Appendix B.

Paging in TRAC

B.1 Introduction

Paging in TRAC is accomplished in three steps:

- task break up
- page transfers
- task resynchronization

The first step prepares all necessary information so that the task can be broken into smaller tasks capable of performing the page transfers. Next the paging tasks and paging groups are formed and page transfers take place. Recall that a paging task consists of a single processor with access to an SMSM, and a paging group consists of a paging task and all other processors whose pages are stored in the SMSM in the domain of the paging task. It must also be clear that only one processor (the paging task processor) of the group has access to the SMSM. The last step of the paging process consists of the

re-activation of the task-wide instruction tree in order to establish the SIMD mode of execution of the task.

B.2 Hardware support

To effectively accomplish paging several hardware/microcode supported features are required. Each of these will be discussed in terms of a privilege instruction.

Paging tasks will have to communicate with the JM in order to create and de-activate instruction trees. Communication with the JM is handled by IGM. Therefore the paging task needs to constitute itself as a TH-PRM to be capable of sending this type of message. Two instructions are required to accomplish this task.

- TESTPRM(<param>): this instruction will test the hardware TH-PRM bit and will store its value in a location defined by <param>.
- SETPRM(<param>): this instruction will set the TH-PRM bit to <param>.

The purpose of the first instruction is to save the status of the TH-PRM bit so the original task can be reconstructed when paging is finished. The second instruction will be used first to constitute the paging

task into a TH-PRM and then to reset the processors to their original status.

The creation of paging tasks implies the deletion of the task-wide instruction tree. Page in/out is accomplished among two processors which must be connected by an instruction tree. Therefore these trees must be created and deleted dynamically at the request of the paging tasks. The DINST and CINST instructions are used for this purpose.

Both of these instructions will be used repeatedly to create and delete instruction trees between the paging task and a selected processor from the group.

Two types of synchronization primitives are required. The first is for processors waiting on an instruction tree and the second for processors waiting on a SSP signal to resume lockstep execution. Actually both instructions have the same function but they must be distinct in the signal they receive because the SSP signal will vector execution to a predefined location. The WAIT(TSP) and WAIT(SSP) instructions are used for this purpose.

The WAIT(TSP) primitive will be used to wait on an instruction tree which will connect a processor of the group with the paging task, or when a paging task is waiting to continue execution (after finishing a page in/out operation) with another processor of the group. The WAIT(SSP) is required to re-synchronize the task when all pages have been transferred and the task is ready to resume lockstep execution.

A semaphore register is present at the memory module which is at the root of the task's instruction tree. This semaphore is used to count the number of paging groups which have completed their paging operations. When the semaphore reaches a value of zero all paging operations are completed. The SETSEM and GETSEM instructions are used to manipulate the semaphore.

These semaphore instructions are used to read and update the value of the semaphore register.

B.3 Data Structures

The following declarations define the data structures required to support the paging algorithm:

```
CONST
```

```
MAXPROCESSORS = 16

VAR
 PAGETABLE = ARRAY[MAXPROCESSORS*64*3,3] OF INTEGER;
 PAGEGROUP = ARRAY[MAXPROCESSORS] OF INTEGER;
 PAGINGTASK = BOOLEAN;
 NOPAGINGROUP = 1..MAXPROCESSORS;
 PRMSTATUS = BIT;
 PROCTASK = INTEGER*2;

 <page-in> = integer;
 <page-out> = integer;
 <type> = integer;
 <moretransfers> = integer ;
```

where:

- MAXPROCESSORS : the maximum number of processors which can make a group.

- PAGETABLE : a table which contains the processor number, page number, and the space type, for all pages currently resident in memory.

- PAGEGROUP : an array which indicates to which paging group each processor belongs.

- PAGINGTASK : a boolean variable set in those processors which are paging tasks.

- NOPAGINGROUP : number of paging groups (or paging task processors) in the task.

- PRMSTATUS : a bit variable to save the original status of the processor.

- PROCTASK : a MAXPROCESSORS bit variable whose corresponding bits are set for those processors which form part of the task.

- <page-in> : page causing the page fault.

- <page-out> : page to be swapped out.

- <type> : type of space of page causing the fault.
- <moretransfers> : the number of page transfers a paging task has to perform.

Since this information must be accessed by the processors after the task has been broken up it must reside in Data Space, starting at page 0, and must be initialized by the loader at load time.

Two restrictions must be imposed while loading a task. First, a page buffer must be reserved in each memory module which has an SMSM and which will be page 63 of Control Space. Second, page 62 of Control Space is also a reserved page. These two pages are required for the transfer of pages among processors.

B.4 Algorithm 1

This first algorithm is based on the synchronization mechanism which relies on the Control Port to dynamically create and delete instruction trees as discussed in Appendix A. For the purpose of describing the paging algorithm we have divided the process into five levels as shown in Figure B-1. The filled lines represent the execution flow of processors which are paging tasks,

while the dotted lines represent execution flow of non-paging task processors. The fork in the execution flow represents the de-activation of an instruction tree, while a join in the execution flow represents the creation of an instruction tree. See Figure B-2.

Level I is the Page Fault Handler entry. The following pseudo-code describes the actions taken at this level.

```
begin     (*  Level  I
              Page Fault Hander

      - save PRM status
      - find:
            page causing fault  ( < page-in > )
            page to swap by LRU ( < page-out > )
            type of space       ( < type > )
      - de-activate instruction tree
      - all processors wait for tree deletion

                                                    *)
  TESTPRM ( PRMSTATUS ) ;
  FIND ( < page-in > ) ;
  LRU ( < page-out > ) ;

  if  < type > = 'CS' OR 'PS'
   then
    SETSEM ( 1 )
   else
    SETSEM ( < NOPAGINGROUP > ) ;

  DINST ( PROCTASK ) ;
  WAIT ( TSP );

 end;     (*  Level  I                          *)
```

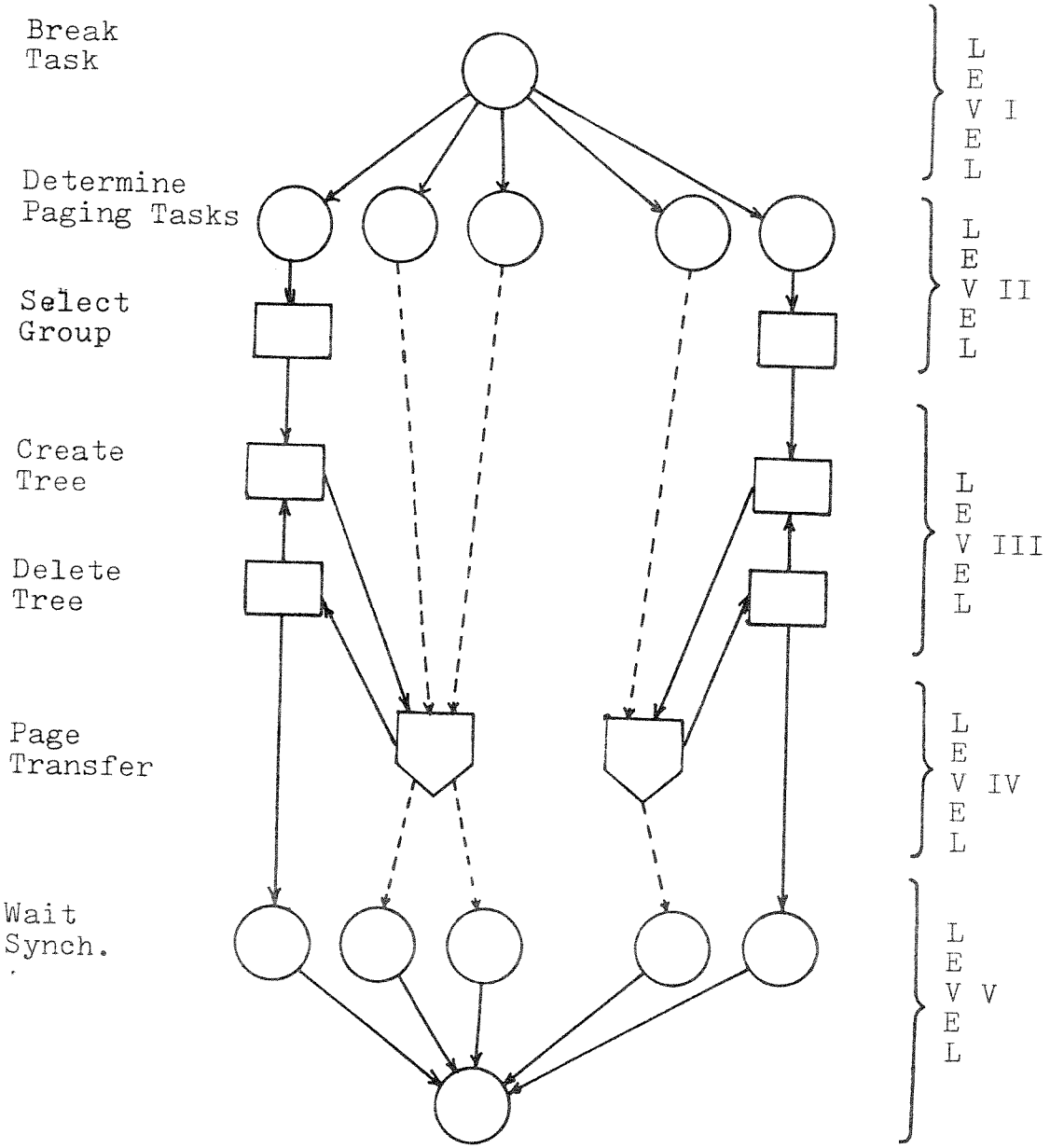

Figure B-1: Execution flow graph

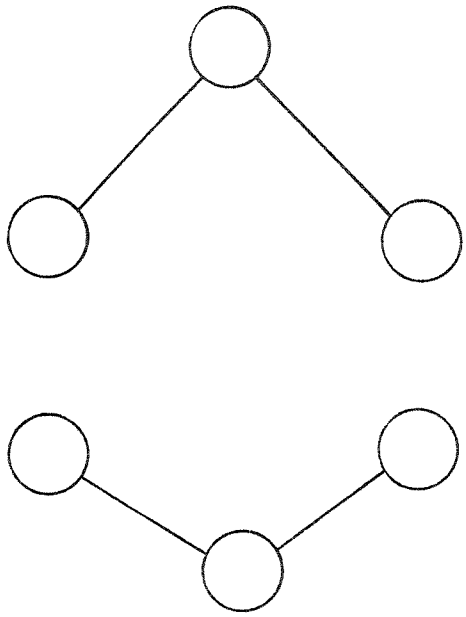

Figure B-2: Fork and join execution

Execution flow is now at Level II. The instruction tree has been deleted. All processors are waiting for a TSP signal. When the processors receive the TSP signal execution resumes with each processor executing as a one-processor task. At this level paging tasks will determine the number of transfers required and will perform any local transfer of pages. Non-paging tasks will convert their <page-out> to CS-62 and will proceed to Level IV. The following pseudo-code maps to Level II of the execution flow graph.

```
begin    (*   Level   II

 each processor starts executing this code
 after the instruction tree has been deactivated
                                                    *)
if < type > = 'PS' OR 'CS'
 then
     (*  fault on one dimensional space
                                                    *)
   if < page-out > ∈ Address-space
    then
     if PAGINGTASK
      then
         (*  paging task owns page
                                                    *)
       begin
        SETPRM ( 1 ) ;
         LOCALTRANSFER ;
        < moretransfers > := 0
       end

      else
         (*  owns page but not a paging task
                                                    *)
       begin
        CONVERT ( < page-out > , CS62 ) ;
        (*   drop to Level  IV
                                                    *)
        PAGETRANSFER
       end

    else
     if < page-out owner > ∈ PAGINGROUP
          AND  PAGINGTASK
      then
       begin
       (*  processor is paging task of owner
                                                    *)
        SETPRM ( 1 ) ;
        < moretransfers > := 1 ;
       end
      else
       (* processor not required in one
          dimensional fault
          drop to Level V
                                                    *)
       WAIT ( SSP ) ;
```

```
else
 (* two dimensional page fault
                                                    *)
 if NOT PAGINGTASK
  then
   (* non-paging task, convert < page-out >,
      and drop to Level IV
                                                    *)
   begin
   CONVERT ( <page-out > , CS62 ) ;
   PAGETRANSFER
   end

end; (*   Level  II                                 *)
```

While non-paging tasks are waiting for a TSP signal at Level IV (or waiting for a SSP signal at Level V in one dimensional page faults) paging task processors proceed to Level III. Each paging task will select a processor from the group, will join this processor at Level IV to form a two processor task, and will then return to Level III. This procedure is repeated until transfers to all processors of the group are accomplished. Once transfers are completed, the synchronization semaphore is updated, the status of the processor is restored, and the processor will proceed to Level V. The following pseudo-code represents Level III of the execution flow graph.

```
begin (*   Level  III
                                                    *)
 while < moretransfers > ≠ 0   do
```

```
   begin
    . select a processor from group
    (* join selected processor at Level IV
                                                    *)
    PAGETRANSFER ;
    < moretransfers > := <moretransfers > - 1 ;
    (*
      delete instruction tree                       *)
    DINST ( < paging task and selected processor > ) ;
    WAIT ( TSP )

   end   (*  while moretransfers                     *)
(*
  paging tasks complete transfers for the group
  update synchronization semaphore and proceed to
  Level  V
                                                    *)
  GETSEM ( < sem-value > ) ;
  < sem-value > := < sem-value > - 1 ;
  SETSEM ( < sem-value > ) ;
   if ( < sem-value > ) ≠  0
    then
     (* some paging task still active
                                                    *)
    begin
     SETPRM ( PRMSTATUS ) ;
     (* drop to   Level  V
                                                    *)
     WAIT ( SSP )
    end

    else
     begin
      (* no other paging task active
         request creation of task-wide instruction
         tree  and proceed to  Level  V
                                                    *)
      CINST ( PROCTASK ) ;
      SETPRM ( PRMSTATUS ) ;
      WAIT ( SSP ) ;
     end

 end  (*  Level III                                 *)
```

At level IV the paging task joins the processor

selected at level III to form a two-processor task. When the instruction tree is created the page transfers take place. While the paging task return to level III the other processor proceeds to level V to wait for the synchronization signal. The following pseudo-code describes execution at level IV.

```
procedure pagetransfer;   (*    Level  IV     *)

 if PAGINGTASK
  then
   (* paging task requests new instruction tree and
      then waits
                                                 *)
   CINST ( <paging task and selected processor> ) ;
   (* non-paging task goes directly to wait state
                                                 *)
 WAIT ( TSP ) ;
 (* paging task joins selected processor in a two
    processor task
                                                 *)
 NONLOCALTRANSFER ;
 (* CS-62  =>  CS-63  =>  SMSM   < page-out >
    SMSM   =>  CS-63  =>  CS-62  < page-in >
                                                 *)
 CONVERT ( CS62 , < page-in > ) ;
 . update PAGETABLE ;
(*
   < page-in > is now resident
   non paging task will proceed to Level V while
   paging task return to Level III
                                                 *)
 WAIT ( SSP AND NOT PAGINGTASK ) ;
 (* a wait ( false ) is equivalent to a NOP
                                                 *)

 end;  (* pagetransfer        Level  IV        *)
```

At level V processors wait for the synchronization

signal when the task-wide instruction has been created. At this point lockstep execution is resumed and control can be returned to the executing task.

B.5 Algorithm 2

This algorithm uses the CLA Logic mechanism as described in Appendix A. The difference between using the CLA logic mechanism and the Control Port mechanism is in the way synchronization is accomplished. Since the paging algorithm is the same the Pascal pseudo-code is omitted to avoid repetition from the previous section.

Again the problem of paging in and out a multi-byte wide space is considered.

After the page to be replaced is decided upon (<page-out>) the task breaks into multiple paging tasks. Again the processors that have access to SMSM's through their data tree are designated as paging task heads. They are responsible for paging their own pages as well as those pages belonging to processors of the paging group. Individual paging tasks execute independently of each other and at the end of their operation are suspended until the last of them reaches completion. In the end all

processors of the task are brought back in synchronism after which they start executing in lockstep. The process of paging a multi-byte wide space takes 9 steps. These are described below.

Step 1: Delete the task-wide instruction tree.

The current task-wide instruction tree (which overlaps the task-wide shared tree of color 0 by convention) is deleted in lockstep. This is accomplished by properly executing the DELBRNCH instruction. At this point the processors cease to be part of the main task.

Step 2: Set appropriate Acquires and Propagates.

The processors realign themselves into different paging groups. Shared trees of color 1 are predefined over paging task processors at task set up time. Also, the total number of processors for its paging group is known to each processors at load time. The paging head processors negate their Acquires and assert their Propagates through the RESET(ACQUIRE) and SET(PROPAGATE) instructions. These processors then wait in a microcode loop for the Grant signal to be true (WAIT(GRANT)).

Step 3: Wake up paging task processors.

The individual paging task head processors assert their Acquires (SET(ACQUIRE)) and thus wake up their subject processors by asserting their Grants. Thus the whole paging group gets into lockstep. In lockstep they create the paging task's instruction tree by converting the shared tree into an instruction tree. of color 1.

Step 4: Mask Page.

Page 63 of Control Space is reserved for paging out to the SMSM. It exists only in the paging task head processors. CS-62 is similarly reserved for paging. This page moves from processor to processor within a paging task. The processors from which a page is to be swapped rename that page to be CS-62.

The processors are also provided with a unique id within their paging group. The following computation is done to select the next processors for paging.

Let the number of processors in the paging task be NUMSUB.

1. if NUMSUB <> paging-id then set mask.
2. under mask put <page-out> into ALU register.
3. convert <page-out> to CS-62.

Step 5: Transfer CS-62 to CS-63.

Data is then transferred from CS-62 to CS-63 through the use of the broadcast bus. The data received from CS-62 comes from the paging candidate and goes to CS-63 which exists in the paging task head processor. A complete page is transferred in lockstep but only two processors are actually active.

Step 6: Transfer page to SMSM and read <page-in>.

The paging task head processor now has the page to be swapped out in its CS-63. It transfers this to the SMSM by executing an I/O instruction. It then reads back <page-in> from the SMSM into CS-63.

Step 7: Transfer page CS-62 to CS-63.

A reverse transfer of data takes place and the new page, <page-in>, is placed in the memory of the current candidate.

Step 8: Change page id and increment counter.

CS-62 is changed to <page-in>. Again only one processor actually does the modification, others simply execute in lockstep mode without any side effects. The number of processors (NUMSUB) is then decremented by one. If there are any more candidates left the process is repeated from Step 4.

Step 9: Resynchronize task.

The instruction tree of color 1 is destroyed (DELBRNCH), and the processors are desynchronized. Each non paging task head processor creates a branch of the taskwide instruction tree, CREBRNCH(0), asserts its Propagate, SET(PROPAGATE) and negate its Generate, RESET(GENERATE). Finally all will enter a microcode loop waiting for their Carry to become true, WAIT(CARRY).

Meanwhile the paging task head processor acquires the shared memory module and decrements the synchronization register. It releases the memory module and goes into a loop like the other processors of the group if the synchronization register is not zero. If the synchronization register is zero it will assert its

Generate, SET(GENERATE), and will join the rest of the processors in lockstep.

Appendix C.

Intra-task Communication in TRAC

Chapter 4 described the instances where Intra-task Communication was required. Communication among processors of the same task is required as a follow up action of an IGM. These messages arrive at the TH-PRM processor as interrupt packets.

There are two steps involved. First, the packet has to be analyzed by the TH-PRM and, second, information may have to be distributed or collected from other processors of the task.

Since the packet is stored in page zero of the TH-PRM's Data Space, only the TH-PRM has access to this information. Therefore it is required that while the packet is analyzed by the TH-PRM, the other processors must be in a wait state. To realize this lockstep mode of execution must first be abandoned.

After the packet has been analyzed, the TH-PRM is free to communicate on a one to one basis with each individual processor. Any message passing is done through NIGM. This is possible since processors are in a wait local packet loop.

Resynchronization is reinstated at the request of the TH-PRM.

C.1 Hardware Support

The desynchronization and synchronization process utilizes the CLA Logic mechanism as described in Appendix A.

C.2 Algorithm

The Interrupt Handler is activated when IGM (interrupt packet) arrives at the TH-PRM's processor. The task must be desynchronized for packet analysis and for processors to communicate with each other via NIGM (non-interupt packets).

Figure C-1 schematically shows the desynchronization algorithm and the following pseudo-Pascal code shows the structure of these process:

```
  SETSEM(# Processors);
            (* delete branch of instruction tree  *)
  DELBRNCH;
    if THPRM
      then    (*    TH-PRM        *)
        begin

          (*    analyze packet          *)
          (*    one-to-one processor communication    *)
          (*    send resynchronization NIGM SIGNAL    *)


        end

      else    (* non THPRM         *)
        begin
          repeat

           WAIT ( NIGM );

           (* process NIGM        *)

           until  NIGM = "resynchronization";
        end;


(* all processors will now wait for the instruction tree
   to be activated                                        *)
GETSEM(SEMVALUE);
if SEMVALUE <> 0
  then
    CREBRNCH(0)
    SEMVALUE := SEMVALUE -1
    SETSEM(SEMVALUE)
    SET(PROPAGATE)
  else
    CREBRNCH(0)
    SET(GENERATE);

WAIT(CARRY);
```

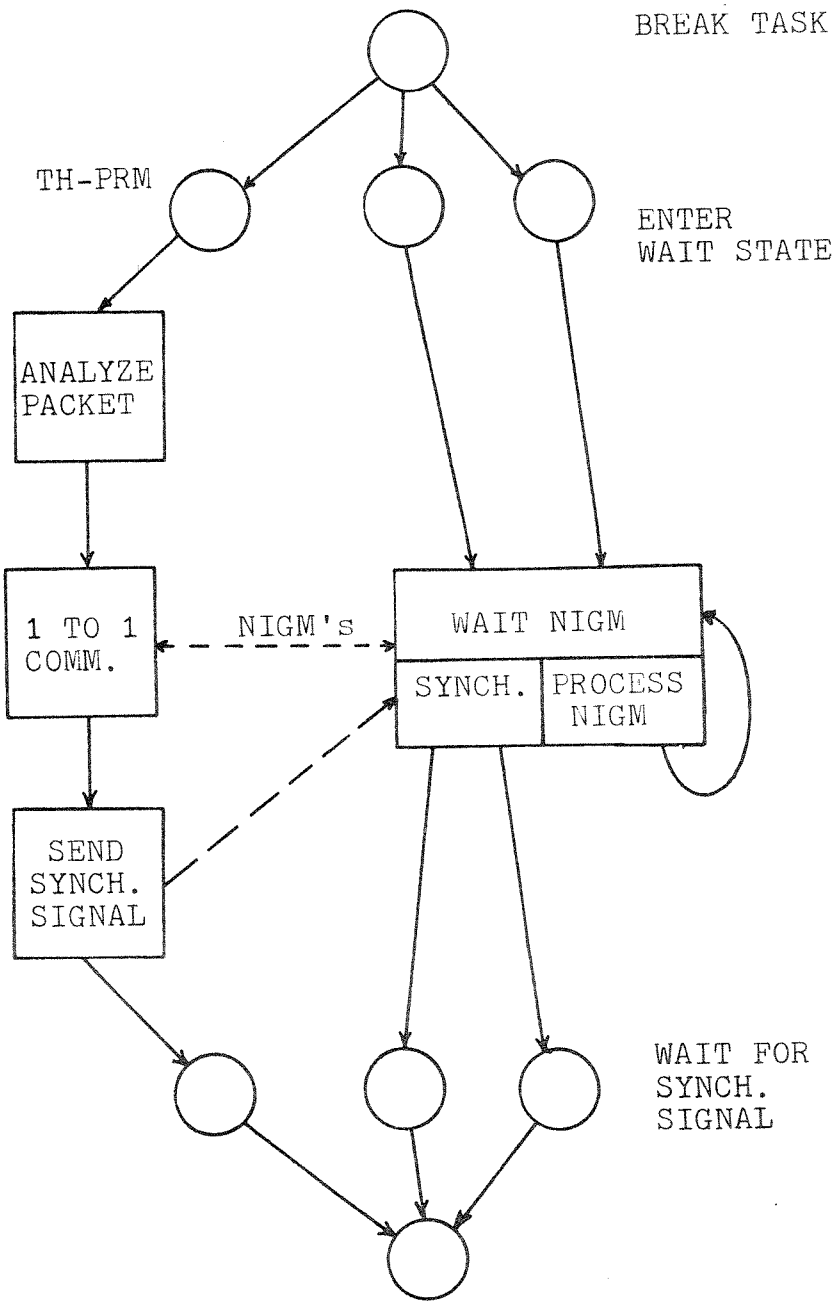

Figure C-1: Process desynchronization

Appendix D.

Shared Memories in TRAC

D.1 Introduction

Shared memories are used to share information among tasks. TRAC's architecture requires that a shared memory module be connected to one shared tree at a time. Therefore processors must cooperate in the use of the shared memory.

The use of shared memories can be viewed as a three step process:

- detection of the absence of a shared module (or share memory fault).
- request for a shared memory.
- acquisition of a shared memory module.

This Appendix will describe the three steps and their implementation in TRAC.

D.2 Shared Memory Fault

When a task executing in lockstep mode tries to access a page, and a page fault situation is encountered, two things could have happened. Either the page in question is stored in a shared memory module currently not attached to the task, or the page is stored in a backup device.

To resolve the situation the PRM will search the Shared Memory Page Table. The table, whose format is shown in Figure D-1, contains an entry for every page stored in a shared memory module along with the description of its memory space and the color of the shared memory module in which it resides.

If the page causing the fault is found in the table, then the shared memory of the specified color must be acquired. If the page in question is not in the table, then a page fault has occurred and the PRM will proceed as explained in Appendix B.

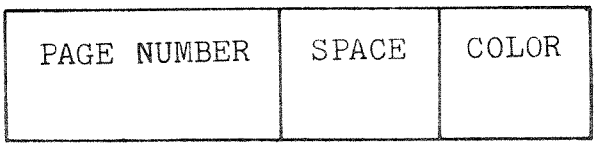

Figure D-1: Shared Memory Page Table

D.3 Shared Memory Request

It is not mandatory that pages with the same page number of different processors of the task be stored in shared memories of the same color. Colors are assigned at load time and must be accurately reflected in the Shared Memory Page Table. For example Figure D-2, shows the case where page 10 of Data Space is stored in different color shared memory modules for each of the three processors of the task.

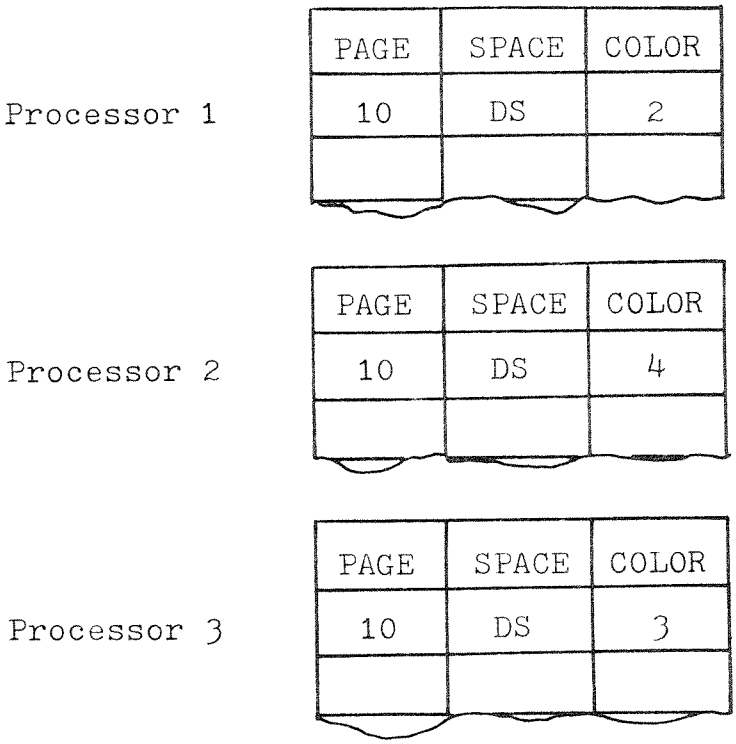

Figure D-2: Shared Memory Page Table Example

In order to avoid deadlock situations, the acquisition of shared modules is under the control of the Job Monitor. This means that the TH-PRM of the task must request the shared modules to the JM. To handle this situation each processor of the task will be requested to send the color of the shared memory it needs to the TH-PRM. The TH-PRM will in turn send a request for all shared memories needed to the JM. The process of sending the messages among the processors of the task is described in Appendix C. Communication with the JM is discussed in Appendix E

At this point the TH-PRM is waiting for a message from the JM instructing it to acquire the shared memories.

D.4 Shared Memory Acquisition

When the JM sends the acquire message, the TH-PRM will in turn send a message to the other processors of the task. Each processor will issue an acquire instruction for its shared memory module. The task is resynchronized as explained in Appendix C.

Appendix E.

Messages in TRAC

E.1 Introduction

Inter-task and intra-task communication in TRAC is done through a message system. Chapter 3 described the Communication System while Chapter 4 discussed the protocols needed for inter-task and intra-task communication.

This Appendix will outline the messages necessary to accomplish communication in TRAC.

E.2 Data Structures

The Communication System will require the support of two basic data structures; one to define the message itself and a second one to define the message table. The first will have the necessary information for the message to be properly routed through the network and for the communication system to process the message. The message

table contains the necessary information for the Upward and Downward Consistency Checking Mechanisms.

E.3 Message Format

Messages will consist of three logical fields:

- processor id. of destination processor.
- message id.
- contents.

The processor id of the destination processor is the physical processor number to which the message must be sent. The message id is an identification number associated with the message and the contents is the message itself. The first two fields are one byte long while the contents field varies in length depending on the message.

E.4 Message Table

The message table consists of two entries:

- message id.
- message type.

As before, the message id will identify the

message while the message type is a coded scheme used by the Consistency Checking Mechanism (see Chapter 3).

The following messages are required for the TRAC implementation:

MESSAGE: SEND OBJECT

MESSAGE ID: 1

TYPE: 56

DESCRIPTION: requests the PRM to send a particular object to another PRM or to a shared memory module.

CONTENTS: channel through which the object is to be sent

if packet the destination processor id else shared memory module and address within the module

type of object

address of type and var descriptors

MESSAGE: RECEIVE OBJECT

MESSAGE ID: 2

TYPE: 76

DESCRIPTION: tells the PRM to receive a particular object

CONTENTS: same as SEND OBJECT

MESSAGE: GET OBJECT

MESSAGE ID: 3

TYPE: 55

DESCRIPTION: this message will contain the actual object the PRM is expecting to receive

CONTENTS: processor id of sender

the actual object

MESSAGE: ACQUIRE SHARED MEMORY

MESSAGE ID: 4

TYPE: 56

DESCRIPTION: this message tells the PRM that it can acquire a shared memory module.

CONTENTS: shared memory modules which can be acquired or null if TH-PRM had previously requested the modules.

MESSAGE: RELEASE SHARED MEMORY

MESSAGE ID: 5

TYPE: 14

DESCRIPTION: tells the TH-PRM that it should release shared memory modules

CONTENTS: shared memory modules to release

MESSAGE: RETURN SHARED MEMORY

MESSAGE ID: 6

TYPE: 42

DESCRIPTION: tells the PRM's to release shared memory modules

CONTENTS: shared memory modules to release.

MESSAGE: NEED SHARED MEMORY

MESSAGE ID: 7

TYPE: 55

DESCRIPTION: the PRM's will request shared memory modules to the TH-PRM which in turn will request these modules to the JM

CONTENTS: needed shared memory modules

MESSAGE: WHICH SHARED MEMORY

MESSAGE ID: 8

TYPE: 42

DESCRIPTION: after a shared memory module page fault the TH-PRM will ask the PRM's for the shared module they need. Each PRM will respond with a NEED SHARED MEMORY message

CONTENTS: null

MESSAGE: TERMINATE

MESSAGE ID: 9

TYPE: 14

DESCRIPTION: tells the TH-PRM to terminate execution of the task

CONTENTS: null

MESSAGE: STOP

MESSAGE ID: 10

TYPE: 14

DESCRIPTION: tells the TH-PRM to temporarily stop execution of the task. The TH-PRM will wait for an IGM while the other PRM's will wait for a synchronization signal

CONTENTS: null

MESSAGE: CONTINUE

MESSAGE ID: 11

TYPE: 14

DESCRIPTION: tells the TH-PRM to resume execution of the task. A synchronization signal is sent by the TH-PRM to the other processors of the task.

CONTENTS: null

MESSAGE: VALUE

MESSAGE ID: 12

TYPE: 56

DESCRIPTION: requests the current value of a conditional value to be sent to the JM

CONTENTS: address of type

address of var descriptor

MESSAGE: DELETE INSTRUCTION TREE

MESSAGE ID: 13

TYPE: 41

DESCRIPTION: requests the deletion of an instruction tree

CONTENTS: processors which are nodes of the tree to be deleted

MESSAGE: CREATE INSTRUCTION TREE

MESSAGE ID: 14

TYPE: 41

DESCRIPTION: requests the creation of an instruction tree

CONTENTS: processors which are nodes of the new instruction tree

References and Bibliography

[BAC 81] Bacon J.
An Approach to Distributed Software Systems.
Operating Systems Review 15(4):62-74, October, 1981.

[BAS 77] Basket F., Howard J., Montague J.
Task Communication in DEMOS.
In **Proceedings of the Sixth Symposium on Operating Systems Principles**, pages 23-31. ACM-SIGOPS, November, 1977.

[BRI 73] Brinch Hansen P.
Operating Systems Principles.
Prentice Hall, 1973.

[BRO 80a] Browne J.C., Charlu D., DeGroot D., Tripathi A.R.
Software and Programming Systems for TRAC.
Technical Report TRAC-19, Dept. Computer Sciences and Electrical Eng., University of Texas at Austin, 1980.

[BRO 80b] Browne J.C., Tripathi A.R., Fedak S., Kapur R., Adiga A.
A Language for Definition and Control of Reconfigurable Parallel Computation Structures.
1980.

[BRO 81] Browne J.C., DeGroot D., Tripathi K., Fedack S. Adiga A.
The Structure of an Operating System for a Reconfigurable Network Architectured System: TRACOS.
1981.

[BRO 82] Browne J.C., Lipovski G.J.
Reconfigurable Network Architectured Computer Systems: An Environment for Parallel Computing.
1982.

[COF 71] Coffman E.G., Elphick M.J., Shoshani A.
System Deadlocks.
ACM Computing Surveys 3(2):67-78, June, 1971.

[COF 73] Coffman E.G., Denning P.J.
Operating Systems Theory.
Prentice Hall, 1973.

[DeG 80] DeGroot D., Hunt W.A.
A Solution to the Paging Problem.
Technical Report TRAC-18, Dept. of Computer Sciences and Electrical Eng., University of Texas at Austin, 1980.

[DeG 81] DeGroot D., Hunt W.A., Browne J.C.
Virtual Memory Management for Network Architectured Vari-Structured Computers.
1981.

[DES 83] Desphante S., Canas D.A., Sejnowski M.C.
Synchronization and Paging in TRAC.
1983.

[DIJ 68] Dijkstra E.W.
The Structure of the THE Multiprograming System.
Communications of the ACM 11(5):341-345, May, 1968.

[ENS 74] Enslow P.H., Editor.
Multiprocessors and Parallel Processing.
Wiley, 1974.

[FED 80] Fedack S.
Implementation of CSL for TRAC.
1980.

[GEN 81] Gentleman W.M.
Message Passing Between Sequential Processes: The Reply Primitive and the Administrator Concept.
Software Practices and Experiences 11:453-466, 1981.

[HAB 76] Haberman A.N.
Introduction to Operating Systems Design. SRA, 1976.

[HOW 73] Howard J.
Mixed Solutions for the Deadlock Problem.
CACM 16(7):427-430, July, 1973.

[JON 77] Jones A.K., Chansler R.J., Durham I.,Feiter A., Schwares K.
Software Management of Cm*, A Distributed Multiprocessor.
In **Proceedings NCC**, pages 657-663. ACM, 1977.

[JON 79a] Jones A.K., Chasler R.J., Durham I., Karster S., Vegdahl S.
StarOs, a Multiprocessor Operating System for support of Task Forces.
In **Proceedings of the Seventh Symposium on Operating Systems Principles**, pages 117-127. ACM-SIGOPS, December, 1979.

[JON 79b] Jones A.K.
The Object Model: A Conceptual Tool for Structuring Software.
In **Operating Systems: An Advance Course**, chapter 2, pages 8-16. Springer-Verlag, 1979.

[JON 80] Jones A.K., Schwartz P.
Experiences Using Multiprocessor Systems- A Status Report.
ACM Computing Surveys 12(2):121-166, June, 1980.

[KAP 80] Kapur R., Premkumar U., Lipovski G.J.
Organization of the TRAC Processor-memory Subsystem.
In **Proceedings NCC**, pages 623-629. ACM, 1980.

[KAR 77] Kartashev S.I., Kartashev S.P.
A Multicomputer System With Software Reconfiguration of The Architecture.
In **Proceedings of The Eight International Conference on Computer Performance**, pages 271-286. IEEE, 1977.

[LIP 79] Lipovski G.J.
The Architecture of the Banyan Switch for TRAC.
Technical Report TRAC-7, Dept. of Computer Sciences and Electrical Eng. University of Texas at Austin, 1979.

[MAN 81] Manning E.G., Wong J.W., Powell P.A., Radia S.R., Tokuda H.
SOSHINI- A Testbed for Distributed Software.
In **Proceedings International Conference on Communication**. IEEE, June, 1981.

[MEH 80] Mehra S.K., Majitha J.C.
Software Issues for Reconfigurable Architectures.
In **Proceedings COMPSAC 80**, pages 484-491. IEEE, 1980.

[NUT 77] Nutt G.J.
A Parallel Processor Operating System.
IEEE-TSE SE-3(6):467-475, November, 1977.

[OUS 80] Ousterhout J.K., Scelza A., Scindler D.S.
MEDUSA: An Experiment in Distributed Operating Systems Structure.
CACM 23(2):92-105, February, 1980.

[PRE 79] Premkumar U.V., Kapur R., Lipovski G.J.
Interprocess Communication on the Texas Reconfigurable Array Computer.
In **Proceedings of the 1st. International Conference on Distributed Computers**, pages 51-62. IEEE, 1979.

[PRE 80] Premkumar U.V., Kapur R., Malek M., Lipovski G.J., Horne P.
Design and Implementation of the Banyan Interconnection Network in TRAC.
In **AFIPS Conference Proceedings**, pages 643-653. AFIPS, 1980.

[QUA 78] Quaynor N., Bernstein A.
Operating Systems for Hierarchical Multiprocessors.
In **Proceedings Seventh Texas Conference on Computing Systems**, pages 9-15. IEEE, 1978.

[SEJ 80] Sejnowski M.C., Upchurch E., Kapur R., Lipovski G.J.
An Overview of the Texas Reconfigurable Computer.
In **Proceedings NCC**, pages 631-641. ACM, 1980.

[SEJ 81] Sejnowski M.
Packet Support in TRAC.
Master's thesis, Dept. of Computer Sciences, University of Texas at Austin, May, 1981.

[SHA 74] Shaw A.C.
The Logical Design of Operating Systems.
Prentice Hall, 1974.

[SMU 79] Smullen J.
Memory Management of TRAC.
Master's thesis, Dept. of Computer Sciences, University of Texas at Austin, May, 1979.

[STO 80] Stone H.S.
Introduction to Computer Architecture.
SRA, 1980.

[SUL 77] Sullivan H., Bashkow T.R., Klappholz D.
The Node Kernel: Resource Management in the Self Organizing Parallel Processor.
In **Proceedings International Conference on Parallel Processing**, pages 157-162. IEEE, 1977.

[TRI 79] Tripathi A.R., Lipovski G.J.
Packet Switching in Banyan Networks.
In **Proceedings of the 6th. Symposium on Computer Architecture**, pages 160-167. IEEE, 1979.

[WAT 79] Watson R.W., Fletcher J.G.
An Architecture for Support of Network Operating Systems Services.
In **Proceedings of the 4th Berkeley Conference on Distributed Data Management and Computer Networks**, pages 18,50. University of California at Berkeley, August, 1979.

[WUL 81] Wulf W., Levin R., Harbeson S.D.
HYDRA/C.mmp: An Experimental Computer System.
McGraw Hill, 1981.