RCSL No: 30-M270

Edition: April 1981

Author: Lars Bone Jørgensen

Title:

CPU820 MICROPROGRAMMED TESTS
User's Guide

RCSL 42-i1592

A/S REGNECENTRALEN
af 1979

Keywords:

CPU820, test, microprogrammed tests, power up test, CPU test, memory test, cache test.

Abstract:

This manual is a user's guide to CPU820 microprogrammed tests.

(32 printed pages)

Printed by A/S Regnecentralen af 1979, Copenhagen

RCSL 42-i1592

TABLE OF CONTENTS PAGE

1. INTRODUCTION 1.

The microtest consists of two parts.

The first part is started after power up, and ensures that the TCP (technician's control panel) is functioning properly.

The second part is a more complete test which is divided in a CPU test, a Memory test, and a Cache test. These tests can be controlled either by the OCP or the TCP.

2. POWER UP TEST 2.

This is a very simple test with the purpose of checking the TCP. The test is using the three indicators on the front panel of the CPU821.

Registers and operations used in the TCP handling is tested, and the test is terminated by checking the TCP input and output procedures.

Description

The test is started at power up if the second mode switch on the CPU821 PCBA is set (fig. 1).

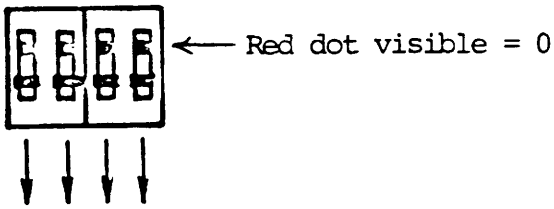
At first the three indicators on the front panels are tested, and a loop is entered where the four different u-instruction cycle lengths are tested (fig. 1).

Pressing the autoload botton on the OCP makes the test continue.

Three different tests are executed and in case of errors the erroneous testnumber is displayed by the indicators, while the test is looping.

The CPU is continuing with the tests until the autoload button is pressed again.
Then the test writes the text "TYPE OCTAL NUMBER:" on the TCP, and the operator is supposed to enter an octal number terminated by a <CR>.

The number is echoed by the test, and in that way the input and output procedures are tested.



0 0 0 0	Normal mode of operation
1 X X X	Power restart
X 1 X X	Power-up test mode
X X 1 X	Continous test mode
X X X 1	Console used as TC

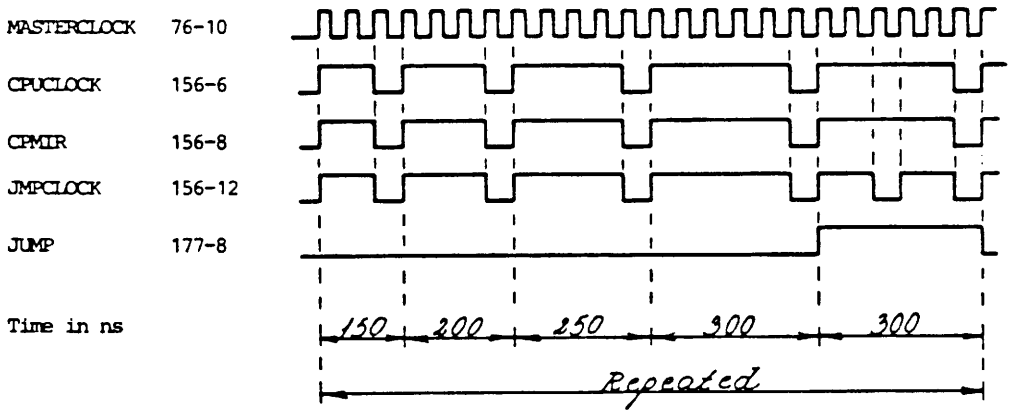


Figure 1: Mode switches on the CPU821, and timing diagram.

3. MICROPROGRAMMED TEST 3.

3.1 Operating via OCP 3.1

The test is started when the autoload botton at the OCP is activated. Depending on the third mode switch on the CPU821 (fig. 1) the test is either run in short mode or in normal mode. In the latter case the test is performed continuously.

The test is broken and restarted if the autoload botton is activated.

On the front panel of the CPU821 there is a switch to disable the test, in case the coredump produced after autoload is needed.

3.2 Operating via TCP 3.2

The test is started with the following commands typed on the TCP:

T0: CPU test, Memory test normal mode, Cache test
T1: CPU test, Memory test short mode, Cache test
T2: CPU test
T3: Memory test normal mode
T4: Memory test short mode
T5: Cache test
T6: Clear and size memory W3 = last location in memory
T7: Loop to adjust deskew delay.

Indications

Before every test the TCP bell is activated, and the indicators on the front edge of CPU821 are showing the kind of the current test.

Break

The test is breaked if any command is typed on the TCP.

3.3 CPU Test 3.3

The CPU test consists of nine independent tests, each formed by a number of test loops.

After an error is detected it is possible to loop in the erroneous test.

1) Test of the immediate u-instruction.
   The Q-register is loaded with pattern of shifting 'zeroes', and checked.

2) Test of arithmetic and logic operations.
   The arithmetic and logic operations of the ALU are checked.

3) Test of the ALU-registers, the scratch-pad-file, and the internal registers IC and LC.
   A one is shifted through both registerfiles and the IC and LC registers.
   The ALU-register stack is read both from the A and B files.

4) Test of shift operations.
   Doubleshift is tested. All values of the SI-field (Shift input) is tested with both left and right shifts.

5) Test of jumpconditions.
   It is tested that all jumpconditions, controlled by microinstructions, can be set and cleared.

6) Test of halfword manipulator.
   The halfword manipulator functions are tested with odd and even i/o address, and with two different datapatterns.

7) Test of external interrupts.
   It is tested that interrupt level 0-7 does not set interrupt.
   It is also verified that simulating interrupt on level 8-63 set the proper interrupt level, and that the interrupt level is able to clear the simulated interrupt.

8) Test of registers defined by instruction register, and test of prefetch.
Four different instructions are stored in the locations 0-6. The instructioncounter is set to 0, the instructionregister is loaded and a prefetch is started. The W-register according to the W-field in the instructionregister is tested and a new prefetch is started.
In that way all four values of the W-field is tested.
The WPRE-field, X-field, DISP, and DISP+IC are tested in a similar way.

9) Test of the i/o protection system.
The value of bit 10:11 and bit 20 of the i/o statusregister are tested after read with protection and write with protection.
The i/o addresses and the protection registers have the value shown in fig. 2.

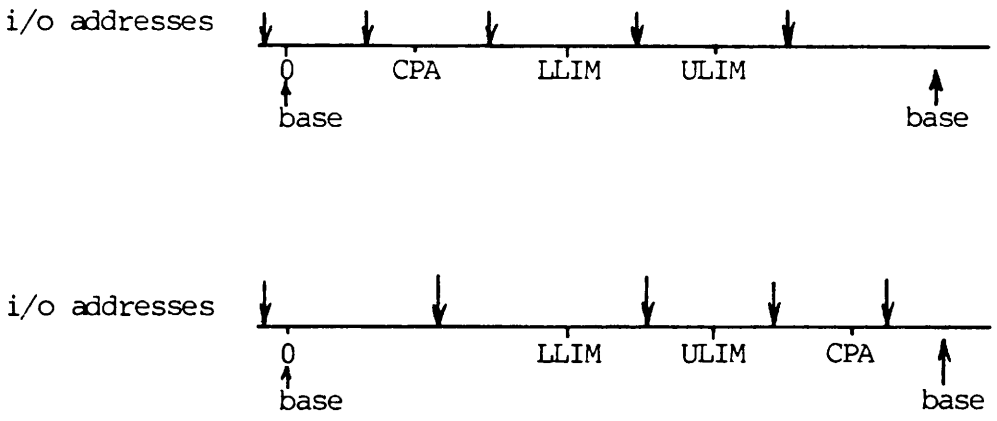


Figure 2: The value of i/o addresses and protection registers.

3.4 Memory Test 3.4

3.4.1 Normal Mode 3.4.1

The total number of i/o operations performed by the memory test is 6*D*N*MEMSIZE, where N = number of addressbits and D = number of different datapatterns.

After the program is started the memory is loaded with 0's. The N address bits indicates up to 2**N = MEMSIZE words in the memory, each address is read and verified to be all 0's.

Then a single 1 is substituted in one bit position, and the altered word is written back in the same location. Finally the word is reread to verify that the newly entered 1 is still there. This is repeated for each of the 24 data bits.

After the word has been filled with 1's the first bit position is substituted with a 0, and this is repeated until the word is all 0's again.

So far 48*MEMSIZE*3 i/o operations has been counted, and the whole procedure is repeated but reading the memory in reverse order from MEMSIZE to 8 giving 48*MEMSIZE*6 operations.

The factor N is accounted for by repeating these series of tests N times using a different bit for incrementing through all addresses by 2, by 4, by 8, and so on.

3.4.2 Short Mode 3.4.2

In this version the 48 data patterns is reduced to two patterns:

1010 .....1,0 and

0101 0,1

Incrementing through the addresses in reduced to N/2 times.

This reduces the total number of operations to 6*MEMSIZE*N.

```
integer procedure getdata (cause);
integer cause;
begin
  if short mode then
  begin
    newdata: = if cause = 1 then 8'25252525 else
               8'52525252;
  end else
  begin
    newdata: = newdata shift 1;
    if cause <=24 then cause: = cause + 1;
  end
end getdata;

comment size memory;

for ic: = 8, ic+2 while -,ioerror and DATI = DATO do
begin
  DATO: = ic;
  write (ic, DATO);
  read (ic, DATI);
end;

comment clear memory;

for wrkl: = ic, wrkl-2 while wrkl>8 do
  write(wrkl,0),
W3: = ic-2; comment last location in memory
comment start of MEM test
olddata: = 0 ; newdata: = 0; size: = ic;
start: = 8; stop: = size;
W1: = 2;
while W1 <size do
```

```
begin
  for down: = false, true do
  begin
    for cause: = 0 step 1 until if shortmode then 1 else 47 do
    begin
      newdata: = getdata (cause);
      if -, down then
      begin
        for W0: = 8 step 2 until W1-2 do
        for ic: = w0 step W1 until last do
        begin
          read (ic, DATI);
          if ioerror then ERROR;
          write (ic, newdata);
          if ioerror then ERROR;
          if DATI <> olddata then olddataerror;
          read (ic, DATI);
          if ioerror then ERROR;
          if DATI <> newdata then newdataerror;
        end for ic;
        olddata: = newdata;
      end <*if up*>
      else
      begin
        for W0: = last step -2 until last - W1 + 2 do
        for ic: = W0 step -W1 until 8 do
        begin
          read (ic, DATI);
          if ioerror then ERROR;
          write (ic, newdata);
          if ioerror then ERROR;
          if DATI <> olddata then olddataerror;
          read (ic, DATI);
          if ioerror then ERROR;
          if DATI <> newdata then newdataerror;
        end for ic;
        olddata: = newdata;
      end down;
    end for cause;
```

```
    W1: = if shortmode then W1 shift 2 else
          W1 shift 1;
end while W1 < size;
```

3.5 Cache Test 3.5

The cache memory is initialized in a consecutive area of 4 k words (the hit area) starting at location 0, by means of testwrite.

The values are 0, 2, 4 ...8'20000 and the valid bit is set.

The memory is checked from the start of the hit area to the top of memory and from location 0 to the start of hit area.

In the hit area the contents of memory locations and hit status are checked.

A new value is written in the same location and reread in a mode where the cache memory is bypassed to see if the writing was sucessful.

Outside the hit area it is checked that the status was a miss, and that the contents of memory are unchanged.

The test is repeated with the cache memory initialized from 4 k to 8 k, and so on, until the top of memory.

If more than one set is present the same test is run with the second set.

If two sets are present it is checked that reading two words with a difference in addresses of 4 k, will initialize both sets.

```
If CAM -, available then goto continue;

Count-and-init-chache-modules;
size-and-char-mem;
for set: = 0, 1 do
begin
mask: = if set = 0 then bit 17 else bit 19;
if set_present then
begin
  camcontrolregister: = if set = 0 then 8'50 else 8'30;
  comment set + testwrite;

  for startwindow: = 0 step 8'20000 until last do
  begin
    for wrk: = 0 step 2 until 8'20000 do
    begin
      CAMTD: = wrk; <*cam testdata*>
      read ( startwindow + wrk, DATI);
      <*write testdata*>
    end;
    camcontrolregister: = camcontrolregister or 3;
    for ic: = startwindow step 2 until startwindow + size do
    begin
      if ic mod size < startwindow then
      begin
        read (ic mod size, DATI, IOSTATUS);
        if IOSTATUS and mask = 0 then camhiterror;
        if DATI <> ic then camreaderror;
      end else
      if ic <startwindow + 8'20000 then
      begin
        read (ic, DATI, IOSTATUS);
        if IOSTATUS and mask <>0 then camhiterror
        if DATI <> ic - startwindow then camreaderror;
        write (ic, ic);
        cambypass: = 1;
        read (ic, DATI);
```

```
            if DATI <> ic then camwriteerror;
            cam bypass: = 0;
          end else
          begin
            read (ic, DATI, IOSTATUS);
            if IOSTATUS and mask = 0 then camhiterror;
            if DATI <> 0 then camreaderror;
          end
        end for ic;
      end for startwindow;
      size_and_clear_mem;
    end if set present;
    end for set;
```

3.6 Error Messages

3.6.1 Operating via OCP

In case of errors the autoload lamp will begin to gleam with a period of 0.5 sec.

3.6.2 Operating via TCP

In case of errors the following text will be displayed:

```
ERR
C(wrk0)
C(wrk1)
C(wrk2)
C(IC)
```

C(wrk0) is the contents of the register wrk0, which is the number of the erroneous test.

After an error it is possible to proceed in three ways:

1) Type P at the TCP: Proceed the test.

2) Type N at the TCP: Loop in the test.
The subsequent error messages are suppressed.

3) Type a TCP command: The test is broken and the command is executed.

3.6.3 Description of Error Types 3.6.3

-2 : Buserror
(wrk1 1=NACK 2=timeout 4=parity)
IC : Address

---

0 : Newdata error in MEM test
wrk1 = read data, wrk2 = expected data
IC : address

---

1 : Olddata error in MEM test
wrk1 = read data, wrk2 = expected data
IC : address

---

2 : Hit error in CAM test
wrk1 : bit 17, 19 of iostatus
wrk2 : expected value, IC : address

---

3 : Read error in CAM test
wrk1 : read data, wrk2 : expected data
IC : address

---

4 : Write error in CAM test
Error when writing data memory
wrk1 : written data, wrk2 : expected data
IC : address

---

5 : No hit with two sets in CAM test.
With two sets present it is not possible to load the same location in both sets.
wrk1 : status of first i/o status.
wrk2 : status of second i/o status.

10-40 : Test of immediate operand instructions
wrk1 : read data, wrk2 : expected data

100-246 : Test of arithmetic and logic operations.
Operations and datapatterns can be found in the
u-program listing.
wrk1 : result of operation, wrk2 : expected result.

300-440 : Test of ALU- and Scratchpad registers,
IC and LC registers.
wrk1 : contents of register, wrk2 : expected contents.

500-507 : Test of u-instruction shiftoperations
wrk1 : result of shift, wrk2 : expected result

500 : Test of double shift, wrk1,Q = 0,8'40000000
is shifted left.

501 : Test of shift input (zero)
wrk1 = -1 is shifted left with S1 = zero

502 : Test of shift input (zero)
wrk1 = -1 is shifted right with S1 = zero

503 : Test of shift input (shiftlink)
wrk1 = 8'40000000 is shifted left twice.
First time with S1 = zero, second with S1 = shift link.

504 : Test of shift input (shift link)
wrk1 = 1 is shifted right twice.
First time with S1 = zero, second with S1 = shift link.

505 : Test of shift input (add condition)
Addcondition: = 1. wrk1 = 0 is shifted left with
S1 = addcondition.

506 : Test of shift input (F(0))
wrk1 : = 8'40000000 and F(0): = 1
wrk1 is shifted right with S1 = F(0).

507 : Test of shift input (sign)
wrk1: = wrk2: = 8'40000000.
wrk1: = wrk1 + wrk2
wrk1 is shifted right with S1 = sign (sign = F10) error overflow).

540-555 : Test of jumpconditions

540 : Condition = DBUS(0) (negative)
wrk2: = 0, test condition is false
wrk2: = 8'40000000, test condition is true.

541 : Condition is F <> 0 (non zero)
wrk2: = 0, test condition is false
wrk2: = 1> test condition is true.

542 : Condition is arithmetic overflow
wrk2: = 0-8'40000000, test condition is true.
wrk2: = 8'40000000+1, test condition is false.

543 : Condition is carry from ALU bit(0)
wrk2: = -1 + 1, test condition is true
wrk2: = 0 + 1, test condition is false.

544 : Condition is F(0) exor overflow (sign)
wrk2: = 8'40000000 - 1, test condition is true.
wrk2: = 8'37777777 + 1, test condition is false.

545 : Condition is DBUS(0) <> DBUS(1)
wrk2: = 8'60000000, test condition is false.
wrk2 = 0, test condition is false.
wrk2: 8'40000000, test condition is true.
wrk2: = 8'20000000, test condition is true.

```
546 : Condition is DBUS(1) <> DBUS(2)
      wrk2: = 8'30000000, test condition is false
      wrk2: = 0, test condition is false.
      wrk2: = 8'20000000, test condition is true.
      wrk2: = 8'10000000, test condition is true.

547 : Condition is link <> carry
      wrk2: = (8'40000000+1) shift 1, test conditions is true.
      wrk2: = (-1-1) shift 1, test condition is false.

550 : Condition is LC(0) = 1 (loop counter <0)
      wrk2: = LC: = 0, test condition is false.
      wrk2: = LC: = 8'40000000, test condition is true.

551 : condition is LC(0) = 1 or maxloop (0) = 0
      wrk2: = 48; LC: = 8'40000000 (maxloop is loaded too)
      repeat
        test condition is false;
        wrk2: = wrk2 -1;
        LC: = LC -1;
      until wrk2 = 0;
      test condition is true;

552 : condition is LADR(0:20)<>0 (not w-address)
      wrk2: LAST: = 0; test condition is false.
      wrk2: LAST: = 8'10, test condition is true.

553 : condition is LADR(0) = 1 (not memory address)
      wrk2: = LAST: = 0, test condition is false.
      wrk2: = LAST: = 8'40000000, test condition is true.

554 : condition is CPUST(0) = 1 (monitormode)
      wrk2: = CPUST(1): = 8'40000000, test condition is true.
      wrk2: = CPUST: = 0, test condition is false.

555 : condition is CPUST(1) = 1 (escapemode)
      wrk2: = CPUST: = 8'20000000, test condition is true.
      wrk2: = CPUST: = 0, test condition is false.
```

600-637 Test of halfword manipulator wrk1 = result, wrk2 = expected result.

600 : Function = load halfword = 0, odd = 1
(SBUS(0:23) = 12 ext 0 con DBUS(12:23))
datapattern = 8'7777;

601 : Function = load halfword, odd = 1
datapattern = 8'7777;

602 : Function = load halfword, odd = 0
(SBUS(0:23) = 12 ext 0 con DBUS(0:11))
datapattern = 8'7777

603 : Function = load halfword, odd = 0
datapattern = 8'77770000.

604 : Function = extended load = 1, odd = 1
(SBUS(0:23) = 12 ext DBUS(12) con DBUS(12:23))
datapattern = 8'7777.

605 : Function = extended load, odd = 1
datapattern = 8'77770000.

606 : Function = extended load, odd = 0
(SBUS(0:23) = 12 ext DBUS(0) con DBUS(0:11))
datapattern = 8'7777.

607 : Function = extended load, odd = 0
datapattern = 8'7777.

610 : Function = store halfword = 2, odd = 1
(SBUS(0:23) = 12 ext 0 con DBUS(12:23))
datapattern = 8'7777.

611 : Function = store halfword, odd = 1
datapattern = 8'77770000.

```
612 : Function = store halfword, odd = 0
      (SBUS(0:23) = DBUS(12:23) con 12 ext 0))
      datapattern = 8'7777.

613 : Function = store halfword, odd = 0
      datapattern = 8'77770000.

614 : Function = set mask = 3, odd = 1
      (SBUS(0:23) = 8'77770000)
      datapattern = 8'7777.

615 : Function = set mask, odd = 1
      datapattern = 8'77770000.

616 : Function = set mask, odd = 0
      (SBUS(0:23) = 8'7777)
      datapattern = 8'7777.

617 : Function = set mask, odd = 0
      datapattern + 8'77770000.

620,622 : Function = extend halfword = 4
          (SBUS(0:23) = 12 ext DBUS(12) con DBUS(12:23))
          datapattern = 8'7777.

621,623 : Function = extend halfword
          datapattern = 8'77770000.

624,626 : Function = swop word = 5
          (SBUS(0:23) = DBUS(12:23) con DBUS(0:12))
          datapattern = 8'7777.

625,627 : Function = swop word
          datapattern = 8'77770000.

630,632 : Function = swop left halword = 6
          (SBUS(0:23) = 12 ext 0 con DBUS(0:11))
          datapattern = 8'7777.
```

631,633 : Function = swop left halfword
datapattern = 8'77770000.

634,636 : Function = swop right halfword
(SBUS(0:23) = DBUS(12:23) con 12 ext 0)
datapattern 8'7777.

635,637 : Function = swop right halfword
datapattern = 8'77770000.

701-713 : Test of external interrupts.

701 : Interrupt occurred with ILEV = 0-7
wrk2 = simulated interrupt level.

711 : Interrupt did not occur when ILEV = 8-63
wrk2 = simulated interrupt level.

712 : The received ILEV and the simulated ILEV are not equal.
wrk1 = received ILEV, wrk2 = simulated ILEV.

713 : The simulated interrupt can not be cleared
wrk2 = simulated ILEV.

1000-1023 : Test of registers defined by instruction register, and
test of prefetch instructioncounter.
wrk1 = contents of register,
wrk2 = expected contents.

1000 : Test of W-register
W-FIELD(0:1) = 0

1001 : Test of W-register
W-FIELD(0:1) = 1

1002 : Test of W-register
W-FIELD(0:1) = 2

1003 : Test of W-register
W-FIELD(0:1) = 3

1004 : Test of WPRE-register
W-FIELD(0:1) = 0

1005 : Test of WPRE-register
W-FIELD(0:1) = 1

1006 : Test of WPRE-register
W-FIELD(0:1) = 2

1007 : Test of WPRE-register
W-FIELD(0:1) = 3

1010 : Test of X-register
X-FIELD(0:1) = 0

1011 : Test of X-register
X-FIELD(0:1) = 1

1012 : Test of X-register
X-FIELD(0:1) = 2

1013 : Test of X-register
X-FIELD(0:1) = 3

1014 : Test of DISP_register
DISP = 12

1015 : Test of DISP-register
DISP = 0

1016 : Test of DISP-register
DISP = 2

1017 : Test of DISP-register
DISP = 4

1020 : Test of DIC-register (DISP+IC)
DISP = -2, IC = 8'37777776.

1021 : Test of DIC-register
DISP = 0, IC = 0

1022 : Test of DIC-register
DISP = 2, IC = 2

1023 : Test of DIC-register
DISP = 4, IC = 4

1100-1127 : Test of i/o protection system.
wrk1 = bit(10:11) + bit(20) of i/o status register.
wrk2 = expected bit(10:11) + bit(20)
IC = Base register
LAD: logical address
CPA: common protected area register
LLIM: lower limit register
ULIM: upper limit register

- -

1100 : LAD <0<CPA<LLIM<ULIM
operation = read with protection

1101 : LAD <0<CPA<LLIM<ULIM
operation = write with protection

1102 : 0<LAD<CPA<LLIM<ULIM
operation = read with protection

1103 : 0<LAD<CPA<LLIM<ULIM
operation = write with protection

1104 : 0<CPA<LAD<LLIM<ULIM
operation = read with protection

1105 : 0<CPA<LAD<LLIM<ULIM
operation = read with protection

1106 : 0<CPA<LLIM<LAD<ULIM
operation = read with protection

1107 : 0<CPA<LLIM<LAD<ULIM
operation = write with protection

1110,1112 : 0<CPA<LLIM<ULIM<LAD
operation = read with protection

1111,1113 : 0<CPA<LLIM<ULIM<LAD
operation = write with protection

1114 : LAD<0<LLIM<ULIM<CPA
operation = read with protection

1115 : LAD<0<LLIM<ULIM<CPA
operation = write with protection

1116,1120 : 0<LAD<LLIM<ULIM<CPA
operation = read with protection

1117,1121 : 0<LAD<LLIM<ULIM<CPA
operation = write with protection

1122 : 0<LLIM<LAD<ULIM<CPA
operation = read with protection

1123 : 0<LLIM<LAD<ULIM<CPA
operation = write with protection

1124 : 0<LLIM<ULIM<LAD<CPA
operation = read with protection

1125 : 0<LLIM<ULIM<LAD<CPA
operation = write with protection

1127 : 0<LLIM<ULIM<CPA<LAD
operation = write with protection

1177 : Error in BASE register
a wrong absolute address is calculated.

A. LIST OF THE TCP COMMANDS A.

Commands for u-program diagnostic:

R<loc><CR>

RUN. Starts u-instruction execution from <loc>. (R0 <CR>, will reset the CPU as by power up)

XW<CR>

Examine the six working registers used in the u-program. If you are interested in the contents of the registers, the command must be entered first, as the registers are used in the common TCP procedures.

T<testnumber><CR>

TEST. Start execution of the test with the number <test numbers>

RC8000 commands:

When the CPU is running it is only possible to stop it from the TCP by typing the character <BELL> (=cntr G).

XR<no><CR>

Examine register number <no>.

LR<no> : <cont><CR>

Loads register <no> with the value <cont>

XM<loc><CR>

Examine memory location <loc>.

XN<no><CR>

Examines <no> memory locations, starting after the last location inspected.

LM<loc> : <cont><CR>

Load memory location <loc> with the value <cont>.

LN<CR>

Load the next memory location.

XD<CR>

Examine the eight dynamic registers (W0, W1, W2, W3, STAT, IC, CAUSE, SB)

XS<CR>

Examine the eight static registers (CPA, BASE, LLIM, ULIM, ILIM, INF, SIZE, MONTOP)

RETURN LETTER

Title: CPU820 MICROPROGRAMMED TESTS, User's Guide

RCSI. No.: 30-M270

A/S Regnecentralen af 1979/RC Computer A/S maintains a continual effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback, your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability:

_____

_____

_____

_____

Do you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

How can this manual be improved?

_____

_____

_____

_____

Other comments?

_____

_____

_____

_____

_____

Name: _____ Title: _____

Company: _____

Address: _____

Date: _____

Thank you

42-i 1288

..... Fold here .....

..... Do not tear - Fold here and staple .....

Affix
postage
here

REGNECENTRALEN
af 1979

Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark