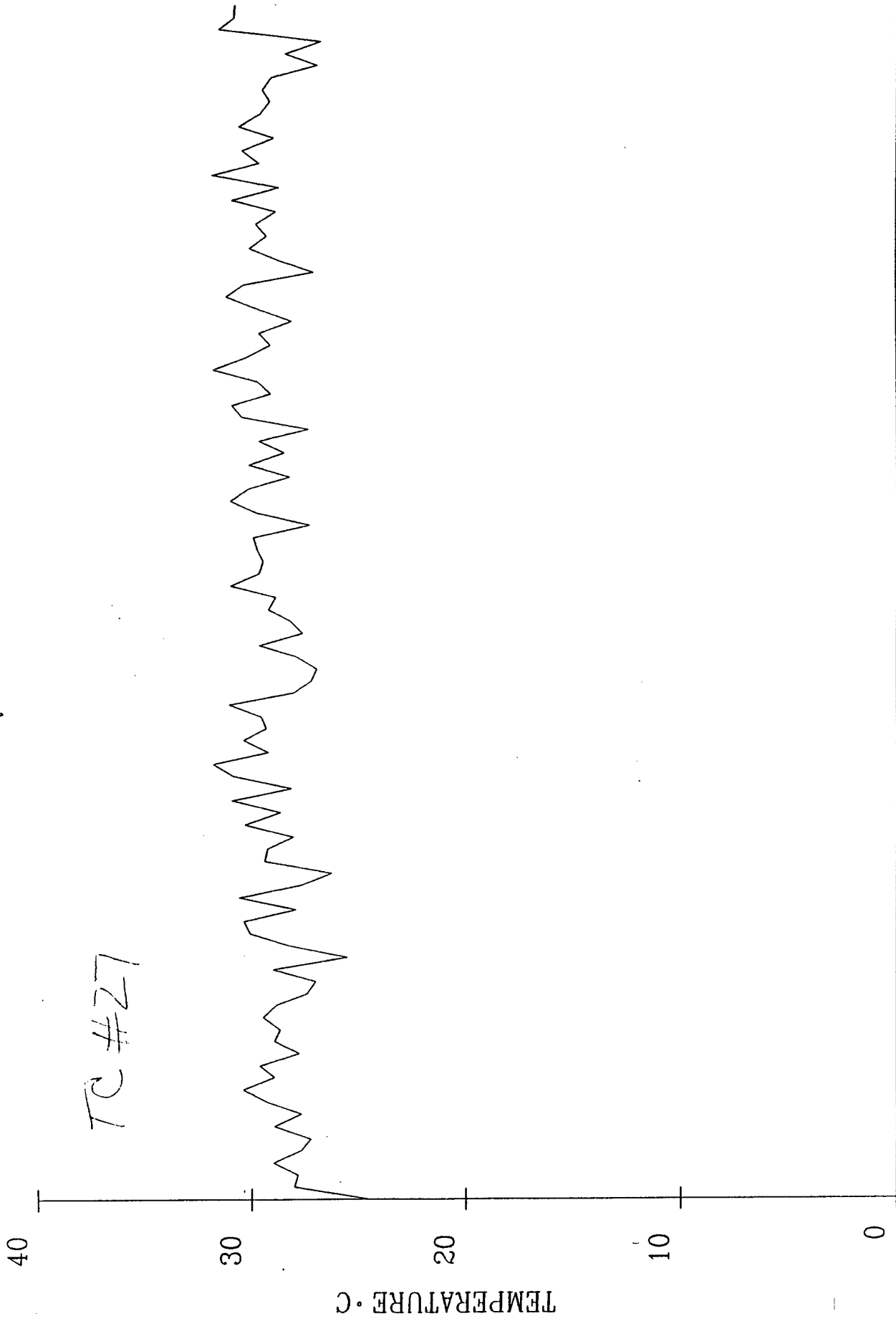
BOTTOM OF RIGHT CABINET



RUNTIME = 103 HOURS

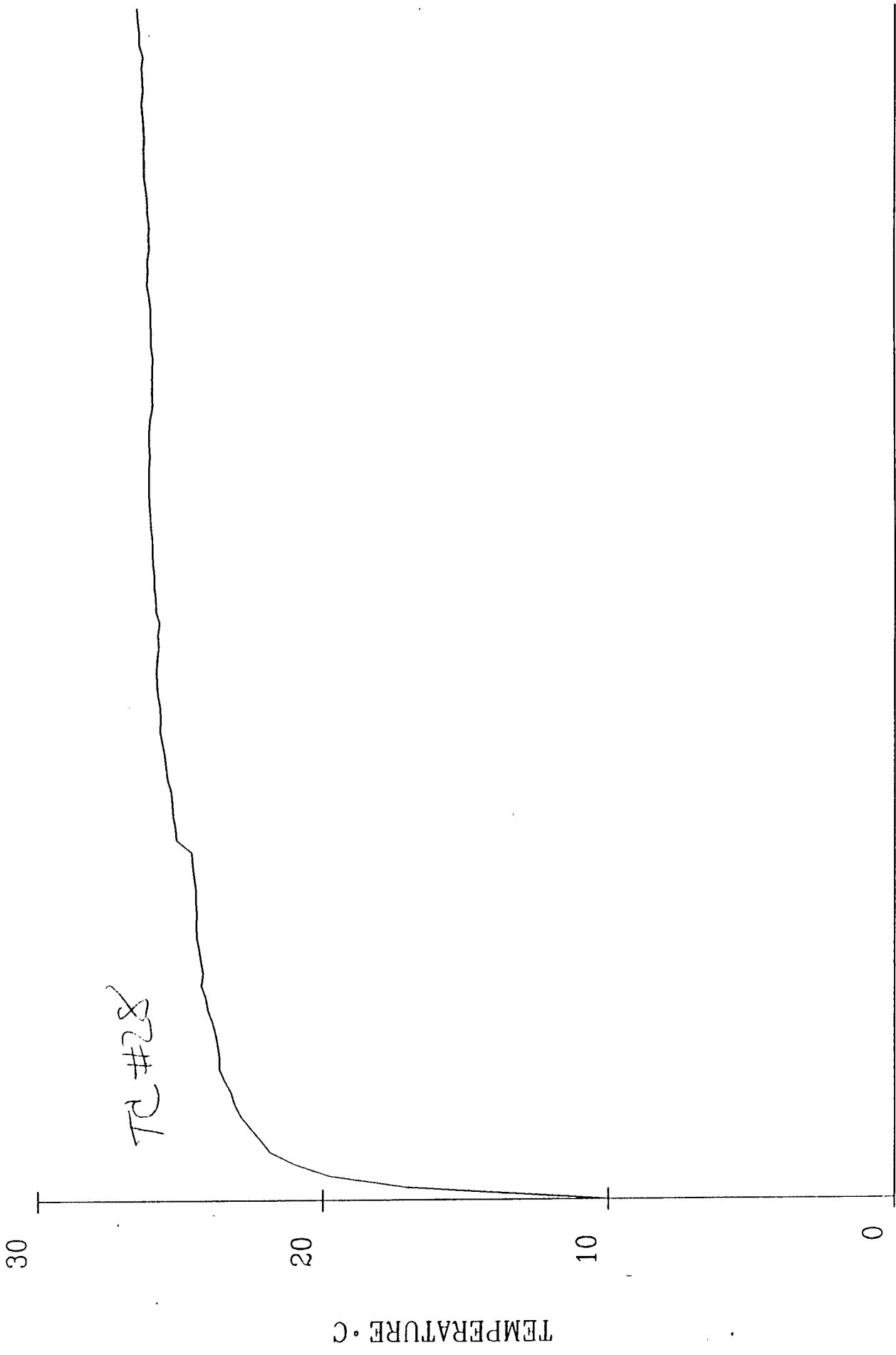
LEFT BOTTOM SEQUENCER BREAKOUT BOARD

TC #27



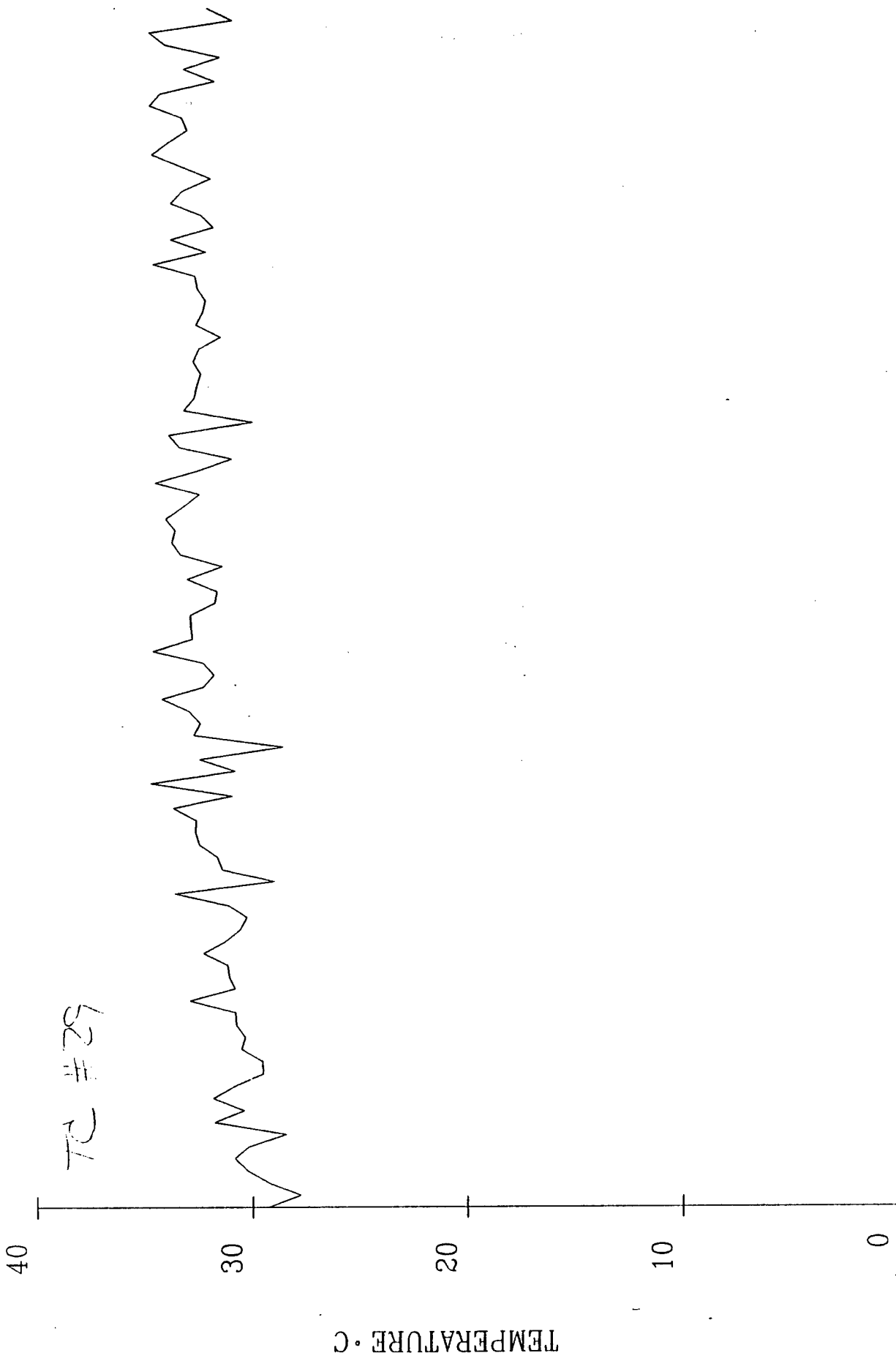
RUNTIME = 103 HOURS

RIGHT TOP SEQUENCER BREAKOUT BOARD



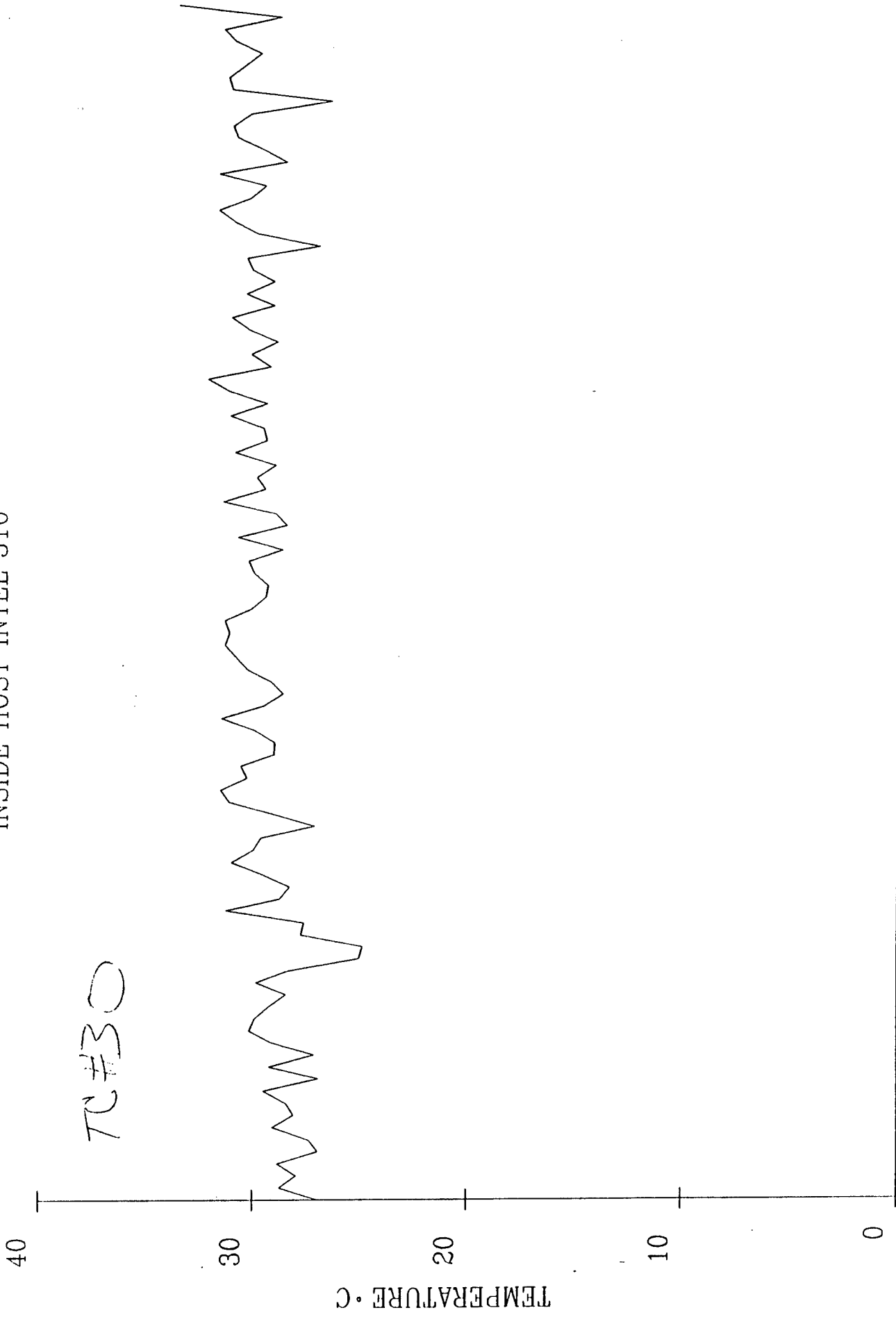
RUNTIME = 103 HOURS

RIGHT BOTTOM SEQUENCER BREAKOUT BOARD



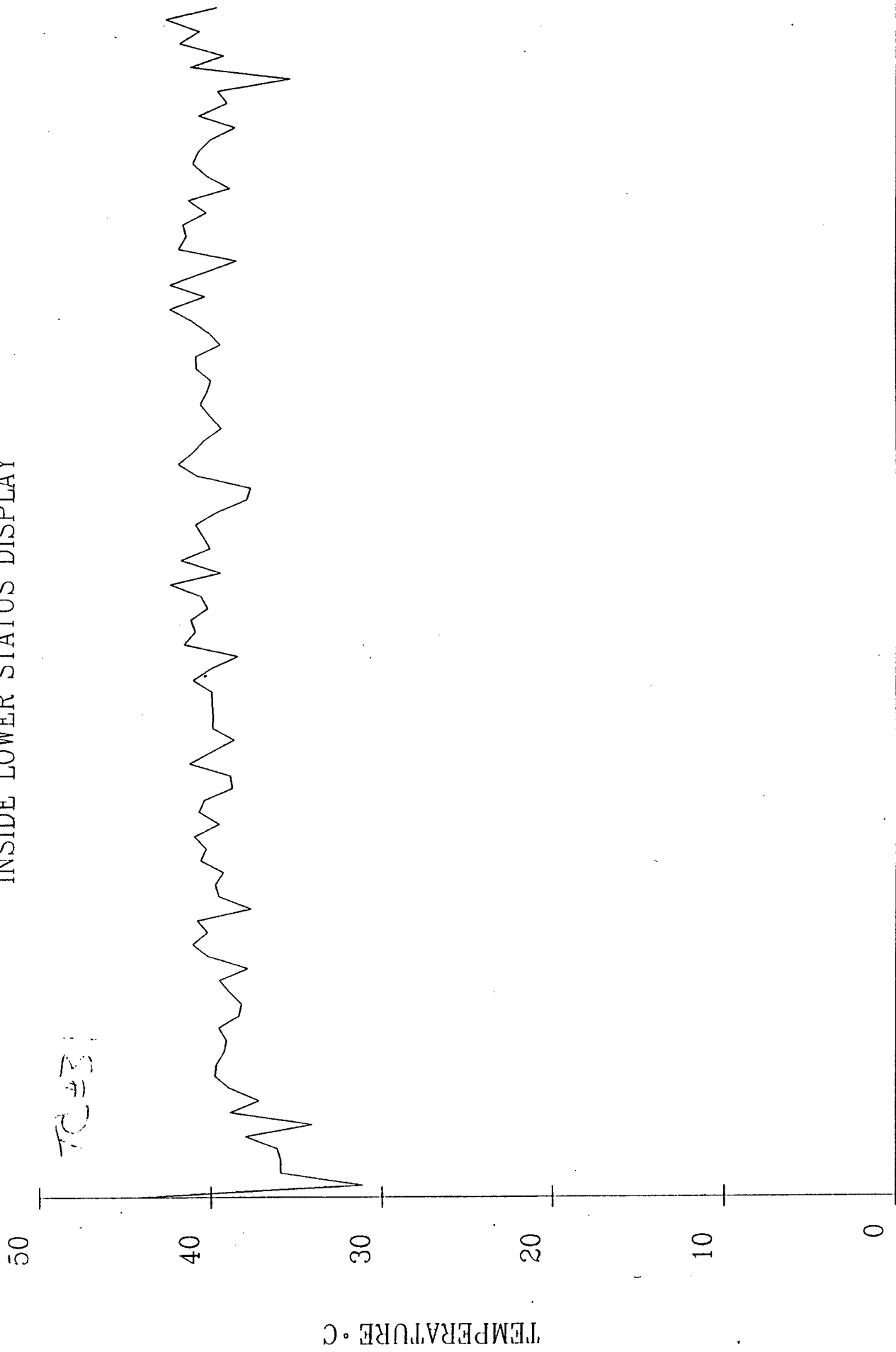
INSIDE HOST INTEL 310

TC #30



RUNTIME = 103 HOURS

INSIDE LOWER STATUS DISPLAY



RUNTIME = 103 HOURS

ROOM TEMPERATURE



RUNTIME = 103 HOURS

## 10.2 Appendix B Test Programs

```
Module hostiot;
```

```
$include('/spock/def/bitfunc.def')  
$include('/spock/def/sysinfo.def')  
$include('/spock/def/multiio.def')  
$include('/spock/def/slaveio.def')  
$include(/spock/fpp/amd/hostio.def)
```

```
{ This is the host program for the T2 or handshake diagnostic.  
This program takes the processors listed in t2proc and writes  
a value to the first one. That processor (processor 1) in turn  
(over the crossbar) sequentially writes that value to and  
reads that value from the remaining processors in the list to  
validate the transfer. (The approach for this test looks much  
like the center of a wheel reaching out one at a time to  
processors that are on the end of the spokes. Where the  
center of the wheel is processor 1.) Four different patterns  
are transfered (0000,5555,AAAA,FFFF). }
```

```
program hostiot(input,output,outlog,proclist,TESTINFO);  
const  
    maxPROC = 64;
```

```
var  
    rs,rr : real;  
    NumProc,i,k,loop,error,count:integer;  
    l : longint;  
    outlog,proclist,TESTINFO :text;  
    fpp_page_port ,fpp_page :array [1..maxPROC] of word;  
    kount,lsend,lrec : longint;  
    T1,T2,I1,I2,cntr : integer;
```

```
$include ('/spock/util/screen.src')
```

```
{ This included procedure writes a long integer in hex format.  
}
```

```
$include (/spock/fpp/amd/hex.hst)
```

```
begin
```

```
{ Read the file TESTINFO for the number of times to run this  
test. }
```

```
    T2:=0;  
    reset(TESTINFO);  
    if eof(TESTINFO) then  
        writeln('File TESTINFO not found, run TESTSETUP');  
    if not eof(TESTINFO) then  
        begin  
            readln(TESTINFO,T1,I1);  
            readln(TESTINFO,T2,I2);  
        end;
```

```

    cntr:=0;
{ Loop for the number of times specified in TESTINFO. }

    while cntr < I1 do
    begin
        cntr:=cntr+1;
        for k := 1 to 4 do
        begin
            gotoxy(28,20);
            print_lngint_hex(output,lsend[k]);
            for i := 1 to NumProc do
            begin
                { Write an 'M' under the processor being tested. }

                if i > 48 then
                    gotoxy(13+((i-49)*3),17)
                else if i > 32 then
                    gotoxy(13+((i-33)*3),14)
                else if i > 16 then
                    gotoxy(13+((i-17)*3),11)
                else
                    gotoxy(13+((i-1)*3),8);
                write('  M');

                { Set-up a communication channel to the processor being
                tested. }

                    turn_on_repeater(fpp_page_port[i],fpp_page[i]);

                { Send a value to the processor. }

                    longint_send(lsend[k]);

                if i > 48 then
                    gotoxy(13+((i-49)*3),17)
                else if i > 32 then
                    gotoxy(13+((i-33)*3),14)
                else if i > 16 then
                    gotoxy(13+((i-17)*3),11)
                else
                    gotoxy(13+((i-1)*3),8);

                { Receive the value back from the processor to validate the
                transfer (The processor just echos the value.). }

                    lrec := longint_receive;

                { If the values are equal, meaning the transfer was okay, the
                'M' under that processor number is blanked out. }

                    if ( (lsend[k]-lrec) = 0.0) then
                        write('  ')

```

```

else

{ If the transfer failed, an 'E' is written over the 'M' and
an error message including the processor being tested, the
value sent to the processor, and the value received from the
processor is written to the outlog. }

begin
  write(' E');
  error := error + 1;
  gotoxy(20,22);
  write('Last error : Processor ',i:2,' sent ');
  print_lngint_hex(output,lsend[k]);
  write('received ');
  print_lngint_hex(output,lrec);

  write(outlog,'Error Processor ',i:2,' sent ');
  print_lngint_hex(outlog,lsend[k]);
  write(outlog,' and received ');
  print_lngint_hex(outlog,lrec);
  writeln(outlog,' ');

end;
count := count + 1;
if (count > 32000) then
  count := 1;

  gotoxy (10,24);
  write(' Number of transfers = ',count:12,' Number
of Errors = ',error);
  turn_off_repeater(fpp_page_port[i]);
end;
end;
end;

{ The total number of transfers and errors are written to the
outlog after the test is finished. }

  writeln(outlog,'Number of transfers = ',count:12,'
Number of errors = ',error);
  writeln(outlog,' ');
end;
end.

```

```
Module hostiot;
```

```
$include('/spock/def/bitfunc.def')  
$include('/spock/def/sysinfo.def')  
$include('/spock/def/multiio.def')  
$include('/spock/def/slaveio.def')  
$include(/spock/fpp/amd/hostio.def)
```

```
{ This is the host program for the fppmu diagnostic. This  
program takes a list of processors (fppmuproc) and in turn  
sequentially writes a value to and reads that value from each  
processor over the multibus to validate the transfer. Four  
patterns (0000,5555,AAAA,FFFF) are used. }
```

```
program hostiot(input,output,outlog,TESTINFO,proclist);  
const
```

```
    maxPROC = 64;
```

```
var
```

```
    rs,rr : real;  
    NumProc,i,k,error,count,T1,I1,cntr:integer;  
    l,lrec : longint;  
    fpp_page_port ,fpp_page :array [1..maxPROC] of word;  
    proclist,outlog,TESTINFO:text;  
    lsend : array[1..10] of longint;
```

```
$include ('/spock/util/screen.src')
```

```
{ This included procedure prints a long integer in hex format.  
}
```

```
$include (/spock/fpp/amd/hex.hst)
```

```
begin
```

```
{ Read the file TESTINFO for the number of times to run this  
test. }
```

```
T1:=0;  
reset(TESTINFO);  
if eof(TESTINFO) then  
    writeln('File TESTINFO not found, run TESTSETUP.');
```

```
if not eof(TESTINFO) then  
    readln(TESTINFO,T1,I1);  
if T1 = 1 then  
    begin  
  
        rewrite(outlog);  
        writeln(outlog,'*****');  
        writeln(outlog,'***** Starting FPPMU test. *****');  
        writeln(outlog,'***** Iterations = ',I1:6,' *****');  
        writeln(outlog,'*****');  
        writeln(outlog,' ');
```

```

reset(proclist);

i := 0;

{ Read all the processor addresses from the file FPPMUPROC,
and initialize each one. }

while not eof(proclist) do
begin
  i := i + 1;
  writeln('reading processor ',i:2);
  readln(proclist,fpp_page_port[i],fpp_page[i]);
  writeln('Processor ',i:2,' at page ',fpp_page[i],' of
page port ',fpp_page_port[i]);
  initialize(fpp_page_port[i],fpp_page[i]);
end;
NumProc := i;

{ Set-up the screen. }

clearscreen;
gotoxy(0,7);
write('Processor : ');

{ Write the numbers of each processor to be tested to the
screen. }

for i := 1 to NumProc do
begin
  if i > 48 then
    gotoxy(13+((i-49)*3),16)
  else if i > 32 then
    gotoxy(13+((i-33)*3),13)
  else if i > 16 then
    gotoxy(13+((i-17)*3),10);
  write(i:3);
end;

gotoxy(5,20);
write('Pattern transferring : ');

{ Values to be sent. When the constant to be sent had the most
significant bit set it caused problems, but if you summed
numbers that didn't have the MSB set, the result could be sent
without problems. }

lsend[1] := 02aaaaaaaah + 40000000h + 40000000h;
lsend[2] := 07ffffffffh + 40000000h + 40000000h;
lsend[3] := 055555555h;
lsend[4] := 000000000;

error := 0;
count := 0;

```

```

if T2 = 1 then
begin

    rewrite(outlog);
    writeln(outlog,'*****');
    writeln(outlog,'***** Handshake (T2) test. *****');
    writeln(outlog,'***** Iterations = ',I2:6,' *****');
    writeln(outlog,'*****');
    writeln(outlog,' ');
    i := 0;
    error := 0;

{ read all the processor addresses from t2proc. }

    reset(proclist);
    while not eof(proclist) do
    begin
        i := i + 1;
        writeln('reading processor ',i:2);
        readln(proclist,fpp_page_port[i],fpp_page[i]);
        writeln('Processor ',i:2,' at page ',fpp_page[i],' of
page port ',fpp_page_port[i]);
        end;

        NumProc := i;

{ Set-up the screen. }

        clearscreen;
        gotoxy(5,8);
        write('value sent ');
        gotoxy(22,2);
        write('# cycled patterns ');

        kount := 0;

{ Initialize processor 1. This will be the processor which
broadcasts to each of the other processors, one at a time,
over the crossbar, then reads the value back from each
processor to validate the transfer. The approach for this test
looks much like the center of a wheel reaching out one at a
time to processors that are on the end of the spokes. Where
the center of the wheel is processor 1. }

        initialize(fpp_page_port[1],fpp_page[1]);

        cntr:=0;

{ Loop for the number of times specified in TESTINFO. }

        while cntr < I2 do
        begin
            cntr:=cntr+1;
            for l := 2 to Numproc do

```

```

    for k := 1 to 4 do
    begin
        kount := kount +1;
        gotoxy(40,2);
        write(kount);

{ Initialize the processor to be sent to, and set-up a
communication channel. }

        initialize(fpp_page_port[1],fpp_page[1]);
        turn_on_repeater(fpp_page_port[1],fpp_page[1]);
        case k of

{ Values to be sent. (When the constant to be sent had the
most significant bit set it caused problems, but if you summed
numbers that didn't have the MSB set, the result could be sent
without problems. )

        {
            1: lsend := 02aaaaaaah + 40000000h +
40000000H; }
            1,3: lsend := 07ffffffh + 40000000h +
40000000h;
            {
                3: lsend := 0555555555h; }
                2,4: lsend := 0h {+ 40000000h + 40000000h};
        end;
        gotoxy(17,8);
        write('bfor snd');

{ Send data over the crossbar from processor 1 to processor L.
}

        longint_send(lsend);
        gotoxy(17,8);
        print_lngint_hex(output,lsend);
        write(' to check P',(l-1):2);
        gotoxy(43,11);
        write(' ');

{ Close the communication channel between processor 1 and the
host. }

        turn_off_repeater(fpp_page_port[1]);

        begin

{ Open a communication channel to the processor that data was
just sent to. }

            turn_on_repeater(fpp_page_port[1],fpp_page[1]);

{ Receive data from processor L (Processor L just echos the
data.). }

            lrec := longint_receive;

```

```

gotoxy(17,10);
print_lngint_hex(output,lrec);
write(' received from P',(l-1):2);
gotoxy(17,11);
print_lngint_hex(output,lrec);
write(' sending to P');
write(0:2);
lrec := longint_receive;
write(' sent');

```

```

{ If the processor under test didn't echo the data correctly,
an error message including the processor number under test and
the difference between what was sent and what was received
back. }

```

```

if (lsend <> lrec) then
begin
gotoxy(26,20);
write('error ');
write(outlog,'error ');
error := error + 1;
print_lngint_hex(output,(lsend-lrec));
write(' while checking P',(l-1):2,' # ',error);
print_lngint_hex(outlog,(lsend-lrec));
writeln(outlog,' while checking P',(l-1):2,' #
',error);
end;

```

```

{ Close the communication channel from processor L. }

```

```

stop_processor;
turn_off_repeater(fpp_page_port[1]);
end;

```

```

{ Open the communication channel from processor 1. }

```

```

turn_on_repeater(fpp_page_port[1],fpp_page[1]);
lrec := longint_receive

```

```

{ If the processor under test didn't echo the data correctly,
an error message including the processor number under test and
the difference between what was sent and what was received
back. }

```

```

if (lsend <> lrec) then
begin
gotoxy(26,20);
write('error ');
write(outlog,'error ');
error := error + 1;
print_lngint_hex(output,(lsend-lrec));
write(' while checking P',(l-1):2,' # ',error);
print_lngint_hex(outlog,(lsend-lrec));

```

```
        writeln(outlog,' while checking P',(1-1):2,' #
',error);
        end;
        end;
        end;

{ The number of cycled patterns and the running total of
errors are written to the outlog after the test finishes. }

writeln(outlog,'Number of cycled patterns = ',kount,' Number
of errors = ',error);
        writeln(outlog,' ');
end.
```

Module t3;

```
$include('/spock/def/bitfunc.def')
$include('/spock/def/sysinfo.def')
$include('/spock/def/multiio.def')
$include('/spock/def/slaveio.def')
$include(/spock/fpp/amd/hostio.def)
```

```
{ This is the host program for the first run of the T3
diagnostic. This program takes a list of processors (t3proc) and
cycles through the list beginning with the first making each a
sending processor and the rest of the list receiving processors.
This is then done for the second and so on to the end of the
list and then the sequence is repeated. During this test cycle
each receiving processor is checked to verify the validity of
the transfer [errors are written to the screen and to outlog
(:lp:?)]. The number of transfers, the number of errors, list of
processors, and the value being sent are displayed. An 'S'
adjacent to the sending processor number indicates such. The
purpose of this program is to stress various parts of the system
(e.g., processors, crossbar boards, cables, etc.) The scheme of
this program is to read in an ordered list of processors and
sets the sender and receivers and then send the value to the
sending processor (which sends it across network) and then
checks the receiving processors for the data received. }
```

```
program t3(input,output,outlog,proclist,TESTINFO);
```

```
const
```

```
    maxPROC = 64;
```

```
var
```

```
    rs : array [1..10] of real;
```

```
    rr,count : real;
```

```
    NumProc,i,k,loop,error,T1,T2,T_3,I1,I2,I3,cntr : integer;
```

```
    l : longint;
```

```
    lsend : array[1..10] of longint;
```

```
    lrec : array [1..maxProc] of longint;
```

```
    outlog,proclist,TESTINFO : text;
```

```
    fpp_page_port ,fpp_page : array [1..maxPROC] of word;
```

```
    ProcErr : boolean;
```

```
$include ('/spock/util/screen.src')
```

```
$include (/spock/fpp/amd/hex.hst)
```

```
begin
```

```
{ Read the file TESTINFO for the number of times to run this
test. }
```

```
    T_3:=0;
```

```
    reset(TESTINFO);
```

```
    if eof(TESTINFO) then
```

```
        writeln('File TESTINFO not found, run TESTSETUP.');
```

```
    if not eof(TESTINFO) then
```

```

begin
  readln(TESTINFO,T1,I1);
  readln(TESTINFO,T2,I2);
  readln(TESTINFO,T_3,I3);
end;
if T_3 = 1 then
begin

  reset(proclist);
  rewrite(outlog);
  writeln(outlog,'*****');
  writeln(outlog,'***** Starting Xbar (T3) test. *****');
  writeln(outlog,'***** Iterations = ',I3:6,' *****');
  writeln(outlog,'*****');
  writeln(outlog,' ');

  { Read the addresses of all the processors from T3PROC and
  initialize each of them. }

  i := 0;

  while not eof(proclist) do
  begin
    i := i + 1;
    writeln('reading processor ',i:2);
    readln(proclist,fpp_page_port[i],fpp_page[i]);
    writeln('Processor ',i:2,' at page ',fpp_page[i],' of page
port ',fpp_page_port[i]);
    initialize(fpp_page_port[i],fpp_page[i]);
  end;
  NumProc := i;

  { Set-up the screen. }

  clearscreen;
  gotoxy(0,7);
  write('Xbar test : ');
  for i := 1 to NumProc do
  begin
    if i > 48 then
      gotoxy(13+((i-49)*3),16)
    else if i > 32 then
      gotoxy(13+((i-33)*3),13)
    else if i > 16 then
      gotoxy(13+((i-17)*3),10);
    write((i-1):3);
  end;

  gotoxy(5,20);
  write('Pattern transferring : ');

  { Values to be transfered. When the constant to be sent had the
  most significant bit set it caused problems, but if you summed

```

numbers that didn't have the MSB set, the result could be sent without problems. }

```
lsend[1] := 02aaaaaaah + 40000000h + 40000000H;  
lsend[2] := 07ffffffh + 40000000h + 40000000h;  
lsend[3] := 05555555h;  
lsend[4] := 00000000h;
```

```
error := 0;  
count := 0.0;  
cntr:=0;
```

```
{ Loop the number of times specified in the file TESTINFO. }
```

```
while cntr < I3 do  
begin  
  cntr:=cntr+1;  
  for k := 1 to 4 do  
    for loop := 1 to NumProc do      { cycle through list,  
loop = sender)  
      begin  
        ProcErr := False;  
  
        for i := 1 to NumProc do    {set one to send and others          if i = loop then  
            begin  
              turn_on_repeater(fpp_page_port[i],fpp_page[i]);  
              real_send(1.0);  
              turn_off_repeater(fpp_page_port[i]);  
            end  
          else  
            begin  
              turn_on_repeater(fpp_page_port[i],fpp_page[i]);  
              real_send(0.0);  
              turn_off_repeater(fpp_page_port[i]);  
            end;  
  
          { send value ,lsend[k], to sending processor }  
  
          turn_on_repeater(fpp_page_port[loop],fpp_page[loop]);  
          gotoxy(30,20);  
          print_lngint_hex(output,lsend[k]);  
          longint_send(lsend[k]);  
          turn_off_repeater(fpp_page_port[loop]);  
  
          { read values received by the processors and compare to value  
          for i := 1 to NumProc do  
            if i <> loop then  
              begin
```

```

{ read value from processor }

    turn_on_repeater(fpp_page_port[i],fpp_page[i]);
    lrec[i] := longint_receive;
    turn_off_repeater(fpp_page_port[i]);

{ compare values and set error flag }

    if (lsend[k] <> lrec[i]) then
        ProcErr := True;
    end;

{ position cursor }

    if loop > 48 then
        gotoxy(13+((loop-49)*3),17)
    else if loop > 32 then
        gotoxy(13+((loop-33)*3),14)
    else if loop > 16 then
        gotoxy(13+((loop-17)*3),11)
    else
        gotoxy(13+((loop-1)*3),8);

    if not ProcErr then
    begin
        write(' S');
        if loop > 48 then
            gotoxy(13+((loop-49)*3),17)
        else if loop > 32 then
            gotoxy(13+((loop-33)*3),14)
        else if loop > 16 then
            gotoxy(13+((loop-17)*3),11)
        else
            gotoxy(13+((loop-1)*3),8);
        write(' ');
    end
    else
    begin
        write(' E');
        for i := 1 to NumProc do
            if i <> loop then
                if (lsend[k] <> lrec[i]) then
                    begin
                        error := error + 1;
                        gotoxy(0,22);
                    end;
        end;

{ write out sending and receiving processors, sent and received
values and error }

        write('Last error : P',(loop-1):2,' sent ');
        print_lngint_hex(output,lsend[k]);
        write(' P',(i-1):2,' received ');
        print_lngint_hex(output,lrec[i]);
        write(' error ');

```

```

        print_lngint_hex(output,lsend[k]-lrec[i]);
        writeln;

(write to error to outlog (:LP:?) )

        write(outlog,'error : P',(loop-1):2,' sent
');
        print_lngint_hex(outlog,lsend[k]);
        write(outlog,' P',(i-1):2,' received ');
        print_lngint_hex(outlog,lrec[i]);
        write(outlog,' error ');
        print_lngint_hex(outlog,lsend[k]-lrec[i]);
        writeln(outlog);
    end;
end;
count := count + (NumProc-1);

gotoxy (10,24);
write(' Number of transfers = ',count:12:1,' Number
of Errors : ',error);
    end;
end;
writeln(outlog,'Number of transfers = ',count:12:1,' Number
of errors = ',error);
writeln(outlog,' ');
end;
end.

```

```
Module hostiot;
```

```
$include('/spock/def/bitfunc.def')  
$include('/spock/def/sysinfo.def')  
$include('/spock/def/multiio.def')  
$include('/spock/def/slaveio.def')  
$include(/spock/fpp/amd/hostio.def)
```

```
{ This is the host program for the ftest diagnostic. This  
program tests the function calculation of the processors  
listed in funcproc. For each of the 10 functions  
(sine, cosine, tangent, arcsine, arccosine, arctangent, reciprocal,  
exponential, square root, and natural log) numbers over some  
respective range are sent to the processor and the processor  
returns the values of the function. These results are each  
compared to the values computed by the host to validate the  
calculation. All 10 functions are tested on one processor  
before testing the next processor in the list. }
```

```
program hostiot(input, output, proclist, TESTINFO, outlog);  
const  
    maxPROC = 64;
```

```
var
```

```
numproc, i, k, error, tot_err, count, bad, cntr, T1, T2, T3, T4, I1, I2, I3,  
I4 : integer;  
    fpp_page_port , fpp_page : array [1..maxPROC] of word;  
    proclist, outlog, TESTINFO : text;  
    lsend : array[1..10] of longint;  
    done : boolean;  
    choice : char;
```

```
$include ('/spock/util/screen.src')
```

```
{ This procedure computes the value of sin(x) for  $-1 \leq x \leq +1$  in  
0.01 increments. If the result is off by more than 0.00000026  
it is flagged as bad by setting the variable bad==1. When an  
error occurs, the value of x, sin(x) computed by the FPP,  
sin(x) computed by the HOST, and the difference is written to  
the outlog. }
```

```
procedure test_sine;
```

```
var  
    line_count, page, errors : integer;  
    x, y, z, diff, tolerance : longreal;
```

```
begin
```

```
    line_count := 1;  
    page := 1;  
    tolerance := 0.00000026;  
    errors := 0;  
    x := -1.0;  
    while (x <= 1.0) do
```

```

begin
  send(1);
  real_send(x);
  y := real_receive;
  diff := (abs(y-sin(x)));
  if diff > tolerance then
  begin
    if line_count = 1 then
    begin
      write(outlog,' X           Remote Sine           Local
Sine');
      writeln(outlog,'           Difference',
                ' on ',fpp_page_port[i],
                ',fpp_page[i]);
      writeln(outlog,' -----
-----');
      line_count := 2;
    end;
    errors := errors + 1;
    write(outlog,x:4:2,'           ',y:10:6);
    writeln(outlog,'           ',sin(x):10:6,'
,diff:8:8);
    end;
    x := x + 0.01;
    z := z + 1;
  end;
  if errors <> 0 then
  begin
    bad := 1;
    writeln(outlog);
    writeln(outlog,errors,' errors have occurred in Sine
function');
    writeln(outlog,'Processor ',i:2);
    writeln(outlog);
    tot_err:=tot_err+errors;
  end;
end;
end;

```

{ This procedure computes the value of cos(x) for  $0 \leq x \leq \pi$  in 1 degree increments. If the result is off by more than 0.00000025 it is flagged as bad by setting the variable bad==1. When an error occurs, the value of x (radians), the value of x (degrees), cos(x) computed by the HOST, cos(x) computed by the FPP, and the difference is written to the outlog. }

```

procedure test_cosine;
const
  PI = 3.1415927;
var
  line_count,page,errors : integer;
  rad,degr,y,tolerance,diff : longreal;

```

```

begin
  errors := 0;
  tolerance := 0.00000025;
  line_count := 1;
  page := 1;
  degr := 0.0;
  rad := 0.0; { Just to get inside the loop}
  while (rad <= PI) do
  begin
    rad := (degr*PI)/180;
    send(2);
    real_send(rad);
    y := real_receive;
    diff := (abs(y-cos(rad)));
    if diff > tolerance then
    begin
      errors := errors + 1;
      if line_count = 1 then
      begin
        write(outlog, 'X(Radians)      X(Degrees)      Local ');
        writeln(outlog, 'Cos          Remote Cos
Difference',
                                ' on ', fpp_page_port[i],
', fpp_page[i]);
        writeln(outlog, '-----
-----');
        line_count := 2;
      end;
      write(outlog, rad:8:6, '      ', degr:5:1,
', cos(rad):11:6);
      writeln(outlog, '      ', y:11:6, '      ', diff:8:8);
      end;
      degr := degr + 1.0;
    end;
    if errors <> 0 then
    begin
      bad := 1;
      writeln(outlog);
      writeln(outlog, errors, ' errors occured in Cosine
function. ');
      writeln(outlog, 'Processor', i:2);
      writeln(outlog);
      tot_err:=tot_err+errors;
    end;
  end;
end;

```

```

{ This procedure computes the value of tan(x) for 0<=x<=PI
x<>PI/2 in 1 degree increments. If the result is off by more
than 0.00014287 it is flagged as bad by setting the variable
bad==1. When an error occurs, the value of x(radians), the
value of x(degrees), tan(x) computed by the HOST, tan(x)
computed by the FPP, and the difference is written to the
outlog. }

```

```

procedure test_tangent;
const
  PI = 3.1415927;
var
  line_count,page,errors : integer;
  rad,degr,y,tolerance,diff : longreal;
begin
  errors := 0;
  tolerance := 0.00014287;
  line_count := 1;
  page := 1;
  degr := 0.0;
  rad := 0.0;
  while (rad <= PI) do
  begin
    rad := (degr*PI)/180;
    if degr <> 90 then
    begin
      send(3);
      real_send(rad);
      y := real_receive;
      diff := (abs(y-tan(rad)));
      if diff > tolerance then
      begin
        errors := errors + 1;
        if line_count = 1 then
        begin
          write(outlog,'X(Radians)      X(Degrees)      ');
          writeln(outlog,' Tangent      FFS Tangent
Difference',
                    ' on ',fpp_page_port[i],',
',fpp_page[i]);
          writeln(outlog,'-----
-----');
          line_count := 2;
          end;
          write(outlog,rad:8:6,'      ',degr:5:1,'
',tan(rad):11:6);
          writeln(outlog,'      ',y:11:6,'      ',diff:8:8);
          end;
        end;

        degr := degr + 1.0;
      end;
    if errors <> 0 then
    begin
      bad :=1;
      writeln(outlog);
      writeln(outlog,errors,' errors in tangent function. ');
      writeln(outlog,'Processor ',i:2);
      writeln(outlog);
      tot_err:=tot_err+errors;
    end;
  end;
end;

```

```
{ This function is the coding of the algorithm used on the FPP
for arcsin }
```

```
function asin(x : longreal) : longreal;
```

```
const
```

```
  PI = 3.14159265;
  P1 = 1.5707963050;
  P2 = -0.2145988016;
  P3 = 0.0889789874;
  P4 = -0.0501743046;
  P5 = 0.0308918810;
  P6 = -0.0170881256;
  P7 = 0.0066700901;
  P8 = -0.0012624911;
```

```
var t,temp,temp2 : longreal;
```

```
begin
```

```
  t := abs(x);
  temp2 := P8 * t;
  temp2 := temp2 + P7;
  temp2 := temp2 * t;
  temp2 := temp2 + P6;
  temp2 := temp2 * t;
  temp2 := temp2 + P5;
  temp2 := temp2 * t;
  temp2 := temp2 + P4;
  temp2 := temp2 * t;
  temp2 := temp2 + P3;
  temp2 := temp2 * t;
  temp2 := temp2 + P2;
  temp2 := temp2 * t;
  temp2 := temp2 + P1;
  temp2 := temp2 * sqrt(1-t);
  temp := (PI/2) - temp2;
  if x < 0 then
    temp := -temp;
  asin := temp;
```

```
end;
```

```
{ This procedure computes the value of arcsin(x) for -1<=x<=+1
in 0.01 increments. If the result is off by more than
0.00000031 it is flagged as bad by setting the variable
bad==1. When an error occurs, the value of x, arcsin(x)
computed by the HOST, arcsin(x) computed by the FPP, and the
difference is written to the outlog. }
```

```
procedure test_arcsine;
```

```
var
```

```
  line_count,page,errors : integer;
  x,y,tolerance,diff : longreal;
```

```
begin
```

```
  errors := 0;
  tolerance := 0.00000031;
```

```

line_count := 1;
page := 1;

x := -1.0;
while (x <= 1.0) do
begin
  send(4);
  real_send(x);
  y := real_receive;
  diff := (abs(y-asin(x)));
  if diff > tolerance then
  begin
    errors := errors + 1;
    if line_count = 1 then
    begin
      write(outlog,' X          ArcSine          FFS
ArcSine');
      writeln(outlog,'          Difference',
          ' on ',fpp_page_port[i],
',fpp_page[i]);
      writeln(outlog,'-----
-----');
      line_count := 2;
    end;
    write(outlog,x:4:2,'          ',asin(x):10:6);
    writeln(outlog,'          ',y:10:6,'          ',diff:8:8);
  end;
  x := x + 0.01;
end;
if errors <> 0 then
begin
  bad := 1;
  writeln(outlog);
  writeln(outlog,errors,' errors in ArcSine function. ');
  writeln(outlog,'Processor ',i:2);
  writeln(outlog);
  tot_err:=tot_err+errors;
end;
end;

```

{ This procedure computes the value of arccos(x) for  $-1 \leq x \leq +1$  in 0.01 increments. If the result is off by more than 0.00000036 it is flagged as bad by setting the variable bad==1. When an error occurs, the value of x, arccos(x) computed by the HOST, arccos(x) computed by the FPP, and the difference is written to the outlog. }

```

procedure test_arccosine;
var
  line_count,page,errors : integer;
  x,y,tolerance,diff : longreal;

function acos(x : real) : real;

```

```

const
  PI = 3.14159265;
begin
  acos := PI/2 - asin(x);
end;

begin
  errors := 0;
  tolerance := 0.00000036;
  line_count := 1;
  page := 1;
  x := -1.0;
  while (x <= 1.0) do
    begin
      send(5);
      real_send(x);
      y := real_receive;
      diff := (abs(y-acos(x)));
      if diff > tolerance then
        begin
          errors := errors + 1;
          if line_count = 1 then
            begin
              write(outlog,' X          ArcCosine          FFS
ArcCosine');
              writeln(outlog,'          Difference',
                          ' on ',fpp_page_port[i],',
',fpp_page[i]);
              writeln(outlog,'-----
-----');
              line_count := 2;
            end;
            write(outlog,x:4:2,'          ',acos(x):10:6);
            writeln(outlog,'          ',y:10:6,'          ',diff:8:8);
          end;
          x := x + 0.01;
        end;
      if errors <> 0 then
        begin
          bad := 1;
          writeln(outlog);
          writeln(outlog,errors,' errors in ArcCosine function.');
```

```

          writeln(outlog,'Processor ',i:2);
          writeln(outlog);
          tot_err:=tot_err+errors;
        end;
      end;
    end;

{ This procedure computes the value of arctan(x) for -1<=x<=+1
in 0.01 increments. If the result is off by more than
0.00000008 it is flagged as bad by setting the variable
bad:=1. When an error occurs, the value of x, arctan(x)
```

computed by the HOST, arctan(x) computed by the FPP, and the difference is written to the outlog. }

```
procedure test_arctangent;
var
  line_count,page,errors : integer;
  x,y,tolerance,diff : longreal;
begin
  errors := 0;
  tolerance := 0.00000008;
  line_count := 1;
  page := 1;
  x := -1.00000000000000;

  while (x <= 1.0000) do
  begin
    send(6);
    real_send(x);
    y := real_receive;
    diff := (abs(y-arctan(x)));
    if diff > tolerance then
    begin
      errors := errors + 1;
      if line_count = 1 then
      begin
        write(outlog,'X          Local ');
        writeln(outlog,'ArcTan  Remote ArcTan  Difference',
                  ' on ',fpp_page_port[i],
',fpp_page[i]);
        writeln(outlog,'-----');
        -----');
        line_count := 2;
      end;
      write(outlog,x:8:6,'          ',arctan(x):11:6);
      writeln(outlog,'          ',y:11:6,'          ',diff:8:8);
      end;
      x := x + 0.0100000000000000;
    end;
    if errors <> 0 then
    begin
      bad := 1;
      writeln(outlog);
      writeln(outlog,errors,' errors have occurred in
ArcTangent. ');
      writeln(outlog,'Processor ',i:2);
      writeln(outlog);
      tot_err:=tot_err+errors;
    end;
  end;
end;

{ This procedure computes the reciprocal(x) for x=.00001,
0<=x<=1 by increments of .1, 10<=x<=100 by increments of 1,
100<=x<=1000 by increments of 10, 1000<=x<=10000 by increments
of 100, 10000<=x<=100000 by increments of 1000, and
```

100000<=x<=1000000 by increments of 10000. If the result of reciprocal(.00001) is off more than 0.0078126 or any other result is off by more than 0.00000045, the result is flagged as bad. When an error occurs, the value of x, the reciprocal(x) computed by the HOST, the reciprocal(x) computed by the FFP, and the difference is written to the outlog. }

```

procedure test_reciprical;
var
  first_pass : boolean;
  line_count,page,loops,errors : integer;
  x,y,diff,multiplier,tolerance : longreal;
begin
  errors := 0;
  tolerance := 0.0078126;
  line_count := 1;
  loops := 0;{required so that the special case of 0.00001 is
not counted}
  first_pass := true;
  page := 1;
  x := 0.00001;
  while (x <= 1000000) do
  begin
    if (not first_pass) then
      tolerance := 0.00000045;
    send(7);
    real_send(x);
    y := real_receive;
    diff := (abs(y-(1/x)));
    if diff > tolerance then
      begin
        errors := errors + 1;
        if line_count = 1 then
          begin
            write(outlog,' X          Reciprical          ');
            writeln(outlog,'FFS Reciprical          Difference',
              ' on ',fpp_page_port[i],
',fpp_page[i]);
            writeln(outlog,'-----');
            line_count := 2;
          end;
        if first_pass then
          begin
            write(outlog,x:9:6,'          ',1/x:9:6);
            writeln(outlog,'          ',y:9:6,'          ',diff:9:6);
          end
        else
          begin
            write(outlog,x:9:6,'          ',1/x:9:6);
            writeln(outlog,'          ',y:9:6,'          ',diff:9:6);
          end;
        end;
      end;
    if first_pass then

```

```

begin
  x := 0.00001;
  multiplier := 0.1;
  first_pass := false;
end;
if loops = 10 then
begin
  multiplier := multiplier * 10.0;
  loops := 1;
end;
x := x + multiplier;
loops := loops + 1;
end;
if errors <> 0 then
begin
  bad := 1;
  writeln(outlog);
  writeln(outlog,errors,' errors in Reciprical function. ');
  writeln(outlog,'Processor ',i:2);
  writeln(outlog);
  tot_err:=tot_err+errors;
end;
end;

```

{ This procedure computes the exp(x) for x=-1.0e-10, x=1.0e-10, and 1<=x<=10 by increments of .1. If the result of exp(x) is off more than 0.009110106 the result is flagged as bad. When an error occurs, the value of x, the exp(x) computed by the HOST, the exp(x) computed by the FFP, and the difference is written to the outlog. }

```

procedure test_exponential;
var
  first_pass, second_pass : boolean;
  line_count,page_number,errors : integer;
  x,y,diff,multiplier,tolerance : longreal;
begin
  errors := 0;
  tolerance := 0.009110106;
  line_count := 1;
  page_number := 1;
  first_pass := true;
  second_pass := true;
  x := -1.0e-8;
  while x <= 10 do
  begin
    send(8);
    real_send(x);
    y := real_receive;
    diff := (abs(y-(exp(x)))));
    if diff > tolerance then
    begin
      errors := errors + 1;
      if line_count = 1 then

```

```

begin
  write(outlog,' X           Exponential ');
  writeln(outlog,'           FFS Exponential
Difference',
           ' on ',fpp_page_port[i],
',fpp_page[i]);
  writeln(outlog,'-----
-----');
  line_count := 2;
end;
if first_pass or second_pass then
  write(outlog,x:8:8,' ',exp(x):11:6)
else
  write(outlog,x:9:1,' ',exp(x):12:6);

  writeln(outlog,' ',y:12:6,' ',diff:9:9);
end;
if first_pass then
begin
  x := 1e-8;
  first_pass := false;
end
else
if second_pass then
begin
  x := 1;
  multiplier := 0.1;
  second_pass := false;
end
else
  x := x + multiplier;
end;
if errors <> 0 then
begin
  bad := 1;
  writeln(outlog);
  writeln(outlog,errors,' errors in Exponential
Function.');
```

```

  writeln(outlog,'Processor ',i:2);
  writeln(outlog);
  tot_err:=tot_err+errors;
end;
end;
```

{ This procedure computes the square\_root(x) for 1<=x<=100 in increments of 1. If the result of square\_root(x) is off more than 0.00000122 the result is flagged as bad. When an error occurs, the value of x, the square\_root(x) computed by the HOST, the square\_root(x) computed by the FFP, and the difference is written to the outlog. }

```

procedure test_square_root;
var
  line_count,page,errors : integer;
```

```

x,y,tolerance,diff : longreal;
begin
  errors := 0;
  tolerance := 0.00000122;
  line_count := 1;
  page := 1;
  x := 0.0;
  while (x <= 100) do
    begin
      send(9);
      real_send(x);
      y := real_receive;
      diff := (abs(y-sqrt(x)));
      if diff > tolerance then
        begin
          errors := errors + 1;
          if line_count = 1 then
            begin
              write(outlog,'X          Local ');
              writeln(outlog,'Sqrt          Remote Sqrt
Difference',
                      ' on ',fpp_page_port[i],
',fpp_page[i]);
              writeln(outlog,'-----
-----');
              line_count := 2;
            end;
            write(outlog,x:5:1,'          ',sqrt(x):11:6);
            writeln(outlog,'          ',y:11:6,'          ',diff:8:8);
          end;
          x := x + 1.0;
        end;
        if errors <> 0 then
          begin
            bad := 1;
            writeln(outlog);
            writeln(outlog,errors,' errors have ocured in Square
Root. ');
            writeln(outlog,'Processor ',i:2);
            writeln(outlog);
            tot_err:=tot_err+errors;
          end;
        end;
      end;
    end;

{ This procedure computes the Ln(x) for x=.00001, 0<=x<=1 in
increments of .1, 1<=x<=10 in increments of 1, 10<=x<=100 in
increments of 10. If the result of Ln(x) is off more than
0.00000122 the result is flagged as bad. When an error occurs,
the value of x, the Ln(x) computed by the HOST, the Ln(x)
computed by the FFP, and the difference is written to the
outlog. }

procedure test_natural_log;
var

```

```

first_pass : boolean;
line_count,page,loops,errors : integer;
x,y,diff,multiplier,tolerance : longreal;
begin
  errors := 0;
  tolerance := 0.00000064;
  line_count := 1;
  loops := 0;{required so that the special case of 0.00001 is
not counted}
  first_pass := true;
  page := 1;
  x := 0.00001;
  while (x <= 100) do
    begin
      send(0);
      real_send(x);
      y := real_receive;
      diff := (abs(y-ln(x)));
      if diff > tolerance then
        begin
          errors := errors + 1;
          if errors = 1 then
            begin
              write(outlog,'X                Local ');
              writeln(outlog,'ln                Remote ln
Difference',
                                ' on ',fpp_page_port[i],
',fpp_page[i]);
              writeln(outlog,'-----
-----');
            end;
            write(outlog,x:9:5,'      ',ln(x):11:6);
            writeln(outlog,'      ',y:11:6,'
',diff:8:8);
          end;
          if first_pass then
            begin
              x := 0.0;
              multiplier := 0.1;
              first_pass := false;
            end;
            if loops = 10 then
              begin
                multiplier := multiplier * 10.0;
                loops := 1;
              end;
              x := x + multiplier;
              loops := loops + 1;
            end;
          if errors <> 0 then
            begin
              bad := 1;
              writeln(outlog);
            end;
        end;
      end;
    end;
  end;
end;

```

```

        writeln(outlog,errors,' errors have occurred in Natural Log
Function. ');
        writeln(outlog,'Processor ',i:2);
        writeln(outlog);
        tot_err:=tot_err+errors;
        end;
end;

```

```
begin
```

```
{ Read the file TESTINFO for the number of times to run this
test. }
```

```

T4:=0;
reset(TESTINFO);
if eof(TESTINFO) then
    writeln('File TESTINFO not found, run TESTSETUP');
if not eof(TESTINFO) then
begin
    readln(TESTINFO,T1,I1);
    readln(TESTINFO,T2,I2);
    readln(TESTINFO,T3,I3);
    readln(TESTINFO,T4,I4);
end;
if T4 = 1 then
begin
    rewrite(outlog);
    writeln(outlog,'*****');
    writeln(outlog,'***** Starting FUNCTION test. *****');
    writeln(outlog,'***** Iterations = ',I4:6,' *****');
    writeln(outlog,'*****');
    writeln(outlog,' ');

```

```
{ Read all the processor addresses from the file funcproc }
```

```

reset(proclist);
i := 0;
while not eof(proclist) do
begin
    i := i + 1;
    readln(proclist,fpp_page_port[i],fpp_page[i]);
end;
numProc := i;
tot_err:=0;

```

```
{ Set-up the screen }
```

```

clearscreen;
gotoxy(8,2);
write(' ** Floating Point Processor Function Board Test **
');
gotoxy(0,20);
write('Testing : ');

```

```

gotoxy(0,21);
write('No. of cycles through all boards : ');
gotoxy(0,7);
write('Processor : ');

{ Write the number of each processor to be tested on the
screen. }

for i := 1 to NumProc do
begin
  if i > 48 then
    gotoxy(13+((i-49)*3),16)
  else if i > 32 then
    gotoxy(13+((i-33)*3),13)
  else if i > 16 then
    gotoxy(13+((i-17)*3),10);
  write(i:3);
end;

{ Loop for the number of times specified in TESTINFO. }

cntr:=0;
while cntr < I4 do
begin
  cntr:=cntr+1;

{ Run all function tests on each processor. }

  for i := 1 to numproc do
  begin
    gotoxy(36,21);
    write('          ');
    gotoxy(36,21);
    write(cntr-1);
    bad := 0;

{ Set-up a communication channel to the processor being
tested. }

    initialize(fpp_page_port[i],fpp_page[i]);
    turn_on_repeater(fpp_page_port[i],fpp_page[i]);

{ Write an 'M' under the processor number that is to be
tested. }

    if i > 48 then
      gotoxy(13+((i-49)*3),17)
    else if i > 32 then
      gotoxy(13+((i-33)*3),14)
    else if i > 16 then
      gotoxy(13+((i-17)*3),11)
    else
      gotoxy(13+((i-1)*3),8);
    write(' M');

```

```
{ Write the function to be tested to the screen, then run that function test. }
```

```
gotoxy(11,20);  
write('Sin          ');  
test_sine;  
gotoxy(11,20);  
write('Cos          ');  
test_cosine;  
gotoxy(11,20);  
write('Tan          ');  
test_tangent;  
gotoxy(11,20);  
write('Asin         ');  
test_arcsine;  
gotoxy(11,20);  
write('Acos         ');  
test_arccosine;  
gotoxy(11,20);  
write('Atan         ');  
test_arctangent;  
gotoxy(11,20);  
write('Recip        ');  
test_reciprical;  
gotoxy(11,20);  
write('Exp          ');  
test_exponential;  
gotoxy(11,20);  
write('Square Root ');  
test_square_root;  
gotoxy(11,20);  
write('Natural Log ');  
test_natural_log;
```

```
{ Close the communication channel to the processor. }
```

```
stop_processor;  
turn_off_repeater(fpp_page_port[i]);
```

```
{ If any result was bad (was out of tolerance), an 'E' is written over the 'M' under the processor just tested. If all results passed, then the 'M' is blanked out. }
```

```
if i > 48 then  
  gotoxy(13+((i-49)*3),17)  
else if i > 32 then  
  gotoxy(13+((i-33)*3),14)  
else if i > 16 then  
  gotoxy(13+((i-17)*3),11)  
else  
  gotoxy(13+((i-1)*3),8);  
if bad <> 0 then  
  write(' E');
```

```
        if bad = 0 then
            write(' ');
        end;
    end;
```

```
{ The running total of errors is written to the outlog after
the program finishes. }
```

```
        writeln(outlog, 'Number of errors = ', tot_err);
        writeln(outlog, ' ');
    end;
end.
```

```
program hostio;  
var x : real;  
begin  
  while true do  
    begin  
      receive(host,x);  
      send(host,x);  
    end;  
  end.  
end.
```

```
program send;
var mode,x : real;
begin
  x := 0.0;
  while true do
  begin
    receive(host,x);
    send(network,x);
    receive(network,x);
    send(host,x);
  end;
end.
```

```
program receive;
var mode,x : real;
begin
  while true do
    begin
      receive(network,x);
      send(host,x);
      send(network,x);
      send(host,x);
    end;
  end.
end.
```

{ This program for the processor is the first cut at the T3 diagnostic test. T3 takes a list of processors and beginning with the first processor in the list it sends a value to all the other processors. Then the next processor does the same thing as does the whole list and then it repeats. Each transfer (crossbar cycle) the sender/receiver status is determined by the mode bit (0-send, 1-receive) sent to each processor by the host. }

```
program testxbar;  
var mode,x : real;  
begin
```

```
    while true do  
        begin  
            receive(host,mode);  
  
{receive sender/receiver status from the host}  
  
            if mode = 1.0 then  
                begin  
                    receive(host,x);  
  
{if 1 receive value to send from the host}  
  
                    send(network,x);  
  
{and send it accross the network}  
  
                end  
            else  
                begin  
                    receive(network,x);  
  
{if 0 receive the value from the network}  
  
                    send(host,x);  
  
{and send it back to the host}  
  
                end;  
            end;  
        end.  
end.
```

```
program function;  
{
```

```
    Program      : Function.PAS  
    Programmer   : Richard A. Armstrong  
    Date         : 2/20/89  
    Execution State : To be run as a remote process
```

```
    Purpose      : The purpose of this program is to receive  
an integer from a host program, then to receive a real number  
and based on what integer was passed, return a real number  
calculated by the function specified by the integer.
```

```
}
```

```
var
```

```
    funct : integer;  
    x : real;
```

```
begin
```

```
    while true do  
        begin
```

```
            funct := 0;  
            receive_lsw(host, funct);  
            receive(host, x);  
            if funct = 1 then  
                send(host, sin(x))  
            else if funct = 2 then  
                send(host, cos(x))  
            else if funct = 3 then  
                send(host, tan(x))  
            else if funct = 4 then  
                send(host, asin(x))  
            else if funct = 5 then  
                send(host, acos(x))  
            else if funct = 6 then  
                send(host, atan(x))  
            else if funct = 7 then  
                send(host, 1/x)  
            else if funct = 8 then  
                send(host, exp(x))  
            else if funct = 9 then  
                send(host, sqrt(x))  
            else if funct = 0 then  
                send(host, ln(x));
```

```
        end;
```

```
    end.
```

;pretest

; RUN FPPMU

%0/spock/fpp/amd/reset  
%0/spock/fpp/amd/load(proclist=fppmuproc)  
%0fppmu(outlog=:LP:,proclist=fppmuproc)

; RUN T2

%1/spock/fpp/amd/reset  
%1/spock/reset/resetxbar 5  
%1/spock/reset/resetxbar 4  
%1/spock/fpp/amd/load(proclist=t2proc)  
%1/spock/util/sxbload 5 t2.seq.lft t2.xbar.lft  
%1/spock/util/sxbload 4 t2.seq.rgt t2.xbar.rgt  
%1/spock/util/masseq 5  
%1t2(outlog=:LP:,proclist=t2proc)

; RUN T3

%2/spock/fpp/amd/reset  
%2/spock/reset/resetxbar 4  
%2/spock/reset/resetxbar 5  
%2/spock/fpp/amd/load(proclist=t3proc)  
%2/spock/util/sxbload 5 t3.seq.lft t3.xbar.lft  
%2/spock/util/sxbload 4 t3.seq.rgt t3.xbar.rgt  
%2/spock/util/masseq 5  
%2t3(outlog=:LP:,proclist=t3proc)

; RUN FUNCTION

%3/spock/fpp/amd/reset  
%3/spock/fpp/amd/load(proclist=funcproc)  
%3ftest(outlog=:LP:,proclist=funcproc)

;pretest

; RUN FPPMU

%0/spock/fpp/amd/reset  
%0/spock/fpp/amd/load(proclist=fppmuproc)  
%0fppmu(outlog=:LP:,proclist=fppmuproc)

; RUN T2

%1/spock/fpp/amd/reset  
%1/spock/reset/resetxbar 5  
%1/spock/reset/resetxbar 4  
%1/spock/fpp/amd/load(proclist=t2proc)  
%1/spock/util/sxbload 5 t2.seq.lft t2.xbar.lft  
%1/spock/util/sxbload 4 t2.seq.rgt t2.xbar.rgt  
%1/spock/util/masseq 5  
%1t2(outlog=:LP:,proclist=t2proc)

; RUN T3

%2/spock/fpp/amd/reset  
%2/spock/reset/resetxbar 4  
%2/spock/reset/resetxbar 5  
%2/spock/fpp/amd/load(proclist=t3proc)  
%2/spock/util/sxbload 5 t3.seq.lft t3.xbar.lft  
%2/spock/util/sxbload 4 t3.seq.rgt t3.xbar.rgt  
%2/spock/util/masseq 5  
%2t3(outlog=:LP:,proclist=t3proc)

; RUN FUNCTION

%3/spock/fpp/amd/reset  
%3/spock/fpp/amd/load(proclist=funcproc)  
%3ftest(outlog=:LP:,proclist=funcproc)

;pretest

; RUN FPPMU

%0/spock/fpp/amd/reset  
%0/spock/fpp/amd/load(proclist=fppmuproc)  
%0fppmu(outlog=:LP:,proclist=fppmuproc)

; RUN T2

%1/spock/fpp/amd/reset  
%1/spock/reset/resetxbar 5  
%1/spock/reset/resetxbar 4  
%1/spock/fpp/amd/load(proclist=t2proc)  
%1/spock/util/sxbload 5 t2.seq.lft t2.xbar.lft  
%1/spock/util/sxbload 4 t2.seq.rgt t2.xbar.rgt  
%1/spock/util/masseq 5  
%1t2(outlog=:LP:,proclist=t2proc)

; RUN T3

```
%2/spock/fpp/amd/reset
%2/spock/reset/resetxbar 4
%2/spock/reset/resetxbar 5
%2/spock/fpp/amd/load(proclist=t3proc)
%2/spock/util/sxbload 5 t3.seq.lft t3.xbar.lft
%2/spock/util/sxbload 4 t3.seq.rgt t3.xbar.rgt
%2/spock/util/masseq 5
%2t3(outlog=:LP:,proclist=t3proc)
```

```
; RUN FUNCTION
```

```
%3/spock/fpp/amd/reset
%3/spock/fpp/amd/load(proclist=funcproc)
%3ftest(outlog=:LP:,proclist=funcproc)
```

```
;pretest
```

```
; RUN FPPMU
```

```
%0/spock/fpp/amd/reset
%0/spock/fpp/amd/load(proclist=fppmuproc)
%0fppmu(outlog=:LP:,proclist=fppmuproc)
```

```
; RUN T2
```

```
%1/spock/fpp/amd/reset
%1/spock/reset/resetxbar 5
%1/spock/reset/resetxbar 4
%1/spock/fpp/amd/load(proclist=t2proc)
%1/spock/util/sxbload 5 t2.seq.lft t2.xbar.lft
%1/spock/util/sxbload 4 t2.seq.rgt t2.xbar.rgt
%1/spock/util/masseq 5
%1t2(outlog=:LP:,proclist=t2proc)
```

```
; RUN T3
```

```
%2/spock/fpp/amd/reset
%2/spock/reset/resetxbar 4
%2/spock/reset/resetxbar 5
%2/spock/fpp/amd/load(proclist=t3proc)
%2/spock/util/sxbload 5 t3.seq.lft t3.xbar.lft
%2/spock/util/sxbload 4 t3.seq.rgt t3.xbar.rgt
%2/spock/util/masseq 5
%2t3(outlog=:LP:,proclist=t3proc)
```

```
; RUN FUNCTION
```

```
%3/spock/fpp/amd/reset
%3/spock/fpp/amd/load(proclist=funcproc)
%3ftest(outlog=:LP:,proclist=funcproc)
```

```
;pretest
```

```
; RUN FPPMU
```

```

%0/spock/fpp/amd/reset
%0/spock/fpp/amd/load(proclist=fppmuproc)
%0fppmu(outlog=:LP:,proclist=fppmuproc)

; RUN T2
%1/spock/fpp/amd/reset
%1/spock/reset/resetxbar 5
%1/spock/reset/resetxbar 4
%1/spock/fpp/amd/load(proclist=t2proc)
%1/spock/util/sxbload 5 t2.seq.lft t2.xbar.lft
%1/spock/util/sxbload 4 t2.seq.rgt t2.xbar.rgt
%1/spock/util/masseq 5
%1t2(outlog=:LP:,proclist=t2proc)

; RUN T3

%2/spock/fpp/amd/reset
%2/spock/reset/resetxbar 4
%2/spock/reset/resetxbar 5
%2/spock/fpp/amd/load(proclist=t3proc)
%2/spock/util/sxbload 5 t3.seq.lft t3.xbar.lft
%2/spock/util/sxbload 4 t3.seq.rgt t3.xbar.rgt
%2/spock/util/masseq 5
%2t3(outlog=:LP:,proclist=t3proc)

; RUN FUNCTION

%3/spock/fpp/amd/reset
%3/spock/fpp/amd/load(proclist=funcproc)
%3ftest(outlog=:LP:,proclist=funcproc)

;pretest

; RUN FPPMU

%0/spock/fpp/amd/reset
%0/spock/fpp/amd/load(proclist=fppmuproc)
%0fppmu(outlog=:LP:,proclist=fppmuproc)

; RUN T2
%1/spock/fpp/amd/reset
%1/spock/reset/resetxbar 5
%1/spock/reset/resetxbar 4
%1/spock/fpp/amd/load(proclist=t2proc)
%1/spock/util/sxbload 5 t2.seq.lft t2.xbar.lft
%1/spock/util/sxbload 4 t2.seq.rgt t2.xbar.rgt
%1/spock/util/masseq 5
%1t2(outlog=:LP:,proclist=t2proc)

; RUN T3

%2/spock/fpp/amd/reset
%2/spock/reset/resetxbar 4

```

```
%2/spock/reset/resetxbar 5
%2/spock/fpp/amd/load(proclist=t3proc)
%2/spock/util/sxbload 5 t3.seq.lft t3.xbar.lft
%2/spock/util/sxbload 4 t3.seq.rgt t3.xbar.rgt
%2/spock/util/masseq 5
%2t3(outlog=:LP:,proclist=t3proc)
```

```
; RUN FUNCTION
```

```
%3/spock/fpp/amd/reset
%3/spock/fpp/amd/load(proclist=funcproc)
%3ftest(outlog=:LP:,proclist=funcproc)
```

1 200  
1 25  
1 840  
1 250