SNAP Controller

BY

Hirendu Vaishnav

Technical Report CENG 90-01

Electrical Engineering - Systems Department

University of Southern California

Los Angeles, CA. 90089-0781

SNAP Controller

Technical Report No. CENG 90-01

Hirendu Vaishnav, Dan Moldovan

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California

January 8, 1990

[0]This research has been funded by the National Science Foundation Grant No. MIP-89/02426

Contents

1 System Description

1.1 System Block Diagram

A block diagram of the complete SNAP system is shown in Figure 1.The SNAP Array is distributed among four boards, each containing 64 SNAP chips. Each chip can store 64 semantic network nodes. The whole system is accessible through the SUN workstation. Application programs are compiled into the SNAP Primitive Instruction (SNAPRIM) Set and transmitted to the SNAP array from the SUN workstation. The SNAP controller provides the interface between the SUN workstation and the SNAP array. The SNAP controller is connected to the SUN workstation through the VME Bus, and through a specially designed SNAP Bus to the SNAP array.

1.2 Design Specifications

The design specifications can be broadly divided into the following two sections:

1. Functional Specifications
2. Hardware Specifications

1.2.1 Functional Specifications

The SNAP controller is expected to perform these functions:

1. Load the instruction set from the SUN workstation when the host is ready with a compiled application software.
2. Convert the SNAP primitive instruction set to an instruction set , which could be directly executed by SNAP array and transmit these instructions to the SNAP array.
3. Assign physical Node-ID's to the nodes created in the program and supervise these nodes for any overflows or other erroneous conditions.
4. Allow for self-test software to be executed at some specified times. Allow a little compilation in case of loops, etc. in the program.

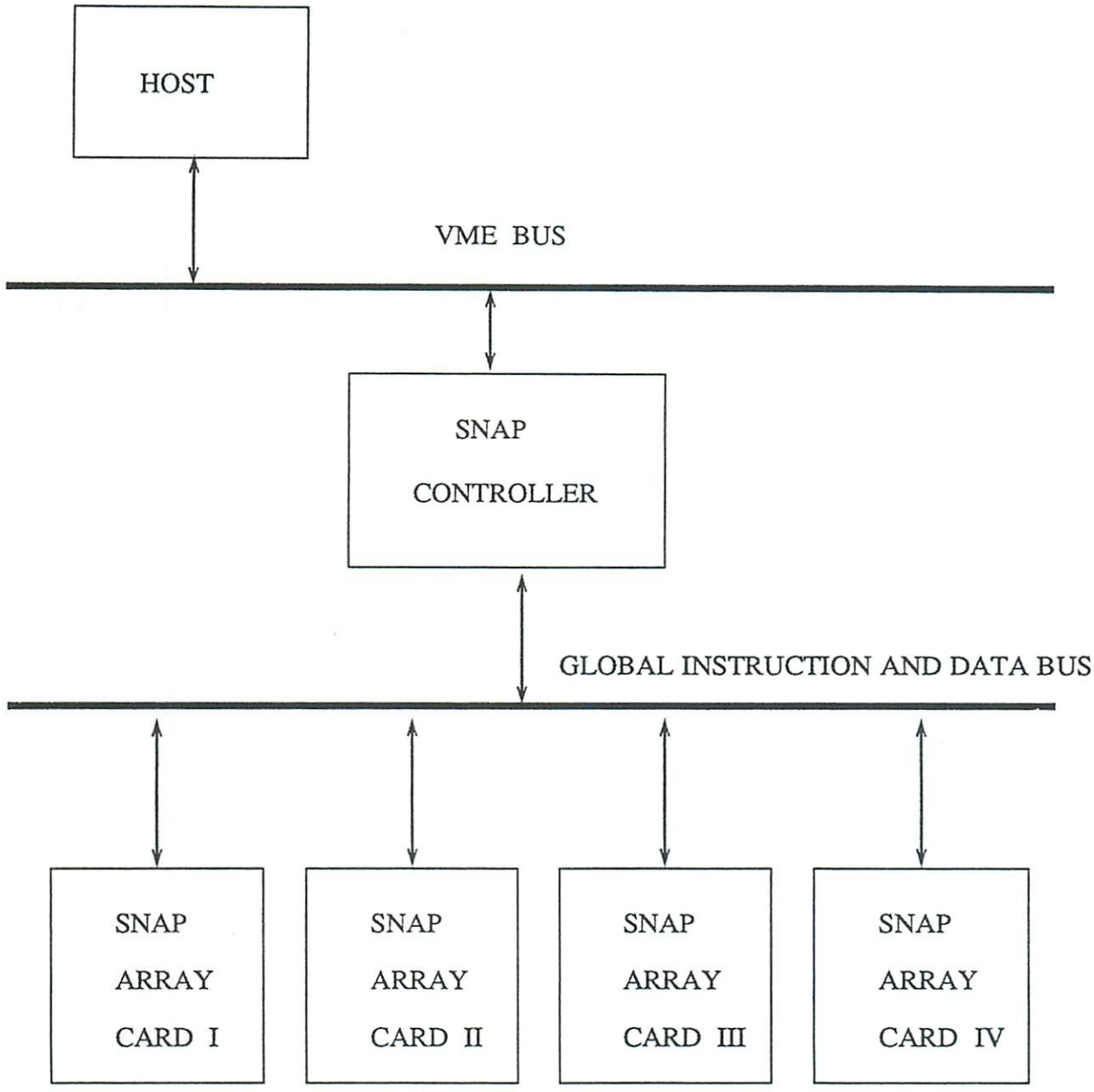


Figure 1: Block Diagram of The SNAP system

1.2.2 Hardware Specifications

The environmental aspects of the SNAP system and the speed considerations require that the SNAP controller meet the following specifications:

1. The SUN workstation communicates to the SNAP controller through the VME Bus. This requires that the SNAP controller be designed so that it satisfies all VME Bus interface specifications.

2. The width of each SNAP primitive instruction is 32 bits, same width of data bus of the VME Bus. This requires that the controller hardware support 32 bit operations.

3. The width of SNAP Data Bus is 8 bits, which means that these data should be converted to 32 bits to use the VME Bus at its peak performance.

4. Since only 64 user defined pins are available for SNAP Bus we should try to minimize the number of pins in the SNAP Bus.

1.3 Design Objectives

Apart from the specifications mentioned above, the following design objectives are included for more efficient operation of the SNAP system:

1. The speed of the SNAP controller should not prove to be a bottleneck for the SNAP system.

2. The controller design should be such that data transfer rate on the VME Bus as well as on the SNAP Bus be compatible with the SNAP array operation.

3. The controller design should accommodate some PsuedoInstructions for the controller to facilitate optimization in the node management at the compiler level. This mainly includes the capability to transmit back the Node Memory Map to the SUN workstation.

4. The controller should be capable of accommodating minor changes in the hardware specification, Node management techniques, system architecture, working environment, and a possible expansion of the system.

2 Design Approach

Considering these specifications and objectives, the following design approach was considered most suitable:

1. 256 bytes of the compiled instruction set is going to be loaded in the controller as a block, after which the instructions are executed on the SNAP array one by one. Similarily, the the data to be sent back to the workstation is also being send in a block. Considering these two factors, it was found most appropriate to configure the SNAP controller as a slave supporting 32 bit data transfer in the Quad Block Transfer mode. This also reduces the hardware required for the VME interface, making it possible to carry on the data transfer almost independent of the rest of the board, increasing the speed of the data transfer.

2. Considering the need for higher speed and flexibility, the control unit of the controller is microprogrammed.

3. Since the board supports 32 bit operations it was found most appropriate to use 32 bit microprocessor. Apart from increasing the speed this approach also allows inclusion of the compiling and self-testing facilities. This might also be used in the future for any expansion of the system.

4. To provide operations of data transfer and instruction execution to be carried out in parallel, two memory banks are provided. One is controlled by the control unit for SNAP primitive execution, the other is controlled by the VME interface to allow data transfer to and from the SNAP controller.

5. To provide efficient node management it was necessary to use a Content Addressable Memory as node memory. The required CAM is implemented using static RAM. This works at the same speed and reduces the cost significantly.

6. Similar to the VME interface, the SNAP interface is also implemented fully in hardware, resulting in a higher data transfer rate.

3 Hardware Description-Block Diagram

The block diagram of the SNAP array is shown in the figure 2. Each block of the circuit is described briefly below. A detailed hardware description will

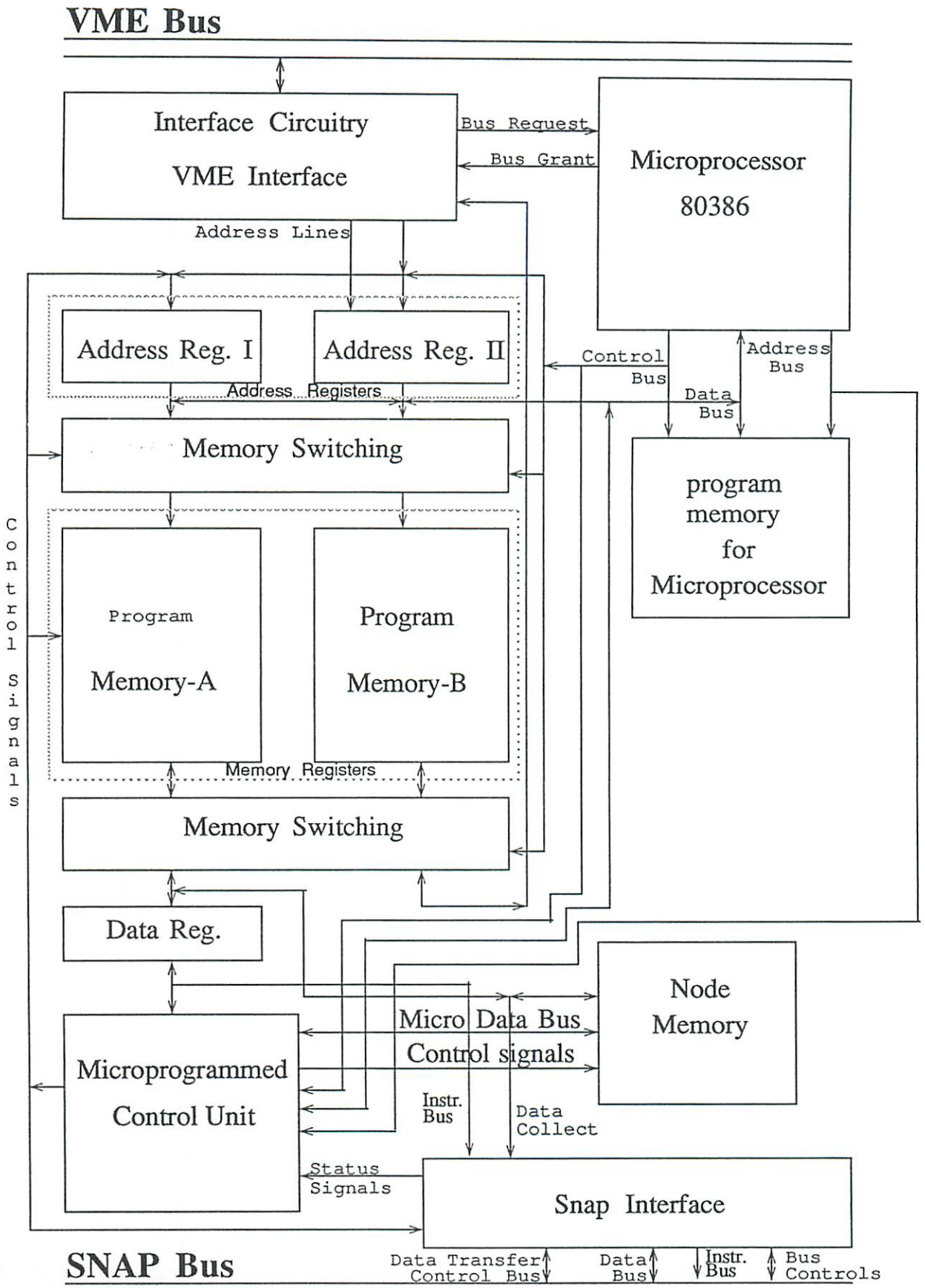


Figure 2 - SNAP Controller Block Diagram

be given after describing the operation of the SNAP controller in the next section.

3.1 VME Interface

This part of the controller consists mainly of circuits supporting the VME interface of the controller. It performs the following functions:

1. After receiving address location on the address bus assigned to the SNAP controller, and after checking the correct data transfer mode, this circuit generates an onboard bus request signal indicating to the microprocessor that the HOST wants to read or write data.
2. It checks for the bus grant signal from the microprocessor after giving a bus request. On receiving the bus grant, it enables the data transfer on the VME Bus, providing the control signals for the data transfer. The Quad Block Transfer mode of the VME Bus is used for data transfer.
3. It generates the interrupt request on the assigned intrrupt level and also facilitates transfer of the status-ID.
4. It interfaces the SNAP controller board to the VME Bus by using appropriate driver and receiver circuits.

3.2 Address Registers

These registers are used as the address registers for the memory banks. Since two memory banks are used, two address registers are required.

3.2.1 Address Registers I

These are the address registers used by the control unit to access the program memory currently being executed. The control signals provided to these registers are either from the control unit (while executing the program) or from the microprocessor (for initialising and for reading and writing the address).

There are two registers used here; one for the program address and the other for the result address. Before giving the Begin signal to the Control Unit, the microprocessor writes the beginning address of the program,to the program address register and the result address, where the result of all collect operations are going to be stored.

The program register counter is incremented by the control unit, while the result address register is incremented by the SNAP interface circuit.

3.2.2 Address Register II

This address register is controlled by the VME interface circuit after bus grant signal is given to the circuit by the microprocessor. This register is used solely for data transfer to or from the SNAP controller.

3.3 Memory Switching

This circuit allows the simultaneous execution and data transfer. According to the control signal given by the microprocessor it connects both the address registers to their allotted memory bank. The second circuit does the same thing for the data of the memory chips, except in this case it outputs the data of each memory bank to associated data registers.

3.4 Program Memory

As explained earlier this part is divided into two banks, Program Memory-A and Program Memmory-B. According to the controls given by the microprocessor, one of these memories is accessed by the control unit to execute the SNAP primitives, while the other is used by the VME interface for data transfer. These are static memory 256k byte, each and are managed by the microprocessor.This ensures that memory space occupied by the board on the VME Bus is only 256k byte.

3.5 Control Unit

This is a microprogrammed control unit, implemented by using a bit slice microsequencer, a control memory, an opcode map, a status register, along with associated MUX circuits for branching. This unit performs the following function:

- Generates control signals for the proper execution of each SNAP primitive.
- Breaks down each instruction in a set of microinstructions which perform the operations related to the Node management, variable binding,

collect execution, and controls corresponding to the different execution sequence,branching, etc.

- Indicates the microprocessor end of the execution or an overflow, if any.
- Facilitates execution of any Pseudoinstructions to the controller and generates controls for them.

3.6 Node Memory

This is one of the most important unit because it contains the Node-Memory Map, indicating which Node-ID is assigned to which Node-name and how many relations are associated with each Node-ID. It also contains information about the chip count (number of relations assigned with each chip), and gives an overflow signal calling for remedial actions. The action of this memory is totally controlled by the control unit.

The node memory is accessed each time with different arguments (Node-ID, Node-name and counter). Considering the fact that the operation involved is essentially searching, it is proposed to use a Content Addressable Memory for the node memory part of the circuit. The chip counter, however, can be implemented using static memory. The node memory unit also contains the mask generating circuit, for the mask register and input field selector for the comparand register. Since there are so few possible combinations, these circuits require very few control signals from the CU.

An alternative arrangement can be obtained by emulating Content Addressable Memory with static memory. This involves breaking down the CAM in static Ram, fieldwise. Each field is implemented as an independent memory, having its own address field. This memory can be linked to get full function of a CAM. Since we have only three fields the worst delay time would be twice the time obtained with CAM.

This node memory is also connected to the Program Memory-1 where it can be transferred in case of a Pseudoinstruction execution. Here the node memory Map is stored in the Program Memory-1 from where it will be transferred to the host workstation after memory switching.

3.7 SNAP Interface

This section performs the function of communicating with the SNAP array through the SNAP Instruction Bus,SNAP Data Bus, SNAP Bus Control Bus and SNAP Data Transfer Control Bus. The instruction is transmitted to the

SNAP array through the instruction bus by latching the instruction on the instruction register from the data register. The data collect is initiated by a collect begin control provided by the CU. This will transfer the control of the Program Memory-1 to the SNAP interface. After packing it into 4 bytes and adding the variable name in the beginning of each data packet for each collect operation, it transfers the data collected byte by byte from the SNAP.

3.8 Microprocessor

The microprocessor performs the following important functions:

1. The main function of the microprocessor is supervising the operation of different units in the SNAP controller. This includes initiating the data transfer to and from the SNAP controller, initiating the Control Unit, and setting and keeping track of each unit by providing appropriate signals to each.

2. The microprocessor is responsible for the memory management aspect of the SNAP controller. It looks after the memory switching, provides the address registers with corresponding locations, ensures continuous execution of otherwise disjointed instruction sets of the same program, avoides any overwriting, keeps track of result storage locations, etc.

3. The microprocessor deals with undesirable conditions by providing remedial action required for a specific condition. For example, a chip count overflow can be handled by loading a microinstruction assigning the Node-name multiply. On more difficult conditions it can instruct the control unit to re-execute the program from a certain location after rectifying the problem. It can also send some error messages to the Host.

4. The microprocessor can be used to backtrack the program from a particular SNAP primitive instruction onwards. This is accomplished by simply starting at the last instruction of the block to be backtracked (negated) and negating all the instructions untill the beginning of the block. By negating an instruction, any effect created by execution of that instruction on the SNAP database would be removed. This proccess of negation should be done in exact reverse order to guarantee proper operation.

5. Microprocessor does the lowest level compilation in case of nested loops in the SNAP primitive instructions. This reduces the excessive time

taken during loading of such programs and results in savings in memory space.

6. The microprocessor can also contain a self-test program which is executed every time the board is reset, or after a certain period of time. This program can check the proper functioning of the SNAP controller or even the SNAP array.

4 Hardware Description

A detailed hardware description of the SNAP controller design is given below. The functions performed by each of the circuits is briefly explained.

4.1 VME Interface

The main operations done by the interface can be divided among four sections.

- Location Monitor
- Interrupter
- Data Transfer Unit
- Bus Error Generator

Because all these sections in the hardware design overlap, the following section describes the VME interface as a single unit.

The main function of the location monitor is to determine whether or not the data transfer cycle addresses the memory space on the board. The type of data transfer specified on the VME Bus is checked to make sure it is compatible with the capability of the SNAP controller board. If both this conditions are met, it generates a signal indicating that the board is going to be involved in the next data tansfer over the VME Bus. The location monitor also generates on board READ/WRITE* signals.

To transfer the results from the controller board to the host computer the controller generates an interrupt, the interrupt service routine of which, reads back the result from the controller board. The interrupter generates an interrupt on a prespecified interrupt line and waits for an IACKIN* from the VME Bus. Meanwhile it latches the lower three address bits to determine

VME Interface

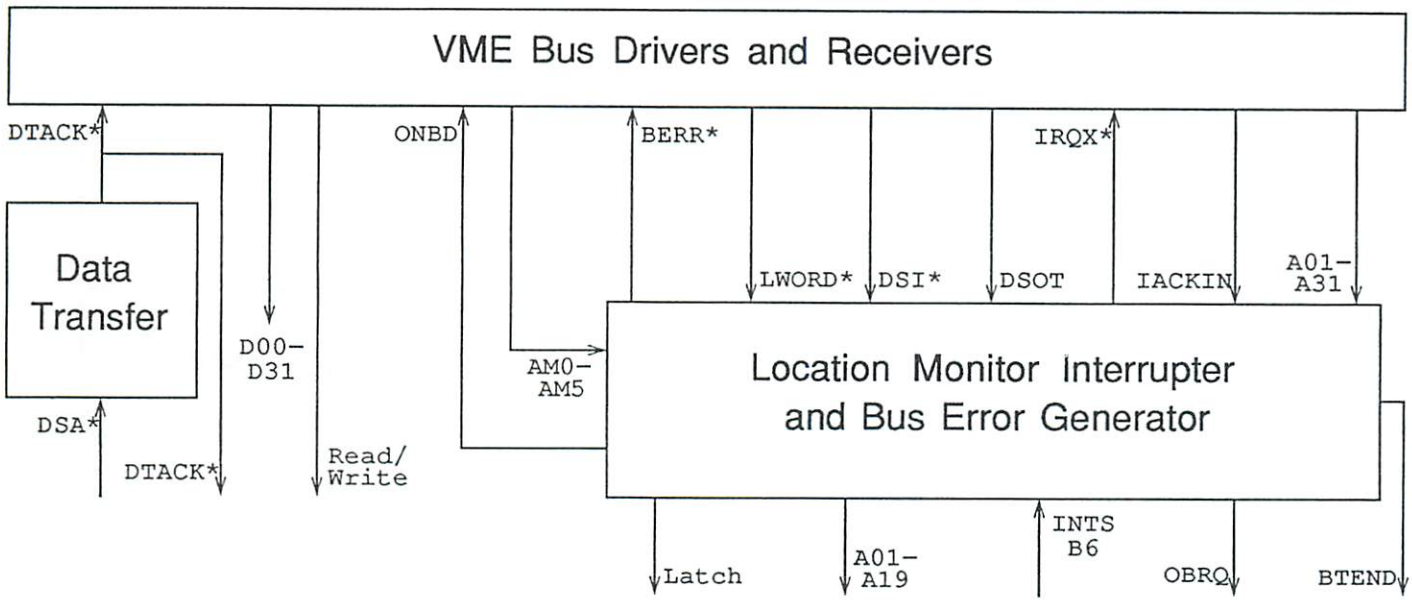


Figure 3: VME Interface

Address Registers

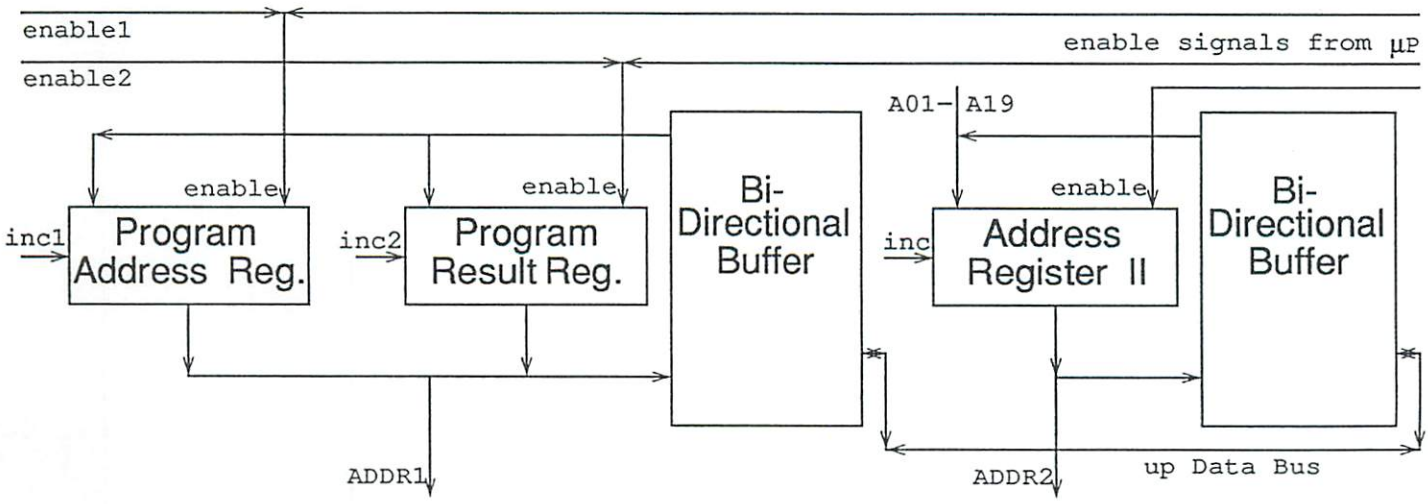


Figure 4: Address Registers

whether interrupt handler is responding to the same interrupt level. If so, and if the board receives an IACKIN* low it generates a signal requesting the status-ID transfer. After receiving the data, it puts the status-ID on the data bus and drives DTACK* low. If the interrupter operates in ROAK (Release On Acknowledge) mode, it can release the IRQ* line along with DTACK* going low.

The data transfer unit basically enables the data buffers after obtaining an ONBD signal, taking care of the direction of the data flow. It also generates a DTACK* signal after receiving DSA* low, and allows for the memory access time.

The bus error generator is used to drive the BERR* signal low in case of fault. It can drive the BERR* low if the DTACK* has not been low for more than 2T(called the timeout period), after receiving a DSA* low. It may also generate in case wrong data transfer type is indicated on the VME Bus, during the data transfer or interrupt acknowledge cycle.

The VME interface design is as follows:

All the signals are bufferred from the VME Bus by using the driver and receiver chips as shown. The specifications for SNAP controller are as follows.

None priviledged Extended Mode:

AM5, AM4, AM2 = 0; AM3, AM1, AM0 = 1

Quad Byte Block Transfer:

DS0*, DS1*, A01, LWORD* = 0

Slave Type: D32

Using these specifications the circuit was designed, the block diagram of which is as shown in Fig.3. The memory space occupied by the SNAP controller can be specified using straps or hardwired contacts. The same is also done for the interrupt level verifier. The on board request signal ORBQ is generated when in the data transfer mode(IACK* = 1), the address on the SNAP board is accessed, in the extended mode(EXTMODE= 1), or when in an interrupt acknowledge cycle the interrupt level match while IRQX*= 0 and IACKIN*= 0. The OBRQ signal is positive edge triggered and active high. This can be connected to the microprocessor interrupt input.

When a bus grant signal is received from the microprocessor, the interface circuit generates an ONBD signal indicating that the local data bus is now controlled by the interface circuit. The ONBD signal is also used to enable the data buffer the DTACK* generator. ONBD combined with DSA* gives a LATCH signal which is used as a trigger signal for the DTACK* monoshot.

The DSA* generated as shown is used to increment the address register along with the ONBD signal. AS* signal is used to latch the address so that the OBRQ remains enabled during the whole data transfer.

Another interrupt, BTEND, indicating end of transfer, is given to the microprocessor when the ONBD signal is high and OBRQ goes low.

4.2 Address Registers

As described earlier this unit addresses the two memory banks. Address Register-I can be enabled by the microprocessor or the CU. Address Register-II can be enabled by the microprocessor or the VME interface.

The control unit accesses the program address register (which is also a counter) to increment it each time one instruction execution is completed. The program result register is disabled while the instruction execution is taking place. On encountering a collect or read operation the CU enables the program result register. The program result register is incremented every time four data bytes are transferred to the SNAP controller.

Similarily the Address Reg. II is enabled by the microprocessor on reception of a OBRQ from the VME interface. Once the data transfer has started the counter is incremented by the LATCH signal generated by the SNAP interface.

The bidirectional buffer shown in the diagram allows the microprocessor to access these registers to read or to write. This is required for the memory management capability of the microprocessor.

4.3 Memory Switching

As explained earlier this unit switches the memory units, depending upon the select signals provided by the microprocessor. These circuits, as described below, can be divided into two sections.

4.3.1 Memory Switching I

This circuit switches the addresses provided from the address registers to the appropriate memory bank. As shown in the Figure 5 it consists of 2:1 Muxes. This multiplexer selects the input lines which are to be placed on the output lines depending on the select signals generated by the select register(in the control status Register).

The address decoding circuits generate the chip select signals for the program memory.

The write enable generator generates WE2* for the Program Memory-B. It can be a simple NOR gate enabled by a low on the Read/Write* signal.

4.3.2 Memory Switching II

This section of the circuit consists of a pair of bi-directional buffers, which connect the data outputs of the program memory either to the IRData register or to the VME interface. Again the select signal provided by the microprocessor is used.

4.4 Program Memory

The program memory is divided in two sections of 256k byte, i.e. 64k*32 bits, each. One of the memory banks can be used by the CU to execute the program stored earlier, while the second memory bank can be used by the VME interface for program loading or for the result reading.

The access time of this memory is taken care of by the CU and the DTACK* generator monoshot.

4.5 Control Unit

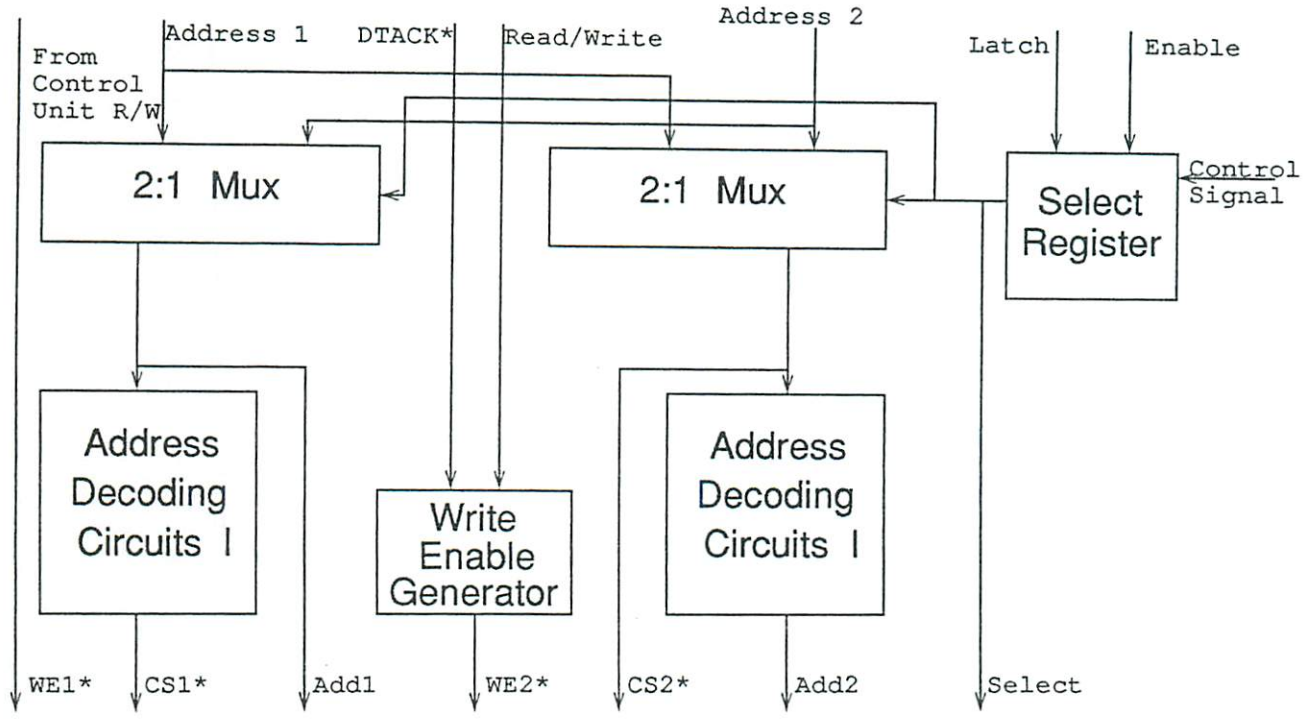
The CU is the most important unit of the SNAP controller. This is the unit which actually decodes the instructions and excutes them. The same section also performs the node manangement functions. The timing information of this unit is the predominant factor which affects the overall time taken by the SNAP system. Therefore, it is required that CU operate at high speed. This consideration explains why the control unit is implemented using microprogrammed logic.

The CU is described in three sections. The first section describes functional requirement for execution of each instruction. The second section describes the sequence of events (Program flow) for the CU. The third section describes the hardware implementation.

4.5.1 Functional Requirements

For execution of each instruction , the following operations are required:

Memory Switching I



Memory Switching II

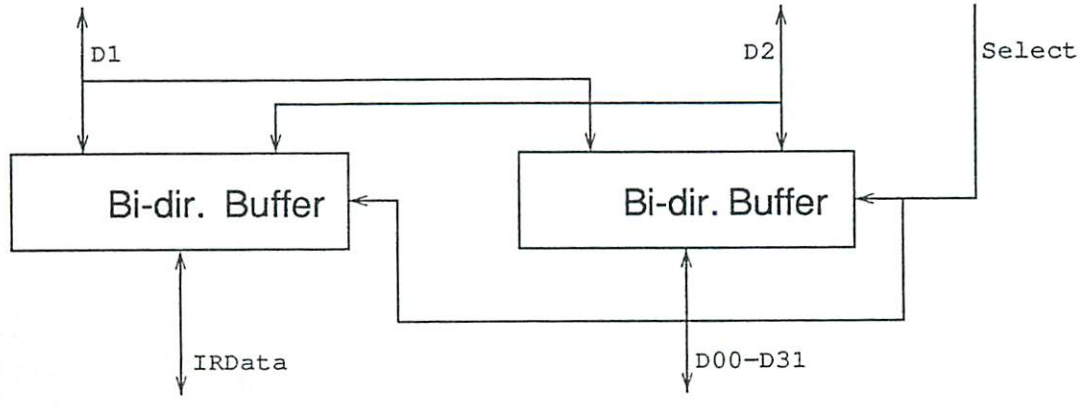


Figure 5: Memory Switching

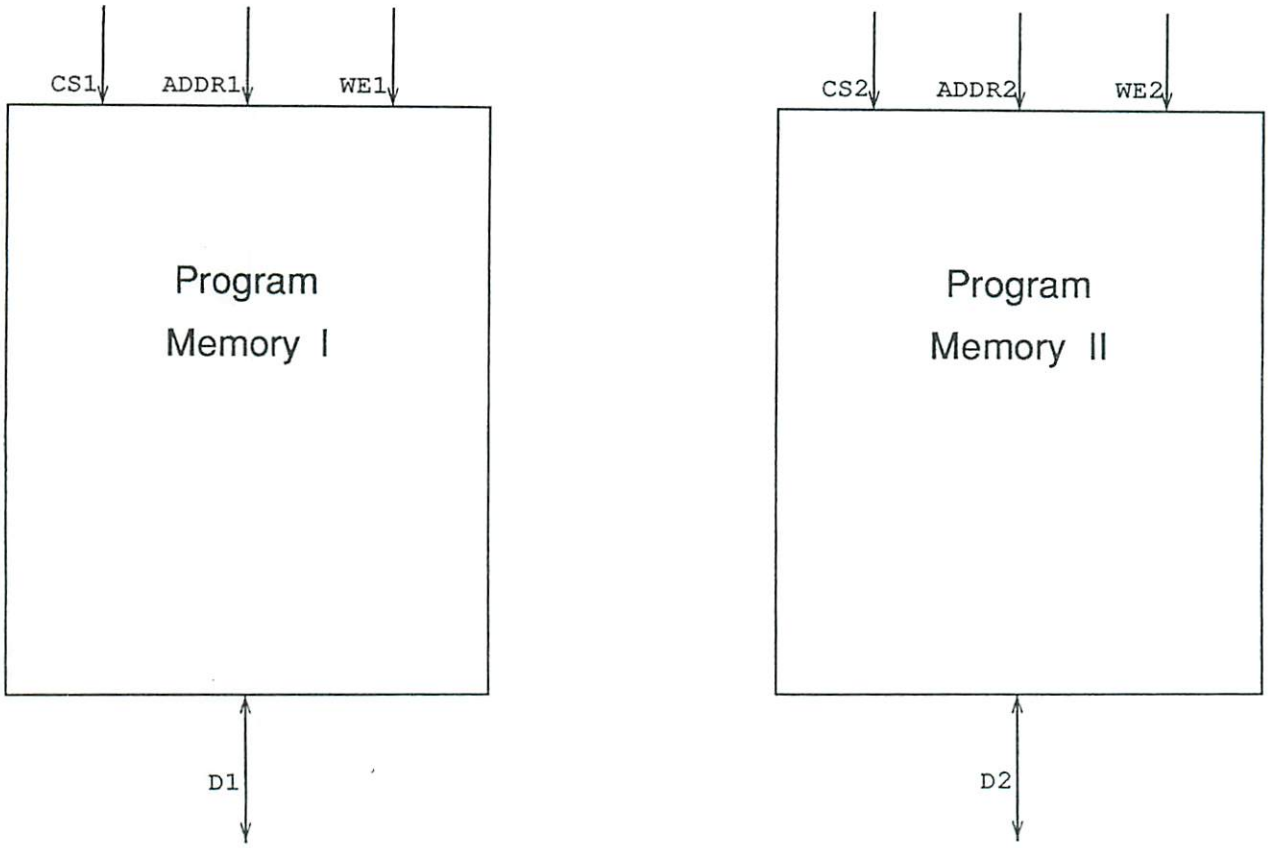


Figure 6: Program Memory

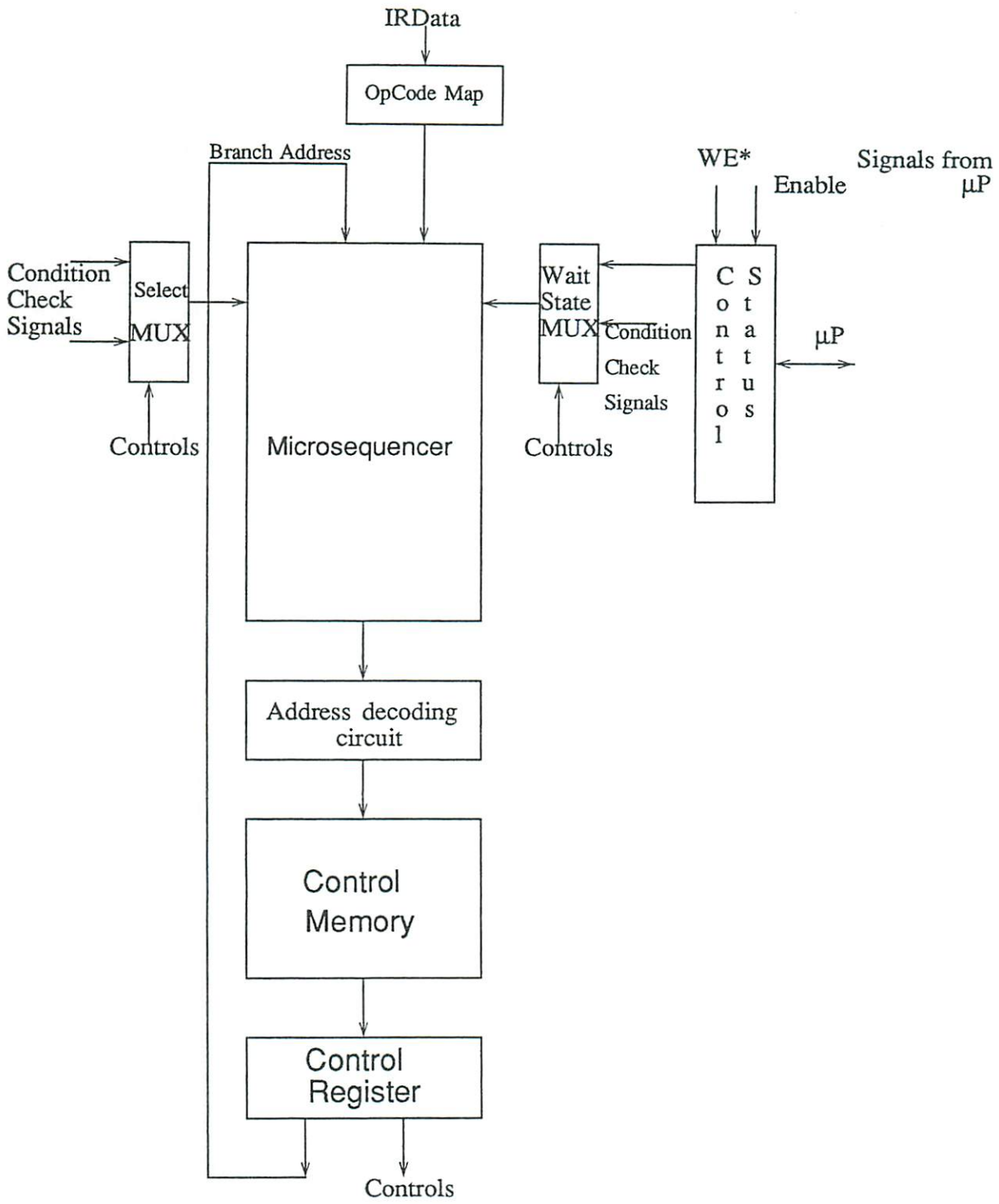


Figure 7: Control Unit

CREATE <node 1>, <relation>, <node 2>

We need to transmit two CREATE instructions to the array, namely CREATE <node 1>, <F-relation>, <node 2> and CREATE <node 2>, <R-relation>, <node 1>.

1. • Check in the controller CAM to determine if the nodes are allocatted Node-ID's. If not, assign them Node-ID's.
   • If all the location in the controller CAM are assigned then the controller invokes garbage collection subroutine (described in the next section).
2. • Convey this information to the SNAP array to set corresponding nodes.
   • If the chip in which the node is selected does not have any memory space, the Controller should reallocate the node.
   • Increment the relation counter associated with that node and the chip by one.
3. If all the nodes are alloted then send a message to the user.
4. Transmit the instruction, CREATE <node 1>, <F-relation>, <node 2>, on the Global Instruction Bus.
5. Transmit the instruction, CREATE <node 2>, <R-relation>, <node 1>, on the Global Instruction Bus.

DELETE <node 1>, <relation>, <node 2>

Here also, like CREATE, we have to transmit two instructions to the SNAP array.

1. Check if both nodes exist. If not, give error message.(Can be done in the software domain.)
2. Separately both the instructions to the SNAP array.
3. Reduce the counter of both nodes and chips associated with them by one.

COLLECT [marker #], <variable>

1. Generate a table of Node-IDs denoting it as "variable"

2. Transmit the instruction to the SNAP array.

3.
   - If only one response, take the Node-ID of the node and store it in the "variable" table.
   - If more than one response, keep storing the Node-IDs in the "variable" table.

COLLECT-COLOR [marker #],<variable>

Same as above.

COLLECT-RELATION [marker #], <variable>

Same as above ,but now the table contains the Node-color and relations.

READ [marker #], [register #]

1. Generate a temporary table.
2. Store in the table the values transferred from the SNAP.
3. Send the results to the USER.

All other instructions need no action from the controller. These instructions are transmitted directly to the SNAP array.

4.5.2 Hardware

The CU hardware is a typical microprogrammed unit hardware. The OpCode Map shown maps the opcode to its corresponding control memory location. The control status register is used by the microcontroller to indicate to the microprocessor different conditions such as chip overflow, delete error, execution complete, etc. It is also used to give a begin signal to the CU. The two multiplexers shown here are controlled by their corresponding control fields. The select mux is used to implement conditional jumps, while the wait state Mux is used to allow CU to remain idle till some condition is met.

4.5.3 Software

The software part of the control unit is described in section "Instruction Execution", as it requires understanding of the total hardware. The software here essentially means the microprograms in the control memory.

4.6 Node Memory

This circuit contains node map of the SNAP array, and relations associated with each node and each chip. Two possible implementations are explained. One approach involves use of CAM, while the other involves use of static RAM emulating functions of a CAM. A conceptual CAM implementation is explained in brief. The actual design is obtained using static RAM.

4.6.1 CAM Implementation

CAM implementation is shown in Figure 8. Since there are only three fields in each location a two bit control field is sufficient. The argument select circuit selects the input arguments which are to be used to recollect the data. The chip count memory can be implemented using static memory. The address for this memory is going to be derived from the chip address bits of the Node-ID. The counter register and the relation field of the CAM output are implemented using counters as they need to be incremented and decremented. This implementation of the node memory optimizes the speed because, ideally speaking, a CAM is most suitable for search operation.

4.6.2 Static RAM implementation

This implementation is possible for this particular application, because it is possible to mimic the CAM by using the RAM, but still maintaining the speed obtained by using a CAM. This is achieved in a very interesting manner by dividing the node memory into three fields. Each of these three fields is then implemented by a separate memory unit, using appropriate fields as the address to these units. For example: CAM operation of finding out a Node-ID, given the Node-name, can be implemented by having a static Node-ID memory, which is addressed by the Node-name and vice-versa. This implementation gives us the same speed as the CAM, with the same amount of hardware.

The relation count part of the Node-memory is still to be implemented by either a CAM or should be controlled by an independent unit. The static RAM implementation cannot be extended to relation count since there is not always a one-to-one correspondence between nodes and number of relations associated with each of them.

The circuit is shown in the Figure 9. Since the counter part of the node memory requires a continuous search for empty nodes it is controlled by the microprocessor. The rest of the circuit is controlled by the control unit.

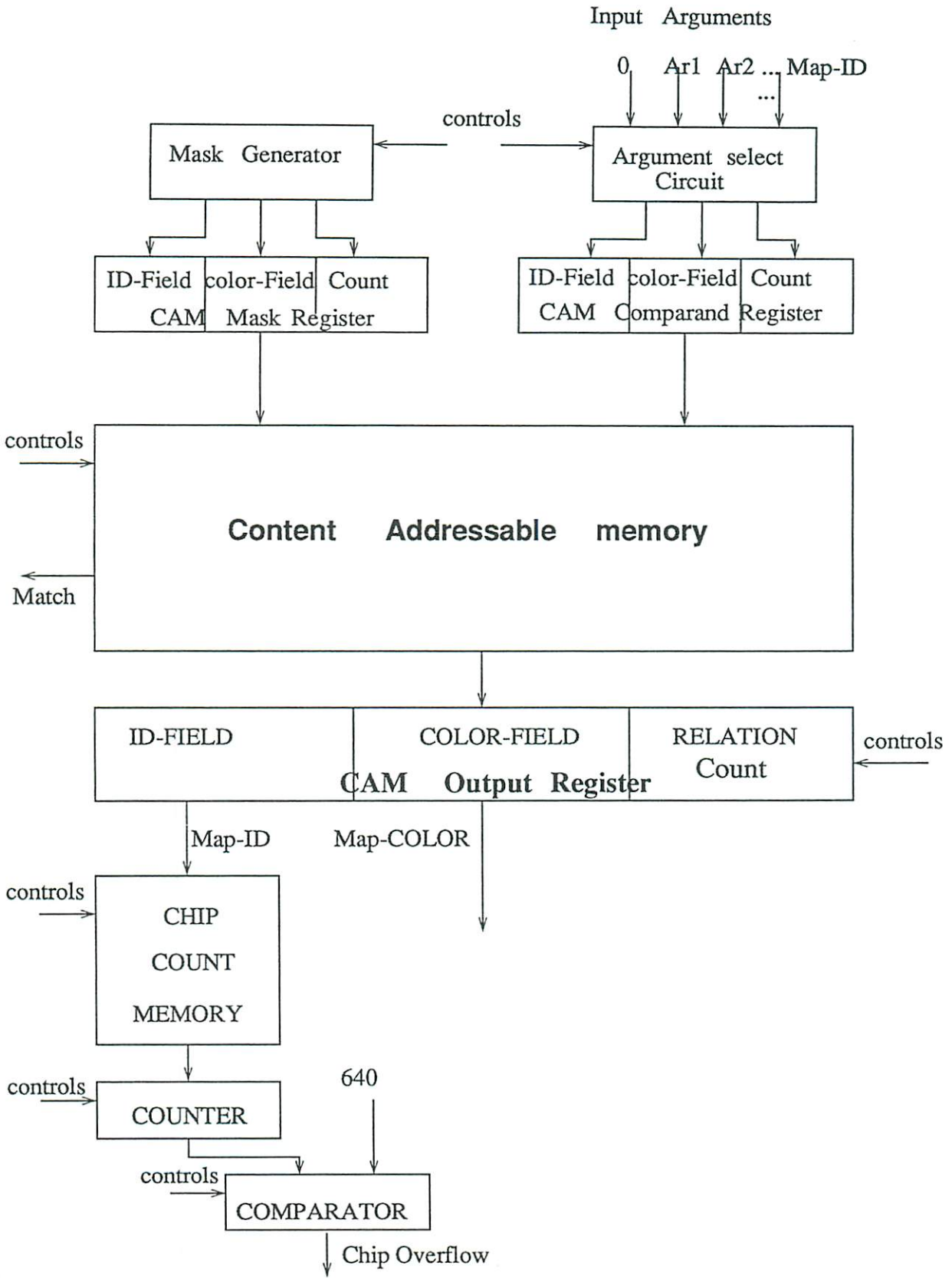


Figure 8: Node Memory

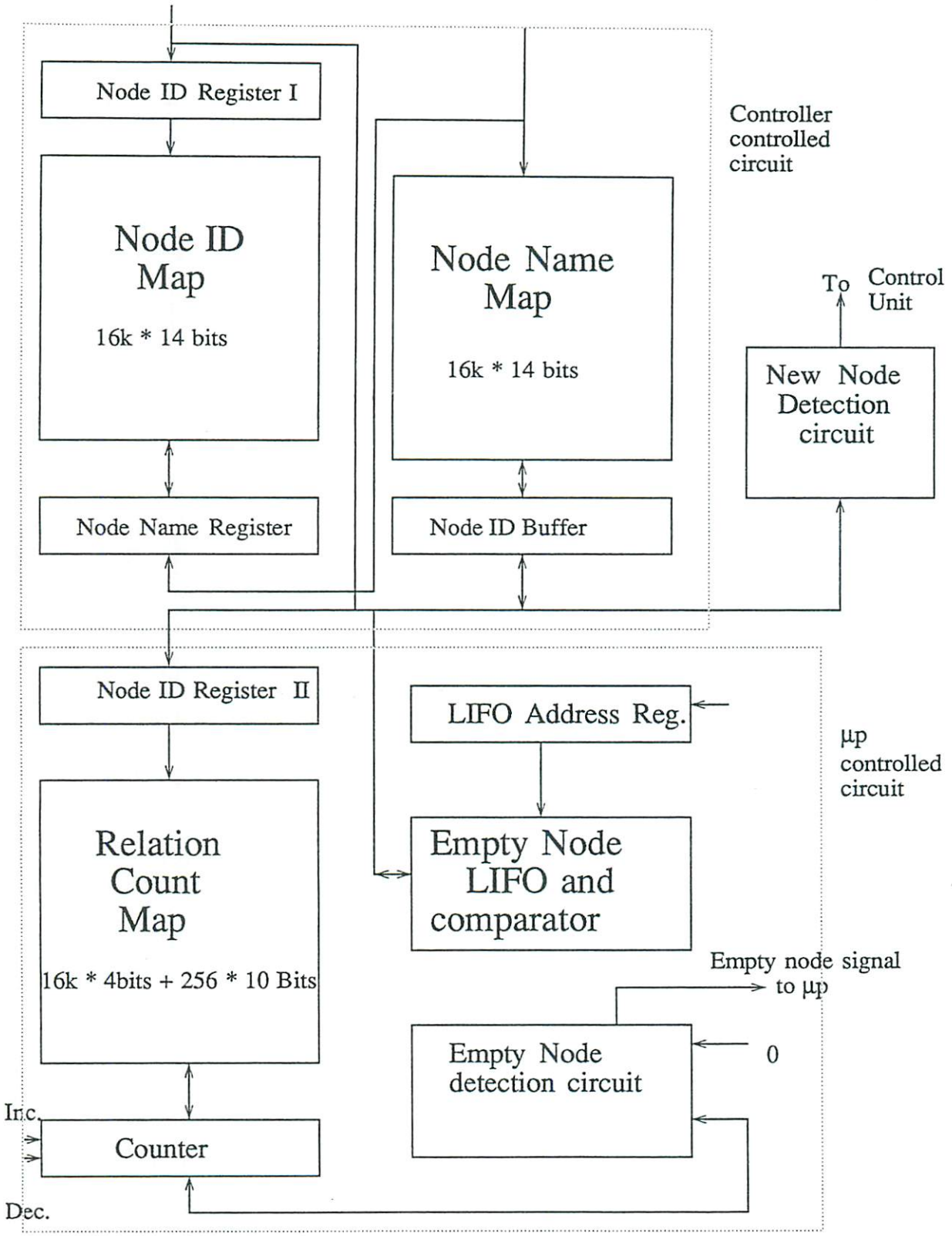


Figure 9: Node Memory (RAM implementation)

While the relation count map is free the onboard controller traverses through the memory to detect empty nodes and stores the Node-ID in the empty node LIFO.

During the execution the control unit obtains the Node-ID of the node specified in the SNAPRIM by using the Node-name map. If the node is not assigned it is detected by the new node detection circuit. This is made possible by the fact that this memory is reset, in the beginning, to zero. Therefore, if we obtain Node-ID zero for a Node-name, the Node-name in question is not yet assigned a Node-ID. This, unfortunately, also means that we can not assign node 0 to any Node-name.

The control unit, on the other end, takes the empty nodes sorted out by the microprocessor from the LIFO and assignes the new Node-name to that Node-ID. With this exception, the control unit operates in the same way as CAM.

4.7 SNAP Interface

The main function of the SNAP interface is to transmit the instruction on the SNAP bus and Collect the data requested from the SNAP array in an orderly manner.

The instruction transmission part is performed by the instruction buffer under the control of the CU. The opcode and the argument are selected by the CU in the argument select section. To take care of the lengthy time taken by certain SNAPRIMs, an instruction queue in ther SNAP interface circuit, is provided. The length of the queue can be determined by calculating the average instruction execution time on the SNAP and then dividing the maximum instruction execution time taken by any instruction on the SNAP controller by the average instruction execution time. This measure is taken to guarantee that SNAP controller is not a bottleneck to the SNAP array.

The bus control circuit gives out a bus grant signal(BG) if there is a bus request signal coming in(BRIN) from the SNAP array, in response to a data collect instruction, if the CU has given the control of the data bus to the SNAP interface.(In form of collect begin signal).

The timing specifications for the data transfer are as shown in the Figure 11. As shown in the timing diagram SDTA (SNAP Data Acknowledge) is given to the SNAP after the data is latched in the progarm memory. SDTA is generated after the access time of the memory and SDTA itself is used to latch the data in the memory. The SDTA generator circuit generates SDTA. The same circuit also generates a collection Complete signal when no DR (

Data Ready) signal arrive from the SNAP array for more than Ts. Ts is the time out period as shown in the Figure 11.

The collect Status generator generates the CC/CB* signal depending on the collect begin and collect complete signals.

The unidirectional data register byte serial to parallel circuit converts four 8 bit data into 32 bit data. It contains three 8 bit registers each latched consecutively by three DR signals. The fourth DR signal generates RLATCH which is used to latch all the 32 bits to the program memory.

All the signals coming from the SNAP array and transmitted to the SNAP array are passed through driver and receiver circuits as shown in Figure 10.

4.8 Microprocessor and related Memory

The reason for using 80386 for our pupose has been explained. As mentioned earlier, the main function of the microprocessor is that of memory management, minor compilation, VME Bus transfer initiation and self-testing capabilities. It can also involve itself with error handling capabilities. The section is implemented using the microprocessor, program memory(ROM), RAM, and corresponding address decoding circuits.

The OBRQ comes as an interrupt to the microprocessor. The Interrupt subroutine may be executed as shown below.

OBRQ:

1. If WRITE* is high then go to COLLECT
2. Reads in the address to which the SUN host wants to transfer the data. (It can safely be assumed that the AS* has gone low already latching the address in the address register II.
3. It compares the address with the presently allocated program boundaries and drives BERR* if address lies between any of these boundries.
4. Repeats the same for address reg + 256.
5. If none of the above is true, it stores the address as another program boundary and gives BG signal on the status register.

BTEND:

1. On reception of BTEND the microprocessor reads in the Address register II.

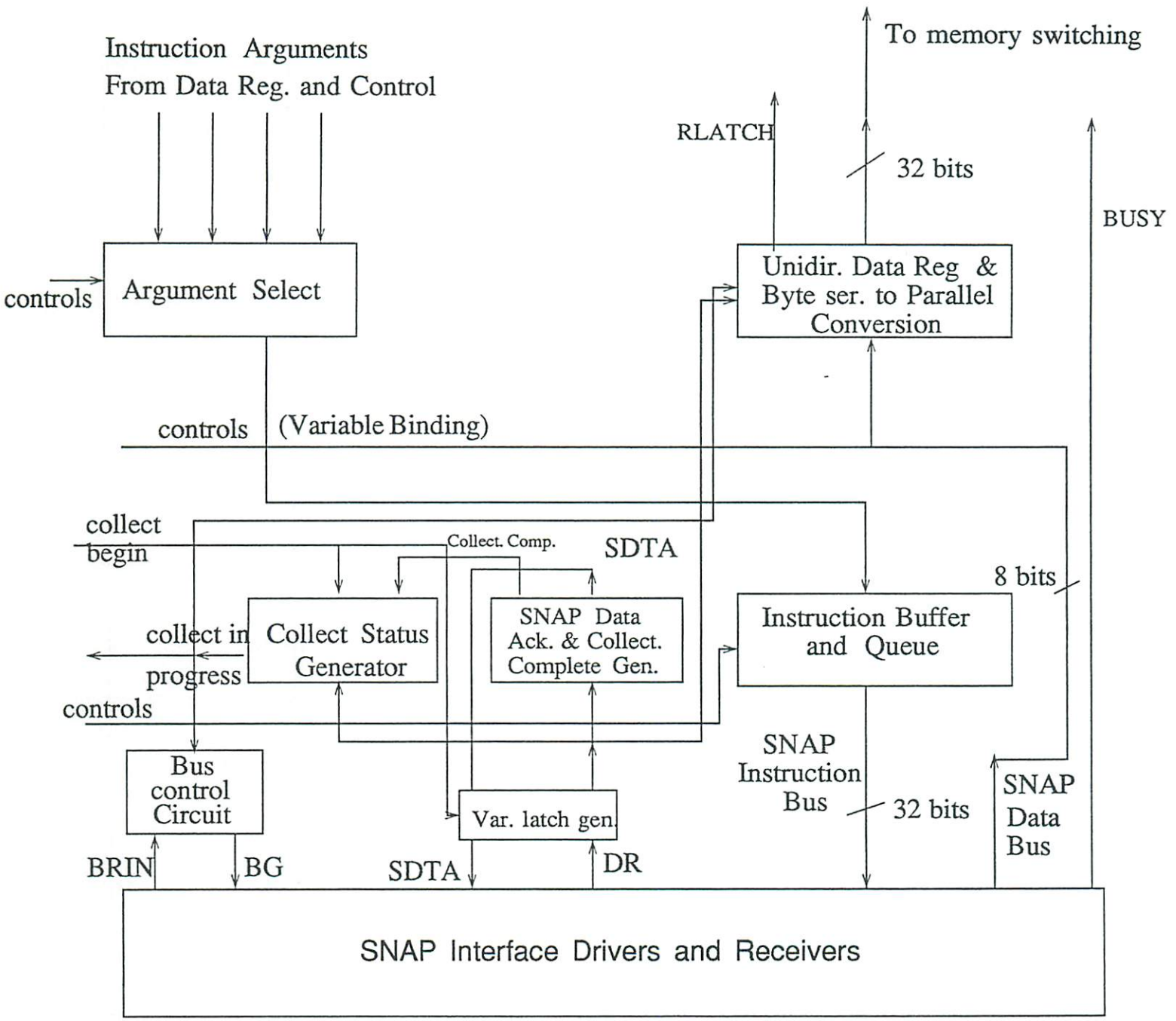


Figure 10: SNAP Interface

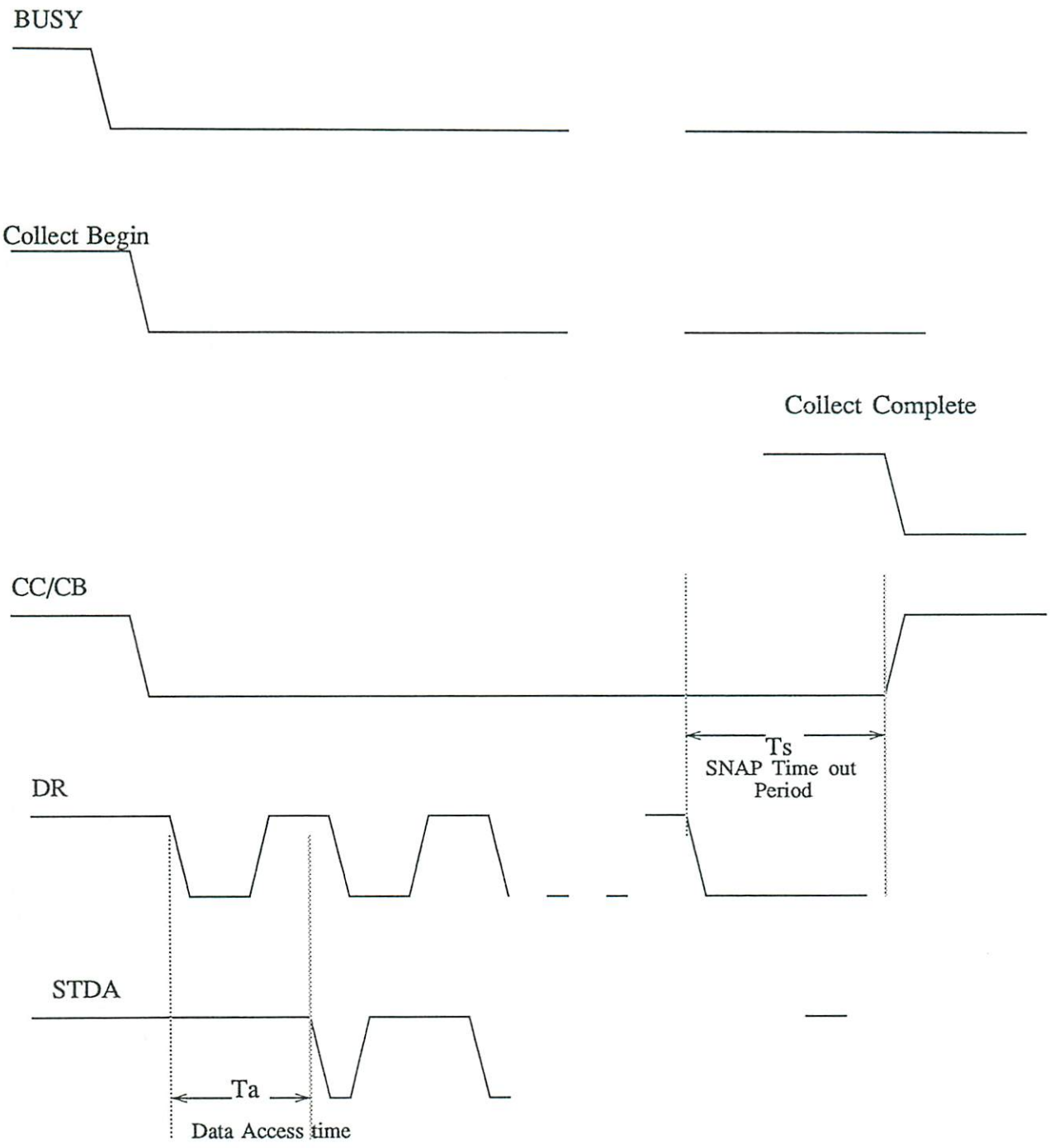


Figure 11: SNAP Interface signals

2. If the data transfer was a read operation then go to COLEND.

3. Using the address value create the upper boundary of this program segment.

COLLECT:

1. Ignore the address latched in the address register and latch in the lower bound of the result space created as a result of previous program execution.

2. Give the BG signal to the VME interface.

COLEND:

1. It reads in the Address Register II, and fixes new upper bound on the result space.

2. If a result space still exists it gives another VME interrupt to the host.

Similarly, the microprocessor controls the sequence of program execution by the CU. It feeds in the initial Address in the Program Address register, giving a begin signal to the CU. It also loads the location in which the results can be stored. When an execution complete signal is received, it again reads in both these registers and determines new program space and result space boundries.

5 Instruction Execution

Each instruction execution in the SNAP Controller and therefore, in the control unit can be divided into two cycles. The first cycle is the instruction fetch cycle. The instruction is brought to the instruction register from the program memory assigned using the address from the program address register. In the second cycle the instruction is executed by decoding it in a sequence of microinstructions which are executed sequencially.

This can be implemented as shown below. Attention is drawn to the fact that a given sequence of events is not optimized and few of the operations can be performed in parallel, thus reducing the execution time.

Certain modification required for the control, taking into account the static memory implementation of the node memory are given along with, denoted by '**'.

We can also pipeline the FETCH and EXECUTION CYCLES, if it does not increase the control memory size significantly.

The approximate timing information is provided for execution of each instruction in terms of the sequencer clock period. Also to be noted here is the fact that AMD2909 has a maximum clock frequency of 25MHZ.

FETCH CYCLE

Since all the instruction decoding is preceded by instruction fetch, the given sequence of events will take place before every instruction execution.

Sequence of events follows.

FETCH:

1. Keep idle till Begin/End* from the control status register HIGH.
2. • Enable the address decoder.
   • Drive R/W* HIGH.
3. Wait till the memory access time.
4. • Latch the instruction into the data register.
   • Enable the OpCode Map.
5. • Latch the control memory address given out by the OpCode Map into the address register of the sequencer.
   • Disable the OpCode Map.
   • Increment the address register.
   • Select the register R in the Sequencer (AMD2909).

After these four microinstructions the instruction register takes over and, depending on the instruction, corresponding sequence of operations take place as described below.

EXECUTE CYCLE

Each instruction is executed in microinstructions in the sequence given below.

CREATE <node 1>, <relation>, <node 2>

CREATE:

NODE1:

1. • Generate mask pattern for Node-name in mask field.
   • Select node 1 as the input to the comparand register.
2. Latch the argument in the comparand register.
3. Enable the CAM.
4. Wait for a MATCH or CTO (CAM Time Out, in case of no match).
5. If MATCH go to MATCH1, else go to CTO1.

MATCH1:

1. Read the data in CAM Register.
2. Increment the relation counter in the CAM register.
3. Enable the CHIP count memory
4. Latch the chip count output
5. • Increment the chip count register
   • Enable the comparator.
6. If Chip overflow, then go to NEW.
7. Write the chip count register back to the memory.
8. Write the CAM register back to the CAM.
9. Go to NODE2.

CTO1:

1. • Generate mask pattern for relation counter in mask field.
   • Select Relation Counter= 0 as the input to the comparand register.
2. Latch the argument in the comparand register.
3. Enable the CAM.
4. Wait for a MATCH or CTO (CAM Time Out, in case of no match).
5. If MATCH go to MATCH2, otherwise go to CTO2.

MATCH2:

1. • Disable the MATCH from selecting.
   • Read the data in the CAM register.
2. Enable the node 1 to the Node-name of the CAM register.
3. Latch the node 1 in Node-name field of the CAM register.
4. Increment the relation counter.
5. Write the CAM register back in the CAM.
6. Go to NODE2.

CTO2:

1. • Signal overflow to the microprocessor.
   • HALT.

NODE2:

The same procedure shown above is followed here with the following changes.

1. node 1 is replaced by node 2.
2. Go to NODE2 instructions will be changed to CALL TRAN.
3. Another CALL TRAN for broadcast of the reverse relation Instruction.
4. Go to FETCH.

DELETE <node 1>, <relation>, <node 2>

DEL:

DNODE1:

1. • Generate mask pattern for Node-name in Mask field.
   • Select node 1 as the input to the comparand register.
2. Latch the argument in the comparand register.

3. Enable the CAM.

4. Wait for a MATCH or CTO (CAM Time Out, in case of no match).

5. If MATCH, go to MATCH3, otherwise go to CTO3.

MATCH3:

1. Read the data in CAM register.

2. Decrement the relation counter in the CAM register.

3. Write the CAM register back to the CAM.

4. Go to DNODE2.

CTO3:

1. Send a DE (delete error) signal to the microprocessor.

2. HALT.

DNODE2:

The same procedure shown above is followed here with the following changes.

1. node 1 is replaced by node 2.

2. Go to NODE2 instructions will be changed to CALL TRAN.

3. CALL TRAN to broadcast reverse relation Instruction.

4. Go to FETCH.

COLLECT [marker #], <variable>

Since the COLLECT oriented communication is performed by the hardware in the SNAP interface, the operation required from the CU is quite simple.

1. • Enable the result address register.
   • Disable the program address register.

- Enable BUSY* at the sequencer Cin input.

2. Latch the Variable name from the Data Register into the first location.

3. CALL TRAN.

4. - Send collect begin (CB) to the SNAP interface.
   - Enable CC/CB* at the Cin input of the sequencer.

5. Go to FETCH.

COLLECT-COLOR [marker #],<variable>

Same as above.

COLLECT-RELATION [marker #], <variable>

Same as above.

READ [marker #], [register #]

Same as above.

All the other instructions do not require any action of the SNAP controller and can be microprogrammed as given below.

REST:

1. CALL TRAN.

2. Go to FETCH.

Thus requiring about 6 clock cycles (4 for FETCH) for each instruction.

** Using the static RAM for the node memory the following changes in the microprogram are required. The changes are only in the execute phase of the instruction.

CREATE <node 1>, <relation>, <node 2>

CREATE:

NODE1:

1. - Enable node-1 as the Node-name memory address.
   - Enable Node-name map memory.

2. If NEW NODE, CALL NEWNODE, otherwise go to MATCH1.

MATCH1:

1. • Enable Node-ID buffer and latch it in Node-ID register-II.
   • Enable relation count map.
2. Latch the relation count and chip count in the counter.
3. If Overflow, HALT.
4. Increment the relation and the chip count.
5. Write the counts back in the relation count map.
6. CALL TRANS with F-relation enabled.
7. Go to NODE2

NODE2:

1. • Enable node-2 as the Node-name memory address.
   • Enable node-name map memory.
2. If NEW NODE, CALL NEWNODE, otherwise go to MATCH2.

MATCH2:

1. • Enable Node-ID buffer and latch it in Node-ID register-II.
   • Enable relation count map.
2. Latch the relation count and chip count in the counter.
3. If Overflow, HALT.
4. Increment the relation and the chip count.
5. Write the counts back in the relation count map.
6. CALL TRANS with R-relation enabled.
7. Go to FETCH.

NEWNODE:

1. • Latch the LIFO output in the Node-name map.

- Decrement the LIFO address register.
- Latch the LIFO output to the Node-ID register-I.
- Enable the Node-ID memory map.

2. Latch the Node-name associated with that Node-ID into the Node-name register.
   - Enable the Node-name register to the Node-name memory
   - Enable the Node-name memory.
3. Latch Zero in the Node-name memory, through the Node-ID buffer.
4. - Disable the Node-name register to the Node-name memory
   - Enable the Node-name memory.

DELETE <node 1>, <relation>, <node 2> DEL:

DNODE1:

1. - Enable node-1 to the Node-name map.
   - Enable Node-name map.
2. If NEWNODE, go to DE(Delete Error), otherwise go to DMATCH1.

DMATCH1:

1. - Latch the data in Node-ID register-II.
   - Enable relation count and chip count memory.
2. Latch the count in the counter.
3. Decrement both the counts.
4. - Write the counts back.
   - CALL TRANS with F-relation enabled.
5. Go to DNODE2.

DNODE2:

1. - Enable node-2 to the Node-name map.
   - Enable Node-name map.

2. If NEWNODE, go to DE(Delete Error), otherwise go to DMATCH2.

DMATCH2:

1. • Latch the data in Node-ID register-II.
   • Enable relation count and chip count memory.
2. Latch the count in the counter.
3. Decrement both the counts.
4. • Write the counts back.
   • CALL TRANS with R-relation enabled.
5. Go to FETCH.

DE:

1. Transmit the error code and HALT.

The remaining SNAPRIMs do not involve the node memory and therefore are essentially unchanged.

6 Performance Details

The approximate number of cycles required for the execution of each instruction is given below. This count is obtained by assuming that no wait cycles are required. Moreover, the time taken by the COLLECT and READ depends on the BUSY signal as well as the number of nodes responding. This is not considered here because it depends on the database rather than controller.

The timing given below are timings, which are expected to be obtained under the most adverse conditions. If a probability distribution of the occurence of each SNAPRIM is available, mean controller execution time can be obtained.

CREATE <node 1>, <relation>, <node 2>

Using CAM: Approx. 34 clock cycles. Using Static RAM: Approx. 22 clock cycles.

DELETE <node 1>, <relation>, <node 2>

Using CAM: Approx. 24 clock cycles. Using Static RAM: Approx. 14 clock cycles.

COLLECT [marker #], <variable>

COLLECT-COLOR [marker #],<variable>

COLLECT-RELATION [marker #], <variable>

READ [marker #], [register #]

Approx. 9 clock cycles each.

All other instructions do not require any action of the SNAP controller and can be microprogrammed as follows:

REST:

1. CALL TRAN.
2. Go to FETCH.

This requires about 6 clock cycles (4 for FETCH) for each instruction.

To ensure that SNAP controller does not prove to be a bottleneck to the SNAP array, an instruction queue is implemented in the SNAP interface. The length of this queue depends on the mean instruction execution time of the SNAP array and the worst execution time on the SNAP controller (Here 22 cycles, if RAM used for Node-memory). This facility demands that the queue be empty, in case broadcast of an instruction depends on the feedback provided by SNAP array(i.e. COLLECT instructions).

This pipeline effectively reduces the affect caused by SNAPRIMs requiring long microinstruction sequences to make it run in harmony with the SNAP Array.